Package ‘Matrix’

February 19, 2015

Version 1.1-5
Date 2015-01-18
Priority recommended
Title Sparse and Dense Matrix Classes and Methods
Author Douglas Bates <bates@stat.wisc.edu> and Martin Maechler
Maintainer Martin Maechler <mmaechler+Matrix@gmail.com>
Contact Doug and Martin <Matrix-authors@R-project.org>
Description Classes and methods for dense and sparse matrices and 
operations on them using LAPACK and SuiteSparse.
Depends R (>= 2.15.2), methods
Imports graphics, grid, stats, utils, lattice
Suggests expm, MASS
Enhances MatrixModels, graph, SparseM, sfsmisc
Encoding UTF-8
LazyData no
LazyDataNote not possible, since we use data/*.R *and* our classes
ByteCompile yes
BuildResaveData no
License GPL (>= 2)
LicenseNote The Matrix package includes libraries AMD, CHOLMOD, 
COLAMD, CSparse and SPQR from the SuiteSparse collection of Tim 
Davis. All sections of that code are covered by the GPL or 
LGPL licenses. See the directory doc/UFsparse for details.
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-01-18 19:49:49
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abIndex-class</td>
<td>4</td>
</tr>
<tr>
<td>abIseq</td>
<td>6</td>
</tr>
<tr>
<td>all-methods</td>
<td>7</td>
</tr>
<tr>
<td>all.equal-methods</td>
<td>7</td>
</tr>
<tr>
<td>atomicVector-class</td>
<td>8</td>
</tr>
<tr>
<td>band</td>
<td>9</td>
</tr>
<tr>
<td>bandSparse</td>
<td>10</td>
</tr>
<tr>
<td>bdiag</td>
<td>11</td>
</tr>
<tr>
<td>BunchKaufman-methods</td>
<td>13</td>
</tr>
<tr>
<td>CAex</td>
<td>14</td>
</tr>
<tr>
<td>cBind</td>
<td>15</td>
</tr>
<tr>
<td>CHMfactor-class</td>
<td>16</td>
</tr>
<tr>
<td>chol</td>
<td>19</td>
</tr>
<tr>
<td>chol2inv-methods</td>
<td>21</td>
</tr>
<tr>
<td>Cholesky</td>
<td>22</td>
</tr>
<tr>
<td>Cholesky-class</td>
<td>24</td>
</tr>
<tr>
<td>colSums</td>
<td>26</td>
</tr>
<tr>
<td>compMatrix-class</td>
<td>27</td>
</tr>
<tr>
<td>condest</td>
<td>28</td>
</tr>
<tr>
<td>CsparseMatrix-class</td>
<td>30</td>
</tr>
<tr>
<td>ddenseMatrix-class</td>
<td>31</td>
</tr>
<tr>
<td>ddiMatrix-class</td>
<td>32</td>
</tr>
<tr>
<td>denseMatrix-class</td>
<td>33</td>
</tr>
<tr>
<td>dgCMatrix-class</td>
<td>34</td>
</tr>
<tr>
<td>dgeMatrix-class</td>
<td>35</td>
</tr>
<tr>
<td>dgRMatrix-class</td>
<td>36</td>
</tr>
<tr>
<td>dgTMatrix-class</td>
<td>37</td>
</tr>
<tr>
<td>Diagonal</td>
<td>39</td>
</tr>
<tr>
<td>diagonalMatrix-class</td>
<td>40</td>
</tr>
<tr>
<td>diagU2N</td>
<td>42</td>
</tr>
<tr>
<td>dMatrix-class</td>
<td>43</td>
</tr>
<tr>
<td>dpoMatrix-class</td>
<td>44</td>
</tr>
<tr>
<td>drop0</td>
<td>46</td>
</tr>
<tr>
<td>dsCMatrix-class</td>
<td>47</td>
</tr>
<tr>
<td>dsparseMatrix-class</td>
<td>49</td>
</tr>
<tr>
<td>dsRMatrix-class</td>
<td>50</td>
</tr>
<tr>
<td>dsyMatrix-class</td>
<td>51</td>
</tr>
<tr>
<td>dtCMatrix-class</td>
<td>52</td>
</tr>
<tr>
<td>dtpMatrix-class</td>
<td>54</td>
</tr>
<tr>
<td>dtRMatrix-class</td>
<td>56</td>
</tr>
<tr>
<td>dtrMatrix-class</td>
<td>57</td>
</tr>
<tr>
<td>expand</td>
<td>58</td>
</tr>
<tr>
<td>expm</td>
<td>59</td>
</tr>
<tr>
<td>externalFormats</td>
<td>60</td>
</tr>
<tr>
<td>facmul</td>
<td>62</td>
</tr>
<tr>
<td>forceSymmetric</td>
<td>63</td>
</tr>
</tbody>
</table>
R topics documented:

- formatSparseM ........................................ 64
- generalMatrix-class .................................. 65
- graph-sparseMatrix .................................. 66
- Hilbert ................................................. 67
- image-methods ........................................ 68
- index-class ............................................ 70
- indMatrix-class ....................................... 70
- invPerm ................................................ 73
- is.na-methods .......................................... 74
- is.null.DN ............................................. 75
- isSymmetric-methods .................................. 76
- isTriangular .......................................... 77
- KhatriRao .............................................. 78
- KNex .................................................... 79
- kronecker-methods .................................... 80
- ldenseMatrix-class .................................... 81
- ldiMatrix-class ...................................... 82
- lgeMatrix-class ...................................... 83
- lsparseMatrix-classes ................................ 84
- lsyMatrix-class ...................................... 85
- ltrMatrix-class ...................................... 86
- lu ....................................................... 87
- LU-class .............................................. 89
- Matrix .................................................. 90
- Matrix-class ......................................... 92
- matrix-products ..................................... 94
- MatrixFactorization-class ......................... 96
- ndenseMatrix-class .................................. 97
- nearPD .................................................. 98
- ngeMatrix-class ...................................... 101
- nMatrix-class ........................................ 102
- nnzero .................................................. 103
- norm ................................................... 104
- nsparseMatrix-classes ............................... 105
- nsyMatrix-class ...................................... 107
- ntrMatrix-class ...................................... 108
- number-class ......................................... 109
- pMatrix-class ........................................ 110
- printSpMatrix ........................................ 112
- qr-methods ........................................... 114
- rankMatrix ............................................ 115
- rcond .................................................. 118
- rep2abI ............................................... 120
- replValue-class ...................................... 121
- rleDiff-class ......................................... 121
- rsparsematrix ........................................ 122
- RsparseMatrix-class .................................. 123
- Schur .................................................. 124
Class "abIndex" of Abstract Index Vectors

Description

The "abIndex" class, short for "Abstract Index Vector", is used for dealing with large index vectors more efficiently, than using integer (or numeric) vectors of the kind 2:1000000 or c(0:1e5, 1000:1e6).

Note that the current implementation details are subject to change, and if you consider working with these classes, please contact the package maintainers (packageDescription("Matrix")$Maintainer).

Objects from the Class

Objects can be created by calls of the form new("abIndex", ...), but more easily and typically either by as(x, "abIndex") where x is an integer (valued) vector, or directly by abIseq() and combination c(...) of such.

Slots

kind: a character string, one of ("int32", "double", "rleDiff"), denoting the internal structure of the abIndex object.

x: Object of class "numLike"; is used (i.e., not of length 0) only iff the object is not compressed, i.e., currently exactly when kind != "rleDiff".

rleD: object of class "rleDiff", used for compression via rle.
Methods

**as.numeric, as.integer, as.vector** signature(x = "abIndex"): ...
[ signature(x = "abIndex", i = "index", j = "ANY", drop = "ANY"): ...

**coerce** signature(from = "numeric", to = "abIndex"): ...

**coerce** signature(from = "abIndex", to = "numeric"): ...

**coerce** signature(from = "abIndex", to = "integer"): ...

length signature(x = "abIndex"): ...

**Ops** signature(e1 = "numeric", e2 = "abIndex"): These and the following arithmetic and logic operations are not yet implemented; see **Ops** for a list of these (S4) group methods.

**Ops** signature(e1 = "abIndex", e2 = "abIndex"): ...

**Ops** signature(e1 = "abIndex", e2 = "numeric"): ...

**Summary** signature(x = "abIndex"): ...

**show** ("abIndex"): simple show method, building on show(<rleDiff>).

**is.na** ("abIndex"): works analogously to regular vectors.

**is.finite, is.infinite** ("abIndex"): ditto.

Note

This is currently experimental and not yet used for our own code. Please contact us (packageDescription("Matrix")$Maintainer) if you plan to make use of this class.

Partly builds on ideas and code from Jens Oehlschlaegel, as implemented (around 2008, in the GPL’ed part of) package **ff**.

See Also

**rle** (base) which is used here; **numeric**

Examples

```r
showClass("abIndex")
ii <- c(-3:40, 20:70)
str(ai <- as(ii, "abIndex"))# note
ai # -> show() method

stopifnot(identical(-3:20, as(ab1seql(-3,20), "vector")))
```
Description

Generation of abstract index vectors, i.e., objects of class "abIndex".

`abIseq` is designed to work entirely like `seq`, but producing "abIndex" vectors.

`abIseq()` is its basic building block, where `abIseq1(n,m)` corresponds to `n:m`.

`c(x, ...)` will return an "abIndex" vector, when `x` is one.

Usage

```r
abIseq1(from = 1, to = 1)
abIseq (from = 1, to = 1, by = ((to - from)/(length.out - 1)),
       length.out = NULL, along.with = NULL)
```

```r
## S3 method for class 'abIndex'
c(...)
```

Arguments

- `from`, `to` the starting and (maximal) end value of the sequence.
- `by` number: increment of the sequence.
- `length.out` desired length of the sequence. A non-negative number, which for `seq` and `seq.int` will be rounded up if fractional.
- `along.with` take the length from the length of this argument.
- `...` in general an arbitrary number of R objects; here, when the first is an "abIndex" vector, these arguments will be concatenated to a new "abIndex" object.

Value

An abstract index vector, i.e., object of class "abIndex".

See Also

the class `abIndex` documentation; `rep2abI()` for another constructor; `rle` (base).

Examples

```r
stopifnot(identical(-3:20,
                    as(abIseq1(-3,20), "vector")))
```

```r
try( ## (arithmetic) not yet implemented
    abIseq(1, 50, by = 3)
)
```
**all-methods**  
"Matrix" Methods for Functions `all()` and `any()`

**Description**

The basic R functions `all` and `any` now have methods for `Matrix` objects and should behave as for `matrix` ones.

**Methods**

```r
all signature(x = "Matrix", ..., na.rm = FALSE): ...
any signature(x = "Matrix", ..., na.rm = FALSE): ...
all signature(x = "ldenseMatrix", ..., na.rm = FALSE): ...
all signature(x = "lsparseMatrix", ..., na.rm = FALSE): ...
```

**Examples**

```r
M <- Matrix(1:12 + 0, 3, 4)
all(M >= 1)  # TRUE
any(M < 0)  # FALSE
MN <- M; MN[2,3] <- NA; MN
all(MN >= 0)  # NA
any(MN < 0)  # NA
any(MN < 0, na.rm = TRUE)  # -> FALSE
```

**all.equal-methods**  
Matrix Package Methods for Function `all.equal()`

**Description**

Methods for function `all.equal()` (from R package `base`) are defined for all `Matrix` classes.

**Methods**

```r
target = "Matrix", current = "Matrix" \n
target = "ANY", current = "Matrix" \n
target = "Matrix", current = "ANY"  these three methods are simply using `all.equal.numeric`
directly and work via `as.vector()`.
```

There are more methods, notably also for "sparseVector"'s, see `showMethods("all.equal")`. 
Examples

showMethods("all.equal")

(A <- spMatrix(3, 3, i= c(1:3,2:1), j=c(3:1,1:2), x = 1:5))
ex <- expand(lu. <- lu(A))
stopifnot(all.equal(as(A@i + 1L, lu.@q + 1L), "CsparseMatrix"),
           lu.@U %*% lu.@U),
         with(ex, all.equal(as(P %*% A %*% Q, "CsparseMatrix"),
                        L %*% U)),
         with(ex, all.equal(as(A, "CsparseMatrix"),
                          t(P) %*% L %*% U %*% t(Q))))

atomicVector-class

Virtual Class "atomicVector" of Atomic Vectors

Description

The class "atomicVector" is a virtual class containing all atomic vector classes of base R, as also implicitly defined via is.atomic.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

In the Matrix package, the "atomicVector" is used in signatures where typically "old-style" "matrix" objects can be used and can be substituted by simple vectors.

Extends

The atomic classes "logical", "integer", "double", "numeric", "complex", "raw" and "character" are extended directly. Note that "numeric" already contains "integer" and "double", but we want all of them to be direct subclasses of "atomicVector".

Author(s)

Martin Maechler

See Also

is.atomic, integer, numeric, complex, etc.

Examples

showClass("atomicVector")
band

Extract bands of a matrix

Description

Returns a new matrix formed by extracting the lower triangle (\texttt{tril}) or the upper triangle (\texttt{triu}) or a general band relative to the diagonal (\texttt{band}), and setting other elements to zero. The general forms of these functions include integer arguments to specify how many diagonal bands above or below the main diagonal are not set to zero.

Usage

\begin{verbatim}
band(x, k1, k2, 
tril(x, k = 0, 
triu(x, k = 0, 
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{x} a matrix-like object
\item \texttt{k, k1, k2} integers specifying the diagonal bands that will not be set to zero. These are given relative to the main diagonal, which is \texttt{k=0}. A negative value of \texttt{k} indicates a diagonal below the main diagonal and a positive value indicates a diagonal above the main diagonal.
\item \ldots Optional arguments used by specific methods. (None used at present.)
\end{itemize}

Value

An object of an appropriate matrix class. The class of the value of \texttt{tril} or \texttt{triu} inherits from \texttt{triangularMatrix} when appropriate. Note that the result is of class \texttt{sparseMatrix} only if \texttt{x} is.

Methods

\begin{itemize}
\item \texttt{x = "CsparseMatrix"} method for compressed, sparse, column-oriented matrices.
\item \texttt{x = "TsparseMatrix"} method for sparse matrices in triplet format.
\item \texttt{x = "RsparseMatrix"} method for compressed, sparse, row-oriented matrices.
\item \texttt{x = "ddenseMatrix"} method for dense numeric matrices, including packed numeric matrices.
\end{itemize}

See Also

\texttt{bandSparse} for the \textit{construction} of a banded sparse matrix directly from its non-zero diagonals.
Examples

```r
## A random sparse matrix :
set.seed(7)
m <- matrix(0, 5, 5)
m[sample(length(m), size = 14)] <- rep(1:9, length=14)
(mm <- as(m, "CsparseMatrix"))

tril(mm)      # lower triangle
tril(mm, -1)  # strict lower triangle
triu(mm, 1)   # strict upper triangle
band(mm, -1, 2) # general band

(m5 <- Matrix(rnorm(25), nc = 5))
tril(m5)      # lower triangle
tril(m5, -1)  # strict lower triangle
triu(m5, 1)   # strict upper triangle
band(m5, -1, 2) # general band

(m65 <- Matrix(rnorm(30), nc = 5)) # not square
triu(m65)     # result in not dtrMatrix unless square

(sm5 <- crossprod(m65)) # symmetric
band(sm5, -1, 1) # symmetric band preserves symmetry property
as(band(sm5, -1, 1), "sparseMatrix") # often preferable
```

### bandSparse

**Construct Sparse Banded Matrix from (Sup-/Super-) Diagonals**

**Description**

Construct a sparse banded matrix by specifying its non-zero sup- and super-diagonals.

**Usage**

```r
bandSparse(n, m = n, k, diagonals, symmetric = FALSE, giveCsparse = TRUE)
```

**Arguments**

- `n, m`  
  the matrix dimension \((n, m) = (nrow, ncol)\).

- `k`  
  integer vector of “diagonal numbers”, with identical meaning as in `band(*, k)`.

- `diagonals`  
  optional list of sub-/super- diagonals; if missing, the result will be a pattern matrix, i.e., inheriting from class `nMatrix`.

  Diagonals can also be \(n' \times d\) matrix, where \(d \leftarrow \text{length}(k)\) and \(n' \geq \text{min}(n, m)\). In that case, the sub-/super- diagonals are taken from the columns of diagonals, where only the first several rows will be used (typically) for off-diagonals.

- `symmetric`  
  logical; if true the result will be symmetric (inheriting from class `symmetricMatrix`) and only the upper or lower triangle must be specified (via `k` and `diagonals`).

- `giveCsparse`  
  logical indicating if the result should be a `CsparseMatrix` or a `TsparseMatrix`. The default, `TRUE` is very often more efficient subsequently, but not always.
Value

a sparse matrix (of class CsparseMatrix) of dimension \( n \times m \) with diagonal “bands” as specified.

See Also

band, for extraction of matrix bands; bdiag, diag, sparseMatrix, Matrix.

Examples

diags <- list(1:30, 10*(1:20), 100*(1:20))
s1 <- bandSparse(13, k = -c(0:2, 6), diag = c(diags, diags[2]), symm=TRUE)
s1
s2 <- bandSparse(13, k =  c(0:2, 6), diag = c(diags, diags[2]), symm=TRUE)
stopifnot(identical(s1, t(s2)), is(s1,"dsCMatrix"))

## a pattern Matrix of *full* (sub-)diagonals:
bk <- c(0:4, 7, 9)
(s3 <- bandSparse(30, k = bk, symm= TRUE))

## If you want a pattern matrix, but with "sparse"-diagonals,
## you currently need to go via logical sparse:
llis <- lapply(list(rpois(20, 2), rpois(20,1), rpois(20,3))[c(1:3,2:3,3:2)],
              as.logical)
(s4 <- bandSparse(20, k = bk, symm = TRUE, diag = llis))
(s4. <- as(drop0(s4), "n sparseMatrix"))
n <- 1e4
bk <- c(0:5, 7,11)
bMat <- matrix(c(1:8, n, 8, byrow=TRUE)
blis <- as.data.frame(bMat)
B <- bandSparse(n, k = bk, diag = blis)
Bs <- bandSparse(n, k = bk, diag = blis, symmetric=TRUE)
B [1:15, 1:30]
Bs[1:15, 1:30]
## can use a list *or* a matrix for specifying the diagonals:
stopifnot(identical(B, bandSparse(n, k = bk, diag = bMat)),
identical(Bs, bandSparse(n, k = bk, diag = bMat, symmetric=TRUE)))

bdiag

Construct a Block Diagonal Matrix

Description

Build a block diagonal matrix given several building block matrices.

Usage

bdiag(...)
.bdiag(lst)
Arguments

... individual matrices or a list of matrices.
1st non-empty list of matrices.

Details

For non-trivial argument list, bdiag() calls .bdiag(). The latter maybe useful to programmers.

Value

A sparse matrix obtained by combining the arguments into a block diagonal matrix.
The value of bdiag() inheris from class CsparseMatrix, whereas .bdiag() returns a TsparseMatrix.

Author(s)

Martin Maechler, built on a version posted by Berton Gunter to R-help; earlier versions have been posted by other authors, notably Scott Chasalow to S-news. Doug Bates's faster implementation builds on TsparseMatrix objects.

See Also

Diagonal for constructing matrices of class diagonalMatrix, or kronecker which also works for "Matrix" inheriting matrices.

bandSparse constructs a banded sparse matrix from its non-zero sub-/super - diagonals.

Note that other CRAN R packages have own versions of bdiag() which return traditional matrices.

Examples

bdiag(matrix(1:4, 2), diag(3))
## combine "Matrix" class and traditional matrices:
bdiag(Diagonal(2), matrix(1:3, 3, 4), diag(3:2))

mlist <- list(1, 2:3, diag(x=5:3), 27, cbind(1,3:6), 100:101)
bdiag(mlist)
stopifnot(identical(bdiag(mlist),
    bdiag(lapply(mlist, as.matrix)))))

ml <- c(as(matrix((1:24)%% 11 == 0, 6, 4),"nMatrix"),
    rep(list(Diagonal(2, x=TRUE)), 3))
mln <- c(ml, Diagonal(x = 1:3))
stopifnot(is(bdiag(ml), "lsparseMatrix"),
    is(bdiag(mln),"dsparseMatrix") )
Bunch-Kaufman Decomposition Methods

Description
The Bunch-Kaufman Decomposition of a square symmetric matrix $A$ is $A = PLDL'P'$ where $P$ is a permutation matrix, $L$ is unit-lower triangular and $D$ is block-diagonal with blocks of dimension 1 $\times$ 1 or 2 $\times$ 2.

Usage
BunchKaufman(x, ...)

Arguments
- `x`: a symmetric square matrix.
- `...`: potentially further arguments passed to methods.

Value
an object of class `BunchKaufman`, which can also be used as a (triangular) matrix directly.

Methods
Currently, only methods for `dense` numeric symmetric matrices are implemented.
- `x = "dspMatrix"` uses Lapack routine `dsptrf`.
- `x = "dsyMatrix"` uses Lapack routine `dsytrf`, computing the Bunch-Kaufman decomposition.

References
The original LAPACK source code, including documentation: http://www.netlib.org/lapack/\ndouble/dsytrf.f and http://www.netlib.org/lapack/\ndouble/dsptrf.f

See Also
The resulting class, `BunchKaufman`. Related decompositions are the LU, `lu`, and the Cholesky, `chol` (and for `sparse` matrices, `Cholesky`).

Examples
```
data(CAex)
dim(CAex)
isSymmetric(CAex)# TRUE
CAs <- as(CAex, "symmetricMatrix")
if(FALSE) # no method defined yet for *sparse* :
  bk. <- BunchKaufman(CAs)
  # does apply to *dense* symmetric matrices:
```
Albers' example Matrix with "Difficult" Eigen Factorization

Description

An example of a sparse matrix for which `eigen()` seemed to be difficult, an unscaled version of this has been posted to the web, accompanying an E-mail to R-help (https://stat.ethz.ch/mailman/listinfo/r-help), by Casper J Albers, Open University, UK.

Usage

data(CAex)

Format

This is a $72 \times 72$ symmetric matrix with 216 non-zero entries in five bands, stored as sparse matrix of class `dgCMatrix`.

Details

In some versions of R, `eigen(CAex)` fell into an infinite loop (whereas `eigen(CAex, EISPACK=TRUE)` has been okay).

Examples

data(CAex)
str(CAex) # of class "dgCMatrix"

image(CAex)# -> it's a simple band matrix with 5 bands
## and the eigen values are basically 1 (42 times) and 0 (30 x):
zapsmall(ev <- eigen(CAex, EISPACK=TRUE, only.values=TRUE)$values)
## i.e., the matrix is symmetric, hence
sCA <- as(CAex, "symmetricMatrix")
## and
stopifnot(class(sCA) == "dsCMatrix",
  as(sCA, "matrix") == as(CAex, "matrix"))
cBind

Versions of 'cbind' and 'rbind' recursively built on cbind2/rbind2

Description

The base functions cbind and rbind are defined for an arbitrary number of arguments and hence have the first formal argument .... For that reason, S4 methods cannot easily be defined for binding together matrices inheriting from Matrix.

For that reason, cbind2 and rbind2 have been provided for binding together two matrices, and we have defined methods for these and the 'Matrix'-matrices.

As a substitute for S4-enabled versions of cbind and rbind, you can use cBind and rBind with identical syntax and semantic in order to bind together multiple matrices ("matrix" or "Matrix" and vectors.

Usage

```r
cBind(..., deparse.level = 1)
rBind(..., deparse.level = 1)
```

Arguments

... matrix-like R objects to be bound together, see cbind and rbind.
deparse.level integer determining under which circumstances column and row names are built from the actual arguments’ ‘expression’, see cbind.

Details

The implementation of these is recursive, calling cbind2 or rbind2 respectively, where these have methods defined and so should dispatch appropriately.

Value

typically a ‘matrix-like’ object of a similar class as the first argument in ....

Note that sometimes by default, the result is a sparseMatrix if one of the arguments is (even in the case where this is not efficient). In other cases, the result is chosen to be sparse when there are more zero entries is than non-zero ones (as the default sparse in Matrix()).

Author(s)

Martin Maechler

See Also

cbind2, cbind, Methods.
Examples

(a <- matrix(c(2:1,1:2), 2,2))
cbind(0, rBind(a, 7)) # remains traditional matrix

D <- Diagonal(2)
cBind(4, a, D, -1, D, 0) # a sparse Matrix

CHMfactor-class  

**CHOLMOD-based Cholesky Factorizations**

Description

The virtual class "CHMfactor" is a class of CHOLMOD-based Cholesky factorizations of symmetric, sparse, compressed, column-oriented matrices. Such a factorization is simplicial (virtual class "CHMsimpl") or supernodal (virtual class "CHMsuper"). Objects that inherit from these classes are either numeric factorizations (classes "dCHMsimpl" and "dCHMsuper") or symbolic factorizations (classes "nCHMsimpl" and "nCHMsuper").

Usage

`isLDL(x)`

```
## S4 method for signature 'CHMfactor'
update(object, parent, mult = 0, ...) .updateCHMfactor(object, parent, mult)

## and many more methods, notably,
## solve(a, b, system = c("A","LDL","LD","DLt","Lt","D","P","Pt"), ...)
## ------    see below
```

Arguments

- `x`, `object`, `a`  
  a "CHMfactor" object (almost always the result of `Cholesky()`).
- `parent`  
  a "dsCMatrix" or "dgCMatrix" matrix object with the same nonzero pattern as the matrix that generated object. If parent is symmetric, of class "dsCMatrix", then object should be a decomposition of a matrix with the same nonzero pattern as parent. If parent is not symmetric then object should be the decomposition of a matrix with the same nonzero pattern as tcrossprod(parent).
  Since Matrix version 1.0-8, other "sparseMatrix" matrices are coerced to `dsparseMatrix` and `CsparseMatrix` if needed.
- `mult`  
  a numeric scalar (default 0). mult times the identity matrix is (implicitly) added to parent or tcrossprod(parent) before updating the decomposition object.
- `...`  
  potentially further arguments to the methods.
Objects from the Class

Objects can be created by calls of the form `new("dCHMsuper", ...)` but are more commonly created via `Cholesky()`, applied to `dCMatrix` or `lCMatrix` objects.

For an introduction, it may be helpful to look at the `expand()` method and examples below.

Slots

of "CHMfactor" and all classes inheriting from it:

- **perm**: An integer vector giving the 0-based permutation of the rows and columns chosen to reduce fill-in and for post-ordering.
- **colcount**: Object of class "integer" ....
- **type**: Object of class "integer" ....

Slots of the non virtual classes “[dl]CHM(super|simpl)”: 

- **p**: Object of class "integer" of pointers, one for each column, to the initial (zero-based) index of elements in the column. Only present in classes that contain "CHMsimpl".
- **i**: Object of class "integer" of length nnzero (number of non-zero elements). These are the row numbers for each non-zero element in the matrix. Only present in classes that contain "CHMsimpl".
- **x**: For the "d*" classes: "numeric" - the non-zero elements of the matrix.

Methods

- **isLDL** *(x)* returns a **logical** indicating if *x* is an $LDL'$ decomposition or (when FALSE) an $LL'$ one.
- **coerce** signature(from = "CHMfactor", to = "sparseMatrix") (or equivalently, to = "Matrix" or to = "triangularMatrix")
  - `as(*, "sparseMatrix")` returns the lower triangular factor $L$ from the $LL'$ form of the Cholesky factorization. Note that (currently) the factor from the $LL'$ form is always returned, even if the "CHMfactor" object represents an $LDL'$ decomposition. Furthermore, this is the factor after any fill-reducing permutation has been applied. See the expand method for obtaining both the permutation matrix, $P$, and the lower Cholesky factor, $L$.
- **coerce** signature(from = "CHMfactor", to = "pMatrix") returns the permutation matrix $P$, representing the fill-reducing permutation used in the decomposition.
- **expand** signature(* = "CHMfactor") returns a list with components $P$, the matrix representing the fill-reducing permutation, and $L$, the lower triangular Cholesky factor. The original positive-definite matrix $A$ corresponds to the product $A = P'LL'P$. Because of fill-in during the decomposition the product may apparently have more non-zeros than the original matrix, even after applying `drop0` to it. However, the extra "non-zeros" should be very small in magnitude.
- **image** signature(* = "CHMfactor") : Plot the image of the lower triangular factor, $L$, from the decomposition. This method is equivalent to `image(as(*, "sparseMatrix"))` so the comments in the above description of the `coerce` method apply here too.
The solve methods for a "CHMfactor" object take an optional third argument system whose value can be one of the character strings "A", "LDLt", "LD", "DLt", "L", "Lt", "D", "P" or "Pt". This argument describes the system to be solved. The default, "A", is to solve $Ax = b$ for $x$ where $A$ is the sparse, positive-definite matrix that was factored to produce $a$.

Analogous, system = "L" returns the solution $x$, of $Lx = b$. Similarly, for all system codes but "P" and "Pt" where, e.g., $x <- \text{solve}(a, b, \text{system="P"})$ is equivalent to $x <- P %*% b$.

See also \texttt{solve-methods}.

\textbf{determinant} signature($x = \text{"CHMfactor"}$, logarithm = \text{"logical"}) returns the determinant (or the logarithm of the determinant, if logarithm = \text{TRUE}, the default) of the factor $L$ from the $LL'$ decomposition (even if the decomposition represented by $x$ is of the $LDL'$ form (!)). This is the square root of the determinant (half the logarithm of the determinant when logarithm = \text{TRUE}) of the positive-definite matrix that was decomposed.

\textbf{update} signature(object = \text{"CHMfactor"}), parent. The update method requires an additional argument parent, which is \textit{either} a "dsCMatrix" object, say $A$, (with the same structure of nonzeros as the matrix that was decomposed to produce object) or a general "dgCMatrix", say $M$, where $A := MM'$ (== tcrossprod(parent)) is used for $A$. Further it provides an optional argument mult, a numeric scalar. This method updates the numeric values in object to the decomposition of $A + mI$ where $A$ is the matrix above (either the parent or $MM'$) and $m$ is the scalar mult. Because only the numeric values are updated this method should be faster than creating and decomposing $A + mI$. It is not uncommon to want, say, the determinant of $A + mI$ for many different values of $m$. This method would be the preferred approach in such cases.

\textbf{See Also}

\texttt{cholesky}, also for examples; \texttt{class dgCMatrix}.

\textbf{Examples}

```r
## An example for the expand() method
n <- 1000; m <- 200; nnz <- 2000
set.seed(1)
M1 <- spMatrix(n, m,
i = sample(n, nnz, replace = \text{TRUE}),
j = sample(m, nnz, replace = \text{TRUE}),
x = \text{round(rnorm(nnz)),1})
XX <- crossprod(M1) ## = M1'M1 = M'M where M <- t(M1)
CX <- \text{Cholesky}(XX)
isLDL(CX)
str(CX) ## a \text{"dCHMsimpl" object}
r <- expand(CX)
L.P <- with(r, crossprod(L,P)) ## = L'P
PLL.P <- crossprod(L,P) ## = (L'P)'L'P = P'LL'P = XX = M'M
b <- sample(m)
stopifnot(all.equal(PLL.P, XX),
  all(as.vector(solve(CX, b, \text{system="P"})) = r$P %*% b),
  all(as.vector(solve(CX, b, \text{system="Pt"})) = t(r$P) %*% b ))
```
chol

Choleski Decomposition - 'Matrix' S4 Generic and Methods

Description
Compute the Choleski factorization of a real symmetric positive-definite square matrix.

Usage
chol(x, ...)
## S4 method for signature 'dsCMatrix'
chol(x, pivot = FALSE, ...)
## S4 method for signature 'dsparseMatrix'
chol(x, pivot = FALSE, cache = TRUE, ...)

Arguments
x a (sparse or dense) square matrix, here inheriting from class Matrix; if x is not positive definite, an error is signalled.
pivot logical indicating if pivoting is to be used. Currently, this is not made use of for dense matrices.
cache logical indicating if the result should be cached in x@factors; note that this argument is experimental and only available for some sparse matrices.
... potentially further arguments passed to methods.

Details
Note that these Cholesky factorizations are typically cached with x currently, and these caches are available in x@factors, which may be useful for the sparse case when pivot = TRUE, where the permutation can be retrieved; see also the examples.

However, this should not be considered part of the API and made use of. Rather consider Cholesky() in such situations, since chol(x, pivot=TRUE) uses the same algorithm (but not the same return value!) as Cholesky(x, LDL=FALSE) and chol(x) corresponds to Cholesky(x, perm=FALSE, LDL=FALSE).

Value
a matrix of class Cholesky, i.e., upper triangular: R such that R'R = x (if pivot=FALSE) or P'R'R'P' = x (if pivot=TRUE and P is the corresponding permutation matrix).
Methods

Use `showMethods(chol)` to see all; some are worth mentioning here:

- **chol** signature(`x = "dgeMatrix"`): works via "dpoMatrix", see class `dpomatix`.
- **chol** signature(`x = "dpoMatrix"`): Returns (and stores) the Cholesky decomposition of `x`, via LAPACK routines `dlacpy` and `dpotrf`.
- **chol** signature(`x = "dppMatrix"`): Returns (and stores) the Cholesky decomposition via LAPACK routine `dpptrf`.
- **chol** signature(`x = "dsCMatrix", pivot = "logical"`): Returns (and stores) the Cholesky decomposition of `x`. If `pivot` is true, the Approximate Minimal Degree (AMD) algorithm is used to create a reordering of the rows and columns of `x` so as to reduce fill-in.

References


See Also

The default from base, chol; for more flexibility (but not returning a matrix!) Cholesky.

Examples

showMethods(chol, inherited = FALSE) # show different methods

sy2 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, NA,32,77))
(c2 <- chol(sy2))#-> "Cholesky" matrix
stopifnot(all.equal(c2, chol(as(sy2, "dpoMatrix")), tolerance = 1e-13))
str(c2)

## An example where chol() can't work
(sy3 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, -1, 2, -7)))
try(chol(sy3)) # error, since it is not positive definite

## A sparse example --- exemplifying 'pivot'
(mm <- toeplitz(as(c(10, 0, 1, 0, 3), "sparseVector"))) # 5 x 5
(R <- chol(mm)) ## default: pivot = FALSE
R2 <- chol(mm, pivot=FALSE)
stopifnot( identical(R, R2), all.equal(crossprod(R), mm ) )
(R. <- chol(mm, pivot=TRUE))# nice band structure,
## but of course crossprod(R.) is *NOT* equal to mm
## -- see Cholesky() and its examples, for the pivot structure & factorization
stopifnot(all.equal(sqrt(det(mm)), det(R)),
  all.equal(prod(diag(R)), det(R)),
  all.equal(prod(diag(R.)), det(R)))

## a second, even sparser example:
(M2 <- toeplitz(as(c(1,.5, rep(0,12), -.1), "sparseVector")))
Inverse from Choleski or QR Decomposition – Matrix Methods

Description

Invert a symmetric, positive definite square matrix from its Choleski decomposition. Equivalently, compute \((X'X)^{-1}\) from the \((R\text{ part})\) of the QR decomposition of \(X\). Even more generally, given an upper triangular matrix \(R\), compute \((R'R)^{-1}\).

Methods

- **x = "ANY"** the default method from base, see chol2inv, for traditional matrices.
- **x = "dtrMatrix"** method for the numeric triangular matrices, built on the same LAPACK DPOTRI function as the base method.
- **x = "denseMatrix"** if \(x\) is coercable to a triangularMatrix, call the "dtrMatrix" method above.
- **x = "sparseMatrix"** if \(x\) is coercable to a triangularMatrix, use solve() currently.

See Also

chol (for Matrix objects); further, chol2inv (from the base package), solve.

Examples

```r
M <- Matrix(cbind(1, 1:3, c(1,3,7)))
cM <- chol(M) # a "Cholesky" object, inheriting from "dtrMatrix"
chol2inv(cM) %*% M # the identity
stopifnot(all(chol2inv(cM) %*% M - Diagonal(nrow(M))) < 1e-10)
```
Description

Computes the Cholesky (aka “Choleski”) decomposition of a sparse, symmetric, positive-definite matrix. However, typically \texttt{chol()} should rather be used unless you are interested in the different kinds of sparse Cholesky decompositions.

Usage

\texttt{Cholesky(A, perm = TRUE, LDL = !super, super = FALSE, Imult = 0, ...)}

Arguments

- \texttt{A} \hspace{1cm} \text{sparse symmetric matrix. No missing values or IEEE special values are allowed.}
- \texttt{perm} \hspace{1cm} \text{logical scalar indicating if a fill-reducing permutation should be computed and applied to the rows and columns of } A. \text{Default is TRUE.}
- \texttt{LDL} \hspace{1cm} \text{logical scalar indicating if the decomposition should be computed as LDL’ where } L \text{ is a unit lower triangular matrix. The alternative is LL’ where } L \text{ is lower triangular with arbitrary diagonal elements. Default is TRUE. Setting it to } NA \text{ leaves the choice to a CHOLMOD-internal heuristic.}
- \texttt{super} \hspace{1cm} \text{logical scalar indicating if a supernodal decomposition should be created. The alternative is a simplicial decomposition. Default is FALSE. Setting it to } NA \text{ leaves the choice to a CHOLMOD-internal heuristic.}
- \texttt{Imult} \hspace{1cm} \text{numeric scalar which defaults to zero. The matrix that is decomposed is } A + m \times I\text{ where } m \text{ is the value of } Imult \text{ and } I \text{ is the identity matrix of order } ncol(A).

Details

This is a generic function with special methods for different types of matrices. Use \texttt{showMethods("Cholesky")} to list all the methods for the \texttt{Cholesky} generic.

The method for class \texttt{dsCMATRIX} of sparse matrices — the only one available currently — is based on functions from the CHOLMOD library.

Again: If you just want the Cholesky decomposition of a matrix in a straightforward way, you should probably rather use \texttt{chol(.)}.

Note that if \texttt{perm=TRUE} (default), the decomposition is

\[ A = P' \tilde{L} \tilde{L}' P = P' LL' P, \]

where \( L \) can be extracted by \texttt{as(*, "Matrix")}, \( P \) by \texttt{as(*, "pMatrix")} and both by \texttt{expand(*)}, see the class \texttt{CHMfactor} documentation.

Note that consequently, you cannot easily get the “traditional” cholesky factor \( R \), from this decomposition, as

\[ R' R = A = P' LL' P = P' \tilde{R} \tilde{R} P = (\tilde{R} P)' (\tilde{R} P), \]

but \( \tilde{R} P \) is not triangular even though \( \tilde{R} \) is.
Cholesky

Value

an object inheriting from either "CHMspuper", or "CHMsimpl", depending on the super argument; both classes extend "CHMfactor" which extends "MatrixFactorization".

In other words, the result of Cholesky() is not a matrix, and if you want one, you should probably rather use chol(), see Details.

References

Tim Davis (2005) CHOLMOD: sparse supernodal Cholesky factorization and update/downdate
http://www.cise.ufl.edu/research/sparse/cholmod/

See Also

Class definitions CHMfactor and dsCMatrix and function expand. Note the extra solve(*, system = .) options in CHMfactor.

Note that chol() returns matrices (inheriting from "Matrix") whereas Cholesky() returns a "CHMfactor" object, and hence a typical user will rather use cholA().

Examples

data(KNex)
mtm <- with(KNex, crossprod(mm))
str(mtm@factors) # empty list()
(CI <- Cholesky(mtm)) # uses show(<MatrixFactorization>)
str(mtm@factors) # 'sPDCholesky' (simpl)
(Cm <- Cholesky(mtm, super = TRUE))
c(C1 = isLDL(CI), Cm = isLDL(Cm))
str(mtm@factors) # 'sPDCholesky' *and* 'SPdCholesky'
str(cm1 <- as(C1, "sparseMatrix"))
str(mat <- as(Cm, "sparseMatrix"))## hmm: super is *less* sparse here
cm1[1:20, 1:20]

b <- matrix(c(rep(0, 711), 1), nc = 1)## solve(Cm, b) by default solves Ax = b, where A = Cm'Cm (= mtm)!
## hence, the identical() check *should* work, but fails on some GOTOblas:
x <- solve(Cm, b)
stopifnot(identical(x, solve(Cm, b, system = "A")),
            all.equal(x, solve(mtm, b)))

Cn <- Cholesky(mtm, perm = FALSE)# no permutation -- much worse:
sizes <- c(simple = object.size(C1),
            super = object.size(Cm),
            noPerm = object.size(Cn))
## simple is 100, super= 137, noPerm= 812 :
noquote(cbind(format(100 * sizes / sizes[1], digits=4)))

## Visualize the sparseness:
Cholesky-class

Description

The "Cholesky" class is the class of Cholesky decompositions of positive-semidefinite, real dense matrices. The "BunchKaufman" class is the class of Bunch-Kaufman decompositions of symmetric, real matrices. The "pCholesky" and "pBunchKaufman" classes are their packed storage versions.

Objects from the Class

Objects can be created by calls of the form `new("Cholesky", ...)` or `new("BunchKaufman", ...)` etc., or rather by calls of the form `chol(pm)` or `BunchKaufman(pm)` where `pm` inherits from the "dpoMatrix" or "dsyMatrix" class or as a side-effect of other functions applied to "dpoMatrix" objects (see `dpoMatrix`).
Cholesky-class

Slots

A Cholesky decomposition extends class `MatrixFactorization` but is basically a triangular matrix extending the "dtrMatrix" class.

- `uplo`: inherited from the "dtrMatrix" class.
- `diag`: inherited from the "dtrMatrix" class.
- `x`: inherited from the "dtrMatrix" class.
- `Dim`: inherited from the "dtrMatrix" class.
- `Dimnames`: inherited from the "dtrMatrix" class.

A Bunch-Kaufman decomposition also extends the "dtrMatrix" class and has a `perm` slot representing a permutation matrix. The packed versions extend the "dtpMatrix" class.

Extends

Class "MatrixFactorization" and "dtrMatrix", directly. Class "dgeMatrix", by class "dtrMatrix". Class "Matrix", by class "dtrMatrix".

Methods

Both these factorizations can directly be treated as (triangular) matrices, as they extend "dtrMatrix", see above. There are currently no further explicit methods defined with class "Cholesky" or "BunchKaufman" in the signature.

Note

1. Objects of class "Cholesky" typically stem from `chol(D)`, applied to a dense matrix D.
   On the other hand, the function `Cholesky(S)` applies to a sparse matrix S, and results in objects inheriting from class `CHMfactor`.

2. For traditional matrices m, chol(m) is a traditional matrix as well, triangular, but simply an \( n \times n \) numeric matrix. Hence, for compatibility, the "Cholesky" and "BunchKaufman" classes (and their "p*" packed versions) also extend triangular Matrix classes (such as "dtrMatrix"). Consequently, `determinant(R)` for R \( \leftarrow \) chol(A) returns the determinant of R, not of A. This is in contrast to class `CHMfactor` objects C, where determinant(C) gives the determinant of the original matrix A, for C \( \leftarrow \) Cholesky(A), see also the determinant method documentation on the class `CHMfactor` page.

See Also

Classes `dtrMatrix`, `dpoMatrix`; function `chol`.

Function `Cholesky` resulting in class `CHMfactor` objects, not class "Cholesky" ones, see the section ‘Note’.
Examples

(sm <- as(as(Matrix(diag(5) + 1), "dsyMatrix"), "dspMatrix"))
signif(csm <- chol(sm), 4)

(pm <- crossprod(Matrix(rnorm(18), nrow = 6, ncol = 3)))
(ch <- chol(pm))
if (toupper(ch@uplo) == "U") # which is TRUE
crossprod(ch)
stopifnot(all.equal(as(crossprod(ch), "matrix"),
                     as(pm, "matrix"), tolerance=1e-14))

---

colSums

Form Row and Column Sums and Means

Description

Form row and column sums and means for Matrix objects.

Usage

colSums (x, na.rm = FALSE, dims = 1, ...)
rowSums (x, na.rm = FALSE, dims = 1, ...)
colMeans(x, na.rm = FALSE, dims = 1, ...)
rowMeans(x, na.rm = FALSE, dims = 1, ...)

## S4 method for signature 'CsparseMatrix'
colSums(x, na.rm = FALSE,
         dims = 1, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
rowSums(x, na.rm = FALSE,
         dims = 1, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
colMeans(x, na.rm = FALSE,
         dims = 1, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
rowMeans(x, na.rm = FALSE,
         dims = 1, sparseResult = FALSE)

Arguments

x a Matrix, i.e., inheriting from Matrix.
na.rm logical. Should missing values (including NaN) be omitted from the calculations?
dims completely ignored by the Matrix methods.
... potentially further arguments, for method <-> generic compatibility.
sparseResult logical indicating if the result should be sparse, i.e., inheriting from class sparseVector.
Value

returns a numeric vector if `sparseResult` is `FALSE` as per default. Otherwise, returns a `sparseVector`.

See Also

`colSums` and the `sparseVector` classes.

Examples

```r
(M <- bdiag(Diagonal(2), matrix(1:3, 3, 4), diag(3:2))) # 7 x 8
colSums(M)
d <- Diagonal(10, c(0,0,10,0,2,rep(0,5)))
MM <- kronecker(d, M)
dim(MM) # 70 80
length(MM@x) # 160, but many are '0'; drop those:
MM <- drop0(MM)
length(MM@x) # 32

cm <- colSums(MM)
(scm <- colSums(MM, sparseResult = TRUE))
stopifnot(is(scm, "sparseVector"),
          identical(cm, as.numeric(scm)))
rowSums (MM, sparseResult = TRUE) # 14 of 70 are not zero
colMeans(MM, sparseResult = TRUE) # 16 of 80 are not zero
## Since we have no 'NA's, these two are equivalent:
stopifnot(identical(rowMeans(MM, sparseResult = TRUE),
                      rowMeans(MM, sparseResult = TRUE, na.rm = TRUE)),
          rowMeans(Diagonal(16)) == 1/16,
          colSums(Diagonal(7)) == 1)
```

compMatrix-class

*Class* "compMatrix" of Composite (Factorizable) Matrices

Description

Virtual class of composite matrices; i.e., matrices that can be factorized, typically as a product of simpler matrices.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

- `factors`: Object of class "list" - a list of factorizations of the matrix. Note that this is typically empty, i.e. `list()`, initially and is updated automagically whenever a matrix factorization is computed.
- `Dim`
- `Dimnames`: inherited from the `Matrix` class, see there.
**Extends**

Class "Matrix", directly.

---

**condest**

*Compute Approximate CONDITION number and 1-Norm of (Large) Matrices*

---

**Description**

“Estimate”, i.e. compute approximately the CONDITION number of a (potentially large, often sparse) matrix A. It works by apply a fast randomized approximation of the 1-norm, \(\|A\|_1\), through onenormest(.)

**Usage**

```r
condest(A, t = min(n, 5), normA = norm(A, "1"),
    silent = FALSE, quiet = TRUE)
```

```r
onenormest(A, t = min(n, 5), A.x, At.x, n,
    silent = FALSE, quiet = silent,
    iter.max = 10, eps = 4 * .Machine$double.eps)
```

**Arguments**

- **A**
  a square matrix, optional for onenormest(), where instead of A, A.x and At.x can be specified, see there.

- **t**
  number of columns to use in the iterations.

- **normA**
  number; (an estimate of) the 1-norm of A, by default \(\|A\|_1\); may be replaced by an estimate.

- **silent**
  logical indicating if warning and (by default) convergence messages should be displayed.

- **quiet**
  logical indicating if convergence messages should be displayed.

- **A.x, At.x**
  when A is missing, these two must be given as functions which compute \(A \%\% x\), or \(t(A) \%\% x\), respectively.

- **n**
  = nrow(A), only needed when A is not specified.

- **iter.max**
  maximal number of iterations for the 1-norm estimator.

- **eps**
  the relative change that is deemed irrelevant.

**Details**

`condest()` calls `lu(A)`, and subsequently `onenormest(A.x = , At.x = )` to compute an approximate norm of the inverse of A, \(A^{-1}\), in a way which keeps using sparse matrices efficiently when A is sparse.

Note that `onenormest()` uses random vectors and hence both functions’ results are random, i.e., depend on the random seed, see, e.g., `set.seed()`.
condest

Value

Both functions return a list; condest() with components,

- **est**: a number > 0, the estimated (1-norm) condition number \( \tilde{\kappa} \); when \( r := rcond(A) \), \( 1/\tilde{\kappa} \approx r \).
- **v**: the maximal \( Ax \) column, scaled to norm(\( v \)) = 1. Consequently, \( \text{norm}(Av) = \text{norm}(A)/\text{est} \); when est is large, \( v \) is an approximate null vector.

The function onenormest() returns a list with components,

- **est**: a number > 0, the estimated \( \text{norm}(A, "1") \).
- **v**: 0-1 integer vector length n, with an 1 at the index \( j \) with maximal column \( a[ , j] \) in \( A \).
- **w**: numeric vector, the largest \( Ax \) found.
- **iter**: the number of iterations used.

Author(s)

This is based on octave’s condest() and onenormest() implementations with original author Jason Riedy, U Berkeley; translation to R and adaption by Martin Maechler.

References


See Also

- norm, rcond.

Examples

data(KNex)
mtm <- with(KNex, crossprod(mm))
system.time(ce <- condest(mtm))
sum(abs(ce$v)) ## || v ||_1 == 1
## Prove that || Av || = || A || / est (as ||v|| = 1):
stopifnot(all.equal(norm(mtm %*% ce$v),
                   norm(mtm) / ce$est))

## reciprocal
1 / ce$est
system.time(rc <- rcond(mtm)) # takes ca 3 x longer
rc
all.equal(rc, 1/ce$est) # TRUE -- the approximation was good

one <- onenormest(mtm)
str(one) ## est = 12.3
## CsparseMatrix-class

The `CsparseMatrix` class is the virtual class of all sparse matrices coded in sorted compressed column-oriented form. Since it is a virtual class, no objects may be created from it. See `showClass("CsparseMatrix")` for its subclasses.

### Slots

- **i**: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the 0-based row numbers for each non-zero element in the matrix, i.e., `i` must be in \(0:(\text{nnrow}()-1)\).
- **p**: integer vector for providing pointers, one for each column, to the initial (zero-based) index of elements in the column. \(\Delta p\) is of length `ncol(.) + 1`, with `p[1] == 0` and `p[\text{length}(p)] == \text{nnzero}`, such that in fact, `diff(\Delta p)` are the number of non-zero elements for each column.

In other words, `m@p[1:ncol(m)]` contains the indices of those elements in `m@x` that are the first elements in the respective column of `m`.

- **Dim, Dimnames**: inherited from the superclass, see the `sparseMatrix` class.

### Extends

Class "sparseMatrix", directly. Class "Matrix", by class "sparseMatrix".

### Methods

- matrix products `%*%`, `crossprod()` and `tcrossprod()`, several `solve` methods, and other matrix methods available:
  
  ```r
  signature(e1 = "CsparseMatrix", e2 = "numeric"): ...
  
  Arithmetic signature(e1 = "numeric", e2 = "CsparseMatrix"): ...
  
  Math signature(x = "CsparseMatrix"): ...
  
  band signature(x = "CsparseMatrix"): ...
  - signature(e1 = "CsparseMatrix", e2 = "numeric"): ...
  - signature(e1 = "numeric", e2 = "CsparseMatrix"): ...
  + signature(e1 = "CsparseMatrix", e2 = "numeric"): ...
  + signature(e1 = "numeric", e2 = "CsparseMatrix"): ...
  
  coerce signature(from = "CsparseMatrix", to = "TsparseMatrix"): ...
  coerce signature(from = "CsparseMatrix", to = "denseMatrix"): ...
  coerce signature(from = "CsparseMatrix", to = "matrix"): ...
  ```
coerce signature(from = "CsparseMatrix", to = "lsparseMatrix"): ...
coerce signature(from = "CsparseMatrix", to = "nsparseMatrix"): ...
coerce signature(from = "TsparseMatrix", to = "CsparseMatrix"): ...
coerce signature(from = "denseMatrix", to = "CsparseMatrix"): ...

diag signature(x = "CsparseMatrix"): ...
gamma signature(x = "CsparseMatrix"): ...
lgamma signature(x = "CsparseMatrix"): ...
log signature(x = "CsparseMatrix"): ...
t signature(x = "CsparseMatrix"): ...
tril signature(x = "CsparseMatrix"): ...
triu signature(x = "CsparseMatrix"): ...

Note

All classes extending CsparseMatrix have a common validity (see validObject) check function. That function additionally checks the i slot for each column to contain increasing row numbers. In earlier versions of Matrix (<= 0.999375-16), validObject automatically re-sorted the entries when necessary, and hence new() calls with somewhat permuted i and x slots worked, as new(...) (with slot arguments) automatically checks the validity.

Now, you have to use sparseMatrix to achieve the same functionality or know how to use .validateCsparse() to do so.

See Also

colSums, kronecker, and other such methods with own help pages.

Further, the super class of CsparseMatrix, sparseMatrix, and, e.g., class dgCMatrix for the links to other classes.

Examples

glass("CsparseMatrix")

## The common validity check function (based on C code):
glass(getClass("CsparseMatrix"))

---

**ddenseMatrix-class**

**Virtual Class "ddenseMatrix" of Numeric Dense Matrices**

**Description**

This is the virtual class of all dense numeric (i.e., double, hence “dense”) S4 matrices.

Its most important subclass is the dgeMatrix class.
ddiMatrix-class

Description

The class "ddiMatrix" of numerical diagonal matrices.

Note that diagonal matrices now extend sparseMatrix, whereas they did extend dense matrices earlier.

Objects from the Class

Objects can be created by calls of the form new("ddiMatrix", ...) but typically rather via Diagonal.

Slots

  x: numeric vector. For an \( n \times n \) matrix, the x slot is of length \( n \) or 0, depending on the diag slot:
  diag: "character" string, either "U" or "N" where "U" denotes unit-diagonal, i.e., identity matrices.
  Dim,Dimnames: matrix dimension and dimnames, see the Matrix class description.
**denseMatrix-class**

**Virtual Class "denseMatrix" of All Dense Matrices**

**Description**

This is the virtual class of all dense (S4) matrices. It is the direct superclass of `ddenseMatrix`, `ldenseMatrix`.

**Extends**

class "Matrix" directly.

**Slots**

exactly those of its superclass "Matrix".

**Methods**

Use `showMethods(class = "denseMatrix", where = "package:Matrix")` for an overview of methods. Extraction ("[" methods, see `-methods`.

**Examples**

```r
(d2 <- Diagonal(x = c(10,1)))
str(d2)
## slightly larger in internal size:
str(as(d2, "sparseMatrix"))

M <- Matrix(cbind(1:2:4))
M %*% d2 # 'fast' multiplication

chol(d2) # trivial
stopifnot(is(cd2 <- chol(d2), "ddiMatrix"),
  all.equal(cd2@x, c(sqrt(10), 1)))
```
See Also
colSums, kronecker, and other such methods with own help pages. Its superclass Matrix, and main subclasses, ddenseMatrix and sparseMatrix.

Examples
showClass("denseMatrix")

dgCMatrix-class  Compressed, sparse, column-oriented numeric matrices

Description
The dgCMatrix class is a class of sparse numeric matrices in the compressed, sparse, column-oriented format. In this implementation the non-zero elements in the columns are sorted into increasing row order. dgCMatrix is the "standard" class for sparse numeric matrices in the Matrix package.

Objects from the Class
Objects can be created by calls of the form new("dgCMatrix", ...), more typically via as(*, "CsparseMatrix") or similar. Often however, more easily via Matrix(*, sparse = TRUE), or most efficiently via sparseMatrix().

Slots
x: Object of class "numeric" - the non-zero elements of the matrix.
... all other slots are inherited from the superclass "CsparseMatrix".

Methods
Matrix products (e.g., crossprod-methods), and (among other)

  coerce signature(from = "matrix", to = "dgCMatrix")
  coerce signature(from = "dgCMatrix", to = "matrix")
  coerce signature(from = "dgCMatrix", to = "dgTMatrix")
  diag signature(x = "dgCMatrix"): returns the diagonal of x
  dim signature(x = "dgCMatrix"): returns the dimensions of x
  image signature(x = "dgCMatrix"): plots an image of x using the levelplot function
  solve signature(a = "dgCMatrix", b = "..."): see solve-methods, notably the extra argument sparse.
  lu signature(x = "dgCMatrix"): computes the LU decomposition of a square dgCMatrix object
See Also

Classes `dsCMatrix`, `dtCMatrix`, `lu`

Examples

```r
(m <- Matrix(c(0,0,2:0), 3,5))
str(m)
m[,1]
```


\begin{verbatim}
chol signature(x = "dgeMatrix"): see chol.
coerce signature(from = "dgeMatrix", to = "dgeMatrix"): ...
coerce signature(from = "dgeMatrix", to = "matrix"): ...
coerce signature(from = "matrix", to = "dgeMatrix"): ...
colMeans signature(x = "dgeMatrix"): columnwise means (averages)
colSums signature(x = "dgeMatrix"): columnwise sums
diag signature(x = "dgeMatrix"): ...
dim signature(x = "dgeMatrix"): ...
dimnames signature(x = "dgeMatrix"): ...
eigen signature(x = "dgeMatrix", only.values = "logical"): ...
eigen signature(x = "dgeMatrix", only.values = "missing"): ...
norm signature(x = "dgeMatrix", type = "character"): ...
norm signature(x = "dgeMatrix", type = "missing"): ...
rcond signature(x = "dgeMatrix", norm = "character") or norm = "missing": the reciprocal condition number. rcond().
rowMeans signature(x = "dgeMatrix"): rowwise means (averages)
rowSums signature(x = "dgeMatrix"): rowwise sums
t signature(x = "dgeMatrix"): matrix transpose

See Also

Classes Matrix, dtrMatrix, and dsyMatrix.
\end{verbatim}

---

\textbf{dgRMatrix-class} \hspace{1cm} \textit{Sparse Compressed, Row-oriented Numeric Matrices}

\section*{Description}

The \texttt{dgRMatrix} class is a class of sparse numeric matrices in the compressed, sparse, row-oriented format. In this implementation the non-zero elements in the rows are sorted into increasing column order.

\textbf{Note:} The column-oriented sparse classes, e.g., \texttt{dgCMatrix}, are preferred and better supported in the \texttt{Matrix} package.

\section*{Objects from the Class}

Objects can be created by calls of the form \texttt{new("dgeMatrix", ...).}
Slots

j: Object of class "integer" of length nnzero (number of non-zero elements). These are the column numbers for each non-zero element in the matrix.

p: Object of class "integer" of pointers, one for each row, to the initial (zero-based) index of elements in the row.

x: Object of class "numeric" - the non-zero elements of the matrix.

Dim: Object of class "integer" - the dimensions of the matrix.

Methods

coerce signature(from = "matrix", to = "dgRMatrix")
coerce signature(from = "dgRMatrix", to = "matrix")
coerce signature(from = "dgRMatrix", to = "dgTMatrix")
diag signature(x = "dgRMatrix"): returns the diagonal of x
dim signature(x = "dgRMatrix"): returns the dimensions of x
image signature(x = "dgRMatrix"): plots an image of x using the levelplot function

See Also

the RsparseMatrix class, the virtual class of all sparse compressed row-oriented matrices, with its methods. The dgCMatrix class (column compressed sparse) is really preferred.

dgTMatrix-class Sparse matrices in triplet form

Description

The "dgTMatrix" class is the class of sparse matrices stored as (possibly redundant) triplets. The internal representation is not at all unique, contrary to the one for class dgCMatrix.

Objects from the Class

Objects can be created by calls of the form new("dgTMatrix", ...), but more typically via as(*, "dgTMatrix"), spMatrix(), or sparseMatrix(*, giveCsparse=FALSE).

Slots

i: Object of class "integer" - the row indices of non-zero entries in 0-base, i.e., must be in 0:(nrow(.)-1).

j: Object of class "integer" - the column indices of non-zero entries. Must be the same length as slot i and 0-based as well, i.e., in 0:(ncol(.)-1).

x: Object of class "numeric" - the (non-zero) entry at position (i,j). Must be the same length as slot i. If an index pair occurs more than once the corresponding values of slot x are added to form the element of the matrix.

Dim: Object of class "integer" of length 2 - the dimensions of the matrix.
Methods

+ signature(e1 = "dgTMatrix", e2 = "dgTMatrix")
coerce signature(from = "dgTMatrix", to = "dgCMatrix")
coerce signature(from = "dgTMatrix", to = "dgeMatrix")
coerce signature(from = "dgTMatrix", to = "matrix"), and typically coercion methods for more specific signatures, we are not mentioning here.

Note that these are not guaranteed to continue to exist, but rather you should use calls like as(x, "CsparseMatrix"), as(x, "generalMatrix"), as(x, "dMatrix"), i.e. coercion to higher level virtual classes.
coerce signature(from = "matrix", to = "dgTMatrix"), (direct coercion from tradition matrix).
image signature(x = "dgTMatrix"): plots an image of x using the levelplot function

t signature(x = "dgTMatrix"): returns the transpose of x

Note

Triplet matrices are a convenient form in which to construct sparse matrices after which they can be coerced to dgCMatrix objects.

Note that both new(.) and spMatrix constructors for "dgTMatrix" (and other "linkS4class(TsparseMatrix)" classes) implicitly add \( x_{ik} \)’s that belong to identical \((i_k,j_k)\) pairs.

See Also

Class dgCMatrix or the superclasses dsparseMatrix and TsparseMatrix;

Examples

m <- Matrix(0+1:28, nrow = 4)
m[3, c(2, 4:5, 7)] <- m[3, 1:4] <- m[1:3, 6] <- 0
(mT <- as(m, "dgTMatrix"))
str(mT)
mT[,]
mT[4, drop = FALSE]
stopifnot(identical(mT[lower.tri(mT)],
                     m[lower.tri(m) ]))
mT[lower.tri(mT, diag=TRUE)] <- 0
mT

## Triplet representation with repeated (i,j) entries
## *adds* the corresponding x’ s:
T2 <- new("dgTMatrix",
          i = as.integer(c(1,1,0,3,3)),
          j = as.integer(c(2,2,4,0,0)),
          x = 10*1:5, Dim = 4:5)
str(T2) # contains (i,j,x) slots exactly as above, but
T2 ## has only three non-zero entries, as for repeated (i,j)'s,
      ## the corresponding x's are "implicitly" added
stopifnot(nnzero(T2) == 3)
Description
Create a diagonal matrix object, i.e., an object inheriting from diagonalMatrix.

Usage
Diagonal(n, x = NULL)
.symDiagonal(n, x = rep.int(1,n), uplo = "U")
.sparseDiagonal(n, x = 1, uplo = "U",
    shape = if(missing(cols)) "t" else "g",
    unitri, kind, cols = if(n) 0:(n - 1L) else integer(0))

Arguments
n integer specifying the dimension of the (square) matrix. If missing, length(x) is used.
x numeric or logical; if missing, a unit diagonal n × n matrix is created.
uplo for .symDiagonal, the resulting sparse symmetricMatrix will have slot uplo set from this argument, either "U" or "L". Only rarely will it make sense to change this from the default.
shape string of 1 character, one of c("t", "s", "g"), to choose a triangular, symmetric or general result matrix.
unitri optional logical indicating if a triangular result should be “unit-triangular”, i.e., with diag = "U" slot, if possible. The default, missing, is the same as TRUE.
kind string of 1 character, one of c("d","l","n"), to choose the storage mode of the result, from classes dsparseMatrix, lsparseMatrix, or nsparseMatrix, respectively.
cols integer vector with values from 0:(n-1), denoting the columns to subselect conceptually, i.e., get the equivalent of Diagonal(n,\times)[, cols + 1].

Value
Diagonal() returns an object of class ddiMatrix or ldiMatrix (with “superclass” diagonalMatrix).
.symDiagonal() returns an object of class dsCMatrix or lsCMatrix, i.e., a sparse symmetric matrix. This can be more efficient than Diagonal(n) when the result is combined with further symmetric (sparse) matrices, however not for matrix multiplications where Diagonal() is clearly preferred.
.sparseDiagonal(), the workhorse of .symDiagonal returns a CsparseMatrix (the resulting class depending on shape and kind) representation of Diagonal(n), or, when cols are specified, of Diagonal(n)[, cols+1].
Author(s)
Martin Maechler

See Also
the generic function diag for extraction of the diagonal from a matrix works for all “Matrices”. 
bandSparse constructs a banded sparse matrix from its non-zero sub-/super - diagonals. band(A) returns a band matrix containing some sub-/super - diagonals of A. Matrix for general matrix construction; further, class diagonalMatrix.

Examples
Diagonal(3)
Diagonal(x = 10^(3:1))
Diagonal(x = (1:4) >= 2) #-> "ldiMatrix"

## Use Diagonal() + kronecker() for "repeated-block" matrices:
(M1 <- Matrix(0+0.5, 2,3))
(M <- kronecker(Diagonal(3), M1))

(S <- crossprod(Matrix(rbinom(60, size=1, prob=0.1), 10,6)))
(SI <- S + 10*symDiagonal(6)) # sparse symmetric still
stopifnot(is(SI, "dsCMatrix"))
(I4 <- .sparseDiagonal(4, shape="t")) # now (2012-10) unitriangular
stopifnot(I4@diag == "U", all(I4 == diag(4)))

diagonalMatrix-class  Class "diagonalMatrix" of Diagonal Matrices

Description
Class "diagonalMatrix" is the virtual class of all diagonal matrices.

Objects from the Class
A virtual Class: No objects may be created from it.

Slots
 diag: code"character" string, either "U" or "N", where "U" means ‘unit-diagonal’.
 Dim: matrix dimension, and
 Dimnames: the dimnames, a list, see the Matrix class description. Typically list(NULL,NULL) for diagonal matrices.

Extends
Class "sparseMatrix", directly.
Methods

These are just a subset of the signature for which defined methods. Currently, there are (too) many explicit methods defined in order to ensure efficient methods for diagonal matrices.

coerce signature(from = "matrix", to = "diagonalMatrix"): ...
coerce signature(from = "Matrix", to = "diagonalMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "generalMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "triangularMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "nMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "matrix"): ...
coerce signature(from = "diagonalMatrix", to = "sparseVector"): ...
t signature(x = "diagonalMatrix"): ...
and many more methods

solve signature(a = "diagonalMatrix", b, ...): is trivially implemented, of course; see also solve-methods.

which signature(x = "nMatrix"), semantically equivalent to base function which(x, arr.ind).

"Math" signature(x = "diagonalMatrix"): all these group methods return a "diagonalMatrix", apart from cumsum() etc which return a vector also for base matrix.

signature(e1 = "ddiMatrix", e2="denseMatrix"): arithmetic and other operators from the Ops group have a few dozen explicit method definitions, in order to keep the results diagonal in many cases, including the following:

/ signature(e1 = "ddiMatrix", e2="denseMatrix"): the result is from class ddiMatrix which is typically very desirable. Note that when e2 contains off-diagonal zeros or NAs, we implicitly use 0/x = 0, hence differing from traditional R arithmetic (where 0/0 → NaN), in order to preserve sparsity.

summary (object = "diagonalMatrix"): Returns an object of S3 class "diagSummary" which is the summary of the vector object@x plus a simple heading, and an appropriate print method.

See Also

Diagonal() as constructor of these matrices, and isDiagonal. ddiMatrix and ldiMatrix are "actual" classes extending "diagonalMatrix".

Examples

I5 <- Diagonal(5)
D5 <- Diagonal(x = 10*(1:5))
## trivial (but explicitly defined) methods:
stopifnot(identical(crossprod(I5), I5),
identical(tcrossprod(I5), I5),
identical(crossprod(I5, D5), D5),
identical(tcrossprod(D5, I5), D5),
identical(solve(D5), solve(D5, I5)),
all.equal(D5, solve(solve(D5)), tolerance = 1e-12)
solve(D5)# efficient as is diagonal

# an unusual way to construct a band matrix:
rrbind2(cbind2(15, D5),
cbind2(D5, 15))

<table>
<thead>
<tr>
<th>diagU2N</th>
<th>Transform Triangular Matrices from Unit Triangular to General and back</th>
</tr>
</thead>
</table>

**Description**

Transform a triangular matrix x, i.e., of class "triangularMatrix", from (internally!) unit triangular ("unitriangular") to "general" (diagU2N(x)) or back (diagN2U(x)). Note that the latter, diagN2U(x), also sets the diagonal to one in cases where diag(x) was not all one.

.diagU2N(x) assumes but does not check that x is a triangularMatrix with diag slot "U", and should hence be used with care.

**Usage**

```r
diagN2U(x, cl = getClassDef(class(x)), checkDense = FALSE)
diagU2N(x, cl = getClassDef(class(x)), checkDense = FALSE)
.diagU2N(x, cl, checkDense = FALSE)
```

**Arguments**

- **x**: a triangularMatrix, often sparse.
- **cl** (optional, for speedup only:) class (definition) of x.
- **checkDense**: logical indicating if dense (see denseMatrix) matrices should be considered at all; i.e., when false, as per default, the result will be sparse even when x is dense.

**Details**

The concept of unit triangular matrices with a diag slot of "U" stems from LAPACK.

**Value**

a triangular matrix of the same class and (semantically) with identical entries as x, but with a different diag slot.

**Note**

Such internal storage details should rarely be of relevance to the user. Hence, these functions really are rather internal utilities.
See Also

"triangularMatrix", "dtCMatrix".

Examples

(T <- Diagonal(7) + triu(Matrix(rpois(49, 1/4), 7, 7), k = 1))
(uT <- diagN2U(T)) # "unitriangular"
(t.u <- diagN2U(10*xT))# changes the diagonal!
stopifnot(all(T == uT), diag(t.u) == 1,
          identical(T, diagU2N(uT)))
T[upper.tri(T)] <- 5
T <- diagN2U(as(T, "triangularMatrix"))
stopifnot(T@diag == "U")
dT <- as(T, "denseMatrix")
dt <- diagN2U(dt)
dtU <- diagN2U(dt, checkDense=TRUE)
stopifnot(is(dtU, "denseMatrix"), is(dt., "sparseMatrix"),
          all(dt == dt.), all(dt == dtU),
          dt@diag == "U", dtU@diag == "U")

---

dMatrix-class

(Virtual) Class "dMatrix" of "double" Matrices

Description

The dMatrix class is a virtual class contained by all actual classes of numeric matrices in the Matrix package. Similarly, all the actual classes of logical matrices inherit from the lMatrix class.

Slots

Common to all matrix object in the package:

Dim: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.

Dimnames: list of length two; each component containing NULL or a character vector length equal the corresponding Dim element.

Methods

There are (relatively simple) group methods (see, e.g., Arith)

Arith signature(e1 = "dMatrix", e2 = "dMatrix"): ...
Arith signature(e1 = "dMatrix", e2 = "numeric"): ...
Arith signature(e1 = "numeric", e2 = "dMatrix"): ...
Math signature(x = "dMatrix"): ...
Math2 signature(x = "dMatrix", digits = "numeric"): this group contains round() and signif().
The "dpoMatrix" class is the class of positive-semidefinite symmetric matrices in nonpacked storage. The "dppMatrix" class is the same except in packed storage. Only the upper triangle or the lower triangle is required to be available.

The "corMatrix" class extends "dpoMatrix" with a slot sd.
Objects from the Class

Objects can be created by calls of the form `new("dpoMatrix", ...)` or from `crossprod` applied to an "dgeMatrix" object.

Slots

- **uplo**: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- **x**: Object of class "numeric". The numeric values that constitute the matrix, stored in column-major order.
- **Dim**: Object of class "integer". The dimensions of the matrix which must be a two-element vector of non-negative integers.
- **Dimnames**: inherited from class "Matrix"
- **factors**: Object of class "list". A named list of factorizations that have been computed for the matrix.
- **sd**: (for "corMatrix") a numeric vector of length n containing the (original) $\sqrt{\text{var}(.)}$ entries which allow reconstruction of a covariance matrix from the correlation matrix.

Extends

Class "dsyMatrix", directly. Classes "dgeMatrix", "symmetricMatrix", and many more by class "dsyMatrix".

Methods

- **chol** signature(x = "dpoMatrix"): Returns (and stores) the Cholesky decomposition of x, see chol.
- **determinant** signature(x = "dpoMatrix"): Returns the determinant of x, via chol(x), see above.
- **rcond** signature(x = "dpoMatrix", norm = "character"): Returns (and stores) the reciprocal of the condition number of x. The norm can be "0" for the one-norm (the default) or "1" for the infinity-norm. For symmetric matrices the result does not depend on the norm.
- **solve** signature(a = "dpoMatrix", b = "...."), and
- **solve** signature(a = "dppMatrix", b = "....") work via the Cholesky composition, see also the Matrix solve-methods.
- **Arith** signature(e1 = "dpoMatrix", e2 = "numeric") (and quite a few other signatures): The result of ("elementwise" defined) arithmetic operations is typically not positive-definite anymore. The only exceptions, currently, are multiplications, divisions or additions with positive length(.) == 1 numbers (or logicals).

See Also

Classes dsyMatrix and dgeMatrix; further, Matrix, rcond, chol, solve, crossprod.
Examples

```r
h6 <- Hilbert(6)
rcond(h6)
str(h6)
str(h6 * 27720) # is "integer"
solve(h6)
str(hp6 <- as(h6, "dppMatrix"))

### Note that as(*, "corMatrix") *scales* the matrix
(ch6 <- as(h6, "corMatrix"))
stopifnot(all.equal(h6 * 27720, round(27720 * h6), tolerance = 1e-14),
           all.equal(ch6@sd(-2), 2*(1:6)-1, tolerance= 1e-12))
chch <- chol(ch6)
stopifnot(identical(chch, ch6@factors$Cholesky),
           all(abs(crossprod(chch) - ch6) < 1e-10))
```

---

**drop0**

*Drop "Explicit Zeroes" from a Sparse Matrix*

**Description**

Returns a sparse matrix with no “explicit zeroes”, i.e., all zero or FALSE entries are dropped from the explicitly indexed matrix entries.

**Usage**

```r
drop0(x, tol = 0, is.Csparse = NA)
```

**Arguments**

- `x`: a Matrix, typically sparse, i.e., inheriting from `sparseMatrix`.
- `tol`: non-negative number to be used as tolerance for checking if an entry $x_{i,j}$ should be considered to be zero.
- `is.Csparse`: logical indicating prior knowledge about the “Csparseness” of `x`. This exists for possible speedup reasons only.

**Value**

A Matrix like `x` but with no explicit zeros, i.e., `!any(x@x == 0)`, always inheriting from `CsparseMatrix`.

**Note**

When a sparse matrix is the result of matrix multiplications, you may want to consider combining `drop0()` with `zapsmall()`, see the example.

**See Also**

`spMatrix`, `class sparseMatrix`
Examples

m <- spMatrix(10,20, i= 1:8, j=2:9, x = c(0:2,3:-1))
m
drop0(m)

## A larger example:
t5 <- new("dsCMatrix", Dim = c(5L, 5L), uplo = "L",
    x = c(10, 1, 3, 10, 1, 10, 1, 10, 10),
    i = c(0L,2L,4L, 1L, 3L,2L,4L, 3L, 4L),
    p = c(0L, 3L, 5L, 7:9))
TT <- kronecker(t5, kronecker(kronecker(t5,t5), t5))
IT <- solve(TT)
I. <- TT %*% IT ; nnzero(I.) # 697 ( = 625 + 72 )
I.0 <- drop0(zapsmall(I.))
## which actually can be more efficiently achieved by
I.. <- drop0(I., tol = 1e-15)
stopifnot(all(I.0 == Diagonal(625)),
    nnzero(I..) == 625)

---

dsCMatrix-class Numeric Symmetric Sparse (column compressed) Matrices

Description

The dsCMatrix class is a class of symmetric, sparse numeric matrices in the compressed, column-oriented format. In this implementation the non-zero elements in the columns are sorted into increasing row order.

The dstMatrix class is the class of symmetric, sparse numeric matrices in triplet format.

Objects from the Class

Objects can be created by calls of the form new("dsCMatrix", ...) or new("dstMatrix", ...), or automatically via e.g., as(*, "symmetricMatrix"), or for dsCMatrix also from Matrix(.

Creation “from scratch” most efficiently happens via sparseMatrix(*, symmetric=TRUE).

Slots

uplo: A character object indicating if the upper triangle ("U") or the lower triangle ("L") is stored.

i: Object of class "integer" of length nnZ (half number of non-zero elements). These are the row numbers for each non-zero element in the lower triangle of the matrix.

p: (only in class "dsCMatrix") an integer vector for providing pointers, one for each column, see the detailed description in CsparseMatrix.

j: (only in class "dstMatrix") Object of class "integer" of length nnZ (as i). These are the column numbers for each non-zero element in the lower triangle of the matrix.

x: Object of class "numeric" of length nnZ – the non-zero elements of the matrix (to be duplicated for full matrix).
factors: Object of class "list" - a list of factorizations of the matrix.
Dim: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.

Extends
Both classes extend classes and symmetricMatrix dsparseMatrix directly; dsCMatrix further directly extends CsparseMatrix, where dsTMatrix does TsparseMatrix.

Methods

solve signature(a = "dsCMatrix", b = "..."): x <- solve(a,b) solves $Ax = b$ for $x$; see solve-methods.

chol signature(x = "dsCMatrix", pivot = "logical"): Returns (and stores) the Cholesky decomposition of $x$, see chol.

Cholesky signature(A = "dsCMatrix",...): Computes more flexibly Cholesky decompositions, see Cholesky.

determinant signature(x = "dsCMatrix", logarithm = "missing"): Evaluate the determinant of $x$ on the logarithm scale. This creates and stores the Cholesky factorization.

determinant signature(x = "dsCMatrix", logarithm = "logical"): Evaluate the determinant of $x$ on the logarithm scale or not, according to the logarithm argument. This creates and stores the Cholesky factorization.

t signature(x = "dsCMatrix"): Transpose. As for all symmetric matrices, a matrix for which the upper triangle is stored produces a matrix for which the lower triangle is stored and vice versa, i.e., the uplo slot is swapped, and the row and column indices are interchanged.

t signature(x = "dsTMatrix"): Transpose. The uplo is swapped from "U" to "L" or vice versa, as for a "dsCMatrix", see above.

coerce signature(from = "dsCMatrix", to = "dgTMatrix")
coerce signature(from = "dsCMatrix", to = "dgeMatrix")
coerce signature(from = "dsCMatrix", to = "matrix")
coerce signature(from = "dsTMatrix", to = "dgeMatrix")
coerce signature(from = "dsTMatrix", to = "dsCMatrix")
coerce signature(from = "dsTMatrix", to = "dsyMatrix")
coerce signature(from = "dsTMatrix", to = "matrix")

See Also
Classes dgCMatrix, dgTMatrix, dgeMatrix and those mentioned above.

Examples

mm <- Matrix(toeplitz(c(10, 0, 1, 0, 3)), sparse = TRUE)
mm # automatically dsCMatrix
str(mm)
## Description

The Class "dsparseMatrix" is the virtual (super) class of all numeric sparse matrices.

## Slots

- **Dim**: the matrix dimension, see class "Matrix".
- **Dimnames**: see the "Matrix" class.
- **x**: a numeric vector containing the (non-zero) matrix entries.

## Extends

- Class "dMatrix" and "sparseMatrix", directly.
- Class "Matrix", by the above classes.

## See Also

- the documentation of the (non virtual) sub classes, see `showClass("dsparseMatrix")`; in particular, dgTMatrix, dgCMatrix, and dgRMatrix.

## Examples

- `showClass("dsparseMatrix")`
**Description**

The dsRMatrix class is a class of symmetric, sparse matrices in the compressed, row-oriented format. In this implementation the non-zero elements in the rows are sorted into increasing column order.

**Objects from the Class**

These ".RMMatrix" classes are currently still mostly unimplemented!

Objects can be created by calls of the form `new("dsRMatrix", ...)`.

**Slots**

- `uplo`: A character object indicating if the upper triangle ("U") or the lower triangle ("L") is stored. At present only the lower triangle form is allowed.
- `j`: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the row numbers for each non-zero element in the matrix.
- `p`: Object of class "integer" of pointers, one for each row, to the initial (zero-based) index of elements in the row.
- `factors`: Object of class "list" - a list of factorizations of the matrix.
- `x`: Object of class "numeric" - the non-zero elements of the matrix.
- `Dim`: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.
- `Dimnames`: List of length two, see `Matrix`.  

**Extends**

Classes `RsparseMatrix`, `dsparseMatrix` and `symmetricMatrix`, directly.

Class "dMatrix", by class "dsparseMatrix", class "sparseMatrix", by class "dsparseMatrix" or "RsparseMatrix": class "compMatrix" by class "symmetricMatrix" and of course, class "Matrix".

**Methods**

- **forceSymmetric** signature(x = "dsRMatrix", uplo = "missing"): a trivial method just returning x
- **forceSymmetric** signature(x = "dsRMatrix", uplo = "character"): if uplo == x@uplo, this trivially returns x; otherwise t(x).
- **coerce** signature(from = "dsCMatrix", to = "dsRMatrix")
**dsyMatrix-class**

**Symmetric Dense Numeric Matrices**

**Description**

The "dsyMatrix" class is the class of symmetric, dense matrices in non-packed storage and "dspMatrix" is the class of symmetric dense matrices in packed storage. Only the upper triangle or the lower triangle is stored.

**Objects from the Class**

Objects can be created by calls of the form `new("dsyMatrix", ...).`

**Slots**

- `uplo`: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- `x`: Object of class "numeric". The numeric values that constitute the matrix, stored in column-major order.
- `Dim,Dimnames`: The dimension (a length-2 "integer") and corresponding names (or NULL), see the `Matrix`.
- `factors`: Object of class "list". A named list of factorizations that have been computed for the matrix.

**Extends**

"dsyMatrix" extends class "dgeMatrix", directly, whereas "dspMatrix" extends class "ddenseMatrix", directly.

Both extend class "symmetricMatrix", directly, and class "Matrix" and others, indirectly, use `showClass("dsyMatrix")`, e.g., for details.
Methods

coerce signature(from = "ddenseMatrix", to = "dgeMatrix")
coerce signature(from = "dspMatrix", to = "matrix")
coerce signature(from = "dsyMatrix", to = "matrix")
coerce signature(from = "dsyMatrix", to = "dspMatrix")
coerce signature(from = "dspMatrix", to = "dsyMatrix")
norm signature(x = "dspMatrix", type = "character"), or x = "dsyMatrix" or type = "missing": Computes the matrix norm of the desired type, see, norm.
rcond signature(x = "dspMatrix", type = "character"), or x = "dsyMatrix" or type = "missing": Computes the reciprocal condition number, rcond().
solve signature(a = "dspMatrix", b = "...."), and solve signature(a = "dsyMatrix", b = "...."): x <- solve(a,b) solves $Ax = b$ for $x$; see solve-methods.

signature(x = "dsyMatrix"): Transpose; swaps from upper triangular to lower triangular storage, i.e., the uplo slot from "U" to "L" or vice versa, the same as for all symmetric matrices.

See Also

Classes dgeMatrix and Matrix; solve, norm, rcond, t

Examples

## Only upper triangular part matters (when uplo == "U" as per default)
(sy2 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, NA,32,77)))
str(t(sy2)) # uplo = "L", and the lower tri. (i.e. NA is replaced).

chol(sy2) #-> "Cholesky" matrix
(sp2 <- pack(sy2)) # a "dspMatrix"

## Coercing to dpoMatrix gives invalid object:
sy3 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, -1, 2, -7))
try(as(sy3, "dpoMatrix")) # -> error: not positive definite

---

**dtCMatrix-class**

**Triangular, (compressed) sparse column matrices**

**Description**

The "dtCMatrix" class is a class of triangular, sparse matrices in the compressed, column-oriented format. In this implementation the non-zero elements in the columns are sorted into increasing row order. The "dtTMatrix" class is a class of triangular, sparse matrices in triplet format.
Objects from the Class

Objects can be created by calls of the form `new("dtCMatrix", ...)` or calls of the form `new("dtTMatrix", ...)`, but more typically automatically via `Matrix()` or coercion such as `as(x, "dtCMatrix")`.

Slots

- `uplo`: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- `diag`: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see `triangularMatrix`.
- `p`: (only present in "dtCMatrix"): an integer vector for providing pointers, one for each column, see the detailed description in `CsparseMatrix`.
- `i`: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the row numbers for each non-zero element in the matrix.
- `j`: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the column numbers for each non-zero element in the matrix. (Only present in the `dtTMatrix` class.)
- `x`: Object of class "numeric" - the non-zero elements of the matrix.
- `Dim`, `Dimnames`: The dimension (a length-2 "integer") and corresponding names (or `NULL`), inherited from the `Matrix`, see there.

Extends

Class "dgCMatrix", directly. Class "triangularMatrix", directly. Class "dMatrix", "sparseMatrix", and more by class "dgCMatrix" etc, see the examples.

Methods

- `coerce` signature(from = "dtCMatrix", to = "dgTMatrix")
- `coerce` signature(from = "dtCMatrix", to = "dgeMatrix")
- `coerce` signature(from = "dtTMatrix", to = "dgeMatrix")
- `coerce` signature(from = "dtTMatrix", to = "dtrMatrix")
- `coerce` signature(from = "dtTMatrix", to = "matrix")
- `solve` signature(a = "dtCMatrix", b = "..."): sparse triangular solve (aka “backsolve” or “forwardsolve”), see `solve-methods`.
- `t` signature(x = "dtCMatrix"): returns the transpose of x
- `t` signature(x = "dtTMatrix"): returns the transpose of x

See Also

Classes `dgCMatrix`, `dgTMatrix`, `dgeMatrix`, and `dtrMatrix`. 
Examples

```r
showClass("dCMatrix")
showClass("dTMatrix")
t1 <- new("dTMatrix", x= c(3,7), i= 0:1, j=3:2, Dim= as.integer(c(4,4)))
t1
## from 0-diagonal to unit-diagonal (low-level step):
tu <- t1; tu@diag <- "U"
tu
(cu <- as(tu, "dCMatrix"))
str(cu)# only two entries in @i and @x
stopifnot(cu@i == 1:0,
    all(2 * symmpart(cu) == Diagonal(4) + forceSymmetric(cu))

t1[1,2:3] <- -1:-2
diag(t1) <- 10*c(1:2,3:2)
t1 # still triangular
(it1 <- solve(t1))
t1. <- solve(it1)
all(abs(t1 - t1.) < 10 * .Machine$double.eps)

## 2nd example
U5 <- new("dCMatrix", i= c(1L, 0:3), p=0L,0L,0:2, 5L), Dim = c(5L, 5L),
    x = rep(1, 5), diag = "U")
U5
(iu <- solve(U5)) # contains one '0'
validObject(iu2 <- solve(U5, Diagonal(5)))# failed in earlier versions

I5 <- iu %*% U5 # should equal the identity matrix
i5 <- iu2 %*% U5
m53 <- matrix(1:15, 5,3, dimnames=list(NULL,letters[1:3]))
asDiag <- function(M) as(drop0(M), "diagonalMatrix")
stopifnot(
    all.equal(Diagonal(5), asDiag(I5), tolerance=1e-14),
    all.equal(Diagonal(5), asDiag(i5), tolerance=1e-14),
    identical(list(NULL, dimnames(m53)[[2]]), dimnames(solve(U5, m53)))
)
```

**dtpMatrix-class**

### Packed Triangular Dense Matrices - "dtpMatrix"

**Description**

The "dtpMatrix" class is the class of triangular, dense, numeric matrices in packed storage. The "dtrMatrix" class is the same except in nonpacked storage.

**Objects from the Class**

Objects can be created by calls of the form `new("dtpMatrix", ...)` or by coercion from other classes of matrices.
Slots

uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.

diag: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see \texttt{triangularMatrix}.

x: Object of class "numeric". The numeric values that constitute the matrix, stored in column-major order. For a packed square matrix of dimension $d \times d$, length(x) is of length $d(d+1)/2$ (also when diag == "U"!).

\texttt{Dim,Dimnames}: The dimension (a length-2 "integer") and corresponding names (or NULL), inherited from the \texttt{Matrix}, see there.

Extends

Class "ddenseMatrix", directly. Class "triangularMatrix", directly. Class "dMatrix" and more by class "ddenseMatrix" etc, see the examples.

Methods

\texttt{\%\%\% signature(x = "dtpMatrix", y = "dgeMatrix")}: Matrix multiplication; ditto for several other signature combinations, see \texttt{showMethods("\%\%\%", class = "dtpMatrix")}.

\texttt{coerce signature(from = "dtpMatrix", to = "dtrMatrix")}

\texttt{coerce signature(from = "dtpMatrix", to = "matrix")}

\texttt{determinant signature(x = "dtpMatrix", logarithm = "logical")}: the \texttt{determinant(x)} trivially is $\prod\text{diag}(x)$, but computed on log scale to prevent over- and underflow.

\texttt{diag signature(x = "dtpMatrix")}: ...

\texttt{norm signature(x = "dtpMatrix", type = "character")}: ...

\texttt{rcond signature(x = "dtpMatrix", norm = "character")}: ...

\texttt{solve signature(a = "dtpMatrix", b = "..."}): efficiently using internal backsolve or forwardsolve, see \texttt{solve-methods}.

\texttt{t signature(x = "dtpMatrix")}: \texttt{t(x)} remains a "dtpMatrix", lower triangular if \texttt{x} is upper triangular, and vice versa.

See Also

Class \texttt{dtrMatrix}

Examples

\texttt{showClass("dtrMatrix")}

\texttt{example("dtrMatrix-class", echo=FALSE)}

(p1 <- as(T2, "dtpMatrix"))

str(p1)

(pp <- as(T, "dtpMatrix"))

ipl <- solve(p1)

stopifnot(length(p1@x) == 3, length(pp@x) == 3,
dtRMatrix-class

Triangular Sparse Compressed Row Matrices

Description

The dtRMatrix class is a class of triangular, sparse matrices in the compressed, row-oriented format. In this implementation the non-zero elements in the rows are sorted into increasing column order.

Objects from the Class

This class is currently still mostly unimplemented!

Objects can be created by calls of the form `new("dtRMatrix", ...)`.

Slots

- `uplo`: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular. At present only the lower triangle form is allowed.
- `diag`: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see `triangularMatrix`.
- `j`: Object of class "integer" of length `nnzero(.)` (number of non-zero elements). These are the row numbers for each non-zero element in the matrix.
- `p`: Object of class "integer" of pointers, one for each row, to the initial (zero-based) index of elements in the row. (Only present in the dsRMatrix class.)
- `x`: Object of class "numeric" - the non-zero elements of the matrix.
- `Dim`: The dimension (a length-2 "integer")
- `Dimnames`: corresponding names (or NULL), inherited from the `Matrix`, see there.

Extends

Class "dgRMatrix", directly. Class "dsparseMatrix", by class "dgRMatrix". Class "dMatrix", by class "dgRMatrix". Class "sparseMatrix", by class "dgRMatrix". Class "Matrix", by class "dgRMatrix".

Methods

No methods currently with class "dsRMatrix" in the signature.
See Also

Classes `dgCMatrix`, `dgTMatrix`, `dgeMatrix`

Examples

```r
(m0 <- new("dtrMatrix"))
(m2 <- new("dtrMatrix", Dim = c(2L,2L),
         x = c(5, 1:2), p = c(0L,2:3), j = c(0:1,1L)))
str(m2)
(m3 <- as(Diagonal(2), "RsparseMatrix")) # -> dtRMatrix
```

**dtrMatrix-class**  
*Triangular, dense, numeric matrices*

**Description**

The "dtrMatrix" class is the class of triangular, dense, numeric matrices in nonpacked storage. The "dtpMatrix" class is the same except in packed storage.

**Objects from the Class**

Objects can be created by calls of the form `new("dtrMatrix", ...)`. 

**Slots**

- `uplo`: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- `diag`: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see `triangularMatrix`
- `x`: Object of class "numeric". The numeric values that constitute the matrix, stored in column-major order.
- `Dim`: Object of class "integer". The dimensions of the matrix which must be a two-element vector of non-negative integers.

**Extends**

Class "ddenseMatrix", directly. Class "triangularMatrix", directly. Class "Matrix" and others, by class "ddenseMatrix".

**Methods**

Among others (such as matrix products, e.g. `?crossprod-methods`),

- `coerce signature(from = "dgeMatrix", to = "dtrMatrix")`
- `coerce signature(from = "dtrMatrix", to = "matrix")`
- `coerce signature(from = "dtrMatrix", to = "ltrMatrix")`
**coerce** signature(from = "dtrMatrix", to = "matrix")

**coerce** signature(from = "matrix", to = "dtrMatrix")

**norm** signature(x = "dtrMatrix", type = "character")

**rcond** signature(x = "dtrMatrix", norm = "character")

**solve** signature(a = "dtrMatrix", b = "....") efficiently use a “forwardsolve” or backsolve for a lower or upper triangular matrix, respectively, see also **solve-methods**.

+, -, *, ..., ==, >=, ... all the Ops group methods are available. When applied to two triangular matrices, these return a triangular matrix when easily possible.

**See Also**

Classes **ddenseMatrix, dtpMatrix, triangularMatrix**

**Examples**

```r
(m <- rbind(2:3, 0:-1))
(M <- as(m, "dgeMatrix"))

(T <- as(M, "dtrMatrix")) ## upper triangular is default
(T2 <- as(t(M), "dtrMatrix"))
stopifnot(T@uplo == "U", T2@uplo == "L", identical(T2, t(T)))
```

---

**Description**

Expands decompositions stored in compact form into factors.

**Usage**

```r
expand(x, ...)
```

**Arguments**

- **x** a matrix decomposition.
- **...** further arguments passed to or from other methods.

**Details**

This is a generic function with special methods for different types of decompositions, see **showMethods**(expand) to list them all.

**Value**

The expanded decomposition, typically a list of matrix factors.
expm

Note
Factors for decompositions such as lu and qr can be stored in a compact form. The function expand allows all factors to be fully expanded.

See Also
The LU lu, and the Cholesky decompositions which have expand methods; facmul.

Examples
(x <- Matrix(round(rnorm(9),2), 3, 3))
(ex <- expand(lux <- lu(x)))

Description
Compute the exponential of a matrix.

Usage
expm(x)

Arguments
x  
a matrix, typically inheriting from the dMatrix class.

Details
The exponential of a matrix is defined as the infinite Taylor series \( \expm(A) = I + A + A^2/2! + A^3/3! + \ldots \) (although this is definitely not the way to compute it). The method for the dgeMatrix class uses Ward’s diagonal Pade’ approximation with three step preconditioning.

Value
The matrix exponential of x.

Note
The expm package contains newer (partly faster and more accurate) algorithms for expm() and includes logm and sqrtm.

Author(s)
This is a translation of the implementation of the corresponding Octave function contributed to the Octave project by A. Scottedward Hodel <A.S.Hodel@Eng.Auburn.EDU>. A bug in there has been fixed by Martin Maechler.
References

http://en.wikipedia.org/wiki/Matrix_exponential


See Also

Schur; additionally, expm, logm, etc in package expm.

Examples

(ml <- Matrix(c(1,0,1), nc = 2))
(e1 <- expm(ml)); e <- exp(1)
stopifnot(all.equal(e10x, c(e,0,e,e), tolerance = 1e-15))
(m2 <- Matrix(c(-49, -64, 24, 31), nc = 2))
(e2 <- expm(m2))
(m3 <- Matrix(cbind(0, rbind(6*diag(3),0)))# sparse!
(e3 <- expm(m3)) # upper triangular

externalFormats

Read and write external matrix formats

Description

Read matrices stored in the Harwell-Boeing or MatrixMarket formats or write sparseMatrix objects to one of these formats.

Usage

readHB(file)
readMM(file)
writeMM(obj, file, ...)

Arguments

obj a real sparse matrix
file for writeMM - the name of the file to be written. For readHB and readMM the name of the file to read, as a character scalar. The names of files storing matrices in the Harwell-Boeing format usually end in “.rua” or “.rsa”. Those storing matrices in the MatrixMarket format usually end in “.mtx”. Alternatively, readHB and readMM accept connection objects.
...
optional additional arguments. Currently none are used in any methods.
Value

The `readHB` and `readMM` functions return an object that inherits from the "Matrix" class. Methods for the `writeMM` generic functions usually return `NULL` and, as a side effect, the matrix `obj` is written to file in the MatrixMarket format (writeMM).

Note

The Harwell-Boeing format is older and less flexible than the MatrixMarket format. The function `writeHB` was deprecated and has now been removed. Please use `writeMM` instead.

A very simple way to export small sparse matrices `S`, is to use `summary(S)` which returns a `data.frame` with columns `i`, `j`, and possibly `x`, see `summary` in `sparseMatrix-class`, and an example below.

References

http://math.nist.gov/MatrixMarket
http://www.cise.ufl.edu/research/sparse/matrices

Examples

```r
str(pores <- readMM(system.file("external/pores_1.mtx", 
    package = "Matrix")))
str(utm <- readHB(system.file("external/utm300.rua", 
    package = "Matrix")))
str(lunda <- readMM(system.file("external/lund_a.mtx", 
    package = "Matrix")))
str(lunda <- readHB(system.file("external/lund_a.rsa", 
    package = "Matrix")))
## Not run:
## NOTE: The following examples take quite some time
## ----- even on a fast internet connection:
if(FALSE) # the URL has been corrected, but we need an un-tar step!
str(sm <- readHB(gzcon(url("http://www.cise.ufl.edu/research/sparse/RB/Boeing/msc00726.tar.gz"))))

str(jgl009 <- 
    readMM(gzcon(url("ftp://math.nist.gov/pub/MatrixMarket2/Harwell-Boeing/counterx/jgl009.mtx.gz"))))

## End(Not run)
data(KNex)
writeMM(KNex$mm, "mmmm.mtx")

## very simple export - in triplet format - to text file:
data(CAex)
s.CA <- summary(CAex)
message("writing to ", outf <- tempfile())
write.table(s.CA, file = outf, row.names=FALSE)
## and read it back -- showing off sparseMatrix():
dd <- read.table(outf, header=TRUE)
mm <- do.call(sparseMatrix, dd)
stopifnot(all.equal(mm, CAex, tolerance=1e-15))
```
Description

Performs multiplication by factors for certain decompositions (and allows explicit formation of those factors).

Usage

\texttt{facmul(x, factor, y, transpose, left, ...)}

Arguments

\begin{itemize}
  \item \texttt{x} \hspace{1em} a matrix decomposition. No missing values or IEEE special values are allowed.
  \item \texttt{factor} \hspace{1em} an indicator for selecting a particular factor for multiplication.
  \item \texttt{y} \hspace{1em} a matrix or vector to be multiplied by the factor or its transpose. No missing values or IEEE special values are allowed.
  \item \texttt{transpose} \hspace{1em} a logical value. When FALSE (the default) the factor is applied. When TRUE the transpose of the factor is applied.
  \item \texttt{left} \hspace{1em} a logical value. When TRUE (the default) the factor is applied from the left. When FALSE the factor is applied from the right.
  \item \texttt{...} \hspace{1em} the method for "qr.Matrix" has additional arguments.
\end{itemize}

Value

the product of the selected factor (or its transpose) and \texttt{y}

NOTE

Factors for decompositions such as \texttt{lu} and \texttt{qr} can be stored in a compact form. The function \texttt{facmul} allows multiplication without explicit formation of the factors, saving both storage and operations.

References


Examples

\begin{verbatim}
library(Matrix)
x <- Matrix(rnorm(9), 3, 3)
## Not run:
qrx <- qr(x) # QR factorization of x
y <- rnorm(3)
facmul( qr(x), factor = "Q", y) # form Q y
## End(Not run)
\end{verbatim}
forceSymmetric

**Description**

Force a square matrix \( x \) to a `symmetricMatrix`, **without** a symmetry check as it would be applied for `as(x, "symmetricMatrix")`.

**Usage**

`forceSymmetric(x, uplo)`

**Arguments**

- **x**
  - any square matrix (of numbers), either "traditional" (`matrix`) or inheriting from `Matrix`.
- **uplo**
  - optional string, "U" or "L" indicating which “triangle” half of \( x \) should determine the result. The default is "U" unless \( x \) already has a `uplo` slot (i.e., when it is `symmetricMatrix`, or `triangularMatrix`), where the default will be \( x@uplo \).

**Value**

a square matrix inheriting from class `symmetricMatrix`.

**See Also**

`symmpart` for the symmetric part of a matrix, or the coercions `as(x, <symmetricMatrix class>)`.

**Examples**

```r
## Hilbert matrix
i <- 1:6
h6 <- 1/outer(i - 1L, i, "+")
sd <- sqrt(diag(h6))
hh <- t(h6/sd)/sd # theoretically symmetric
isSymmetric(hh, tol=0) # FALSE; hence
try( as(hh, "symmetricMatrix") ) # fails, but this works fine:
H6 <- forceSymmetric(hh)

## result can be pretty surprising:
(M <- Matrix(1:36, 6))
forceSymmetric(M) # symmetric, hence very different in lower triangle
(tm <- tril(M))
forceSymmetric(tm)
```
formatSparseM  Formatting Sparse Numeric Matrices Utilities

Description

Utilities for formatting sparse numeric matrices in a flexible way. These functions are used by the `format` and print methods for sparse matrices and can be applied as well to standard R matrices. Note that all arguments but the first are optional.

`formatSparseM()` is the main “workhorse” of `formatSpMatrix`, the format method for sparse matrices.

`.formatSparseSimple()` is a simple helper function, also dealing with (short/empty) column names construction.

Usage

```r
formatSparseM(x, zero.print = ".", align = c("fancy", "right"),
   m = as(x,"matrix"), asLogical=NULL, digits=NULL,
   cx, iN0, dn = dimnames(m))

.formatSparseSimple(m, asLogical=FALSE, digits=NULL,
   col.names, note.dropping.colnames = TRUE,
   dn=dimnames(m))
```

Arguments

- **x**: an R object inheriting from class `sparseMatrix`.
- **zero.print**: character which should be used for structural zeroes. The default "." may occasionally be replaced by " " (blank); using "0" would look almost like print()ing of non-sparse matrices.
- **align**: a string specifying how the zero.print codes should be aligned, see `formatSpMatrix`.
- **m**: (optional) a (standard R) matrix version of x.
- **asLogical**: should the matrix be formatted as a logical matrix (or rather as a numeric one); mostly for `formatSparseM()`.
- **digits**: significant digits to use for printing, see `print.default`.
- **cx**: (optional) character matrix; a formatted version of x, still with strings such as "0.00" for the zeros.
- **iN0**: (optional) integer vector, specifying the location of the non-zeroes of x.
- **col.names**, **note.dropping.colnames**: see `formatSpMatrix`.
- **dn**: `dimnames` to be used; a list (of length two) with row and column names (or `NULL`).
Value

A character matrix like `cx`, where the zeros have been replaced with (padded versions of) `zeroNprint`. As this is a **dense** matrix, do not use these functions for really large (really) sparse matrices!

Author(s)

Martin Maechler

See Also

`formatSpMatrix` which calls `formatSparseM()` and is the `format` method for sparse matrices.

`printSpMatrix` which is used by the (typically implicitly called) `show` and `print` methods for sparse matrices.

Examples

```r
m <- suppressWarnings(matrix(c(0, 3.2, 0, 0, 11, 0, 0, 0, -7, 0), 4, 9))
fm <- formatSparseM(m)
noquote(fm)
## nice, but this is nicer:
print(fm, quote=FALSE, right=TRUE)
## and "the same" as:
Matrix(m)

## `align = "right"` is cheaper -- the "." are not aligned:
noquote(f2 <- formatSparseM(m, align="r"))
stopifnot(f2 == fm | m == 0, dim(f2) == dim(m),
          (f2 == ".") == (m == 0))
```

generalMatrix-class

Class "generalMatrix" of General Matrices

Description

Virtual class of “general” matrices; i.e., matrices that do not have a known property such as symmetric, triangular, or diagonal.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

- `factors`
- `Dim`

Dimnames: all slots inherited from `compMatrix`; see its description.
Extends

Class "compMatrix", directly. Class "Matrix", by class "compMatrix".

See Also

Classes compMatrix, and the non-general virtual classes: symmetricMatrix, triangularMatrix, diagonalMatrix.

descriptions "graph" – (sparse) Matrix

Description

The Matrix package has supported conversion from and to "graph" objects from (Bioconductor) package graph since summer 2005, via the usual as(.) "<class>" coercion,

as(from, Class)

Since 2013, this functionality is further exposed as the graph2T() and T2graph() functions (with further arguments than just from), which convert graphs to and from the triplet form of sparse matrices (of class "TsparseMatrix").

Usage

graph2T(from, use.weights = )
T2graph(from, need.uniq = is_not_uniqT(from), edgemode = NULL)

Arguments

from for graph2T(), an R object of class "graph";
for T2graph(), a sparse matrix inheriting from "TsparseMatrix".
use.weights logical indicating if weights should be used, i.e., equivalently the result will be numeric, i.e. of class dgTMatrix; otherwise the result will be ngTMatrix or nsTMatrix, the latter if the graph is undirected. The default looks if there are weights in the graph, and if any differ from 1, weights are used.
need.uniq a logical indicating if from may need to be internally "uniqified"; do not set this and hence rather use the default, unless you know what you are doing!
edgemode one of NULL, "directed", or "undirected". The default NULL looks if the matrix is symmetric and assumes "undirected" in that case.

Value

For graph2T(), a sparse matrix inheriting from "TsparseMatrix".
For T2graph() an R object of class "graph".
See Also

Note that the CRAN package `igraph` also provides conversions from and to sparse matrices (of package `Matrix`) via its `graph.adjacency()` and `get.adjacency()`.

Examples

```r
if(isTRUE(try(require(graph)))) { 
  n4 <- LETTERS[1:4]; dns <- list(n4,n4)
  show(a1 <- sparseMatrix(i= c(1:4), j=c(2:4,1), x = 2, dimnames=dns))
  show(g1 <- as(a1, "graph")) # directed
  unlist(edgeWeights(g1)) # all '2'

  show(a2 <- sparseMatrix(i= c(1:4,4), j=c(2:4,1:2), x = TRUE, dimnames=dns))
  show(g2 <- as(a2, "graph")) # directed
  # now if you want it undirected:
  show(g3 <- T2graph(as(a2,"TsparseMatrix"), edgemode="undirected"))
  show(m3 <- as(g3,"Matrix"))
  show( graph2T(g3) ) # a "pattern Matrix" (nTMatrix)

  a. <- sparseMatrix(i= 4:1, j=1:4, dimnames=list(n4,n4), giveC=FALSE) # no 'x'
  show(a.) # "ngTMatrix"
  show(g. <- as(a., "graph"))
}
```

---

### Hilbert

**Generate a Hilbert matrix**

#### Description

Generate the `n` by `n` symmetric Hilbert matrix. Because these matrices are ill-conditioned for moderate to large `n`, they are often used for testing numerical linear algebra code.

#### Usage

```r
Hilbert(n)
```

#### Arguments

- `n` - a non-negative integer.

#### Value

the `n` by `n` symmetric Hilbert matrix as a "`dpoMatrix`" object.

#### See Also

the class `dpoMatrix`
**Examples**

Hilbert(6)

---

**Methods for image() in Package 'Matrix'**

**Description**

Methods for function `image` in package `Matrix`. An image of a matrix simply color codes all matrix entries and draws the \( n \times m \) matrix using an \( n \times m \) grid of (colored) rectangles.

**Usage**

```r
## S4 method for signature 'dgTMatrix'
image(x,
    xlim = c(1, di[2]),
    ylim = c(di[1], 1),
    aspect = "iso",
    sub = sprintf("Dimensions: %d x %d", di[1], di[2]),
    xlab = "Column", ylab = "Row", cuts = 15,
    useRaster = FALSE,
    useAbs = NULL, colorkey = !useAbs,
    col.regions = NULL,
    lwd = NULL, ...)```

**Arguments**

- `x`  
  a Matrix object, i.e., fulfilling `is(x, "Matrix")`.
- `xlim`, `ylim`  
  x- and y-axis limits; may be used to “zoom into” matrix. Note that `x,y` “feel reversed”: `ylim` is for the rows (= 1st index) and `xlim` for the columns (= 2nd index). For convenience, when the limits are integer valued, they are both extended by \( 0.5 \); also, `ylim` is always used decreasingly.
- `aspect`  
  aspect ratio specified as number (y/x) or string; see `levelplot`.
- `sub`, `xlab`, `ylab`  
  axis annotation with sensible defaults; see `plot.default`.
- `cuts`  
  number of levels the range of matrix values would be divided into.
- `useRaster`  
  logical indicating if raster graphics should be used (instead of the tradition rectangle vector drawing). If true, `panel.levelplot.raster` (from `lattice` package) is used, and the colorkey is also done via rasters, see also `levelplot` and possibly `grid.raster`.
  Note that using raster graphics may often be faster, but can be slower, depending on the matrix dimensions and the graphics device (dimensions).
- `useAbs`  
  logical indicating if `abs(x)` should be shown; if TRUE, the former (implicit) default, the default `col.regions` will be `grey` colors (and no colorkey drawn). The default is `FALSE` unless the matrix has no negative entries.
colorkey logical indicating if a color key aka ‘legend’ should be produced. Default is to draw one, unless useAbs is true. You can also specify a list, see levelplot, such as list(raster=TRUE) in the case of rastering.

col.regions vector of gradually varying colors; see levelplot.
lwd (only used when useRaster is false:) non-negative number or NULL (default), specifying the line-width of the rectangles of each non-zero matrix entry (drawn by grid.rect). The default depends on the matrix dimension and the device size.

... further arguments passed to methods and levelplot, notably at for specifying (possibly non equidistant) cut values for dividing the matrix values (superseding cuts above).

Methods

All methods currently end up calling the method for the dgTMatrix class. Use showMethods(image) to list them all.

Examples

showMethods(image)
# If you want to see all the methods' implementations:
showMethods(image, incl=TRUE, inherit=FALSE)

data(CAex)
image(CAex, main = "image(CAex)"
image(CAex, useAbs=TRUE, main = "image(CAex, useAbs=TRUE)"

cCA <- cholesky(crossprod(CAex), Imult = .01)
image(cCA, main="Cholesky(crossprod(CAex), Imult = .01)"
image(cCA, useAbs=TRUE)

data(USCounties)
image(USCounties)# huge
image(sign(USCounties))## just the pattern
  # how the result looks, may depend heavily on
  # the device, screen resolution, antialiasing etc
  # e.g. x11(type="Xlib") may show very differently than cairo-based

## Drawing borders around each rectangle;
# again, viewing depends very much on the device:
image(USCounties[1:400,1:200], lwd=.1)
## Using (xlim,ylim) has advantage : matrix dimension and (col/row) indices:
image(USCounties, c(1,200), c(1,400), lwd=.1)
image(USCounties, c(1,300), c(1,200), lwd=.5 )
image(USCounties, c(1,300), c(1,200), lwd=.01)

if(doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA")) || identical("true", unname(Sys.getenv("R_PKG_CHECKING_doExtras")))) {
  ## Using raster graphics: For PDF this would give a 77 MB file,
  ## however, for such a large matrix, this is typically considerably
  ## *slower* (than vector graphics rectangles) in most cases:
  # Use raster graphics
  # setraster = TRUE

}
if(doPNG <- !dev.interactive())
png("image-USCounties-raster.png", width=3200, height=3200)
image(USCounties, useRaster = TRUE) # should not suffer from anti-aliasing
if(doPNG)
  dev.off()
## and now look at the *.png image in a viewer you can easily zoom in and out
}#only if(doExtras)

---

index-class  Virtual Class "index" - Simple Class for Matrix Indices

Description

The class "index" is a virtual class used for indices (in signatures) for matrix indexing and sub-assignment of "Matrix" matrices.

In fact, it is currently implemented as a simple class union (setClassUnion) of "numeric", "logical" and "character".

Objects from the Class

Since it is a virtual Class, no objects may be created from it.

See Also

[-methods, and

Subassign-methods, also for examples.

Examples

showClass("index")

---

indMatrix-class  Index matrices

Description

The "indMatrix" class is the class of index matrices, stored as 1-based integer index vectors. An index matrix is a matrix with exactly one non-zero entry per row. Index matrices are useful for mapping observations to unique covariate values, for example.

Matrix (vector) multiplication with index matrices is equivalent to replicating and permuting rows, or "sampling rows with replacement", and is implemented that way in the Matrix package, see the ‘Details’ below.
Details

Matrix (vector) multiplication with index matrices from the left is equivalent to replicating and
permuting rows of the matrix on the right hand side. (Similarly, matrix multiplication with the
transpose of an index matrix from the right corresponds to selecting columns.) The crossproduct
of an index matrix \( M \) with itself is a diagonal matrix with the number of entries in each column of \( M \)
on the diagonal, i.e., \( M'M = \text{Diagonal}(x=\text{table}(M@\text{perm})) \).

Permutation matrices (of class \texttt{pMatrix}) are special cases of index matrices: They are square, of
dimension, say, \( n \times n \), and their index vectors contain exactly all of \( 1:n \).

While “row-indexing” (of more than one row or using \texttt{drop=FALSE}) stays within the “indMatrix”
class, all other subsetting/indexing operations (“column-indexing”, including, \texttt{diag}) on “indMatrix”
objets treats them as nonzero-pattern matrices ("ngTMatrix" specifically), such that non-matrix
subsetting results in \texttt{logical} vectors. Sub-assignment (\( M[i,j] \leftarrow v \)) is not sensible and hence an
error for these matrices.

Objects from the Class

Objects can be created by calls of the form \texttt{new("indMatrix", ...)} or by coercion from an
integer index vector, see below.

Slots

\texttt{perm}: An integer, 1-based index vector, i.e. an integer vector of length \texttt{Dim}[1] whose elements are
taken from \texttt{1:Dim[2]}.

\texttt{Dim}: Object of class "integer". The dimensions of the matrix which must skinny, i.e., the first
dimension has to be at least as large as the second.

\texttt{dimnames}: list of length two; each component containing NULL or a \texttt{character} vector length
equal the corresponding \texttt{dim} element.

Extends

Class "\texttt{sparseMatrix}" and "\texttt{generalMatrix}”, directly.

Methods

\texttt{%*%} signature(\( x = \text{"matrix"}, y = \text{"indMatrix"} \)) and other signatures (use \texttt{showMethods("%*%", class="indMatrix"})

\texttt{coerce} signature(\texttt{from} = \text{"integer"}, \texttt{to} = \text{"indMatrix"}): This enables typical "indMatrix"
construction, given an index vector from elements in \texttt{1:Dim[2]}, see the first example.

\texttt{coerce} signature(\texttt{from} = \text{"numeric"}, \texttt{to} = \text{"indMatrix"}): a user convenience, to allow
as(\texttt{perm}, \text{"indMatrix"}) for numeric \texttt{perm} with integer values.

\texttt{coerce} signature(\texttt{from} = \text{"list"}, \texttt{to} = \text{"indMatrix"}): The list must have two (integer-
valued) entries: the first giving the index vector with elements in \texttt{1:Dim[2]}, the second giving
\texttt{Dim[2]}. This allows "indMatrix" construction for cases in which the values represented by
the rightmost column(s) are not associated with any observations, i.e., in which the index does
not contain values \texttt{Dim[2]}, \texttt{Dim[2]-1}, \texttt{Dim[2]-2}, ...

\texttt{coerce} signature(\texttt{from} = \text{"indMatrix"}, \texttt{to} = \text{"matrix"}): coercion to a traditional FALSE/TRUE
matrix of \texttt{mode} \texttt{logical}. 
coerce} signature(x = "indMatrix", to = "ngTMatrix"): coercion to sparse logical matrix of class \texttt{ngTMatrix}.

\texttt{t} signature(x = "indMatrix"): return the transpose of the index matrix (which is no longer an \texttt{indMatrix}, but of class \texttt{ngTMatrix}).

\texttt{colSums}, \texttt{colMeans}, \texttt{rowSums}, \texttt{rowMeans} signature(x = "indMatrix"): return the column or row sums or means.

\texttt{rbind2} signature(x = "indMatrix", y = "indMatrix"): a fast method for rowwise catenation of two index matrices (with the same number of columns).

\texttt{kronecker} signature(X = "indMatrix", Y = "indMatrix"): return the kronecker product of two index matrices, which corresponds to the index matrix of the interaction of the two.

\textbf{Author(s)}

Fabian Scheipl <<...@uni-muenchen.de>> building on existing "pMatrix", after a nice hike's conversation with Martin Maechler and tweaks by the latter.

\textbf{See Also}

The permutation matrices \texttt{pMatrix} are special index matrices. The “pattern” matrices, \texttt{nMatrix} and its subclasses.

\textbf{Examples}

\begin{verbatim}
pl <- as(c(2,3,1), "pMatrix")
(sm1 <- as(rep(c(2,3,1), e=3), "indMatrix"))
stopifnot(all(sm1 == pl[rep(1:3, each=3),]))

## row-indexing of a <pMatrix> turns it into an <indMatrix>:
class(pl[rep(1:3, each=3),])

set.seed(12) # so we know '10' is in sample
## random index matrix for 50 observations and 10 unique values:
(s10 <- as(sample(10, 50, replace=TRUE),"indMatrix"))

## Sample rows of a numeric matrix:
(mm <- matrix(1:10, nrow=10, ncol=3))
s10 %% mm

set.seed(27)
IM1 <- as(sample(1:20, 100, replace=TRUE), "indMatrix")
IM2 <- as(sample(1:10, 100, replace=TRUE), "indMatrix")
(c12 <- crossprod(IM1,IM2))
## same as cross-tabulation of the two index vectors:
stopifnot(all(c12 - unclass(table(IM1@perm, IM2@perm)) == 0))

# 3 observations, 4 implied values, first does not occur in sample:
as(2:4, "indMatrix")
# 3 observations, 5 values, first and last do not occur in sample:
as(list(2:4, 5), "indMatrix")
\end{verbatim}
invPerm

Description

From a permutation vector \( p \), compute its inverse permutation vector.

Usage

\[
\text{invPerm}(p, \text{zero.p} = \text{FALSE}, \text{zero.res} = \text{FALSE})
\]

Arguments

- \( p \)
  an integer vector of length, say, \( n \).
- \( \text{zero.p} \)
  logical indicating if \( p \) contains values \( 0:(n-1) \) or rather (by default, \( \text{zero.p} = \text{FALSE} \)) \( 1:n \).
- \( \text{zero.res} \)
  logical indicating if the result should contain values \( 0:(n-1) \) or rather (by default, \( \text{zero.res} = \text{FALSE} \)) \( 1:n \).

Value

an integer vector of the same length (\( n \)) as \( p \). By default, (\( \text{zero.p} = \text{FALSE}, \text{zero.res} = \text{FALSE} \)), \( \text{invPerm}(p) \) is the same as \( \text{order}(p) \) or \( \text{sort.list}(p) \) and for that case, the function is equivalent to \( \text{invPerm} . \quad \text{<- function}(p) \{ \: p[p] <- \text{seq_along}(p) \; ; \; p \} \).

Author(s)

Martin Maechler

See Also

the class of permutation matrices, \texttt{pMatrix}.
Examples

```r
p <- sample(10) # a random permutation vector
ip <- invPerm(p)
p[ip] # == 1:10
## they are indeed inverse of each other:
stopifnot(
  identical(p[ip], 1:10),
  identical(ip[p], 1:10),
  identical(invPerm(ip), p)
)
```

Description

Methods for function `is.na()`, `is.finite()`, and `is.infinite()` for all Matrices (objects extending the `Matrix` class):

- `x = "denseMatrix"` returns a "nMatrix" object of same dimension as `x`, with TRUE’s whenever `x` is `NA`, finite, or infinite, respectively.
- `x = "sparseMatrix"` ditto.

Usage

```r
## S4 method for signature 'sparseMatrix'
is.na(x)
## S4 method for signature 'dsparseMatrix'
is.finite(x)
## S4 method for signature 'ddenseMatrix'
is.infinite(x)
## ...
## and for other classes

## S4 method for signature 'xMatrix'
anyNA(x)
## S4 method for signature 'nsparseMatrix'
anyNA(x)
## S4 method for signature 'sparseVector'
anyNA(x)
## S4 method for signature 'nsparseVector'
anyNA(x)
```

Arguments

- `x` sparse or dense matrix or sparse vector (here; any R object in general).
Description

Are the `dimnames` `dn` **NULL-like**?

`is.null.DN(dn)` is less strict than `is.null(dn)`, because it is also true (TRUE) when the `dimnames` `dn` are "like" NULL, or `list(NULL,NULL)`, as they can easily be for the traditional R matrices (matrix) which have no formal `class` definition, and hence much freedom in how their `dimnames` look like.

Usage

`is.null.DN(dn)`

Arguments

dn, `dimnames()` of a matrix-like R object.

Value

`logical` TRUE or FALSE.
Note
This function is really to be used on "traditional" matrices rather than those inheriting from `Matrix`, as the latter will always have dimnames `list(NULL,NULL)` exactly, in such a case.

Author(s)
Martin Maechler

See Also
`is.null`, `dimnames.matrix`.

Examples
```r
m <- matrix(round(100 * rnorm(6)), 2, 3); m1 <- m2 <- m3 <- m4 <- m
dimnames(m1) <- list(NULL, NULL)
dimnames(m2) <- list(NULL, character())
dimnames(m3) <- rev(dimnames(m2))
dimnames(m4) <- rep(list(character()), 2)

m4 # prints absolutely identically to m

stopifnot(m == m1, m1 == m2, m2 == m3, m3 == m4,
          identical(capture.output(m) -> cm,
                     capture.output(m1)),
          identical(cm, capture.output(m2)),
          identical(cm, capture.output(m3)),
          identical(cm, capture.output(m4))
```

Methods for Function `isSymmetric` in Package 'Matrix'

Description
`isSymmetric(M)` returns a `logical` indicating if `M` is a symmetric matrix. This (now) is a base function with a default method for the traditional matrices of class "matrix". Methods here are defined for virtual Matrix classes such that it works for all objects inheriting from class `Matrix`.

See Also
`forceSymmetric`, `symmpart`, and the formal class (and subclasses) "symmetricMatrix".

Examples
```r
isSymmetric(Diagonal(4)) # TRUE of course
M <- Matrix(c(1,2,2,1), 2,2)
isSymmetric(M) # TRUE (*and* of formal class "dsyMatrix")
isSymmetric(as(M, "dgeMatrix")) # still symmetric, even if not "formally"
isSymmetric(triu(M)) # FALSE
```
## isTriangular

### isTriangular() and isDiagonal() Methods

#### Description

`isTriangular(M)` returns a logical indicating if `M` is a triangular matrix. Analogously, `isDiagonal(M)` is true iff `M` is a diagonal matrix.

Contrary to `isSymmetric()`, these two functions are generically from package `Matrix`, and hence also define methods for traditional (`class "matrix"`) matrices.

#### Usage

- `isDiagonal(object)`
- `isTriangular(object, upper = NA, ...)`

#### Arguments

- `object`: any R object, typically a matrix (traditional or `Matrix` package).
- `upper`: logical, one of `NA` (default), `FALSE`, or `TRUE` where the last two cases require a lower or upper triangular object to result in `TRUE`.
- `...`: potentially further arguments for other methods.

#### Value

A (“scalar”) logical, `TRUE` or `FALSE`, never `NA`. For `isTriangular()`, if the result is `TRUE`, it may contain an attribute (see `attributes "kind"`, either "L" or "U" indicating if it is a lower or upper triangular matrix.

#### See Also

- `isSymmetric`;
- formal class (and subclasses) "`triangularMatrix`" and "`diagonalMatrix`".

#### Examples

```r
isTriangular(Diagonal(4))
#> is TRUE: a diagonal matrix is also (both upper and lower) triangular
(M <- Matrix(c(1,2,0,1), 2,2))
isTriangular(M) # TRUE (*and* of formal class "dtrMatrix")
isTriangular(as(M, "dgeMatrix")) # still triangular, even if not "formally"
isTriangular(crossprod(M)) # FALSE
isDiagonal(matrix(c(2,0,0,1), 2,2)) # TRUE
```
KhatriRao  

Khatri-Rao Matrix Product

Description

Computes Khatri-Rao products for any kind of matrices.

The Khatri-Rao product is a column-wise Kronecker product. Originally introduced by Khatri and Rao (1968), it has many different applications, see Liu and Trenkler (2008) for a survey. Notably, it is used in higher-dimensional tensor decompositions, see Bader and Kolda (2008).

Usage

\texttt{KhatriRao(X, Y = X, FUN = "\*", make.dimnames = FALSE)}

Arguments

- \texttt{X, Y} matrices of with the same number of columns.
- \texttt{FUN} the (name of the) \texttt{function} to be used for the column-wise Kronecker products, see \texttt{kronecker}, defaulting to the usual multiplication.
- \texttt{make.dimnames} logical indicating if the result should inherit \texttt{dimnames} from \texttt{X} and \texttt{Y} in a simple way.

Value

a \texttt{CsparseMatrix}, say \texttt{R}, the Khatri-Rao product of \texttt{X} (\texttt{n} \times \texttt{k}) and \texttt{Y} (\texttt{m} \times \texttt{k}), is of dimension \texttt{(n \cdot m) \times k}, where the \texttt{j}-th column, \texttt{R[,j]} is the kronecker product \texttt{kronecker(X[,j], Y[,j])}.

Note

The current implementation is efficient for large sparse matrices.

Author(s)

Michael Cysouw, Univ. Marburg; minor tweaks by Martin Maechler.

References


**See Also**

kronecker.

**Examples**

```r
## Example with very small matrices:
m <- matrix(1:12,3,4)
d <- diag(1:4)
KhatRao(m,d)
KhatRao(d,m)
dimnames(m) <- list(LETTERS[1:3], letters[1:4])
KhatRao(m,d, make.dimnames=TRUE)
KhatRao(d,m, make.dimnames=TRUE)
dimnames(d) <- list(NULL, paste0("D", 1:4))
KhatRao(m,d, make.dimnames=TRUE)
KhatRao(d,m, make.dimnames=TRUE)
dimnames(d) <- list(paste0("d", 10*1:4), paste0("D", 1:4))
KhatRao(m,d, make.dimnames=TRUE)
KhatRao(d,m, make.dimnames=TRUE)

nm <- as(m,"nMatrix")
nm <- as(m,"nMatrix")
KhatRao(nm,nm, make.dimnames=TRUE)
KhatRao(nd,mm, make.dimnames=TRUE)

stopifnot(dim(KhatRao(m,d)) == c(nrow(m)*nrow(d), ncol(d)))
```

---

**Description**

A model matrix \( m \) and corresponding response vector \( y \) used in an example by Koenker and Ng. The matrix \( m \) is a sparse matrix with 1850 rows and 712 columns but only 8758 non-zero entries. It is a "dgCMatrix" object. The vector \( y \) is just numeric of length 1850.

**Usage**

```r
data(KNex)
```

**References**

Examples

data(KNex)
class(KNex$mm)
dim(KNex$mm)
image(KNex$mm)
str(KNex)

system.time(  # a fraction of a second
  sparse.sol <- with(KNex, solve(crossprod(mm), crossprod(mm, y)))))

head(round(sparse.sol,3))

## Compare with QR-based solution ("more accurate, but slightly slower"): system.time(  
  sp.sol2 <- with(KNex, qr.coef(qr(mm), y) ))

all.equal(sparse.sol, sp.sol2, tolerance = 1e-13) # TRUE

---

kronecker-methods  
Methods for Function 'kronecker()' in Package 'Matrix'

Description

Computes Kronecker products for objects inheriting from "Matrix".

In order to preserve sparseness, we treat \( \emptyset \) as \( \emptyset \), not as NA as usually in R (and as used for the base function kronecker).

Methods

kronecker signature(X = "Matrix", Y = "ANY") .......
kroncker signature(X = "ANY", Y = "Matrix") .......
kroncker signature(X = "diagonalMatrix", Y = "ANY") .......
kroncker signature(X = "sparseMatrix", Y = "ANY") .......
kroncker signature(X = "TsparseMatrix", Y = "TsparseMatrix") .......
kroncker signature(X = "dgTMatrix", Y = "dgTMatrix") .......
kroncker signature(X = "dtTMatrix", Y = "dtTMatrix") .......
kroncker signature(X = "indMatrix", Y = "indMatrix") .......

Examples

(t1 <- spMatrix(5, 4, x=c(3,2,-7,11), i=1:4, j=4:1)) # 5 x 4
(t2 <- kroncker(Diagonal(3, 2:4), t1)) # 15 x 12

## should also work with special-cased logical matrices
l3 <- upper.tri(matrix(3,3))
M <- Matrix(l3)
ldenseMatrix-class

Virtual Class "ldenseMatrix" of Dense Logical Matrices

Description

ldenseMatrix is the virtual class of all dense logical (S4) matrices. It extends both denseMatrix and lMatrix directly.

Slots

x: logical vector containing the entries of the matrix.
Dim, Dimnames: see Matrix.

Extends

Class "lMatrix", directly. Class "denseMatrix", directly. Class "Matrix", by class "lMatrix".
Class "Matrix", by class "denseMatrix".

Methods

coerce signature(from = "matrix", to = "ldenseMatrix"): ...

coerce signature(from = "ldenseMatrix", to = "matrix"): ...

as.vector signature(x = "ldenseMatrix", mode = "missing"): ...

which signature(x = "ndenseMatrix"), semantically equivalent to base function which(x, arr.ind); for details, see the lMatrix class documentation.

See Also

Class lgeMatrix and the other subclasses.

Examples

showClass("ldenseMatrix")

as(diag(3) > 0, "ldenseMatrix")

(N <- as(M, "n sparseMatrix"))
N2 <- as(N, "generalMatrix")
MM <- kronecker(M, M)
NN <- kronecker(N, N)
NN2 <- kronecker(N2, N2)
stopifnot(identical(NN, MM),
is(NN, "triangularMatrix"))
Class "ldiMatrix" of Diagonal Logical Matrices

Description

The class "ldiMatrix" of logical diagonal matrices.

Objects from the Class

Objects can be created by calls of the form new("ldiMatrix", ...) but typically rather via Diagonal.

Slots

x: "logical" vector.
diag: "character" string, either "U" or "N", see ddiMatrix.
Dim,Dimnames: matrix dimension and dimnames, see the Matrix class description.

Extends

Class "diagonalMatrix" and class "lMatrix", directly.
Class "sparseMatrix", by class "diagonalMatrix".

See Also

Classes ddiMatrix and diagonalMatrix; function Diagonal.

Examples

(1M <- Diagonal(x = c(TRUE, FALSE, FALSE)))
str(1M)#> gory details (slots)
crossprod(1M) # numeric
(nM <- as(1M, "nMatrix"))# -> sparse (not formally "diagonal")
crossprod(nM) # logical sparse
Class "lgematrix" of General Dense Logical Matrices

Description
This is the class of general dense logical matrices.

Slots
- x: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.
- Dim, Dimnames: The dimension (a length-2 "integer") and corresponding names (or NULL), see the Matrix class.
- factors: Object of class "list". A named list of factorizations that have been computed for the matrix.

Extends
Class "ldenseMatrix", directly. Class "lMatrix", by class "ldenseMatrix". Class "denseMatrix", by class "ldenseMatrix". Class "Matrix", by class "ldenseMatrix". Class "Matrix", by class "ldenseMatrix".

Methods
Currently, mainly t() and coercion methods (for as(.)); use, e.g., showMethods(class="lgematrix") for details.

See Also
Non-general logical dense matrix classes such as ltrMatrix, or lsyMatrix; sparse logical classes such as lgCMatrix.

Examples
showClass("lgematrix")
str(new("lgematrix"))
set.seed(1)
(lM <- Matrix(matrix(rnorm(28), 4,7) > 0)) # a simple random lgematrix
set.seed(11)
(lC <- Matrix(matrix(rnorm(28), 4,7) > 0)) # a simple random lgCMatrix
as(lM, "lgCMatrix")
**lsparseMatrix-classes**  
*Sparse logical matrices*

**Description**

The `lsparseMatrix` class is a virtual class of sparse matrices with `TRUE/FALSE` entries. Only the positions of the elements that are `TRUE` are stored. These can be stored in the “triplet” form (classes `lgTMatrix`, `lsTMatrix`, and `ltTMatrix` which really contain pairs, not triplets) or in compressed column-oriented form (classes `lgCMatrix`, `lsCMatrix`, and `ltCMatrix`) or in compressed row-oriented form (classes `lgRMatrix`, `lsRMatrix`, and `ltRMatrix`). The second letter in the name of these non-virtual classes indicates general, symmetric, or triangular.

**Objects from the Class**

Objects can be created by calls of the form `new("lgMatrix", ...)` and so on. More frequently objects are created by coercion of a numeric sparse matrix to the logical form for use in the symbolic analysis phase of an algorithm involving sparse matrices. Such algorithms often involve two phases: a symbolic phase wherein the positions of the non-zeros in the result are determined and a numeric phase wherein the actual results are calculated. During the symbolic phase only the positions of the non-zero elements in any operands are of interest, hence any numeric sparse matrices can be treated as logical sparse matrices.

**Slots**

- **x**: Object of class "logical", i.e., either `TRUE`, `NA`, or `FALSE`.
- **uplo**: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular. Present in the triangular and symmetric classes but not in the general class.
- **diag**: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N" for non-unit. The implicit diagonal elements are not explicitly stored when `diag` is "U". Present in the triangular classes only.
- **p**: Object of class "integer" of pointers, one for each column (row), to the initial (zero-based) index of elements in the column. Present in compressed column-oriented and compressed row-oriented forms only.
- **i**: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the row numbers for each `TRUE` element in the matrix. All other elements are `FALSE`. Present in triplet and compressed column-oriented forms only.
- **j**: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the column numbers for each `TRUE` element in the matrix. All other elements are `FALSE`. Present in triplet and compressed column-oriented forms only.
- **Dim**: Object of class "integer" - the dimensions of the matrix.

**Methods**

- `coerce` signature(from = "dgCMatrix", to = "lgCMatrix")
- `t` signature(x = "lgCMatrix"): returns the transpose of x
which signature(x = "lsparseMatrix"), semantically equivalent to \texttt{base} function which(x, arr.ind): for details, see the \texttt{lMatrix} class documentation.

See Also

the class \texttt{dgCMatrix}

Examples

\begin{verbatim}
(m <- Matrix(c(0,0,2:0), 3,5, dimnames=list(LETTERS[1:3],NULL)))
(lm <- (m > 1)) # lgc!
!lm # no longer sparse
stopifnot(is(lm,"lsparseMatrix"),
          identical(!lm, m <= 1))
\end{verbatim}

data(KNex)
str(mmG.1 <- (KNex $ mm) > 0.1)# "lgc..."
table(mmG.1@x)# however with many `\textquotesingle\textquotesingle;non-structural zeros\textquotesingle\textquotesingle;''
## from logical to nz_pattern --- okay when there are no NA's:
mmG.1 <- as(mmG.1, "nMatrix") # <<< has "TRUE" also where mmG.1 had FALSE
## from logical to "double"
dmG.1 <- as(mmG.1, "dMatrix") # has '0' and back:
lmG.1 <- as(dmG.1, "lMatrix") # has no extra FALSE, i.e. drop0() included
stopifnot(identical(mmG.1, as((KNex $ mm) != 0,"nMatrix")),
          validObject(lmG.1), all(lmG.1@x),
          # same "logical" but lmG.1 has no 'FALSE' in x slot:
          all(lmG.1 == mmG.1))

class(xnx <- crossprod(mmG.1))# "nsC.."
class(xlx <- crossprod(mmG.1))# "dsC.." : numeric
is0 <- (xlx == 0)
mean(as.vector(is0))# 99.3% zeros: quite sparse, but
table(xlx@x == 0)# more than half of the entries are (non-structural!) 0
stopifnot(isSymmetric(xlx), isSymmetric(xnx),
          # compare xnx and xlx : have the *same* non-structural 0s :
          sapply(slotNames(xnx),
                 function(n) identical(slot(xnx, n), slot(xlx, n))))
\end{verbatim}

\section*{Description}

The "lsyMatrix" class is the class of symmetric, dense logical matrices in non-packed storage and "lsPMatrix" is the class of of these in packed storage. In the packed form, only the upper triangle or the lower triangle is stored.

\section*{Objects from the Class}

Objects can be created by calls of the form \texttt{new("lsyMatrix", ...).}
Slots

uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.

x: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.

Dim,Dimnames: The dimension (a length-2 "integer") and corresponding names (or NULL), see the Matrix class.

factors: Object of class "list". A named list of factorizations that have been computed for the matrix.

Extends

Both extend classes "ldenseMatrix" and "symmetricMatrix", directly; further, class "Matrix" and others, indirectly. Use showClass("lsyMatrix"), e.g., for details.

Methods

Currently, mainly t() and coercion methods (for as(); use, e.g., showMethods(class="dsyMatrix")) for details.

See Also

lgeMatrix, Matrix, t

Examples

(M2 <- Matrix(c(TRUE, NA, FALSE, FALSE), 2, 2)) # logical dense (ltr)
str(M2)
# can
(sM <- M2 | t(M2)) # "lge"
as(sM, "lsyMatrix")
str(sM <- as(sM, "lspMatrix")) # packed symmetric

ltrMatrix-class Triangular Dense Logical Matrices

Description

The "ltrMatrix" class is the class of triangular, dense, logical matrices in nonpacked storage. The "ltpMatrix" class is the same except in packed storage.
Slots

  x: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.

  uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.

  diag: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see triangularMatrix.

  Dim,Dimnames: The dimension (a length-2 "integer") and corresponding names (or NULL), see the Matrix class.

  factors: Object of class "list". A named list of factorizations that have been computed for the matrix.

Extends

  Both extend classes "ldenseMatrix" and "triangularMatrix", directly; further, class "Matrix", "lMatrix" and others, indirectly. Use showClass("ltrMatrix"), e.g., for details.

Methods

  Currently, mainly t() and coercion methods (for as(.); use, e.g., showMethods(class="ltpMatrix") for details.

See Also

  Classes lgeMatrix, Matrix; function t

Examples

  showClass("ltrMatrix")

  str(new("ltpMatrix"))
  (lutr <- as(upper.tri(matrix(,4,4)), "ltrMatrix"))
  str(lutp <- as(lutr, "ltpMatrix")) # packed matrix: only 10 = (4+1)*4/2 entries
  !lutp ## the logical negation (is *not* logical triangular !)
  ## but this one is:
  stopifnot(all.equal(lutp, as(!lutp, "ltpMatrix")))

---

lu (Generalized) Triangular Decomposition of a Matrix

Description

  Computes (generalized) triangular decompositions of square (sparse or dense) and non-square dense matrices.
Usage

```r
lu(x, ...) 
## S4 method for signature 'matrix'
lu(x, warnSing = TRUE, ...) 
## S4 method for signature 'dgeMatrix'
lu(x, warnSing = TRUE, ...) 
## S4 method for signature 'dgCMatrix'
lu(x, errSing = TRUE, order = TRUE, tol = 1, ...) 
```

Arguments

- **x**: A dense or sparse matrix, in the latter case of square dimension. No missing values or IEEE special values are allowed.
- **warnSing**: (when `x` is a "denseMatrix") logical specifying if a warning should be signalled when `x` is singular.
- **errSing**: (when `x` is a "sparseMatrix") logical specifying if an error (see `stop`) should be signalled when `x` is singular. When `x` is singular, `lu(x, errSing=FALSE)` returns `NA` instead of an LU decomposition. No warning is signalled and the user should be careful in that case.
- **order**: Logical or integer, used to choose which fill-reducing permutation technique will be used internally. Do not change unless you know what you are doing.
- **tol**: Positive number indicating the pivoting tolerance used in `cs_lu`. Do only change with much care.
- **...**: Further arguments passed to or from other methods.

Details

`lu()` is a generic function with special methods for different types of matrices. Use `showMethods(\"lu\")` to list all the methods for the `lu` generic.

The method for class `dgeMatrix` (and all dense matrices) is based on LAPACK's "dgetrf" subroutine. It returns a decomposition also for singular and non-square matrices.

The method for class `dgCMatrix` (and all sparse matrices) is based on functions from the CSparse library. It signals an error (or returns `NA`, when `errSing = FALSE`, see above) when the decomposition algorithm fails, as when `x` is (too close to) singular.

Value

An object of class "LU", i.e., "denseLU" (see its separate help page), or "sparseLU", see `sparseLU`; this is a representation of a triangular decomposition of `x`.

References


Tim Davis (2005) [http://www.cise.ufl.edu/research/sparse/CSparse/](http://www.cise.ufl.edu/research/sparse/CSparse/)

LU-class

See Also

Class definitions LU and sparseLU and function expand, qr, chol.

Examples

```r
##--- Dense -------------------------------
x <- Matrix(rnorm(9), 3, 3)
lu(x)
dim(x2 <- round(10 * x[,] - 3)) # non-square
expand(lu2 <- lu(x2))

##--- Sparse (see more in ?"sparseLU-class")----- % ./sparseLU-class.Rd
pm <- as(readMM(system.file("external/pores_1.mtx", package = "Matrix")),
        "CsparseMatrix")
str(pmLU <- lu(pm)) # p is a 0-based permutation of the rows
                    # q is a 0-based permutation of the columns
## permute rows and columns of original matrix
ppm <- pm[pmLU@p + 1L, pmLU@q + 1L]
plU <- drop0(pmLU@U %*% pmLU@U) # *% U -- dropping extra zeros
## equal up to "rounding"
ppm[1:4, 1:5]
plU[1:4, 1:5]
```

LU-class

LU (dense) Matrix Decompositions

Description

The "LU" class is the virtual class of LU decompositions of real matrices. "denseLU" the class of LU decompositions of dense real matrices.

Details

The decomposition is of the form

\[ A = PLU \]

where typically all matrices are of size \( n \times n \), and the matrix \( P \) is a permutation matrix, \( L \) is lower triangular and \( U \) is upper triangular (both of class dtrMatrix).

Note that the dense decomposition is also implemented for a \( m \times n \) matrix \( A \), when \( m \neq n \).

If \( m < n \) ("wide case"), \( U \) is \( m \times n \), and hence not triangular.
If \( m > n \) ("long case"), \( L \) is \( m \times n \), and hence not triangular.

Objects from the Class

Objects can be created by calls of the form `new("denseLU", ...)`. More commonly the objects are created explicitly from calls of the form `lu(mm)` where `mm` is an object that inherits from the "dgeMatrix" class or as a side-effect of other functions applied to "dgeMatrix" objects.
Matrix

Extends

"LU" directly extends the virtual class "MatrixFactorization".
"denseLU" directly extends "LU".

Slots

x: object of class "numeric". The "L" (unit lower triangular) and "U" (upper triangular) factors of the original matrix. These are stored in a packed format described in the Lapack manual, and can retrieved by the expand() method, see below.

perm: Object of class "integer" - a vector of length min(Dim) that describes the permutation applied to the rows of the original matrix. The contents of this vector are described in the Lapack manual.

Dim: the dimension of the original matrix; inherited from class MatrixFactorization.

Methods

expand signature(x = "denseLU"): Produce the "L" and "U" (and "P") factors as a named list of matrices, see also the example below.

solve signature(a = "denseLU", b = "missing"): Compute the inverse of A, A^{-1}, solve(A) using the LU decomposition, see also solve-methods.

See Also

class sparseLU for LU decompositions of sparse matrices; further, class dgeMatrix and functions lu, expand.

Examples

set.seed(1)
mm <- Matrix(round(rnorm(9),2), nrow = 3)
mm
str(lum <- lu(mm))
elu <- expand(lum)
elu # three components: "L", "U", and "P", the permutation
eelu$L %*% elu$U
(m2 <- with(elu, P %*% L %*% U)) # the same as 'mm'
stopifnot(all.equal(as(mm, "matrix"),
                     as(m2, "matrix")))

---

Matrix

Construct a Classed Matrix

Description

Construct a Matrix of a class that inherits from Matrix.
Matrix

Usage

Matrix(data=NA, nrow=1, ncol=1, byrow=FALSE, dimnames=NULL, sparse = NULL, doDiag = TRUE, forceCheck = FALSE)

Arguments

data an optional numeric data vector or matrix.
nrow when data is not a matrix, the desired number of rows
ncol when data is not a matrix, the desired number of columns
byrow logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.
dimnames a dimnames attribute for the matrix: a list of two character components. They are set if not NULL (as per default).
sparse logical or NULL, specifying if the result should be sparse or not. By default, it is made sparse when more than half of the entries are 0. Note that when the resulting matrix is diagonal (“mathematically”), sparse=FALSE results in a diagonalMatrix, unless doDiag=FALSE as well, see the first examples.
doDiag only when sparse = FALSE, logical indicating if a diagonalMatrix object should be considered (default). Otherwise, in such a case, a dense (symmetric) matrix will be returned.
forceCheck logical indicating if the checks for structure should even happen when data is already a "Matrix" object.

Details

If either of nrow or ncol is not given, an attempt is made to infer it from the length of data and the other parameter. Further, Matrix() makes efforts to keep logical matrices logical, i.e., inheriting from class lMatrix, and to determine specially structured matrices such as symmetric, triangular or diagonal ones. Note that a symmetric matrix also needs symmetric dimnames, e.g., by specifying dimnames = list(NULL, NULL), see the examples.

Most of the time, the function works via a traditional (full) matrix. However, Matrix(0, nrow,ncol) directly constructs an "empty" sparseMatrix, as does Matrix(FALSE, *).

Although it is sometime possible to mix unclassed matrices (created with matrix) with ones of class "Matrix", it is much safer to always use carefully constructed ones of class "Matrix".

Value

Returns matrix of a class that inherits from "Matrix". Only if data is not a matrix and does not already inherit from class Matrix are the arguments nrow, ncol and byrow made use of.

See Also

The classes Matrix, symmetricMatrix, triangularMatrix, and diagonalMatrix; further, matrix. Special matrices can be constructed, e.g., via sparseMatrix (sparse), bdiag (block-diagonal), bandSparse (banded sparse), or Diagonal.
Examples

Matrix(0, 3, 2)           # 3 by 2 matrix of zeros -> sparse
Matrix(0, 3, 2, sparse=FALSE) # -> 'dense'
Matrix(0, 2, 2, sparse=FALSE) # diagonal !
Matrix(0, 2, 2, sparse=FALSE, doDiag=FALSE) # -> dense
Matrix(1:6, 3, 2)          # a 3 by 2 matrix (+ integer warning)
Matrix(1:6 + 1, nrow=3)

## logical ones:
Matrix(diag(4) > 0)# -> "ldiMatrix" with diag = "U"
Matrix(diag(4) > 0, sparse=TRUE)# -> sparse...
Matrix(diag(4) >= 0)# -> "lsyMatrix" (of all 'TRUE')
## triangular
l3 <- upper.tri(matrix(c(3,3)))
(M <- Matrix(l3))# -> "ltCMatrix"
Matrix(l3)# -> "ltrMatrix"
as(l3, "CsparseMatrix")

Matrix(1:9, nrow=3,    
dimnames = list(c("a", "b", "c"), c("A", "B", "C")))
(l3 <- Matrix(diag(3)))# identity, i.e., unit "diagonalMatrix"
str(l3) # note the empty 'x' slot

(A <- cbind(a=c(2,1), b=1:2))# symmetric *apart* from dimnames
Matrix(A)                 # hence 'dgeMatrix'
(As <- Matrix(A, dimnames = list(NULL,NULL)))# -> symmetric
stopifnot(is(As, "symmetricMatrix"),
          is(Matrix(0, 3, 3), "sparseMatrix"),
          is(Matrix(FALSE, 1,1), "sparseMatrix"))

---

Matrix-class

Virtual Class "Matrix" Class of Matrices

Description

The Matrix class is a class contained by all actual classes in the Matrix package. It is a “virtual” class.

Slots

Common to all matrix objects in the package:

Dim: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.

Dimnames: list of length two; each component containing NULL or a character vector length equal the corresponding Dim element.
Methods

determinant signature(x = "Matrix", logarithm = "missing"): and
determinant signature(x = "Matrix", logarithm = "logical"): compute the (log) determinant of x. The method chosen depends on the actual Matrix class of x. Note that det also works for all our matrices, calling the appropriate determinant() method. The Matrix::det is an exact copy of base::det, but in the correct namespace, and hence calling the S4-aware version of determinant().

diff signature(x = "Matrix"): As diff() for traditional matrices, i.e., applying diff() to each column.

dim signature(x = "Matrix"): extract matrix dimensions dim.

dim<- signature(x = "Matrix", value = "ANY"): where value is integer of length 2. Allows to reshape Matrix objects, but only when prod(value) == prod(dim(x)).

dimnames signature(x = "Matrix"): extract dimnames.

dimnames<- signature(x = "Matrix", value = "list"): set the dimnames to a list of length 2, see dimnames<-. 

length signature(x = "Matrix"): simply defined as prod(dim(x)) (and hence of mode "double").

show signature(object = "Matrix"): show method for printing.

image signature(object = "Matrix"): draws an image of the matrix entries, using levelplot() from package lattice.

head signature(object = "Matrix"): return only the “head”, i.e., the first few rows.

tail signature(object = "Matrix"): return only the “tail”, i.e., the last few rows of the respective matrix.

as.matrix, as.array signature(x = "Matrix"): the same as as(x, "matrix"); see also the note below.

as.vector signature(x = "Matrix", mode = "missing"): as.vector(m) should be identical to as.vector(as(m,"matrix")), implemented more efficiently for some subclasses.

as(x, "vector"), as(x, "numeric") etc, similarly.

There are many more methods that (conceptually should) work for all "Matrix" objects, e.g., colSums, rowMeans. Even base functions may work automagically (if they first call as.matrix() on their principal argument), e.g., apply, eigen, svd or kappa all do work via coercion to a “traditional” (dense) matrix.

Note

Loading the Matrix namespace “overloads” as.matrix and as.array in the base namespace by the equivalent of function(x) as(x, "matrix"). Consequently, as.matrix(m) or as.array(m) will properly work when m inherits from the "Matrix" class — also for functions in package base and other packages. E.g., apply or outer can therefore be applied to "Matrix" matrices.

Author(s)

Douglas Bates <bates@stat.wisc.edu> and Martin Maechler
See Also
the classes dgeMatrix, dgCMatrix, and function Matrix for construction (and examples).
Methods, e.g., for kronecker.

Examples

```r
slotNames("Matrix")

cl <- getClass("Matrix")
names(cl@subclasses) # more than 40 ..

showClass("Matrix")#> output with slots and all subclasses

(M <- Matrix(c(0,1,0,0), 6, 4))
  dim(M)
  diag(M)
  cm <- M[1:4,] + 10*Diagonal(4)
  diff(M)
  ## can reshape it even :
  dim(M) <- c(2, 12)
  M
  stopifnot(identical(M, Matrix(c(0,1,0,0), 2,12)),
    all.equal(det(cm),
      determinant(as(cm,"matrix"), log=FALSE)$modulus,
      check.attributes=FALSE))
```

Description

The basic matrix product, %*% is implemented for all our Matrix and also for sparseVector
classes, fully analogously to R’s base matrix and vector objects.
The functions crossprod and tcrossprod are matrix products or “cross products”, ideally imple-
mented efficiently without computing t(.)’s unnecessarily. They also return symmetricMatrix
classed matrices when easily detectable, e.g., in crossprod(M), the one argument case.
tcrossprod() takes the cross-product of the transpose of a matrix. tcrossprod(x) is formally
equivalent to, but faster than, the call x %*% t(x), and so is tcrossprod(x, y) instead of
x %*% t(y).

Usage

```r
## S4 method for signature 'CsparseMatrix,diagonalMatrix'
x %*% y
  ## .... and many more

## S4 method for signature 'dgeMatrix,missing'
```
crossprod(x, y = NULL)
  ## .... and many more

## S4 method for signature 'CsparseMatrix,ddenseMatrix'
tcrossprod(x, y = NULL)
  ## .... and many more

Arguments

x    a matrix-like object
y    a matrix-like object, or for [tc]crossprod() NULL (by default); the latter case is formally equivalent to y = x.

Details

For some classes in the Matrix package, such as `dgCMatrix`, it is much faster to calculate the cross-product of the transpose directly instead of calculating the transpose first and then its cross-product.

Value

A Matrix object, in the one argument case of an appropriate symmetric matrix class.

Methods

%*% signature(x = "dgeMatrix", y = "dgeMatrix"): Matrix multiplication; ditto for several other signature combinations, see showMethods("%*%", class = "dgeMatrix").

%*% signature(x = "dtrMatrix", y = "matrix") and other signatures (use showMethods("%*%", class="dtrMatrix")) matrix multiplication. Multiplication of (matching) triangular matrices now should remain triangular (in the sense of class `triangularMatrix`).

crossprod signature(x = "dgeMatrix", y = "dgeMatrix"): ditto for several other signatures, use showMethods("crossprod", class = "dgeMatrix"), matrix crossproduct, an efficient version of t(x) %*% y.

crossprod signature(x = "CsparseMatrix", y = "missing") returns t(x) %*% x as an dsCMatrix object.

crossprod signature(x = "TsparseMatrix", y = "missing") returns t(x) %*% x as an dsCMatrix object.

crossprod,tcrossprod signature(x = "dtrMatrix", y = "matrix") and other signatures, see "%*%" above.

See Also

tcrossprod in R’s base, crossprod and %*%. 

Examples

```r
## A random sparse "incidence" matrix:
m <- matrix(0, 400, 500)
set.seed(12)
m[runif(314, 0, length(m))] <- 1
mm <- as(m, "dgCMatrix")
object.size(m) / object.size(mm) # smaller by a factor of > 200

## tcrossprod() is very fast:
system.time(tCmm <- tcrossprod(mm)) # 0 (PIII, 933 MHz)
system.time(cm <- crossprod(t(m))) # 0.16
system.time(cm. <- tcrossprod(m)) # 0.02

stopifnot(cm == as(tCmm, "matrix"))

## show sparse sub matrix
tCmm[1:16, 1:30]
```

---

MatrixFactorization-class

Class "MatrixFactorization" of Matrix Factorizations

Description

The class "MatrixFactorization" is the virtual (super) class of (potentially) all matrix factorizations of matrices from package Matrix.

The class "CholeskyFactorization" is the virtual class of all Cholesky decompositions from Matrix (and trivial sub class of "MatrixFactorization").

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

- **Dim**: Object of class "integer" - the dimensions of the original matrix - must be an integer vector with exactly two non-negative values.

Methods

- **dim** (x) simply returns x@Dim, see above.
- **expand** signature(x = "MatrixFactorization"): this has not been implemented yet for all matrix factorizations. It should return a list whose components are matrices which when multiplied return the original Matrix object.
- **show** signature(object = "MatrixFactorization"): simple printing, see show.
- **solve** signature(a = "MatrixFactorization", b = .): solve $Ax = b$ for $x$; see solve-methods.
See Also

classes inheriting from "MatrixFactorization", such as LU, Cholesky, CHMfactor, and sparseQR.

Examples

showClass("MatrixFactorization")
getClass("CholeskyFactorization")

ndenseMatrix-class  Virtual Class "ndenseMatrix" of Dense Logical Matrices

Description

ndenseMatrix is the virtual class of all dense logical (S4) matrices. It extends both denseMatrix and lMatrix directly.

Slots

x: logical vector containing the entries of the matrix.
Dim, Dimnames: see Matrix.

Extends

Class "nMatrix", directly. Class "denseMatrix", directly. Class "Matrix", by class "nMatrix".
Class "Matrix", by class "denseMatrix".

Methods

%*% signature(x = "nSparseMatrix", y = "ndenseMatrix"):
%*% signature(x = "ndenseMatrix", y = "nSparseMatrix"):
coerce signature(from = "matrix", to = "ndenseMatrix"):
coerce signature(from = "ndenseMatrix", to = "matrix"):
crossprod signature(x = "nSparseMatrix", y = "ndenseMatrix"):
crossprod signature(x = "ndenseMatrix", y = "nSparseMatrix"):
as.vector signature(x = "ndenseMatrix", mode = "missing"):
diag signature(x = "ndenseMatrix"): extracts the diagonal as for all matrices, see the generic
diag().

which signature(x = "ndenseMatrix"), semantically equivalent to base function which(x, arr.ind);
for details, see the lMatrix class documentation.

See Also

Class ngeMatrix and the other subclasses.
nearPD

Nearest Positive Definite Matrix

Description

Compute the nearest positive definite matrix to an approximate one, typically a correlation or variance-covariance matrix.

Usage

nearPD(x, corr = FALSE, keepDiag = FALSE, do2eigen = TRUE, doSym = FALSE, doDykstra = TRUE, only.values = FALSE, ensureSymmetry = !isSymmetric(x), eig.tol = 1e-06, conv.tol = 1e-07, posd.tol = 1e-08, maxit = 100, conv.norm.type = "I", trace = FALSE)

Arguments

x
 numeric n × n approximately positive definite matrix, typically an approximation to a correlation or covariance matrix. If x is not symmetric (and ensureSymmetry is not false), symmpart(x) is used.

corr
 logical indicating if the matrix should be a correlation matrix.

keepDiag
 logical, generalizing corr: if TRUE, the resulting matrix should have the same diagonal (diag(x)) as the input matrix.

do2eigen
 logical indicating if a posdefify() eigen step should be applied to the result of the Higham algorithm.

doSym
 logical indicating if X ← (X + t(X))/2 should be done, after X ← tcrossprod(Qd, Q); some doubt if this is necessary.

doDykstra
 logical indicating if Dykstra’s correction should be used; true by default. If false, the algorithm is basically the direct fixpoint iteration \( Y_k = P_U(P_S(Y_{k-1})) \).

only.values
 logical; if TRUE, the result is just the vector of eigen values of the approximating matrix.

ensureSymmetry
 logical; by default, symmpart(x) is used whenever isSymmetric(x) is not true. The user can explicitly set this to TRUE or FALSE, saving the symmetry test. Beware however that setting it FALSE for an asymmetric input x, is typically nonsense!

eig.tol
 defines relative positiveness of eigenvalues compared to largest one, \( \lambda_1 \). Eigen values \( \lambda_k \) are treated as if zero when \( \lambda_k / \lambda_1 \leq eig.tol \).

conv.tol
 convergence tolerance for Higham algorithm.

Examples

showClass("ndenseMatrix")
as(diag(3) > 0, "ndenseMatrix")# -> "nge"
posd.tol  tolerance for enforcing positive definiteness (in the final posdefify step when do2eigen is TRUE).
maxit  maximum number of iterations allowed.
conv.norm.type  convergence norm type (norm(*, type)) used for Higham algorithm. The default is "I" (infinity), for reasons of speed (and back compatibility); using "F" is more in line with Higham's proposal.
trace  logical or integer specifying if convergence monitoring should be traced.

Details
This implements the algorithm of Higham (2002), and then (if do2eigen is true) forces positive definiteness using code from posdefify. The algorithm of Knol DL and ten Berge (1989) (not implemented here) is more general in (1) that it allows constraints to fix some rows (and columns) of the matrix and (2) to force the smallest eigenvalue to have a certain value.

Note that setting corr = TRUE just sets diag(.) <- 1 within the algorithm.
Higham (2002) uses Dykstra's correction, but the version by Jens Oehlschlaegel did not use it (accidentally), and has still lead to good results; this simplification, now only via doDykstra = FALSE, was active in nearPD() upto Matrix version 0.999375-40.

Value
If only.values = TRUE, a numeric vector of eigen values of the approximating matrix; Otherwise, as by default, an S3 object of class "nearPD", basically a list with components

mat  a matrix of class dpoMatrix, the computed positive-definite matrix.
eigenvalues  numeric vector of eigen values of mat.
corr  logical, just the argument corr.
normF  the Frobenius norm (norm(x-X, "F")) of the difference between the original and the resulting matrix.
iterations  number of iterations needed.
converged  logical indicating if iterations converged.

Author(s)
Jens Oehlschlaegel donated a first version. Subsequent changes by the Matrix package authors.

References
See Also

A first version of this (with non-optional corr=TRUE) has been available as `nearcor()`, and more simple versions with a similar purpose `posdefify()`, both from package `sfsmisc`.

Examples

```r
## Higham (2002), p. 334f - simple example
A <- matrix(1, 3, 3); A[1,3] <- A[3,1] <- 0
n.A <- nearPD(A, corr=TRUE, do2eigen=FALSE)
str(A[c("mat", "normF")])
stopifnot(all.equal(n.A$mat[1,2], 0.760689917),
  all.equal(n.A$normF, 0.52779033, tolerance=1e-9))

set.seed(27)
m <- matrix(round(rnorm(25),2), 5, 5)
diag(m) <- pmax(0, diag(m)) + 1
(m <- round(cov2cor(m), 2))
str(near.m <- nearPD(m, trace = TRUE))
round(near.m$mat, 2)
norm(m - near.m$mat) # 1.102 / 1.08

if(require("sfsmisc")){
m2 <- posdefify(m) # a simpler approach
  norm(m - m2) # 1.185, i.e., slightly "less near"
}round(nearPD(m, only.values=TRUE), 9)

## A longer example, extended from Jens' original,
## showing the effects of some of the options:
pr <- Matrix(c(1, 0.477, 0.644, 0.478, 0.651, 0.826,
               0.477, 1, 0.516, 0.233, 0.682, 0.75,
               0.644, 0.516, 1, 0.599, 0.581, 0.742,
               0.478, 0.233, 0.599, 1, 0.741, 0.8,
               0.651, 0.682, 0.581, 0.741, 1, 0.798,
               0.826, 0.75, 0.742, 0.8, 0.798, 1),
nrow = 6, ncol = 6)
nC <- nearPD(pr, conv.tol = 1e-7) # default
nc.$iterations # 2
nc.1 <- nearPD(pr, conv.tol = 1e-7, corr = TRUE)
nc.1$iterations # 11 / 12 (!)
ncr <- nearPD(pr, conv.tol = 1e-15)
str(ncr)# still 2 iterations
ncr.1 <- nearPD(pr, conv.tol = 1e-15, corr = TRUE)
ncr.1$iterations # 27 / 30!
ncF <- nearPD(pr, conv.tol = 1e-15, conv.norm = "F")
stopifnot(all.equal(ncr, ncF))# norm type does not matter at all in this example
```
## Description

This is the class of general dense nonzero-pattern matrices, see `nMatrix`.

### Slots

- `x`: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.

- `dim`, `dimnames`: The dimension (a length-2 "integer") and corresponding names (or NULL), see the `Matrix` class.

- `factors`: Object of class "list". A named list of factorizations that have been computed for the matrix.

### Extends

Class "ndenseMatrix", directly. Class "lMatrix", by class "ndenseMatrix". Class "denseMatrix", by class "ndenseMatrix". Class "Matrix", by class "ndenseMatrix". Class "Matrix", by class "ndenseMatrix".

### Methods

Currently, mainly `t()` and coercion methods (for `as(.)`); use, e.g., `showMethods(class="ngeMatrix")` for details.
See Also

Non-general logical dense matrix classes such as ntrMatrix, or nsyMatrix; sparse logical classes such as ngCMMatrix.

Examples

generic.showClass("ngeMatrix")
## "lgeMatrix" is really more relevant

nMatrix-class

Class "nMatrix" of Non-zero Pattern Matrices

Description

The nMatrix class is the virtual "mother" class of all non-zero pattern (or simply pattern) matrices in the Matrix package.

Slots

Common to all matrix object in the package:

Dim: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.

Dimnames: list of length two; each component containing NULL or a character vector length equal the corresponding Dim element.

Methods

There is a bunch of coercion methods (for as(...), e.g.,

\textbf{coerce} signature(from = "nMatrix", to = "matrix"): ...  
\textbf{coerce} signature(from = "nMatrix", to = "dMatrix"): ...
\textbf{coerce} signature(from = "nMatrix", to = "lMatrix"): ...
\textbf{coerce} signature(from = "matrix", to = "nMatrix"): Note that these coercions (must) coerce NA to non-zero, hence conceptually TRUE. This is particularly important when sparseMatrix objects are coerced to "nMatrix" and hence to \textbf{nsparseMatrix}.
\textbf{coerce} signature(from = "dMatrix", to = "nMatrix"): ...
\textbf{coerce} signature(from = "lMatrix", to = "nMatrix"): ...

. Additional methods contain group methods, such as

\textbf{Ops} signature(e1 = "nMatrix", e2 = ".")....,  
\textbf{Arith} signature(e1 = "nMatrix", e2 = ".")....,  
\textbf{Compare} signature(e1 = "nMatrix", e2 = ".")....,  
\textbf{Logic} signature(e1 = "nMatrix", e2 = ".")....,  
\textbf{Summary} signature(x = "nMatrix", ".").,...
See Also

The classes \texttt{Matrix}, \texttt{nsparseMatrix}, and the mother class, \texttt{Matrix}.

Examples

```r
getClass("nMatrix")
L3 <- Matrix(upper.tri(diag(3)))
L3 # an "ltCMatrix"
as(L3, "nMatrix") # -> ntc*

## similar, not using Matrix()
as(upper.tri(diag(3)), "nMatrix")# currently "ngTMMatrix"
```

---

**nnzero**

The Number of Non-Zero Values of a Matrix

Description

Returns the number of non-zero values of a numeric-like \texttt{R} object, and in particular an object \texttt{x} inheriting from class \texttt{Matrix}.

Usage

```r
nnzero(x, na.counted = NA)
```

Arguments

- \texttt{x} an \texttt{R} object, typically inheriting from class \texttt{Matrix} or \texttt{numeric}.
- \texttt{na.counted} a \texttt{logical} describing how \texttt{NAs} should be counted. There are three possible settings for \texttt{na.counted}:
  - \texttt{TRUE} \texttt{NAs} are counted as non-zero (since “they are not zero”).
  - \texttt{NA} (default)the result will be \texttt{NA} if there are \texttt{NA}'s in \texttt{x} (since “\texttt{NA}'s are not known, i.e., may be zero”).
  - \texttt{FALSE} \texttt{NAs} are omitted from \texttt{x} before the non-zero entries are counted.

For sparse matrices, you may often want to use \texttt{na.counted = TRUE}.

Value

the number of non zero entries in \texttt{x} (typically \texttt{integer}).

Note that for a \texttt{symmetric} sparse matrix \texttt{S} (i.e., inheriting from class \texttt{symmetricMatrix}), \texttt{nnzero(S)} is typically \texttt{twice} the length(\texttt{S@x}).
Methods

signature(x = "ANY") the default method for non-\texttt{Matrix} class objects, simply counts the number 0s in x, counting NA's depending on the na.counded argument, see above.

signature(x = "denseMatrix") conceptually the same as for traditional \texttt{matrix} objects, care has to be taken for "\texttt{symmetricMatrix}" objects.

signature(x = "diagonalMatrix"), \textbf{and} signature(x = "indMatrix") fast simple methods for these special "\texttt{sparseMatrix}" classes.

signature(x = "sparseMatrix") typically, the most interesting method, also carefully taking "\texttt{symmetricMatrix}" objects into account.

See Also

The \texttt{Matrix} class also has a \texttt{length} method; typically, length(M) is much larger than nnzero(M) for a sparse matrix M, and the latter is a better indication of the size of M.

Examples

\begin{verbatim}
m <- Matrix(0+1:28, nrow = 4)
m[-3,c(2,4:5,7)] <- m[ 3, 1:4] <- m[1:3, 6] <- 0
(mT <- as(m, "dgTMatrix"))

nnzero(mT)
(S <- crossprod(mT))

nnzero(S)
str(S) # slots are smaller than nnzero()
stopifnot(nnzero(S) == sum(as.matrix(S) != 0))# failed earlier

data(KNex)
M <- KNex$mm
class(M)
dim(M)

length(M); stopifnot(length(M) == prod(dim(M)))

nnzero(M) # more relevant than length

## the above are also visible from
str(M)
\end{verbatim}

---

**norm**

**Matrix Norms**

Description

Computes a matrix norm of \(x\), using Lapack for dense matrices. The norm can be the one norm, the infinity norm, the Frobenius norm, or the maximum modulus among elements of a matrix, as determined by the value of type.

Usage

\[ \text{norm}(x, \text{type}, \ldots) \]
Arguments

- **x**: a real or complex matrix.
- **type**: A character indicating the type of norm desired.
  - "0", "o" or "1" specifies the one norm (maximum absolute column sum);
  - "I" or "i" specifies the infinity norm (maximum absolute row sum);
  - "F" or "f" specifies the Frobenius norm (the Euclidean norm of \( x \) treated as if it were a vector); and
  - "M" or "m" specifies the maximum modulus of all the elements in \( x \).

The default is "0". Only the first character of `type` is used.

Further arguments passed to or from other methods.

Details

For dense matrices, the methods eventually call the Lapack functions `dlange`, `dlansy`, `dlantr`, `zlange`, `zlansy`, and `zlantr`.

Value

A numeric value of class "norm", representing the quantity chosen according to `type`.

References


See Also

*onenormest()*, an approximate randomized estimate of the 1-norm condition number, efficient for large sparse matrices.

Examples

```r
x <- Hilbert(9)
norm(x, "1")
norm(x, "I")
norm(x, "F")
norm(x, "M")
```

### nsparseMatrix-classes

**Sparse "pattern" Matrices**

Description

The `nsparseMatrix` class is a virtual class of sparse "pattern" matrices, i.e., binary matrices conceptually with TRUE/FALSE entries. Only the positions of the elements that are TRUE are stored. These can be stored in the "triplet" form (classes `ngTMatrix`, `nsTMatrix`, and `ntTMatrix` which really contain pairs, not triplets) or in compressed column-oriented form (classes `ngCMatrix`, `nsCMatrix`, and `ntCMatrix`) or in compressed row-oriented form (classes `ngRMatrix`, `nsRMatrix`, and `ntRMatrix`). The second letter in the name of these non-virtual classes indicates general, symmetric, or triangular.
Objects from the Class

Objects can be created by calls of the form `new("ngCMatrix", ...)` and so on. More frequently, objects are created by coercion of a numeric sparse matrix to the pattern form for use in the symbolic analysis phase of an algorithm involving sparse matrices. Such algorithms often involve two phases: a symbolic phase wherein the positions of the non-zeros in the result are determined and a numeric phase wherein the actual results are calculated. During the symbolic phase only the positions of the non-zero elements in any operands are of interest, hence numeric sparse matrices can be treated as sparse pattern matrices.

Slots

- `uplo`: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular. Present in the triangular and symmetric classes but not in the general class.
- `diag`: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N" for non-unit. The implicit diagonal elements are not explicitly stored when `diag` is "U". Present in the triangular classes only.
- `p`: Object of class "integer" of pointers, one for each column (row), to the initial (zero-based) index of elements in the column. Present in compressed column-oriented and compressed row-oriented forms only.
- `i`: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the row numbers for each TRUE element in the matrix. All other elements are FALSE. Present in triplet and compressed column-oriented forms only.
- `j`: Object of class "integer" of length `nnzero` (number of non-zero elements). These are the column numbers for each TRUE element in the matrix. All other elements are FALSE. Present in triplet and compressed column-oriented forms only.
- `dim`: Object of class "integer" - the dimensions of the matrix.

Methods

- `coerce` signature(`from = "dgCMatrix", to = "ngCMatrix"`), and many similar ones; typically you should coerce to "nSparseMatrix" (or "nMatrix"). Note that coercion to a sparse pattern matrix records all the potential non-zero entries, i.e., explicit ("non-structural") zeroes are coerced to TRUE, not FALSE, see the example.
- `t` signature(`x = "ngCMatrix"`): returns the transpose of `x`
- `which` signature(`x = "lSparseMatrix"`), semantically equivalent to base function `which(x, arr.ind)`; for details, see the `lMatrix` class documentation.

See Also

the class `dgCMatrix`

Examples

```r
(m <- Matrix(c(0,0,2:0), 3,5, dimnames=list(LETTERS[1:3],NULL)))
## `extract the nonzero-pattern of (m) into an nMatrix':
mm <- as(m, "nSparseMatrix") ## -> will be a "ngCMatrix"
str(mm) # no `x' slot
```
nsyMatrix-class

The "nsyMatrix" class is the class of symmetric, dense nonzero-pattern matrices in non-packed storage and "nspMatrix" is the class of of these in packed storage. Only the upper triangle or the lower triangle is stored.

Objects from the Class

Objects can be created by calls of the form new("nsyMatrix", ...).

Slots

uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.

x: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.

Dim,Dimnames: The dimension (a length-2 "integer") and corresponding names (or NULL), see the Matrix class.

factors: Object of class "list". A named list of factorizations that have been computed for the matrix.

Extends

"nsyMatrix" extends class "ngeMatrix", directly, whereas "nspMatrix" extends class "ndenseMatrix", directly.

Both extend class "symmetricMatrix", directly, and class "Matrix" and others, indirectly, use showClass("nsyMatrix"), e.g., for details.
**ntrMatrix-class**

**Methods**

Currently, mainly \texttt{t()} and coercion methods (for \texttt{as()}); use, e.g., \texttt{showMethods(class="dsyMatrix")} for details.

**See Also**

\texttt{ngeMatrix}, \texttt{Matrix.t}

**Examples**

\begin{verbatim}
(s0 <- new("nsyMatrix"))

(M2 <- Matrix(c(TRUE, NA, FALSE, FALSE), 2, 2)) # logical dense (ltr)
(sM <- M2 & t(M2)) # "lge"
class(sM <- as(sM, "nMatrix")) # -> "nge"
(sM <- as(sM, "nsyMatrix")) # -> "nsy"
str (sM <- as(sM, "nspMatrix")) # -> "nsp": packed symmetric
\end{verbatim}

---

**ntrMatrix-class**

*Triangular Dense Logical Matrices*

**Description**

The "ntrMatrix" class is the class of triangular, dense, logical matrices in nonpacked storage. The "ntpMatrix" class is the same except in packed storage.

**Slots**

- **x**: Object of class "logical". The logical values that constitute the matrix, stored in column-major order.
- **uplo**: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- **diag**: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see \texttt{triangularMatrix}.
- **Dim,Dimnames**: The dimension (a length-2 "integer") and corresponding names (or NULL), see the \texttt{Matrix} class.
- **factors**: Object of class "list". A named list of factorizations that have been computed for the matrix.

**Extends**

"ntrMatrix" extends class "ngeMatrix", directly, whereas
"ntpMatrix" extends class "ndenseMatrix", directly.

Both extend Class "triangularMatrix", directly, and class "denseMatrix", "lMatrix" and others, indirectly, use \texttt{showClass("nsyMatrix")}, e.g., for details.
Methods

Currently, mainly `t()` and coercion methods (for `as(.)`; use, e.g., `showMethods(class="nsyMatrix")` for details.

See Also

Classes `ngeMatrix, Matrix`; function `t`

Examples

```
showClass("ntrMatrix")
str(new("ntpMatrix"))
(nutr <- as(upper.tri(matrix(4,4), "ntrMatrix"))
str(nutp <- as(nutr, "ntpMatrix"))# packed matrix: only 10 = (4+1)*4/2 entries
!nutp ## the logical negation (is *not* logical triangular !)
## but this one is:
stopifnot(all.equal(nutp, as(!nutp, "ntpMatrix")))
```

---

**number-class**

Class "number" of Possibly Complex Numbers

**Description**

The class "number" is a virtual class, currently used for vectors of eigen values which can be "numeric" or "complex".

It is a simple class union (`setClassUnion`) of "numeric" and "complex".

**Objects from the Class**

Since it is a virtual Class, no objects may be created from it.

**Examples**

```
showClass("number")
stopifnot( is(1i, "number"), is(pi, "number"), is(1:3, "number") )
```
Description

The "pMatrix" class is the class of permutation matrices, stored as 1-based integer permutation vectors.

Matrix (vector) multiplication with permutation matrices is equivalent to row or column permutation, and is implemented that way in the Matrix package, see the ‘Details’ below.

Details

Matrix multiplication with permutation matrices is equivalent to row or column permutation. Here are the four different cases for an arbitrary matrix \( M \) and a permutation matrix \( P \) (where we assume matching dimensions):

\[
\begin{align*}
MP &= M \circ P = M[, i(p)] \\
PM &= P \circ M = M[, p, ] \\
P'M &= \text{crossprod}(P,M) (= t(P) \circ M) = M[i(p), ] \\
MP' &= \text{tcrossprod}(M,P) (= M \circ t(P)) = M[, p ]
\end{align*}
\]

where \( p \) is the “permutation vector” corresponding to the permutation matrix \( P \) (see first note), and \( i(p) \) is short for \texttt{invPerm}(p).

Also one could argue that these are really only two cases if you take into account that inversion (\texttt{solve}) and transposition (\texttt{t}) are the same for permutation matrices \( P \).

Objects from the Class

Objects can be created by calls of the form \texttt{new("pMatrix",...)} or by coercion from an integer permutation vector, see below.

Slots

- \texttt{perm}: An integer, 1-based permutation vector, i.e. an integer vector of length \( \text{Dim}[1] \) whose elements form a permutation of \( 1: \text{Dim}[1] \).
- \texttt{Dim}: Object of class "integer". The dimensions of the matrix which must be a two-element vector of equal, non-negative integers.
- \texttt{dimnames}: list of length two; each component containing NULL or a character vector length equal the corresponding \texttt{dim} element.

Extends

Class "\texttt{indMatrix}" , directly.
Methods

signature(x = "matrix", y = "pMatrix") and other signatures (use showMethods("%*%", class="pMatrix"):)

...  

**coerce** signature(from = "integer", to = "pMatrix"): This is enables typical "pMatrix" construction, given a permutation vector of 1:n, see the first example.

**coerce** signature(from = "numeric", to = "pMatrix"): a user convenience, to allow as(perm, "pMatrix") for numeric perm with integer values.

**coerce** signature(from = "pMatrix", to = "matrix"): coercion to a traditional FALSE/TRUE matrix of mode logical. (in earlier version of Matrix, it resulted in a 0/1-integer matrix; logical makes slightly more sense, corresponding better to the "natural" sparseMatrix counterpart, "ngTMatrix".)

**coerce** signature(from = "pMatrix", to = "ngTMatrix"): coercion to sparse logical matrix of class ngTMatrix.

determinant signature(x = "pMatrix", logarithm="logical"): Since permutation matrices are orthogonal, the determinant must be +1 or -1. In fact, it is exactly the sign of the permutation.

**solve** signature(a = "pMatrix", b = "missing"): return the inverse permutation matrix; note that solve(P) is identical to t(P) for permutation matrices. See solve-methods for other methods.

**t** signature(x = "pMatrix"): return the transpose of the permutation matrix (which is also the inverse of the permutation matrix).

Note

For every permutation matrix P, there is a corresponding permutation vector p (of indices, 1:n), and these are related by

\[
P \leftarrow \text{as}(p, "pMatrix")
\]

\[
p \leftarrow P@\text{perm}
\]

see also the ‘Examples’.

"Row-indexing" a permutation matrix typically returns an "indMatrix". See "indMatrix" for all other subsetting/indexing and subassignment (A[...] <- v) operations.

See Also

invPerm(p) computes the inverse permutation of an integer (index) vector p.

Examples

(pm1 <- as(as.integer(c(2,3,1)), "pMatrix"))

t(pm1) # is the same as

solve(pm1)

pm1 %*% t(pm1) # check that the transpose is the inverse

stopifnot(all(diag(3) == as(pm1 %*% t(pm1), "matrix")),
          is.logical(as(pm1, "matrix")))
set.seed(11)
## random permutation matrix :
(p10 <- as(sample(10),"pMatrix"))

## Permute rows / columns of a numeric matrix :
(mm <- round(array(rnorm(3 * 3), c(3, 3)), 2))

mm %*% pm1

try(as(as.integer(c(3,3,1)), "pMatrix")) # Error: not a permutation

as(pm1, "ngTMatrix")
p10[1:7, 1:4] # gives an "ngTMatrix" (most economic!)

## row-indexing of a <pMatrix> keeps it as an <indMatrix>:
p10[1:3, ]

printSpMatrix

Format and Print Sparse Matrices Flexibly

Description

Format and print sparse matrices flexibly. These are the “workhorses” used by the format, show and print methods for sparse matrices. If x is large, printSpMatrix2(x) calls printSpMatrix() twice, namely, for the first and the last few rows, suppressing those in between, and also suppresses columns when x is too wide.

printSpMatrix() basically prints the result of formatSpMatrix().

Usage

formatSpMatrix(x, digits = NULL, maxp = 1e9,
cld = getClassDef(class(x)), zero.print = ".",
col.names, note.dropping.colnames = TRUE,
align = c("fancy", "right"))

printSpMatrix(x, digits = NULL, maxp = getOption("max.print"),
cld = getClassDef(class(x)),
zero.print = ".", col.names, note.dropping.colnames = TRUE,
col.trailer = "",
align = c("fancy", "right"))

printSpMatrix2(x, digits = NULL, maxp = getOption("max.print"),
zero.print = ".", col.names, note.dropping.colnames = TRUE,
suppRows = NULL, suppCols = NULL,
col.trailer = if(suppCols) "......" else "",
align = c("fancy", "right"))
Arguments

x an R object inheriting from class `sparseMatrix`.
digits significant digits to use for printing, see `print.default`, the default, `NULL`, corresponds to using `getOption("digits")`.
maxp integer, default from `options(max.print)`, influences how many entries of large matrices are printed at all.
cld the class definition of `x`; must be equivalent to `getClassDef(class(x))` and exists mainly for possible speedup.
zero.print character which should be printed for structural zeroes. The default "." may occasionally be replaced by " " (blank); using "0" would look almost like `print()`ing of non-sparse matrices.
col.names logical or string specifying if and how column names of `x` should be printed, possibly abbreviated. The default is taken from `options("sparse.colnames")` if that is set, otherwise FALSE unless there are less than ten columns. When TRUE the full column names are printed.
When col.names is a string beginning with "abb" or "sub" and ending with an integer n (i.e., of the form "abb... <n>"), the column names are abbreviate()d or substring()ed to (target) length n, see the examples.

note.dropping.colnames logical specifying, when col.names is FALSE if the dropping of the column names should be noted, TRUE by default.
col.trailer a string to be appended to the right of each column; this is typically made use of by `show(<sparseMatrix>)` only, when suppressing columns.
suppRows, suppCols logicals or `NULL`, for `printSpMatrix2()` specifying if rows or columns should be suppressed in printing. If `NULL`, sensible defaults are determined from `dim(x)` and `options(c("width", "max.print"))`. Setting both to `FALSE` may be a very bad idea.

align a string specifying how the zero.print codes should be aligned, i.e., padded as strings. The default, "fancy", takes some effort to align the typical zero.print = "." with the position of 0, i.e., the first decimal (one left of decimal point) of the numbers printed, whereas align = "right" just makes use of `print(*, right = TRUE)`.

Details

`formatSpMatrix`: If `x` is large, only the first rows making up the approximately first `maxp` entries is used, otherwise all of `x`. `formatSparseSimple()` is applied to (a dense version of) the matrix. Then, `formatSparseM` is used, unless in trivial cases or for sparse matrices without `x` slot.

Value

`formatSpMatrix()`
returns a character matrix with possibly empty column names, depending on col.names etc, see above.

`printSpMatrix()`
return `x invisibly`, see `invisible`. 
Author(s)

Martin Maechler

See Also

the virtual class sparseMatrix and the classes extending it; maybe sparseMatrix or spMatrix as simple constructors of such matrices.

The underlying utilities formatSparseM and .formatSparseSimple() (on the same page).

Examples

f1 <- gl(5, 3, labels = LETTERS[1:5])
X <- as(f1, "sparseMatrix")
X ## <== show(X) <== print(X)
t(X) ## shows column names, since only 5 columns

X2 <- as(gl(12, 3, labels = paste(LETTERS[1:12], "c", sep=".")), "sparseMatrix")
X2
## less nice, but possible:
print(X2, col.names = TRUE) # use [,1] [,2] .. => does not fit

t(X2) # suppressing column names
print(t(X2), col.names=TRUE)
print(t(X2), zero.print = "", col.names="abbr. 1")
print(t(X2), zero.print = "-", col.names="substring 2")

Description

The "Matrix" package provides methods for the QR decomposition of special classes of matrices. There is a generic function which uses qr as default, but methods defined in this package can take extra arguments. In particular there is an option for determining a fill-reducing permutation of the columns of a sparse, rectangular matrix.

Usage

qr(x, ...)
qrR(qr, complete=FALSE, backPermute=TRUE)
Arguments

- **x**: a numeric or complex matrix whose QR decomposition is to be computed. Logical matrices are coerced to numeric.
- **qr**: a QR decomposition of the type computed by `qr`.
- **complete**: logical indicating whether the \( R \) matrix is to be completed by binding zero-value rows beneath the square upper triangle.
- **backPermute**: logical indicating if the rows of the \( R \) matrix should be back permuted such that `qr()`’s result can be used directly to reconstruct the original matrix \( X \).

Methods

- **x = "dgCMatrix"**: QR decomposition of a general sparse double-precision matrix with \( \text{nr}(X) \geq \text{nc}(X) \). Returns an object of class "sparseQR".
- **x = "sparseMatrix"**: works via "dgCMatrix".

See Also

`qr`; then, the class documentations, mainly sparseQR, and also dgCMatrix.

Examples

```r
### example of pivoting -- from base' qraux.Rd ---------------
X <- Matrix(cbind(int = 1,
  b1=rep(1:0, each=3), b2=rep(0:1, each=3),
  c1=rep(c(1,0,0), 2), c2=rep(c(0,1,0), 2), c3=rep(c(0,0,1),2)),
  sparse=TRUE)
X # is singular, columns "b2" and "c3" are "extra"
(qx <- qr(X))
# both @p and @q are non-trivial permutations
drop0(R. <- qr.R(qx), tol=1e-15) # columns are int b1 c1 c2 b2 c3
Q. <- qr.Q(qx)
qI <- sort.list(qx@q) # the inverse 'q' permutation
(X. <- drop0(Q. .%*% R.[, qI], tol=1e-15))## just = X
stopifnot(all(X. - X.) < 8*.Machine$double.eps,
  ## qr(.) returns R already "back permuted" (as with qI):
  identical(R.[, qI], qrR(qx)) )
```

**rankMatrix**  
*Rank of a Matrix*

Description

Compute ‘the’ matrix rank, a well-defined functional in theory, somewhat ambiguous in practice. We provide several methods, the default corresponding to Matlab’s definition.
Usage

rankMatrix(x, tol = NULL, 
    method = c("tolNorm2", "qr.R", "qrLINPACK", "qr", 
              "useGrad", "maybeGrad"), 
    sval = svd(x, 0, 0)$d, warn.t = TRUE)

Arguments

x numeric matrix, of dimension \(n \times m\), say.

tol nonnegative number specifying a (relative, "scalefree") tolerance for testing of "practically zero" with specific meaning depending on method; by default, 
max(dim(x)) * .Machine$double.eps is according to Matlab's default (for its only method which is our method = "tolNorm2").

method a character string specifying the computational method for the rank, can be abbreviated:

"tolNorm2": the number of singular values \(\geq tol \times \text{max}(\text{sval})\); 
"qrLINPACK": for a dense matrix, this is the rank of \(\text{qr}(x, \text{tol}, \text{LAPACK}=\text{FALSE})\) 
(which is \(\text{qr}(\ldots)\)$rank); 
This ("qr*", dense) version used to be the recommended way to compute a matrix rank for a while in the past. 
For sparse \(x\), this is equivalent to "qr.R".

"qr.R": this is the rank of triangular matrix \(R\), where \(\text{qr}()\) uses LAPACK or a "sparseQR" method (see qr-methods) to compute the decomposition \(QR\). The rank of \(R\) is then defined as the number of "non-zero" diagonal entries \(d_i\) of \(R\), and "non-zero"'s fulfill \(|d_i| \geq tol \times \text{max}(|d_i|)\).

"qr": is for back compatibility; for dense \(x\), it corresponds to "qrLINPACK", whereas for sparse \(x\), it uses "qr.R". 
For all the "qr*" methods, singular values \(\text{sval}\) are not used, which may be crucially important for a large sparse matrix \(x\), as in that case, when \(\text{sval}\) is not specified, the default, computing \(\text{svd()}\) currently coerces \(x\) to a dense matrix.

"useGrad": considering the "gradient" of the (decreasing) singular values, the index of the smallest gap.

"maybeGrad": choosing method "useGrad" only when that seems reasonable; otherwise using "tolNorm2".

sval numeric vector of non-increasing singular values of \(x\); typically unspecified and computed from \(x\) when needed, i.e., unless method = "qr".

warn.t logical indicating if rankMatrix() should warn when it needs \(t(x)\) instead of \(x\). Currently, for method = "qr" only, gives a warning by default because the caller often could have passed \(t(x)\) directly, more efficiently.

Value

positive integer in \(1: \text{min}(\text{dim}(x))\), with attributes detailing the method used.
Note
For large sparse matrices x, unless you can specify sval yourself, currently method = "qr" may
be the only feasible one, as the others need sval and call svd() which currently coerces x to a
denseMatrix which may be very slow or impossible, depending on the matrix dimensions.

Note that in the case of sparse x, method = "qr", all non-strictly zero diagonal entries di where
counted, up to including Matrix version 1.1-0, i.e., that method implicitly used tol = 0, see also
the seed(42) example below.

Author(s)
Martin Maechler; for the "*Grad" methods, building on suggestions by Ravi Varadhan.

See Also
qr, svd.

Examples
rankMatrix(cbind(1, 0, 1:3)) # 2

(meths <- eval(formals(rankMatrix)$method))

## a "border" case:
H12 <- Hilbert(12)
rankMatrix(H12, tol = 1e-20) # 12; but 11 with default method & tol.
sapply(meths, function(.m) rankMatrix(H12, method = .m))

## tolNorm2: qr qr.R qrLINPACK useGrad maybeGrad
## 11 12 11 12 11 11
## The meaning of 'tol' for method="qrLINPACK" and *dense* x is not entirely "scale free"
rML1 <- function(ex, M) rankMatrix(M, method="qrLINPACK", tol = 10^-ex)
rMR <- function(ex, M) rankMatrix(M, method="qr.R", tol = 10^-ex)
sapply(5:15, rML1, M = H12) # result is platform dependent
## 7 7 8 10 10 11 11 12 12 12 (x86_64)
sapply(5:15, rML1, M = 1000 * H12) # not identical unfortunately
## 7 7 8 10 10 11 12 12 12 12
sapply(5:15, rMR, M = H12)
## 5 6 7 8 8 9 9 10 10 11
sapply(5:15, rMR, M = 1000 * H12) # the *same*

## "sparse" case:
M15 <- kronecker(diag(x=c(100,1,10)), Hilbert(5))
sapply(meths, function(.m) rankMatrix(M15, method = .m))
#-- all 15, but 'useGrad' has 14.

## "large" sparse
n <- 250000; p <- 33; nnz <- 10000
L <- sparseMatrix(i = sample.int(n, nnz, replace=TRUE),
  j = sample.int(p, nnz, replace=TRUE), x = rnorm(nnz))
(st1 <- system.time(r1 <- rankMatrix(L))) # warning + ~1.5 sec
(st2 <- system.time(r2 <- rankMatrix(L, method = "qr"))) # considerably faster!
r1[[1]] == print(r2[[1]]) ## --> ( 33 TRUE )

## another sparse-"qr" one, which "failed" till 2013-11-23:
set.seed(42)
f1 <- factor(sample(50, 1000, replace=TRUE))
f2 <- factor(sample(50, 1000, replace=TRUE))
f3 <- factor(sample(50, 1000, replace=TRUE))
D <- t(do.call('rBind', lapply(list(f1,f2,f3), as, 'sparseMatrix')))  
dim(D); nnzero(D) ## 1000 x 150 // 3000 non-zeros (= 2%)
stopifnot(rankMatrix(D, method=qr) == 148,
  rankMatrix(crossprod(D),method=qr') == 148)

rcond

Estimate the Reciprocal Condition Number

Description

Estimate the reciprocal of the condition number of a matrix.

This is a generic function with several methods, as seen by showMethods(rcond).

Usage

rcond(x, norm, ...)

## S4 method for signature 'sparseMatrix,character'
rcond(x, norm, useInv=FALSE, ...)

Arguments

x an R object that inherits from the Matrix class.
norm character string indicating the type of norm to be used in the estimate. The default is "O" for the 1-norm ("0" is equivalent to "1"). For sparse matrices, when useInv=TRUE, norm can be any of the kinds allowed for norm; otherwise, the other possible value is "I" for the infinity norm, see also norm.
useInv logical (or "Matrix" containing solve(x)). If not false, compute the reciprocal condition number as 1/(||x|| · ||x⁻¹||), where x⁻¹ is the inverse of x, solve(x). This may be an efficient alternative (only) in situations where solve(x) is fast (or known), e.g., for (very) sparse or triangular matrices.

... further arguments passed to or from other methods.

Value

An estimate of the reciprocal condition number of x.
BACKGROUND

The condition number of a regular (square) matrix is the product of the norm of the matrix and the norm of its inverse (or pseudo-inverse).

More generally, the condition number is defined (also for non-square matrices $A$) as

$$\kappa(A) = \frac{\max_{\|v\|=1} \|Av\|}{\min_{\|v\|=1} \|Av\|}$$

Whenever $x$ is not a square matrix, in our method definitions, this is typically computed via $rcond(qr.R(qr(X)), \ldots)$ where $X$ is $x$ or $t(x)$.

The condition number takes on values between 1 and infinity, inclusive, and can be viewed as a factor by which errors in solving linear systems with this matrix as coefficient matrix could be magnified.

$rcond()$ computes the reciprocal condition number $1/\kappa$ with values in $[0, 1]$ and can be viewed as a scaled measure of how close a matrix is to being rank deficient (aka “singular”).

Condition numbers are usually estimated, since exact computation is costly in terms of floating-point operations. An (over) estimate of reciprocal condition number is given, since by doing so overflow is avoided. Matrices are well-conditioned if the reciprocal condition number is near 1 and ill-conditioned if it is near zero.

References


See Also

norm, kappa() from package base computes an approximate condition number of a “traditional” matrix, even non-square ones, with respect to the $\rho = 2$ (Euclidean) norm. solve.

condest, a newer approximate estimate of the (1-norm) condition number, particularly efficient for large sparse matrices.

Examples

```r
x <- Matrix(rnorm(9), 3, 3)
rcond(x)
## typically "the same" (with more computational effort):
1 / (norm(x) * norm(solve(x)))
rcond(Hilbert(9))  # should be about 9.1e-13

## For non-square matrices:
rcond(x1 <- cbind(1,1:10))# 0.05278
rcond(x2 <- cbind(x1, 2:11))# practically 0, since x2 does not have full rank

## sparse
(S1 <- Matrix(rbind(0:1,0, diag(3:-2))))
rcond(S1)
m1 <- as(S1, "denseMatrix")
all.equal(rcond(S1), rcond(m1))
```
## Wide and Sparse

\[
\text{rcond(\text{Matrix(cbind(\emptyset, \text{diag}(2:-1))))})
\]

## Large Sparse Example

```r
m <- \text{Matrix(c(3,0:2), 2,2)}
M <- \text{bdiag(kronecker(\text{Diagonal}(2), m), kronecker(m,m))}
36*X(1M <- \text{solve(M)}) \# still sparse
MM <- \text{kronecker(\text{Diagonal}(10), kronecker(\text{Diagonal}(5),\text{kronecker}(m,M)))}
dim(M3 <- \text{kronecker(bdiag(M,M),MM)}) \# 12'800 \times 2
if(\text{interactive}) \# takes about 2 seconds if you have \text{>= 8 GB RAM}
  \text{system.time(r <- rcond(M3))}
## whereas this is \text{*fast*} even though it computes \text{solve(M3)}
if(\text{interactive}) \# the values are not the same
  c(r, r.) \# 0.05555 0.013888
## for all 4 norms available for sparseMatrix :
\text{cbind(rr <- \text{as.list(c("1","1","F","M")),}}
  \text{function(N) rcond(M3, norm=N, useInv=TRUE))}
```

---

### `rep2abI`

**Replicate Vectors into 'abIndex' Result**

#### Description

`rep2abI(x, times)` conceptually computes `rep.int(x, times)` but with an `abIndex` class result.

#### Usage

```r
rep2abI(x, times)
```

#### Arguments

- `x`: numeric vector
- `times`: integer (valued) scalar: the number of repetitions

#### Value

A vector of class `abIndex`

#### See Also

`rep.int()`, the base function; `abIseq, abIndex`.

#### Examples

```r
(ab <- rep2abI(2:7, 4))
stopifnot(identical(as(ab, "numeric"),
  rep(2:7, 4)))
```
replValue-class

Virtual Class "replValue" - Simple Class for subassignment Values

Description

The class "replValue" is a virtual class used for values in signatures for sub-assignment of "Matrix" matrices.

In fact, it is a simple class union (setClassUnion) of "numeric" and "logical" (and maybe "complex" in the future).

Objects from the Class

Since it is a virtual Class, no objects may be created from it.

See Also

Subassign-methods, also for examples.

Examples

showClass("replValue")

rleDiff-class

Class "rleDiff" of rle(diff(.)) Stored Vectors

Description

Class "rleDiff" is for compactly storing long vectors which mainly consist of linear stretches. For such a vector x, diff(x) consists of constant stretches and is hence well compressable via rle().

Objects from the Class

Objects can be created by calls of the form new("rleDiff", ...).

Currently experimental, see below.

Slots

first: A single number (of class "numLike", a class union of "numeric" and "logical").

rle: Object of class "rle", basically a list with components "lengths" and "values", see rle(). As this is used to encode potentially huge index vectors, lengths may be of type double here.

Methods

There is a simple show method only.
Note
This is currently an experimental auxiliary class for the class abIndex, see there.

See Also
rle, abIndex.

Examples

showClass("rleDiff")

ab <- c(abIseq(2, 100), abIseq(20, -2))
ab@rleD  # is "rleDiff"

rsparsematrix

Random Sparse Matrix

Description
Generate a Random Sparse Matrix Efficiently.

Usage

rsparsematrix(nrow, ncol, density, nnz = round(density * maxE),
            symmetric = FALSE,
            rand.x = function(n) signif(rnorm(nnz), 2), ...)

Arguments

nrow, ncol number of rows and columns, i.e., the matrix dimension (dim).
density optional number in [0, 1], the density is the proportion of non-zero entries among all matrix entries. If specified it determines the default for nnz, otherwise nnz needs to be specified.

nnz number of non-zero entries, for a sparse matrix typically considerably smaller than nrow * ncol. Must be specified if density is not.
symmetric logical indicating if result should be a matrix of class symmetricMatrix. Note that in the symmetric case, nnz denotes the number of non zero entries of the upper (or lower) part of the matrix, including the diagonal.

rand.x the random number generator for the x slot, a function such that rand.x(n) generates a numeric vector of length n. Typical examples are rand.x = rnorm, or rand.x = runif; the default is nice for didactical purposes.

... optionally further arguments passed to sparseMatrix(), notably giveCsparse.

Details
The algorithm first samples “encoded” (i, j)s without replacement, via one dimensional indices, if not symmetric sample.int(nrow * ncol, nnz), then gets x <- rand.x(nnz) and calls sparseMatrix(i=i, j=j, x=x, ...
RsparseMatrix-class

Value

a sparseMatrix, say M of dimension (nrow, ncol), i.e., with \text{dim}(M) = c(nrow, ncol). if symmetric is not true, with \text{nzM} \leftarrow \text{nnzero}(M) fulfilling \text{nzM} \leq \text{nnz} and typically, \text{nzM} = \text{nnz}.

Author(s)

Martin Maechler

Examples

```r
set.seed(17) # to be reproducible
M <- rsparsematrix(8, 12, nz = 30) # small example, not very sparse
M
M1 <- rsparsematrix(1000, 20, nz = 123, rand.x = runif)
summary(M1)

## a random *symmetric* Matrix
(S9 <- rsparsematrix(9, 9, nz = 10, symmetric=TRUE)) # dsCMatrix
nnzero(S9)# ~ 20: as 'nnz' only counts one "triangle"

## a [T]riplet representation sparseMatrix:
T2 <- rsparsematrix(40, 12, nz = 99, giveSparse=FALSE)
head(T2)
```

RsparseMatrix-class Class “RsparseMatrix” of Sparse Matrices in Column-compressed Form

Description

The "RsparseMatrix" class is the virtual class of all sparse matrices coded in sorted compressed row-oriented form. Since it is a virtual class, no objects may be created from it. See showClass("RsparseMatrix") for its subclasses.

Slots

- \text{j}: Object of class "integer" of length \text{nnzero} (number of non-zero elements). These are the row numbers for each non-zero element in the matrix.
- \text{p}: Object of class "integer" of pointers, one for each row, to the initial (zero-based) index of elements in the row.

\text{Dim}, \text{Dimnames}: inherited from the superclass, see sparseMatrix.

Extends

Class "sparseMatrix", directly. Class "Matrix", by class "sparseMatrix".
Methods

Only few methods are defined currently on purpose, since we rather use the CsparseMatrix in Matrix. Recently, more methods were added but beware that these typically do not return "RsparseMatrix" results, but rather Csparse* or Tsparse* ones.

\[
\text{t signature(x = "RsparseMatrix"): ...}
\]
\[
\text{coerce signature(from = "RsparseMatrix", to = "CsparseMatrix"): ...}
\]
\[
\text{coerce signature(from = "RsparseMatrix", to = "TsparseMatrix"): ...}
\]

See Also

its superclass, sparseMatrix, and, e.g., class dgRMatrix for the links to other classes.

Examples

showClass("RsparseMatrix")

---

Schur  Schur Decomposition of a Matrix

Description

Computes the Schur decomposition and eigenvalues of a square matrix; see the BACKGROUND information below.

Usage

Schur(x, vectors, ...)

Arguments

x numeric square Matrix (inheriting from class "Matrix") or traditional matrix. Missing values (NAs) are not allowed.

vectors logical. When TRUE (the default), the Schur vectors are computed, and the result is a proper MatrixFactorization of class Schur.

... further arguments passed to or from other methods.

Details

Based on the Lapack subroutine dgees.

Value

If vectors are TRUE, as per default: If x is a Matrix an object of class Schur, otherwise, for a traditional matrix x, a list with components T, Q, and EValues.

If vectors are FALSE, a list with components

T the upper quasi-triangular (square) matrix of the Schur decomposition.

EValues the vector of numeric or complex eigen values of \( T \) or \( A \).
BACKGROUND

If \( A \) is a square matrix, then \( A = QTt(Q) \), where \( Q \) is orthogonal, and \( T \) is upper block-triangular (nearly triangular with either 1 by 1 or 2 by 2 blocks on the diagonal) where the 2 by 2 blocks correspond to (non-real) complex eigenvalues. The eigenvalues of \( A \) are the same as those of \( T \), which are easy to compute. The Schur form is used most often for computing non-symmetric eigenvalue decompositions, and for computing functions of matrices such as matrix exponentials.

References


Examples

```r
Schur(Hilbert(9)) # Schur factorization (real eigenvalues)

(A <- Matrix(round(rnorm(5*5, sd = 100)), nrow = 5))
(Sch.A <- Schur(A))

eTA <- eigen(Sch.A@T)
str(SchA <- Schur(A, vectors=FALSE))# no 'T' ==> simple list
stopifnot(all.equal(eTA$values, eigen(A)$values, tolerance = 1e-13),
  all.equal(eTA$values,
    local({z <- Sch.A@EValues
      z[order(Mod(z), decreasing=TRUE)]}, tolerance = 1e-13),
    identical(SchA$T, Sch.A@T),
    identical(SchA$EValues, Sch.A@EValues))

## For the faint of heart, we provide Schur() also for traditional matrices:

a.m <- function(M) unname(as(M, "matrix"))
a <- a.m(A)
Sch.a <- Schur(a)
stopifnot(identical(Sch.a, list(Q = a.m(Sch.A @ Q),
  T = a.m(Sch.A @ T),
  EValues = Sch.A@EValues)),
  all.equal(a, with(Sch.a, Q *%*% T *%*% t(Q))))
```

Schur-class  

Class ”Schur” of Schur Matrix Factorizations

Description

Class ”Schur” is the class of Schur matrix factorizations. These are a generalization of eigen value (or “spectral”) decompositions for general (possibly asymmetric) square matrices, see the Schur() function.

Objects from the Class

Objects of class ”Schur” are typically created by Schur().
Slots

"Schur" has slots

T: Upper Block-triangular Matrix object.
Q: Square orthogonal "Matrix".
EValues: numeric or complex vector of eigenvalues of T.
Dim: the matrix dimension: equal to c(n,n) of class "integer".

Extends

Class "MatrixFactorization", directly.

See Also

Schur() for object creation: MatrixFactorization.

Examples

showClass("Schur")
Schur(M <- Matrix(c(1:7, 10:2), 4,4))
## Trivial, of course:
str(Schur(Diagonal(5)))

## for more examples, see Schur()

---

solve-methods

Methods in Package Matrix for Function solve()

Description

Methods for function solve to solve a linear system of equations, or equivalently, solve for X in

\[ AX = B \]

where \( A \) is a square matrix, and \( X, B \) are matrices or vectors (which are treated as 1-column matrices), and the R syntax is

\[ X <- \text{solve}(A,B) \]

In solve(a,b) in the Matrix package, a may also be a MatrixFactorization instead of directly a matrix.
Usage

```r
## S4 method for signature 'CHMfactor,ddenseMatrix'
solve(a, b,
    system = c("A", "LDLt", "LD", "DLt", "L", "Lt", "D", "P", "Pt"), ...)

## S4 method for signature 'dgCMatrix,matrix'
solve(a, b, sparse = FALSE, tol = .Machine$double.eps, ...)
```

```r
solve(a, b, ...)
## *the* two-argument version, almost always preferred to
# solve(a) ## *the* rarely needed one-argument version
```

Arguments

- **a**: a square numeric matrix, \( A \), typically of one of the classes in \texttt{Matrix}. Logical matrices are coerced to corresponding numeric ones.
- **b**: numeric vector or matrix (dense or sparse) as RHS of the linear system \( Ax = b \).
- **system**: only if \( a \) is a \texttt{CHMfactor}: character string indicating the kind of linear system to be solved, see below. Note that the default, "A", does not solve the triangular system (but "L" does).
- **sparse**: only when \( a \) is a \texttt{sparseMatrix}, i.e., typically a \texttt{dgCMatrix}: logical specifying if the result should be a (formally) sparse matrix.
- **tol**: only used when \( a \) is sparse, in the \texttt{isSymmetric}(a, tol=*) test, where that applies.
- **...**: potentially further arguments to the methods.

Methods

- \texttt{signature(a = "ANY", b = "ANY")} is simply the \texttt{base} package's S3 generic \texttt{solve}.
- \texttt{signature(a = "CHMfactor", b = "...")}, \texttt{system=} *: The solve methods for a "\texttt{CHMfactor}" object take an optional third argument \texttt{system} whose value can be one of the character strings "A", "LDLt", "LD", "DLt", "L", "Lt", "D", "P" or "Pt". This argument describes the system to be solved. The default, "A", is to solve \( Ax = b \) for \( x \) where \( A \) is sparse, positive-definite matrix that was factored to produce \( a \). Analogously, \texttt{system = "L"} returns the solution \( x \), of \( Lx = b \); similarly, for all system codes \texttt{but} "P" and "Pt" where, e.g., \( x \leftarrow \texttt{solve(a, b, system="P")} \) is equivalent to \( x \leftarrow P \%*% b \).
  - If \( b \) is a \texttt{sparseMatrix}, \texttt{system} is used as above the corresponding sparse CHOLMOD algorithm is called.
- \texttt{signature(a = "ddenseMatrix", b = "...")} (for all b) work via \texttt{as(a, "dgeMatrix")}, using the its methods, see below.
- \texttt{signature(a = "denseLU", b = "missing")} basically computes uses triangular forward- and back-solve.
- \texttt{signature(a = "dgCMatrix", b = "matrix")}, and
signature(a = "dgCMatrix", b = "ddenseMatrix") with extra argument list ( sparse = FALSE, tol = .Machine$double.eps
Uses the sparse lu(a) decomposition (which is cached in a's factor slot). By default, sparse=FALSE, returns a denseMatrix, since $U^{-1}L^{-1}B$ may not be sparse at all, even when $L$ and $U$ are.
If sparse=TRUE, returns a sparseMatrix (which may not be very sparse at all, even if a was sparse).

signature(a = "dgCMatrix", b = "d sparseMatrix") , and
signature(a = "dgCMatrix", b = "missing") with extra argument list ( sparse=FALSE, tol = .Machine$double.eps.
Checks if a is symmetric, and in that case, coerces it to "symmetricMatrix", and then computes a sparse solution via sparse Cholesky factorization, independently of the sparse argument. If a is not symmetric, the sparse lu decomposition is used and the result will be sparse or dense, depending on the sparse argument, exactly as for the above (b = "ddenseMatrix") case.

signature(a = "dgeMatrix", b = ".....") solve the system via internal LU, calling LAPACK routines dgetri or dgetrs.

signature(a = "diagonalMatrix", b = "matrix") and other bs: Of course this is trivially implemented, as $D^{-1}$ is diagonal with entries $1/D[i,i].$

signature(a = "dpoMatrix", b = "....Matrix"), and
signature(a = "dppMatrix", b = "....Matrix") The Cholesky decomposition of a is calculated (if needed) while solving the system.

signature(a = "dsCMatrix", b = ".....") All these methods first try Cholmod's Cholesky factorization; if that works, i.e., typically if a is positive semi-definite, it is made use of. Otherwise, the sparse LU decomposition is used as for the "general" matrices of class "dgCMatrix".

signature(a = "dspMatrix", b = "....."), and
signature(a = "dsyMatrix", b = ".....") all end up calling LAPACK routines dspptri, dspptrs, dsytrs and dsytri.

signature(a = "dtCMatrix", b = "C sparseMatrix"),
signature(a = "dtMatrix", b = "dgeMatrix"), etc sparse triangular solve, in traditional S/R also known as backsolve, or forwardsolve. solve(a,b) is a sparseMatrix if b is, and hence a denseMatrix otherwise.

signature(a = "dtrMatrix", b = "ddenseMatrix"), and
signature(a = "dtpMatrix", b = "matrix"), and similar b, including "missing", and "diagonalMatrix": all use LAPACK based versions of efficient triangular backsolve, or forwardsolve.

signature(a = "Matrix", b = "diagonalMatrix") works via as(b, "C sparseMatrix").

signature(a = "sparseQR", b = "ANY") simply uses qr.coef(a, b).

signature(a = "pMatrix", b = ".....") these methods typically use crossprod(a,b), as the inverse of a permutation matrix is the same as its transpose.

signature(a = "T sparseMatrix", b = "ANY") all work via as(a, "C sparseMatrix").

See Also

solve, lu, and class documentations CHMfactor, sparseLU, and MatrixFactorization.
Examples

```r
## A close to symmetric example with "quite sparse" inverse:
n1 <- 7; n2 <- 3
dd <- data.frame(a = gl(n1,n2), b = gl(n2,1,n1*n2)) # balanced 2-way
X <- sparse.model.matrix(~ -1 + a + b, dd) # no intercept --> even sparser
Xxt <- tcrossprod(X)
diag(Xxt) <- rep(c(0,0,1,0), length.out = nrow(Xxt))

n <- nrow(ZZ <- kronecker(Xxt, Diagonal(x=c(4,1))))
image(a <- 2*Diagonal(n) + ZZ %*% Diagonal(x=c(10, rep(1, n-1))))
isSymmetric(a) # FALSE
image(drop0(skewpart(a)))
image(ia0 <- solve(a)) # checker board, dense [but really, a is singular!]
try(solve(a, sparse=TRUE))##-> error [ TODO: assertError ]
ia. <- solve(a, sparse=TRUE, tol = 1e-19)##-> *no* error
if(R.version$arch == "x86_64")
  ## Falls on 32-bit [Fedora 19, R 3.0.2] from Matrix 1.1-0 on [FIXME ?] only
  stopifnot(all.equal(as.matrix(ia.), as.matrix(ia0)))
a <- a + Diagonal(n)
iah <- solve(a)
ihi <- solve(a, sparse=TRUE)
stopifnot(all.equal(as(iah,"denseMatrix"), iah, tolerance=1e-14))
I. <- iah %*% a 
  ; image(I.)
I0 <- drop0(zapsmall(I.)); image(I0)
.j <- a %*% iah
.i0 <- drop0(zapsmall(.i))
stopifnot(all.equal(as(I0, "diagonalMatrix"), Diagonal(n)),
  all.equal(as(.i0,"diagonalMatrix"), Diagonal(n))
```

---

**sparse.model.matrix**

**Construct Sparse Design / Model Matrices**

**Description**

Construct a sparse model or “design” matrix, form a formula and data frame (sparse.model.matrix) or a single factor (fac2sparse).

The `fac2sparse()` functions are utilities, also used internally in the principal user level function `sparse.model.matrix()`.

**Usage**

```r
sparse.model.matrix(object, data = environment(object),
  contrasts.arg = NULL, xlev = NULL, transpose = FALSE,
  drop.unused.levels = FALSE, row.names = TRUE,
  verbose = FALSE, ...)

fac2sparse(from, to = c("d", "i", "l", "n", "z"),

```
drop.unused.levels = TRUE, giveCsparse = TRUE)

fac2Sparse(from, to = c("d", "i", "l", "n", "z"),
    drop.unused.levels = TRUE, giveCsparse = TRUE,
    factorPatt12, contrasts.arg = NULL)

**Arguments**

- **object**: an object of an appropriate class. For the default method, a model formula or terms object.
- **data**: a data frame created with `model.frame`. If another sort of object, `model.frame` is called first.
- **contrasts.arg**: for `sparse.model.matrix()`: A list, whose entries are contrasts suitable for input to the `contrasts` replacement function and whose names are the names of columns of data containing factors.
  - For `fac2Sparse()`: character string or NULL or (coercable to) "sparseMatrix", specifying the contrasts to be applied to the factor levels.
- **xlev**: to be used as argument of `model.frame` if data has no "terms" attribute.
- **transpose**: logical indicating if the `transpose` should be returned; if the transposed is used anyway, setting `transpose = TRUE` is more efficient.
- **drop.unused.levels**: should factors have unused levels dropped? The default for `sparse.model.matrix` has been changed to `FALSE`, 2010-07, for compatibility with R's standard (dense) `model.matrix()`.
- **row.names**: logical indicating if row names should be used.
- **verbose**: logical or integer indicating if (and how much) progress output should be printed.
- **...**: further arguments passed to or from other methods.
- **from** (for `fac2sparse()`) a factor.
- **to** a character indicating the “kind” of sparse matrix to be returned. The default, "d" is for double.
- **giveCsparse** (for `fac2sparse()`) logical indicating if the result must be a `CsparseMatrix`.
- **factorPatt12** logical vector, say `fp`, of length two; when `fp[1]` is true, return “contrasted” `t(X)`; when `fp[2]` is true, the original (“dummy”) `t(X)`, i.e, the result of `fac2sparse()`.

**Value**

- A sparse matrix, extending `CsparseMatrix` (for `fac2sparse()` if `giveCsparse` is true as per default; a `TsparseMatrix`, otherwise).

For `fac2Sparse()`, a list of length two, both components with the corresponding transposed model matrix, where the corresponding `factorPatt12` is true.

Note that `model.Matrix(*, sparse=TRUE)` from package `MatrixModels` may be often be preferable to `sparse.model.matrix()` nowadays, as `model.Matrix()` returns `modelMatrix` objects with additional slots `assign` and `contrasts` which relate back to the variables used.

`fac2sparse()`, the basic workhorse of `sparse.model.matrix()`, returns the `transpose` (t) of the model matrix.
sparse.model.matrix

Author(s)

Doug Bates and Martin Maechler, with initial suggestions from Tim Hesterberg.

See Also

model.matrix in standard R’s package stats.
model.Matrix which calls sparse.model.matrix or model.matrix depending on its sparse argument may be preferred to sparse.model.matrix.

as(f, "sparseMatrix") (see coerce(from = “factor”, ..) in the class doc sparseMatrix) produces the transposed sparse model matrix for a single factor f (and no contrasts).

Examples

dd <- data.frame(a = gl(3,4), b = gl(4,1,12))# balanced 2-way options("contrasts") # the default: "contr.treatment"
sparse.model.matrix(~ a + b, dd)
sparse.model.matrix(~ -1+ a + b, dd)# no intercept --> even sparser
sparse.model.matrix(~ a + b, dd, contrasts = list(a="contr.sum"))
sparse.model.matrix(~ a + b, dd, contrasts = list(b="contr.SAS"))

## Sparse method is equivalent to the traditional one:
stopifnot(all(sparse.model.matrix(~ a + b, dd) ==
 Matrix(model.matrix(~ a + b, dd), sparse=TRUE)),
 all(sparse.model.matrix(~ 0+ a + b, dd) ==
 Matrix(model.matrix(~ 0+ a + b, dd), sparse=TRUE)))

(ff <- gl(3, 4, c("X", "Y", "Z")))
fac2sparse(ff) # 3 x 12 sparse Matrix of class "dgCMatrix"

##
## X 1 1 1 1 . . . .
## Y . . . 1 1 1 1 . .
## Z . . . . . 1 1 1 1

## can also be computed via sparse.model.matrix():
f30 <- gl(3, 0 )
f12 <- gl(3,0, 12)
stopifnot(
 all.equal(t( fac2sparse(ff ) ),
 sparse.model.matrix(~ 0+ff),
 tolerance = 0, check.attributes=FALSE),
 is(M <- fac2sparse(f30, drop= TRUE),"CsparseMatrix"), dim(M) == c(0, 0),
 is(M <- fac2sparse(f30, drop=FALSE),"CsparseMatrix"), dim(M) == c(3, 0),
 is(M <- fac2sparse(f12, drop= TRUE),"CsparseMatrix"), dim(M) == c(0,12),
 is(M <- fac2sparse(f12, drop=FALSE),"CsparseMatrix"), dim(M) == c(3,12) )
Description

Objects of this class contain the components of the LU decomposition of a sparse square matrix.

Objects from the Class

Objects can be created by calls of the form `new("sparseLU", ...) but are more commonly created by function `lu()` applied to a sparse matrix, such as a matrix of class `dgCMatrix`.

Slots

- `l`: Object of class "`dtCMatrix`", the lower triangular factor from the left.
- `u`: Object of class "`dtCMatrix`", the upper triangular factor from the right.
- `p`: Object of class "`integer`", permutation applied from the left.
- `q`: Object of class "`integer`", permutation applied from the right.
- `dim`: the dimension of the original matrix; inherited from class `MatrixFactorization`.

Extends

Class "`LU`", directly. Class "`MatrixFactorization`", by class "`LU`".

Methods

- `expand` signature(x = "sparseLU") Returns a list with components `P`, `L`, `U`, and `Q`, where `P` and `Q` represent fill-reducing permutations, and `L`, and `U` the lower and upper triangular matrices of the decomposition. The original matrix corresponds to the product `PLUQ`.

Note

The decomposition is of the form

\[ A = P'LUQ, \]

or equivalently `PAQ' = LU`, where all matrices are sparse and of size \( n \times n \). The matrices `P` and `Q`, and their transposes `P'` and `Q'` are permutation matrices, `L` is lower triangular and `U` is upper triangular.

See Also

- `lu`, `solve`, `dgCMatrix`
Examples

```r
## Extending the one in examples(lu), calling the matrix A,
## and confirming the factorization identities :
A <- as(readMM(system.file("external/pores1 mtx", package = "Matrix"),
            "CsparseMatrix")
str(luA <- lu(A)) # p is a 0-based permutation of the rows
               # q is a 0-based permutation of the columns
XA <- expand(luA)
## which is simply doing
stopifnot(identical(XA$L, luA@L),
           identical(XA$U, luA@U),
           identical(XA$P, as(luA@p +1 L, "pMatrix"),
           identical(XA$Q, as(luA@q +1 L, "pMatrix"))

P.LUQ <- with(XA, t(P) %%*% L %%*% U %%*% Q)
stopifnot(all.equal(A, P.LUQ, tolerance = 1e-12))
## permute rows and columns of original matrix
pA <- A[luA@p +1 L, luA@q +1 L]
stopifnot(identical(pA, with(XA, P %%*% A %%*% t(Q))))

pLU <- drop0(luA@L %%*% luA@U) # L %%*% U -- dropping extra zeros
stopifnot(all.equal(pA, pLU))
```

Description

Methods for coercion from and to sparse matrices from package **SparseM** are provided here, for ease of porting functionality to the Matrix package, and comparing functionality of the two packages.

Methods

```r
from = "matrix.csr", to = "dgRMatrix" ...
from = "matrix.csc", to = "dgCMatrix" ...
from = "matrix.coo", to = "dgTMatrix" ...
from = "dgRMatrix", to = "matrix.csr" ...
from = "dgCMatrix", to = "matrix.csc" ...
from = "dgTMatrix", to = "matrix.coo" ...
from = "sparseMatrix", to = "matrix.csr" ...
from = "matrix.csr", to = "dgCMatrix" ...
from = "matrix.coo", to = "dgCMatrix" ...
from = "matrix.csr", to = "Matrix" ...
from = "matrix.csc", to = "Matrix" ...
from = "matrix.coo", to = "Matrix" ...
```
sparseMatrix

General Sparse Matrix Construction from Nonzero Entries

Description

User friendly construction of a compressed, column-oriented, sparse matrix, inheriting from class CsparseMatrix (or TsparseMatrix if giveCsparse is false), from locations (and values) of its non-zero entries.

This is the recommended user interface rather than direct new("***Matrix", ...) calls.

Usage

sparseMatrix(i = ep, j = ep, p, x, dims, dimnames,
             symmetric = FALSE, index1 = TRUE,
             giveCsparse = TRUE, check = TRUE, use.last.ij = FALSE)

Arguments

i, j  integer vectors of the same length specifying the locations (row and column indices) of the non-zero (or non-TRUE) entries of the matrix. Note that for repeated pairs \((i_k, j_k)\), when \(x\) is not missing, the corresponding \(x_k\) are added, in consistency with the definition of the "TsparseMatrix" class, unless use.last.ij is true, in which case only the last of the corresponding \((i_k, j_k, x_k)\) triplet is used.

p  numeric (integer valued) vector of pointers, one for each column (or row), to the initial (zero-based) index of elements in the column (or row). Exactly one of \(i\), \(j\) or \(p\) must be missing.

x  optional values of the matrix entries. If specified, must be of the same length as \(i \sim j\), or of length one where it will be recycled to full length. If missing, the resulting matrix will be a 0/1 pattern matrix, i.e., extending class nsparseMatrix.

dims  optional, non-negative, integer, dimensions vector of length 2. Defaults to \(c(max(i), max(j))\).

dimnames  optional list of dimnames; if not specified, none, i.e., NULL ones, are used.

symmetric  logical indicating if the resulting matrix should be symmetric. In that case, only the lower or upper triangle needs to be specified via \((i/j/p)\).

index1  logical scalar. If TRUE, the default, the index vectors \(i\) and/or \(j\) are 1-based, as is the convention in R. That is, counting of rows and columns starts at 1. If FALSE the index vectors are 0-based so counting of rows and columns starts at 0; this corresponds to the internal representation.

giveCsparse  logical indicating if the result should be a CsparseMatrix or a TsparseMatrix. The default, TRUE is very often more efficient subsequently, but not always.

See Also

The documentation in package SparseM, such as SparseM.ontology, and one important class, matrix.csr.
sparseMatrix

check logical indicating if a validity check is performed; do not set to FALSE unless you know what you’re doing!

use.last.i j logical indicating if in the case of repeated, i.e., duplicated pairs \( (i_k, j_k) \) only the last one should be used. The default, FALSE, corresponds to the "TsparseMatrix" definition.

Details

Exactly one of the arguments \( i, j \) and \( p \) must be missing.

In typical usage, \( p \) is missing, \( i \) and \( j \) are vectors of positive integers and \( x \) is a numeric vector. These three vectors, which must have the same length, form the triplet representation of the sparse matrix.

If \( i \) or \( j \) is missing then \( p \) must be a non-decreasing integer vector whose first element is zero. It provides the compressed, or “pointer” representation of the row or column indices, whichever is missing. The expanded form of \( p, \text{rep}(\text{seq.along}(dp), dp) \) where \( dp \leftarrow \text{diff}(p) \), is used as the (1-based) row or column indices.

The values of \( i, j, p \) and index1 are used to create 1-based index vectors \( i \) and \( j \) from which a TsparseMatrix is constructed, with numerical values given by \( x \), if non-missing. Note that in that case, when some pairs \( (i_k, j_k) \) are repeated (aka “duplicated”), the corresponding \( x_k \) are added, in consistency with the definition of the "TsparseMatrix" class, unless use.last.i j is set to true.

By default, when giveCsparse is true, the CsparseMatrix derived from this triplet form is returned.

The reason for returning a CsparseMatrix object instead of the triplet format by default is that the compressed column form is easier to work with when performing matrix operations. In particular, if there are no zeros in \( x \) then a CsparseMatrix is a unique representation of the sparse matrix.

Value

A sparse matrix, by default (see giveCsparse) in compressed, column-oriented form, as an \( \mathbb{R} \) object inheriting from both CsparseMatrix and generalMatrix.

Note

You do need to use index1 = FALSE (or add + 1 to \( i \) and \( j \)) if you want use the 0-based \( i \) (and \( j \)) slots from existing sparse matrices.

See Also

Matrix\((*, \text{sparse}=\text{TRUE})\) for the constructor of such matrices from a dense matrix. That is easier in small sample, but much less efficient (or impossible) for large matrices, where something like sparseMatrix\(()\) is needed. Further bdiag and Diagonal for (block-)diagonal and bandSparse for banded sparse matrix constructors.

The standard \( \mathbb{R} \) xtabs\((* , \text{sparse}=\text{TRUE})\), for sparse tables and sparse.model.matrix\(()\) for building sparse model matrices.

Consider CsparseMatrix and similar class definition help files.
Examples

```r
## simple example
i <- c(1,3:8); j <- c(2,9,6:10); x <- 7 * (1:7)
(A <- sparseMatrix(i, j, x = x))
summary(A)
str(A) # note that *internally* 0-based row indices are used

## dims can be larger than the maximum row or column indices
(AA <- sparseMatrix(c(1,3:8), c(2,9,6:10), x = 7 * (1:7), dims = c(10,20)))
summary(AA)

## i, j and x can be in an arbitrary order, as long as they are consistent
set.seed(1); (perm <- sample(1:7))
(A[1] <- sparseMatrix(i[perm], j[perm], x = x[perm]))
stopifnot(identical(A, A[1]))

## The slots are 0-index based, so
try( sparseMatrix(i=A@i, p=A@p, x= seq_along(A@x) ) )
## fails and you should say so: 1-indexing is FALSE:
 sparseMatrix(i=A@i, p=A@p, x= seq_along(A@x), index1 = FALSE)

## the (i,j) pairs can be repeated, in which case the x's are summed
(args <- data.frame(i = c(i, 1), j = c(j, 2), x = c(x, 2)))
(Aa <- do.call(sparseMatrix, args))
## explicitly ask for elimination of such duplicates, so
## that the last one is used:
(A. <- do.call(sparseMatrix, c(args, list(use.last.ij = TRUE))))
stopifnot(Aa[1,2] == 9, # 2*7 == 9
        A.[1,2] == 2) # 2 was *after* 7

## for a pattern matrix, of course there is no "summing":
(nA <- do.call(sparseMatrix, args[c("i","j")]))

dn <- list(LETTERS[1:3], letters[1:5])
## pointer vectors can be used, and the (i,x) slots are sorted if necessary:
m <- sparseMatrix(i = c(3,1,3:2,2:1), p= c(0:2, 4,4,6), x = 1:6, dimnames = dn)
m
str(m)
stopifnot(identical(dimnames(m), dn))

sparseMatrix(x = 2.72, i=1:3, j=2:4) # recycling x
sparseMatrix(x = TRUE, i=1:3, j=2:4) # recycling x, |--> "lgCMatrix"

## no 'x' --> pattern matrix:
(n <- sparseMatrix(i=1:6, j=rev(2:7)))# --> ngCMatrix

## an empty sparse matrix:
(e <- sparseMatrix(dims = c(4,6), i=(), j=()))

## a symmetric one:
(sy <- sparseMatrix(i= c(2,4,3:5), j= c(4,7:5:5), x = 1:5,
               dims = c(7,7), symmetric=TRUE))
```
sparseMatrix-class

Virtual Class "sparseMatrix" — Mother of Sparse Matrices

Description

Virtual Mother Class of All Sparse Matrices

Slots

Dim: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.

Dimnames: a list of length two - inherited from class Matrix, see Matrix.

Extends

Class "Matrix", directly.

Methods

show (object = "sparseMatrix"): The show method for sparse matrices prints “structural” zeroes as "." using printSpMatrix() which allows further customization.

print signature(x = "sparseMatrix"). The print method for sparse matrices by default is the same as show() but can be called with extra optional arguments, see printSpMatrix().
format signature(x = "sparseMatrix"),....
   The format method for sparse matrices, see formatSpMatrix() for details such as the extra optional arguments.

summary (object = "sparseMatrix"): Returns an object of S3 class "sparseSummary" which is basically a data.frame with columns (i,j,x) (or just (i,j)) for nspareMatrix class objects) with the stored (typically non-zero) entries. The print method resembles Matlab's way of printing sparse matrices, and also the MatrixMarket format, see writeMM.

cbind2 (x = *, y = *): several methods for binding matrices together, column-wise, see the basic cbind and rbind functions.
   Note that the result will typically be sparse, even when one argument is dense and larger than the sparse one.

rbind2 (x = *, y = *): binding matrices together row-wise, see cbind2 above.
determinant (x = "sparseMatrix", logarithm=TRUE): determinant() methods for sparse matrices typically work via Cholesky or lu decompositions.
diag (x = "sparseMatrix"): extracts the diagonal of a sparse matrix.
dim<signature(x = "sparseMatrix", value = "ANY"): allows to reshape a sparse matrix to a sparse matrix with the same entries but different dimensions. value must be of length two and fulfill prod(value) == prod(dim(x)).

coerce signature(from = "factor", to = "sparseMatrix"): Coercion of a factor to "sparseMatrix" produces the matrix of indicator rows stored as an object of class "dgCMatrix". To obtain columns representing the interaction of the factor and a numeric covariate, replace the "x" slot of the result by the numeric covariate then take the transpose. Missing values (NA) from the factor are translated to columns of all 0s.

See also colSums, norm, ... for methods with separate help pages.

Note
   In method selection for multiplication operations (i.e. %*% and the two-argument form of crossprod) the sparseMatrix class takes precedence in the sense that if one operand is a sparse matrix and the other is any type of dense matrix then the dense matrix is coerced to a dgMatrix and the appropriate sparse matrix method is used.

See Also
   sparseMatrix, and its references, such as xtabs(*, sparse=TRUE), or sparse.model.matrix(), for constructing sparse matrices.

T2graph for conversion of "graph" objects (package graph) to and from sparse matrices.

Examples
   showClass("sparseMatrix") ## and look at the help() of its subclasses
   M <- Matrix(0, 10000, 100)
   M[1,1] <- M[2,3] <- 3.14
   M ## show(.) method suppresses printing of the majority of rows

data(CAex); dim(CAex) # 72 x 72 matrix
   determinant(CAex) # works via sparse lu(.)
Description

Objects class "sparseQR" represent a QR decomposition of a sparse $n \times p$ rectangular matrix $X$, typically resulting from `qr()`.

Details

The decomposition is of the form $A[p+1,] = Q \cdot R$, if the $q$ slot is of length $0$ or $A[p+1,q+1] = Q \cdot R$ where $A$ is a sparse $m \times n$ matrix ($m \geq n$), $R$ is an $m \times n$ matrix that is zero below the main diagonal. The $p$ slot is a 0-based permutation of $1:m$ applied to the rows of the original matrix. If the $q$ slot has length $n$ it is a 0-based permutation of $1:n$ applied to the columns of the original matrix to reduce the amount of "fill-in" in the matrix $R$.

The matrix $Q$ is a "virtual matrix". It is the product of $n$ Householder transformations. The information to generate these Householder transformations is stored in the $V$ and $beta$ slots.

The "sparseQR" methods for the `qr.*` functions return objects of class "dgeMatrix" (see `dgeMatrix`). Results from `qr.coef`, `qr.resid` and `qr.fitted` (when $k = ncol(R)$) are well-defined and should match those from the corresponding dense matrix calculations. However, because the matrix $Q$ is not uniquely defined, the results of `qr$qy` and `qr$qty` do not necessarily match those from the corresponding dense matrix calculations.

Also, the results of `qr$qy` and `qr$qty` apply to the permuted column order when the $q$ slot has length $n$.

Objects from the Class

Objects can be created by calls of the form `new("sparseQR", ...)` but are more commonly created by function `qr` applied to a sparse matrix such as a matrix of class `dgCMatrix`.

Slots

$V$: Object of class "dgeMatrix". The columns of $V$ are the vectors that generate the Householder transformations of which the matrix $Q$ is composed.

$beta$: Object of class "numeric", the normalizing factors for the Householder transformations.
p: Object of class "integer": Permutation (of \(0:(n-1)\)) applied to the rows of the original matrix.

q: Object of class "integer": Permutation applied from the right. Can be of length 0 which implies no permutation.

Methods

qr.R signature(qr = "sparseQR"): compute the upper triangular \(R\) matrix of the QR decomposition. Note that this currently warns because of possible permutation mismatch with the classical \(qr.R()\) result, and you can suppress these warnings by setting \texttt{options()} either "Matrix.quiet.qr.R" or (the more general) either "Matrix.quiet" to \texttt{TRUE}.

qr.Q signature(qr = "sparseQR"): compute the orthogonal \(Q\) matrix of the QR decomposition.

\texttt{qr.coef} signature(qr = "sparseQR", y = "ddenseMatrix"): ...
\texttt{qr.coef} signature(qr = "sparseQR", y = "matrix"): ...
\texttt{qr.coef} signature(qr = "sparseQR", y = "numeric"): ...
\texttt{qr.fitted} signature(qr = "sparseQR", y = "ddenseMatrix"): ...
\texttt{qr.fitted} signature(qr = "sparseQR", y = "matrix"): ...
\texttt{qr.fitted} signature(qr = "sparseQR", y = "numeric"): ...
\texttt{qr.qty} signature(qr = "sparseQR", y = "ddenseMatrix"): ...
\texttt{qr.qty} signature(qr = "sparseQR", y = "matrix"): ...
\texttt{qr.qty} signature(qr = "sparseQR", y = "numeric"): ...
\texttt{qr.qy} signature(qr = "sparseQR", y = "ddenseMatrix"): ...
\texttt{qr.qy} signature(qr = "sparseQR", y = "matrix"): ...
\texttt{qr.qy} signature(qr = "sparseQR", y = "numeric"): ...
\texttt{qr.resid} signature(qr = "sparseQR", y = "ddenseMatrix"): ...
\texttt{qr.resid} signature(qr = "sparseQR", y = "matrix"): ...
\texttt{qr.resid} signature(qr = "sparseQR", y = "numeric"): ...
\texttt{solve} signature(a = "sparseQR", b = "ANY"): For \texttt{solve(a,b)}, simply uses \texttt{qr.coef(a,b)}.

See Also

\texttt{qr}, \texttt{qr.Q}, \texttt{qr.R}, \texttt{qr.fitted}, \texttt{qr.resid}, \texttt{qr.coef}, \texttt{qr.qty}, \texttt{qr.qy}, \texttt{dgCMatrix}, \texttt{dgeMatrix}.

Examples

data(KNex)
\texttt{mm} <- KNex \$ \texttt{mm}
\texttt{y} <- KNex \$ \texttt{y}
\texttt{y.} <- \texttt{as(as.matrix(y), \texttt{dgCMatrix})}
\texttt{str(qrm <- qr(mm))}
\texttt{qc <- qr.coef(qrm, y); qc. <- qr.coef(qrm, y.)} # 2nd failed in Matrix <= 1.1-0
\texttt{qf <- qr.fitted(qrm, y); qf. <- qr.fitted(qrm, y.)}
\texttt{qs <- qr.resid(qrm, y); qs. <- qr.resid(qrm, y.)}
\texttt{stopifnot(all.equal(qc, as.numeric(qc.), tolerance=1e-12),}
sparseVector

Sparse Vector Construction from Nonzero Entries

Description
User friendly construction sparse vectors, i.e., objects inheriting from class `sparseVector`, from indices and values of its non-zero entries.

Usage
sparseVector(x, i, length)

Arguments
- x: vector of the non-zero entries.
- i: integer vector (of the same length as x) specifying the indices of the non-zero (or non-TRUE) entries of the sparse vector.
- length: length of the sparse vector.

Details
zero entries in x are dropped automatically, analogously as `drop0()` acts on sparse matrices.

Value
a sparse vector, i.e., inheriting from class `sparseVector`.

Author(s)
Martin Maechler

See Also
- `sparseMatrix()` constructor for sparse matrices; the class `sparseVector`.

Examples
```r
str(sv <- sparseVector(x = 1:10, i = sample(999, 10), length=1000))

sx <- c(0,0,3, 3.2, 0,0,0,-3:1,0,0,2,0,0,5,0,0)
ss <- as(sx, "sparseVector")
stopifnot(identical(ss,
   sparseVector(x = c(2, -1, -2, 3, 1, -3, 5, 3.2),
     i = c(15L, 10:9, 3L,12L,8L,18L, 4L), length = 20L)))
```
sparseVector-class Sparse Vector Classes

Description

Sparse Vector Classes: The virtual mother class "sparseVector" has the five actual daughter classes "dsparseVector", "isparsesVector", "lsparseVector", "nsparseVector", and "zsparseVector", where we've mainly implemented methods for the ds, ls and ns ones.

Slots

length: class "numeric" - the length of the sparse vector. Note that "numeric" can be considerably larger than the maximal "integer", .Machine$integer.max, on purpose.

i: class "numeric" - the (1-based) indices of the non-zero entries. Must not be NA and strictly sorted increasingly.
   Note that "integer" is "part of" "numeric", and can (and often will) be used for non-huge sparseVectors.

x: (for all but "nsparseVector"): the non-zero entries. This is of class "numeric" for class "dsparseVector", "logical" for class "lsparseVector", etc.
   Note that "nsparseVector"s have no x slot. Further, mainly for ease of method definitions, we've defined the class union (see getClassUnion) of all sparse vector classes which have an x slot, as class "xsparseVector".

Methods

length signature(x = "sparseVector"): simply extracts the length slot.

show signature(object = "sparseVector"): The show method for sparse vectors prints "structural" zeroes as "." using the non-exported prSpVector function which allows further customization such as replacing "." by " " (blank).
   Note that options(max.print) will influence how many entries of large sparse vectors are printed at all.

as.vector signature(x = "sparseVector", mode = "character"): coerces sparse vectors to "regular", i.e., atomic vectors. This is the same as as(x, "vector").

as ..: see coerce below

coerce signature(from = "sparseVector", to = "sparseMatrix").and

coerce signature(from = "sparseMatrix", to = "sparseVector"), etc: coercions to and from sparse matrices (sparseMatrix) are provided and work analogously as in standard R, i.e., a vector is coerced to a 1-column matrix.

dim<- signature(x = "sparseVector", value = "integer") coerces a sparse vector to a sparse Matrix, i.e., an object inheriting from sparseMatrix, of the appropriate dimension.

head signature(x = "sparseVector"): as with R's (package util) head, head(x, n) (for n >= 1) is equivalent to x[1:n], but here can be much more efficient, see the example.

tail signature(x = "sparseVector"): analogous to head, see above.
sparseVector-class

**toeplitz** signature(x = "sparseVector"): as `toeplitz(x)`, produce the \( n \times n \) Toeplitz matrix from \( x \), where \( n = \text{length}(x) \).

**rep** signature(x = "sparseVector") repeat \( x \), with the same argument list \( (x, \text{times}, \text{length.out}, \text{each},...) \) as the default method for rep().

**Ops** signature(e1 = "sparseVector", e2 = ") define arithmetic, compare and logic operations, (see Ops).

**Summary** signature(x = "sparseVector"): define all the Summary methods.

[ signature(x = "atomicVector", i = ") not only can you subset (aka “index into”) sparseVectors \( x[i] \) using sparseVectors \( i \), but we also support efficient subsetting of traditional vectors \( x \) by logical sparse vectors (i.e., \( i \) of class "nsparseVector" or "lsparseVector").

**is.na**, **is.finite**, **is.infinite** (\( x \) = "sparseVector") where \( x \) is \( \text{NA} \) (or \( \text{NaN} \)), finite or infinite, entirely analogously to the corresponding base \( \text{R} \) functions.

**See Also**

sparseVector() for friendly construction of sparse vectors (apart from as(*, "sparseVector").

**Examples**

g(getClass("sparseVector")
glassClass("dsparseVector")
glassClass("xsparseVector")# those with an 'x' slot

sx <- c(0,0,3, 3.2, 0,0,0,-3:1,0,0,2,0,5,0,0)
(s2 <- as(sx, "sparseVector")

ix <- as.integer(round(sx))
(is <- as(ix, "sparseVector"))
## an "isparseVector" (!)

## rep() works too:
(ri <- rep(is, length.out= 25))

## Using `dim<` as in base R :
(ri <- ss
dim(ri) <- c(4,5) # becomes a sparse Matrix:
(ri
## or coercion (as as.matrix() in base R):
(as(ss, "Matrix")
stopifnot(all(ss == print(as(ss, "C sparseMatrix")))

## currently has "non-structural" FALSE -- printing as ":
(lis <- is & FALSE)
(nn <- is[is == 0]) # all "structural" FALSE

## NA-case
SN <- sx; SN[4] <- NA
(svN <- as(svN, "sparseVector")

}
v <- as(c(0,0,3, 3.2, rep(0,9),-3,0,-1, rep(0,20),5,0),  
"sparseVector")
v <- rep(rep(v, 50), 5000)  
set.seed(1); v[sample(0:10, 1e6)] <- 0  
str(v)

system.time(for(i in 1:4) hv <- head(v, 1e6))  
## user  system elapsed  
## 0.033 0.000 0.032  

system.time(for(i in 1:4) h2 <- v[1:1e6])  
## user  system elapsed  
## 1.317 0.000 1.319  

stopifnot(identical(hv, h2),  
  identical(is | FALSE, is != 0),  
  validObject(svN), validObject(lis), as.logical(is.na(svN[4])),  
  identical(is^2 > 0, is & TRUE),  
  all(!lis), !any(lis), length(nn@i) == 0, !any(nn), all(!nn),  
  sum(lis) == 0, !prod(lis), range(lis) == c(0,0))

## create and use the t(.) method:  
t(x20 <- sparseVector(c(0,3:1), i=c(1:2,4,7), length=20))  
(T20 <- toeplitz(x20))  
stopifnot(is[T20, "symmetricMatrix"], is(T20, "sparseMatrix"),  
  identical(unname(as.matrix(T20)),  
    toeplitz(as.vector(x20))))

---

**spMatrix**  
Sparse Matrix Constructor From Triplet

**Description**
User friendly construction of a sparse matrix (inheriting from class `TsparseMatrix`) from the triplet representation.

**Usage**

```r
spMatrix(nrow, ncol, i = integer(), j = integer(), x = numeric())
```

**Arguments**

- `nrow, ncol` integers specifying the desired number of rows and columns.
- `i,j` integer vectors of the same length specifying the locations of the non-zero (or non-TRUE) entries of the matrix.
- `x` atomic vector of the same length as `i` and `j`, specifying the values of the non-zero entries.
**spMatrix**

**Value**

A sparse matrix in triplet form, as an \texttt{R} object inheriting from both \texttt{TsparseMatrix} and \texttt{generalMatrix}.

The matrix \( M \) will have \( M[i[k], j[k]] = x[k] \), for \( k = 1, 2, \ldots, n \), where \( n = \text{length}(i) \) and \( M[i', j'] = 0 \) for all other pairs \((i', j')\).

**See Also**

\texttt{Matrix}(*, sparse=TRUE) for the more usual constructor of such matrices; similarly, \texttt{sparseMatrix} which is a bit more general than \texttt{spMatrix()} and returns a \texttt{CsparseMatrix} which is often slightly more desirable. Further, \texttt{bdiag} and \texttt{Diagonal} for (block-)diagonal matrix constructors.

Consider \texttt{TsparseMatrix} and similar class definition help files.

**Examples**

```r
## simple example
A <- spMatrix(10, 20, i = c(1,3:8),
               j = c(2,9,6:10),
               x = 7 * (1:7))
A # a "dgTMatrix"
summary(A)
str(A) # note that *internally* 0-based indices \((i,j)\) are used

L <- spMatrix(9, 30, i = rep(1:9, 3), 1:27,
              (1:27) %% 4 != 1)
L # an "lgTMatrix"

### This is a useful utility, to be used for experiments:

rSpMatrix <- function(nrow, ncol, nnz,
                      rand.x = function(n) round(rnorm(nnz), 2))
{
  ## Purpose: random sparse matrix
  ## ---------------------------------------------------------------
  ## Arguments: \( (nrow, ncol) \): dimension
  ## \( \text{nnz} \) : number of non-zero entries
  ## \( \text{rand.x} \): random number generator for 'x' slot
  ## ---------------------------------------------------------------
  ## Author: Martin Maechler, Date: 14.-16. May 2007
  stopifnot((nnz <= as.integer(nnz)) == 0,
            nrow >= 0, ncol >= 0,
            nnz <= nrow * ncol)
  spMatrix(nrow, ncol,
           i = sample(nrow, nnz, replace = TRUE),
           j = sample(ncol, nnz, replace = TRUE),
           x = rand.x(nnz))
}

M1 <- rSpMatrix(100000, 20, nnz = 200)
summary(M1)
```
symmetricMatrix-class

Virtual Class of Symmetric Matrices in package:Matrix

Description

The virtual class of symmetric matrices, "symmetricMatrix", from the package Matrix contains numeric and logical, dense and sparse matrices, e.g., see the examples.

The main use is in methods (and C functions) that can deal with all symmetric matrices.

Slots

uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.

Dim, Dimnames: The dimension (a length-2 "integer") and corresponding names (or NULL), inherited from the Matrix, see there.

factors: a list of matrix factorizations, also from the Matrix class.

Extends

Class "Matrix", directly.

Methods

There's a C function symmetricMatrix_validity() called by the internal validity checking functions.

See Also

isSymmetric which has efficient methods (isSymmetric-methods) for the Matrix classes. Classes triangularMatrix, and, e.g., dsyMatrix for numeric dense matrices, or lscMatrix for a logical sparse matrix class.

Examples

showClass("symmetricMatrix")

## The names of direct subclasses:
scl <- getClass("symmetricMatrix")@subclasses
directly <- sapply(lapply(scl, slot, "by"), length) == 0
names(scl)[directly]
Description

`symmpart(x)` computes the symmetric part \((x + t(x))/2\) and `skewpart(x)` the skew symmetric part \((x - t(x))/2\) of a square matrix \(x\), more efficiently for specific Matrix classes.

Note that \(x == symmpart(x) + skewpart(x)\) for all square matrices – apart from extraneous `NA` values in the RHS.

Usage

```r
symmpart(x)
skewpart(x)
```

Arguments

`x` a square matrix; either “traditional” of class "matrix", or typically, inheriting from the `Matrix` class.

Details

These are generic functions with several methods for different matrix classes, use e.g., `showMethods(symmpart)` to see them.

If the row and column names differ, the result will use the column names unless they are (partly) NULL where the row names are non-NULL (see also the examples).

Value

`symmpart()` returns a symmetric matrix, inheriting from `symmetricMatrix` iff \(x\) inherited from `Matrix`.

`skewpart()` returns a skew-symmetric matrix, typically of the same class as \(x\) (or the closest “general” one, see `generalMatrix`).

See Also

`isSymmetric`.

Examples

```r
m <- Matrix(1:4, 2,2)
symmpart(m)
skewpart(m)
stopifnot(all(m == symmpart(m) + skewpart(m)))
dn <- dimnames(m) <- list(row = c("r1", "r2"), col = c("var.1", "var.2"))```
triangularMatrix-class

Virtual Class of Triangular Matrices in package:Matrix

Description

The virtual class of triangular matrices, "triangularMatrix", from the package Matrix contains numeric and logical, dense and sparse matrices, e.g., see the examples.

The main use will be in methods (and C functions) that can deal with all triangular matrices.

Slots

- **uplo**: String (of class "character"). Must be either "U", for upper triangular, and "L", for lower triangular.
- **diag**: String (of class "character"). Must be either "U", for unit triangular (diagonal is all ones), or "N" for non-unit. The diagonal elements are not accessed internally when diag is "U". For denseMatrix classes, they need to be allocated though, i.e., the length of the x slot does not depend on diag.
- **Dim, Dimnames**: The dimension (a length-2 "integer") and corresponding names (or NULL), inherited from the Matrix, see there.

Extends

Class "Matrix", directly.

Methods

There’s a C function triangularMatrix_validity() called by the internal validity checking functions.

Currently, Schur, isSymmetric and as() (i.e. coerce) have methods with triangularMatrix in their signature.

See Also

istriangular() for testing any matrix for triangularity; classes symmetricMatrix, and, e.g., dtrMatrix for numeric dense matrices, or ltCMatrix for a logical sparse matrix subclass of "triangularMatrix".

```r
stopifnot(all(m == symmpart(m) + skewpart(m)))
colnames(m) <- NULL
colnames(m) <- NULL
colnames(m) <- NULL
colnames(m) <- NULL
stopifnot(all(m == symmpart(m) + skewpart(m)))
stopifnot(all(m == symmpart(m) + skewpart(m)))
stopifnot(all(m == symmpart(m) + skewpart(m)))
```

---

```
## investigate the current methods:
showMethods(skewpart, include = TRUE)
```
Examples

showClass("triangularMatrix")

## The names of direct subclasses:
scl <- getClass("triangularMatrix")@subclasses
directly <- sapply(lapply(scl, slot, "by"), length) == 0
names(scl)[directly]

(m <- matrix(c(5,1,0,3), 2))
as(m, "triangularMatrix")

Class "TsparseMatrix": Class "TsparseMatrix" of Sparse Matrices in Triplet Form

Description

The "TsparseMatrix" class is the virtual class of all sparse matrices coded in triplet form. Since it is a virtual class, no objects may be created from it. See showClass("TsparseMatrix") for its subclasses.

Slots

dim, Dimnames: from the "Matrix" class,

i: Object of class "integer" - the row indices of non-zero entries in 0-base, i.e., must be in 0:(nrow(.)-1).

j: Object of class "integer" - the column indices of non-zero entries. Must be the same length as slot i and 0-based as well, i.e., in 0:(ncol(.)-1). For numeric Tsparse matrices, (i,j) pairs can occur more than once, see dgTMatrix.

Extends

Class "sparseMatrix", directly. Class "Matrix", by class "sparseMatrix".

Methods

Extraction ("[" methods, see [-methods.

Note

Most operations with sparse matrices are performed using the compressed, column-oriented or CsparseMatrix representation. The triplet representation is convenient for creating a sparse matrix or for reading and writing such matrices. Once it is created, however, the matrix is generally coerced to a CsparseMatrix for further operations.

Note that all new(.), spMatrix and sparseMatrix(*, giveCsparse=FALSE) constructors for "TsparseMatrix" classes implicitly add $x_k$'s that belong to identical $(i_k,j_k)$ pairs, see, the example below, or also "dgTMatrix".
For convenience, methods for some operations such as `%*%` and `crossprod` are defined for `TsparseMatrix` objects. These methods simply coerce the `TsparseMatrix` object to a `CsparseMatrix` object then perform the operation.

**See Also**

its superclass, `sparseMatrix`, and the `dgTMatrix` class, for the links to other classes.

**Examples**

```r
showClass("TsparseMatrix")
## or just the subclasses' names
names(getClass("TsparseMatrix")@subclasses)

T3 <- spMatrix(3, 4, i=c(1,3:1), j=c(2,4:2), x=1:4)
T3 # only 3 non-zero entries, 5 = 1+4 !
```

---

### Description

"Packed" matrix storage here applies to dense matrices (`denseMatrix`) only, and there is available only for symmetric (`symmetricMatrix`) or triangular (`triangularMatrix`) matrices, where only one triangle of the matrix needs to be stored.

`unpack()` unpacks "packed" matrices, where `pack()` produces "packed" matrices.

**Usage**

```r
pack(x, ...)
## S4 method for signature 'matrix'
pack(x, symmetric = NA, upperTri = NA, ...)

unpack(x, ...)
```

**Arguments**

- **x**
  - for `unpack()`: a matrix stored in packed form, e.g., of class "d?pMatrix" where "?" is "t" for triangular or "s" for symmetric.
  - for `pack()`: a (symmetric or triangular) matrix stored in full storage.

- **symmetric**
  - logical (including `NA`) for optionally specifying if `x` is symmetric (or rather triangular).

- **upperTri**
  - (for the triangular case only) logical (incl. `NA`) indicating if `x` is upper (or lower) triangular.

- **...**
  - further arguments passed to or from other methods.
Unused-classes

Details

These are generic functions with special methods for different types of packed (or non-packed) symmetric or triangular dense matrices. Use `showMethods("unpack")` to list the methods for `unpack()`, and similarly for `pack()`.

Value

- for `unpack()`: A `Matrix` object containing the full-storage representation of x.
- for `pack()`: A packed `Matrix` (i.e. of class ".pMatrix") representation of x.

Examples

```r
showMethods("unpack")
(cp4 <- chol(Hilbert(4))) # is triangular
tp4 <- as(cp4,"dtMatrix")# [t]riangular [p]acked
str(tp4)
(unpack(tp4))
stopifnot(identical(tp4, pack(unpack(tp4)))))

(s <- crossprod(matrix(sample(15), 5, 3))) # traditional symmetric matrix
(sp <- pack(s))
(mt <- as.matrix(tt <- tril(s))
(pt <- pack(mt))
stopifnot(identical(pt, pack(tt)),
  dim(s) == dim(sp), all(s == sp),
  dim(mt) == dim(pt), all(mt == pt), all(mt == tt))
showMethods("pack")
```

Unused-classes

<table>
<thead>
<tr>
<th>Virtual Classes Not Yet Really Implemented and Used</th>
</tr>
</thead>
</table>

Description

- `iMatrix` is the virtual class of all integer (S4) matrices. It extends the `Matrix` class directly.
- `zMatrix` is the virtual class of all `complex` (S4) matrices. It extends the `Matrix` class directly.

Examples

```r
showClass("iMatrix")
showClass("zMatrix")
```
Description

Compute the up- or down-dated Cholesky decomposition

Usage

updown(update, C, L)

Arguments

update

logical (TRUE or FALSE) or "+" or "-" indicating if an up- or a down-date is to be computed.

C

d

any R object, coercable to a sparse matrix (i.e., of subclass of sparseMatrix).

L

a Cholesky factor, specifically, of class "CHMfactor".

Value

an updated Cholesky factor, of the same dimension as L. Typically of class "dCHMsimpl" (a subclass of "CHMfactor").

Methods

signature(update = "character", C = "mMatrix", L = "CHMfactor") ..

signature(update = "logical", C = "mMatrix", L = "CHMfactor") ..

Author(s)

Contributed by Nicholas Nagle, University of Tennessee, Knoxville, USA

References

CHOLMOD manual, currently beginning of chapter~18. ...

See Also

Cholesky,
Examples

dn <- list(LETTERS[1:3], letters[1:5])
## pointer vectors can be used, and the (i,x) slots are sorted if necessary:
m <- sparseMatrix(i = c(3,1, 3:2, 2:1), p= c(0:2, 4,4,6), x = 1:6, dimnames = dn)
cA <- Cholesky(A <- crossprod(m) + Diagonal(5))
166 * as(cA,"Matrix") * 2
uc1 <- updown(“+", Diagonal(5), cA)
## Hmm: this loses positive definiteness:
uc2 <- updown("-", 2*Diagonal(5), cA)
image(show(as(cA, "Matrix")))
image(show(c2 <- as(uc2,"Matrix")))# severely negative entries
##--> Warning

USCounties
USCounties Contiguity Matrix

Description

This matrix represents the contiguities of 3111 US counties using the Queen criterion of at least a single shared boundary point. The representation is as a row standardised spatial weights matrix transformed to a symmetric matrix (see Ord (1975), p. 125).

Usage

data(USCounties)

Format

A $3111^2$ symmetric sparse matrix of class dsMatrix with 9101 non-zero entries.

Details

The data were read into \( \mathbb{R} \) using read.gal, and row-standardised and transformed to symmetry using nb2listw and similar.listw. This spatial weights object was converted to class dsCMatrix using as_dsTMatrix_listw and coercion.

Source

The data were retrieved from http://sal.uiuc.edu/weights/zips/usc.zip, files “usc.txt” and “usc\_q.GAL”, with permission for use and distribution from Luc Anselin.

References

Examples

data(USCounties)
(n <- ncol(USCounties))
IM <- .symDiagonal(n)
nn <- 50
set.seed(1)
rho <- runif(nn, 0, 1)
system.time(MJ <- sapply(rho, function(x)
determinant(IM - x * USCounties, logarithm = TRUE)$modulus))

## can be done faster, by update()ing the Cholesky factor:
nWC <- USCounties
C1 <- Cholesky(nWC, Imult = 2)
system.time(MJ1 <- n * log(rho) +
sapply(rho, function(x)
2 * c(determinant(update(C1, nWC, 1/x)$modulus))))
all.equal(MJ, MJ1)

C2 <- Cholesky(nWC, super = TRUE, Imult = 2)
system.time(MJ2 <- n * log(rho) +
sapply(rho, function(x)
2 * c(determinant(update(C2, nWC, 1/x)$modulus))))
all.equal(MJ, MJ2)

system.time(MJ3 <- n * log(rho) + Matrix:::ldetL2up(C1, nWC, 1/rho))
stopifnot(all.equal(MJ, MJ3))
system.time(MJ4 <- n * log(rho) + Matrix:::ldetL2up(C2, nWC, 1/rho))
stopifnot(all.equal(MJ, MJ4))

Description

Methods for "[", i.e., extraction or subsetting mostly of matrices, in package Matrix.

Methods

There are more than these:

x = "Matrix", i = "missing", j = "missing", drop= "ANY" ...

x = "Matrix", i = "numeric", j = "missing", drop= "missing" ...

x = "Matrix", i = "missing", j = "numeric", drop= "missing" ...

x = "dsparseMatrix", i = "missing", j = "numeric", drop= "logical" ...

x = "dsparseMatrix", i = "numeric", j = "missing", drop= "logical" ...

x = "dsparseMatrix", i = "numeric", j = "numeric", drop= "logical" ...
Methods for "[<-", i.e., extraction or subsetting mostly of matrices, in package Matrix.

Note: Contrary to standard matrix assignment in base R, in x[...] <- val it is typically an error (see stop) when the type or class of val would require the class of x to be changed, e.g., when x is logical, say lsparseMatrix, and val is numeric. In other cases, e.g., when x is a "nsparseMatrix" and val is not TRUE or FALSE, a warning is signalled, and val is “interpreted” as logical, and (logical) NA is interpreted as TRUE.

Methods

There are many many more than these:

x = "Matrix", i = "missing", j = "missing", value = "ANY"  is currently a simple fallback method implementation which ensures “readable” error messages.

x = "Matrix", i = "ANY", j = "ANY", value = "ANY"  currently gives an error

x = "denseMatrix", i = "index", j = "missing", value = "numeric" ...

x = "denseMatrix", i = "index", j = "index", value = "numeric" ...

See Also

[<-methods for subassignment to "Matrix" objects; the index class; Extract about the standard subset assignment (and extraction).
Examples

```r
set.seed(101)
(a <- m <- Matrix(round(rnorm(7*4),2), nrow = 7))

a[] <- 2.2 # <<< replaces **every** entry
a
## as do these:
a[,] <- 3 ; a[TRUE,] <- 4

m[2,3] <- 3.14 # simple number
m[3,3:4] <- 3:4 # simple numeric of length 2

## sub matrix assignment:
m[-(4:7), 3:4] <- cbind(1,2:4) # upper right corner of 'm'
m[3:5, 2:3] <- 0
m[6:7, 1:2] <- Diagonal(2)
m

## rows or columns only:
m[1,] <- 10
m[,2] <- 1:7
m[-(1:6), ] <- 3:0 # not the first 6 rows, i.e. only the 7th
as(m, "sparseMatrix")
```
Index

!,Matrix-method (Matrix-class), 92
!,ldenseMatrix-method
   (ldenseMatrix-class), 81
!,ldiMatrix-method (ldiMatrix-class), 82
!,lgeMatrix-method (lgeMatrix-class), 83
!,lsparseMatrix-method
   (lsparseMatrix-classes), 84
!,ltrMatrix-method (ltrMatrix-class), 86
!,ldenseMatrix-method (ldenseMatrix-class), 86
!,ldimatrixM-method
   (ldimatrixM-class), 82
!,lsparseMatrix-method
   (lsparseMatrix-classes), 84
!,ltrMatrix-method (ltrMatrix-class), 86
!,ldenseMatrix-method (ldenseMatrix-class), 86
!,nMatrix-method
   (nMatrix-class), 101
!,ngeMatrix-method (ngeMatrix-class), 101
!,nsparseMatrix-method
   (nsparseMatrix-classes), 105
!,ntpMatrix-method (ntpMatrix-class), 108
!,ltrMatrix-method (ltrMatrix-class), 108
!,sparseVector-method
   (sparseVector-class), 142

*Topic IO
   externalFormats, 60

*Topic algebra
   band, 9
   bandSparse, 10
   chol, 19
   chol2inv-methods, 21
   Cholesky, 22
   Cholesky-class, 24
   colSums, 26
   dgCMatrix-class, 34
   dgeMatrix-class, 35
   dgrMatrix-class, 36
   dtMMatrix-class, 37
   Diagonal, 39
   dMatrix-class, 43
   dpoMatrix-class, 44
   dsCMatrix-class, 47
   dsRMatrix-class, 50
   dtCMatrix-class, 52
   dtRMatrix-class, 56
   expand, 58
   expm, 59
   externalFormats, 60
   facmul, 62
   Hilbert, 67
   lsparseMatrix-classes, 84
   lu, 87
   LU-class, 89
   Matrix, 90
   Matrix-class, 92
   matrix-products, 94
   nearPD, 98
   nMatrix-class, 102
   norm, 104
   nsparseMatrix-classes, 105
   qr-methods, 114
   rankMatrix, 115
   rcond, 118
   Schur, 124
   sparseQR-class, 139
   unpack, 150

*Topic arithmetic
   invPerm, 73

*Topic arith
   all.equal-methods, 7
   colSums, 26
   symmpart, 147

*Topic array
   [-methods, 154
   [<-methods, 155
   bandSparse, 10
   bdiag, 11
   cBind, 15
   chol, 19
   Cholesky, 22
colSums, 26
Diagonal, 39
drop0, 46
externalFormats, 60
facmul, 62
forceSymmetric, 63
Hilbert, 67
KhatriRao, 78
kronecker-methods, 80
lu, 87
Matrix, 90
nearPD, 98
qr-methods, 114
rankMatrix, 115
rcond, 118
rsparseMatrix, 122
spMatrix, 144
symmpart, 147
unpack, 150

*Topic attribute
nnzero, 103

*Topic classes
abIndex-class, 4
abSeq, 6
atomicVector-class, 8
CHMfactor-class, 16
Cholesky-class, 24
compMatrix-class, 27
CsparseMatrix-class, 30
denseMatrix-class, 31
ddiMatrix-class, 32
denseMatrix-class, 33
dgCMatrix-class, 34
dgeMatrix-class, 35
dgRMatrix-class, 36
dgTMatrix-class, 37
diagonalMatrix-class, 40
diagU2N, 42
dMatrix-class, 43
dpoMatrix-class, 44
dscMatrix-class, 47
dspMatrix-class, 49
dRMatrix-class, 50
dsyMatrix-class, 51
dtCMatrix-class, 52
dtpMatrix-class, 54
dtrMatrix-class, 56
dtrRMatrix-class, 57
generalMatrix-class, 65
index-class, 70
indMatrix-class, 70
ldenseMatrix-class, 81
ldiMatrix-class, 82
lgeMatrix-class, 83
lsparseMatrix-classes, 84
lsyMatrix-class, 85
ltrMatrix-class, 86
LU-class, 89
Matrix-class, 92
MatrixFactorization-class, 96
ndenseMatrix-class, 97
ngeMatrix-class, 101
nMatrix-class, 102
nsparseMatrix-classes, 105
nsyMatrix-class, 107
ntrMatrix-class, 108
number-class, 109
pMatrix-class, 110
replValue-class, 121
rleDiff-class, 121
RsparseMatrix-class, 123
Schur-class, 125
sparseLU-class, 132
sparseMatrix-class, 137
sparseQR-class, 139
sparseVector-class, 142
symmetricMatrix-class, 146
triangularMatrix-class, 148
TsparseMatrix-class, 149
Unused-classes, 151

*Topic datasets
CAex, 14
KNex, 79
USCounties, 153

*Topic distribution
rsparseMatrix, 122

*Topic graph
graph-sparseMatrix, 66

*Topic hplot
image-methods, 68

*Topic manip
abSeq, 6
cBind, 15
rep2ab1, 120
*Topic math expm, 59
*Topic methods
[-methods, 154
[<--methods, 155
all-methods, 7
all.equal-methods, 7
band, 9
BunchKaufman-methods, 13
chol2inv-methods, 21
image-methods, 68
is.na-methods, 74
isSymmetric-methods, 76
isTriangular, 77
KhatriRao, 78
kronecker-methods, 80
matrix-products, 94
qr-methods, 114
solve-methods, 126
SparseM-conversions, 133
updown, 152
*Topic models
sparse.model.matrix, 129
*Topic print
formatSparseM, 64
printSpMatrix, 112
*Topic utilities
diagU2N, 42
drop0, 46
formatSparseM, 64
graph-sparseMatrix, 66
is.null.DN, 75
*.Matrix, ddiMatrix-method
(diagonalMatrix-class), 40
*.Matrix, ldiMatrix-method
(diagonalMatrix-class), 40
*,ldenseMatrix, ddiMatrix-method
(diagonalMatrix-class), 40
*,ldenseMatrix, ldiMatrix-method
(diagonalMatrix-class), 40
*,ldiMatrix, Matrix-method
(diagonalMatrix-class), 40
*,ldiMatrix, ddiMatrix-method
(diagonalMatrix-class), 40
*,ldiMatrix, ldiMatrix-method
(diagonalMatrix-class), 40
*,ldiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
*,ndenseMatrix, ddiMatrix-method
(diagonalMatrix-class), 40
*,ndenseMatrix, ldiMatrix-method
(diagonalMatrix-class), 40
*,Matrix, missing-method (Matrix-class), 92
+,dgTMatrix, dgTMatrix-method
(dgTMatrix-class), 37
-,Matrix, missing-method (Matrix-class), 92
-,ldiMatrix, missing-method
(diagonalMatrix-class), 40
-,denseMatrix, missing-method
(denseMatrix-class), 33
-,dsparseVector, missing-method
(sparseVector-class), 142
-,indMatrix, missing-method
(indMatrix-class), 70
-,ldiMatrix, missing-method
(diagonalMatrix-class), 40
-,lspareMatrix, missing-method
(lspareMatrix-classes), 84
-,nsparseMatrix, missing-method
(nsparseMatrix-classes), 105
-,pMatrix, missing-method
(pMatrix-class), 110
-,sparseMatrix, missing-method
(sparseMatrix-class), 137
.NMachine, 116, 142
SuiteSparse_version (Cholesky), 22
.bdiag (bdiag), 11
.diagU2N (diagU2N), 42
dsy2mat (dsyMatrix-class), 51
.formatSparseSimple, 113
.formatSparseSimple (formatSparseM), 64
Nsparsediagonal (Diagonal), 39
Nsymdiagonal (Diagonal), 39
updateCHMfactor (CHMfactor-class), 16
validateCsparse (CsparseMatrix-class), 30
\[,\text{ddiMatrix}, \text{Matrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ddiMatrix}, \text{ddenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ddiMatrix}, \text{ldenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ddiMatrix}, \text{ndenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ldiMatrix}, \text{Matrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ldiMatrix}, \text{ddenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ldiMatrix}, \text{ldenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{ldiMatrix}, \text{ndenseMatrix-method}
\text{(diagonalMatrix-class), 40}
\[,\text{CsparseMatrix}, \text{index, index, logical-method}
\text{([-methods), 154}}
\[,\text{CsparseMatrix}, \text{index, missing, logical-method}
\text{([-methods), 154}}
\[,\text{CsparseMatrix}, \text{missing, index, logical-method}
\text{([-methods), 154}}
\[,\text{CsparseMatrix}, \text{missing, missing, logical-method}
\text{([-methods), 154}}
\[,\text{abIndex}, \text{index, ANY, ANY-method}
\text{(abIndex-class), 4}}
\[,\text{dMatrix}, \text{lMatrix, missing, ANY-method}
\text{(dMatrix-class), 43}}
\[,\text{dMatrix}, \text{logical, missing, ANY-method}
\text{(dMatrix-class), 43}}
\[,\text{denseMatrix}, \text{index, index, logical-method}
\text{([-methods), 154}}
\[,\text{denseMatrix}, \text{index, missing, logical-method}
\text{([-methods), 154}}
\[,\text{denseMatrix}, \text{matrix, missing, ANY-method}
\text{([-methods), 154}}
\[,\text{denseMatrix}, \text{matrix, missing, missing-method}
\text{([-methods), 154}}
\[,\text{denseVector}, \text{index, ANY, ANY-method}
\text{(sparseVector-class), 142}}
\[,\text{denseVector}, \text{lsparseVector, ANY, ANY-method}
\text{(sparseVector-class), 142}}
\[,\text{denseVector}, \text{nsparseVector, ANY, ANY-method}
\text{(sparseVector-class), 142}}
\text{([-methods, 154}}
\text{([-methods), 154}}
\text{([-methods), 154}}
\text{([-methods), 154}}
\text{([-methods), 154}}
\text{([-methods), 154}}
\text{([-methods), 154}}
INDEX

[[-,diagonalMatrix, index, missing, sparseMatrix-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, index, missing, sparseVector-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, missing, replValue-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, missing, index, replValue-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, missing, index, sparseMatrix-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, missing, index, sparseVector-method]
  [[<-methods), 155 (matrix-products), 94
[[-,diagonalMatrix, missing, missing, ANY-method]
  [[<-methods), 155 (matrix-products), 94
[[-,indMatrix, index, ANY, ANY-method]
  [[<-methods), 155 (matrix-products), 94
[[-,indMatrix, missing, index, ANY-method]
  [[<-methods), 155 (matrix-products), 94
[[-,indMatrix, missing, missing, ANY-method]
  [[<-methods), 155 (matrix-products), 94
[[-,sparseMatrix, ANY, ANY, sparseMatrix-method]
  [[<-methods), 155 (matrix-products), 94
[[-,sparseMatrix, ANY, missing, sparseMatrix-method]
  [[<-methods), 155 (matrix-products), 94
[[-,sparseMatrix, missing, ANY, sparseMatrix-method]
  [[<-methods), 155 (matrix-products), 94
[[-,sparseMatrix, missing, missing, ANY-method]
  [[<-methods), 155 (matrix-products), 94
[[-,sparseVector, index, missing, replValue-method]
  [[sparseVector-class], 142 (matrix-products), 94
[[-,sparseVector, sparseVector, missing, replValue-method]
  [[sparseVector-class], 142 (matrix-products), 94

[<-methods, 155 (matrix-products), 94
  **%, ddenseMatrix-method
  (matrix-products), 94
  **%, ANY, Matrix-method
  (matrix-products), 94
  **%, ANY, TsparseMatrix-method
  (matrix-products), 94
  **%, CsparseMatrix, CsparseMatrix-method
  (matrix-products), 94
  **%, CsparseMatrix, ddenseMatrix-method
  (matrix-products), 94
  **%, CsparseMatrix, diagonalMatrix-method
  (matrix-products), 94
  **%, CsparseMatrix, matrix-method
  (matrix-products), 94
  **%, CsparseMatrix, numLike-method
  (matrix-products), 94
  **%, dmmatrix-method
  (matrix-products), 94
  **%, dgeMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, diagonalMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, dgeMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, dtrMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, sparseMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, sparseVector-method
  (matrix-products), 94
  **%, dgeMatrix, dgeMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, dgeMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, dpMatrix-method
  (matrix-products), 94
  **%, dgeMatrix, matrix-method
  (matrix-products), 94
  **%, dgeMatrix, numLike-method
  (matrix-products), 94
%%, diagonalMatrix, CsparseMatrix-method
(matrix-products), 94
%%, diagonalMatrix, denseMatrix-method
(matrix-products), 94
%%, diagonalMatrix, dgeMatrix-method
(matrix-products), 94
%%, diagonalMatrix, diagonalMatrix-method
(matrix-products), 94
%%, diagonalMatrix, lgeMatrix-method
(matrix-products), 94
%%, diagonalMatrix, matrix-method
(matrix-products), 94
%%, dgeMatrix, lgeMatrix-method
(matrix-products), 94
%%, dgeMatrix, diagonalMatrix-method
(matrix-products), 94
%%, dgeMatrix, matrix-method
(matrix-products), 94
%%, dmmatrix, sparseMatrix-method
(matrix-products), 94
%%, dmmatrix, sparseMatrix-method
(matrix-products), 94
%%, ndenseMatrix, nsparseMatrix-method
(matrix-products), 94
%%, nMatrix, dMatrix-method
(matrix-products), 94
%%, nMatrix, lMatrix-method
(matrix-products), 94
%%, nMatrix, nMatrix-method
(matrix-products), 94
%%, integer, dMatrix-method
(matrix-products), 94
%%, lMatrix, dMatrix-method
(matrix-products), 94
%%, lMatrix, lMatrix-method
(matrix-products), 94
%%, ldenseMatrix, lMatrix-method
(matrix-products), 94
%%, ldenseMatrix, nMatrix-method
(matrix-products), 94
%%, lgeMatrix, diagonalMatrix-method
(matrix-products), 94
%%, lgeMatrix, diagonalMatrix-method
(matrix-products), 94
%%, lsparseMatrix, ldenseMatrix-method
(matrix-products), 94
%%, lsparseMatrix, lMatrix-method
(matrix-products), 94
%%, mmatrix, sparseVector-method
(matrix-products), 94
%%, mmatrix, sparseVector-method
(matrix-products), 94
%%, mMatrix, sparseVector-method
(matrix-products), 94
%%, matrix, CsparseMatrix-method
(matrix-products), 94
%%, matrix, CsparseMatrix-method
(matrix-products), 94
%%, matrix, Matrix-method
(matrix-products), 94
%%, matrix, dgeMatrix-method
(matrix-products), 94
%%, matrix, dtMatrix-method
(matrix-products), 94
%%, matrix, indMatrix-method
(matrix-products), 94
%%, matrix, pMatrix-method
(matrix-products), 94
%%, nMatrix, dMatrix-method
(matrix-products), 94
%%, nMatrix, lMatrix-method
(matrix-products), 94
%%, nMatrix, nMatrix-method
(matrix-products), 94
%%, ndenseMatrix, nsparseMatrix-method
(matrix-products), 94
%%, ngCMatrix, ngCMatrix-method
(matrix-products), 94
%%, nsparseMatrix, ndenseMatrix-method
(matrix-products), 94
%%, nsparseMatrix, nMatrix-method
(matrix-products), 94
%%, nsparseMatrix, nMatrix-method
(matrix-products), 94
%%, nsparsesparseMatrix, nMatrix-method
(matrix-products), 94
%%, nsparsesparseMatrix, nMatrix-method
(matrix-products), 94
%%, sparseMatrix, ndenseMatrix-method
(matrix-products), 94
INDEX

%%, numLike, CsparseMatrix-method
(matrix-products), 94
%%, numLike, Matrix-method
(matrix-products), 94
%%, numLike, dgeMatrix-method
(matrix-products), 94
%%, numLike, sparseVector-method
(matrix-products), 94
%%, pMatrix, Matrix-method
(matrix-products), 94
%%, pMatrix, matrix-method
(matrix-products), 94
%%, pMatrix, pMatrix-method
(matrix-products), 94
%%, sparseMatrix, diagonalMatrix-method
(matrix-products), 94
%%, sparseVector, mMatrix-method
(matrix-products), 94
%%, sparseVector, numLike-method
(matrix-products), 94
%%, sparseVector, sparseVector-method
(matrix-products), 94
%%-methods (matrix-products), 94
%%, ddiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ddiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, Matrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ddenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ldenseMatrix-method
(diagonalMatrix-class), 40
%%, ldiMatrix, ndenseMatrix-method
(diagonalMatrix-class), 40

%%, 30, 35, 95
^, Matrix, ddiMatrix-method
(diagonalMatrix-class), 40
^, Matrix, ldiMatrix-method
(diagonalMatrix-class), 40
^, ddenseMatrix, ddiMatrix-method
(diagonalMatrix-class), 40
^, ddenseMatrix, ldiMatrix-method
(diagonalMatrix-class), 40
^,/ddiMatrix,Matrix-method (diagonalMatrix-class), 40
^,/ddiMatrix,ddenseMatrix-method (diagonalMatrix-class), 40
^,/ddiMatrix,ldenseMatrix-method (diagonalMatrix-class), 40
^,/ddiMatrix,ndenseMatrix-method (diagonalMatrix-class), 40
^,ldenseMatrix,ddiMatrix-method (diagonalMatrix-class), 40
^,ldenseMatrix,ldiMatrix-method (diagonalMatrix-class), 40
^,ldiMatrix,ddiMatrix-method (diagonalMatrix-class), 40
^,ldiMatrix,ldenseMatrix-method (diagonalMatrix-class), 40
^,ldiMatrix,ndenseMatrix-method (diagonalMatrix-class), 40
^,ndenseMatrix,ddiMatrix-method (diagonalMatrix-class), 40
^,ndenseMatrix,ldiMatrix-method (diagonalMatrix-class), 40
all.equal,ANY,Matrix-method (all.equal-methods), 7
all.equal,ANY,sparseMatrix-method (all.equal-methods), 7
all.equal,ANY,sparseVector-method (all.equal-methods), 7
all.equal,Matrix,ANY-method (all.equal-methods), 7
all.equal,Matrix,Matrix-method (all.equal-methods), 7
all.equal,ndenseMatrix,ldiMatrix-method (all.equal-methods), 7
all.equal,abIndex-method (all.equal-methods), 7
all.equal,abIndex-numLike-method (all.equal-methods), 7
abbreviate, 113
abIndex, 6, 120, 122
abIndex-class, 4
abIseq, 4, 6, 120
abIseq1 (abIseq), 6
abs, 68
all, 7, 44
all,ddiMatrix-method (diagonalMatrix-class), 40
all,ldenseMatrix-method (all-methods), 7
all,ldiMatrix-method (diagonalMatrix-class), 40
all,lsparseMatrix-method (all-methods), 7
all,lsyMatrix-method (all-methods), 7
all,Matrix-method (all-methods), 7
all,nsparseMatrix-method (nsparseMatrix-classes), 105
all-methods, 7
all.equal, 7
all.equal,abIndex,abIndex-method (all.equal-methods), 7
all.equal,abIndex,numLike-method (all.equal-methods), 7
all.equal,abIndex-method (all.equal-methods), 7
all.equal,abIndex-numLike-method (all.equal-methods), 7
any, 7, 44
any,ddiMatrix-method (diagonalMatrix-class), 40
any,ldiMatrix-method (diagonalMatrix-class), 40
any,Matrix-method (all-methods), 7
any,nsparseMatrix-method (nsparseMatrix-classes), 105
anyNA, dMatrix-method (is.na-methods), 74
anyNA, iMatrix-method (is.na-methods), 74
anyNA, lMatrix-method (is.na-methods), 74
anyNA, ndenseMatrix-method (is.na-methods), 74
anyNA, nsparseMatrix-method (is.na-methods), 74
anyNA, nsparseVector-method (is.na-methods), 74
anyNA, sparseVector-method (is.na-methods), 74
anyNA, xMatrix-method (is.na-methods), 74
anyNA, zMatrix-method (is.na-methods), 74
apply, 93
Arith, 35, 43
Arith, abIndex, abIndex-method (abIndex-class), 4
Arith, abIndex, numLike-method (abIndex-class), 4
Arith, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class), 30
Arith, CsparseMatrix, numeric-method (CsparseMatrix-class), 30
Arith, ddenseMatrix, ddenseMatrix-method (ddenseMatrix-class), 31
Arith, ddenseMatrix, logical-method (ddenseMatrix-class), 31
Arith, ddenseMatrix, numeric-method (ddenseMatrix-class), 31
Arith, ddenseMatrix, sparseVector-method (sparseVector-class), 142
Arith, ddiMatrix, logical-method (diagonalMatrix-class), 40
Arith, ddiMatrix, numeric-method (diagonalMatrix-class), 40
Arith, dgCMatrix, dgCMatrix-method (dgCMatrix-class), 34
Arith, dgCMatrix, logical-method (dgCMatrix-class), 34
Arith, dgCMatrix, numeric-method (dgCMatrix-class), 34
Arith, dgeMatrix, dgeMatrix-method (dgeMatrix-class), 35
Arith, dgeMatrix, logical-method (dgeMatrix-class), 35
Arith, dgeMatrix, numeric-method (dgeMatrix-class), 35
Arith, dgeMatrix, sparseVector-method (sparseVector-class), 142
Arith, dMatrix, dMatrix-method (dMatrix-class), 43
Arith, dpoMatrix, logical-method (dpoMatrix-class), 44
Arith, dpoMatrix, numeric-method (dpoMatrix-class), 44
Arith, dppMatrix, logical-method (dppMatrix-class), 44
Arith, dppMatrix, numeric-method (dppMatrix-class), 44
Arith, dsCMatrix, dsCMatrix-method (dsCMatrix-class), 47
Arith, dsparseMatrix, logical-method (dsparseMatrix-class), 49
Arith, dsparseMatrix, numeric-method (dsparseMatrix-class), 49
Arith, dsparseVector, dsparseVector-method (sparseVector-class), 142
Arith, dtCMatrix, dtCMatrix-method (dtCMatrix-class), 52
Arith, ldiMatrix, logical-method (diagonalMatrix-class), 40
Arith, ldiMatrix, numeric-method (diagonalMatrix-class), 40
Arith, lgCMatrix, lgCMatrix-method (lsparseMatrix-classes), 84
Arith, lgeMatrix, lgeMatrix-method (lgeMatrix-class), 83
Arith, lgTMatrix, lgTMatrix-method (lsparseMatrix-classes), 84
Arith, lMatrix, logical-method (dMatrix-class), 43
Arith, lMatrix, numeric-method (dMatrix-class), 43
Arith, logical, ddenseMatrix-method (ddenseMatrix-class), 31
Arith, logical, ddiMatrix-method (diagonalMatrix-class), 40
Arith, logical, dgCMatrix-method (dgCMatrix-class), 34
Arith, logical, dgeMatrix-method (dgeMatrix-class), 35
Arith, logical, dpoMatrix-method (dpoMatrix-class), 44
Arith, logical, dppMatrix-method (dppMatrix-class), 44
Arith, logical, dsparseMatrix-method (dsparseMatrix-class), 49
Arith, logical, ldiMatrix-method (diagonalMatrix-class), 40
Arith, logical, lMatrix-method (dMatrix-class), 43
Arith, logical, nMatrix-method (nMatrix-class), 102
Arith, lsparseMatrix, Matrix-method (lsparseMatrix-classes), 84
Arith, Matrix, lsparseMatrix-method (lsparseMatrix-classes), 84
Arith, Matrix, Matrix-method (Matrix-class), 92
attributes, 77
backsolve, 128
band, 9, 10, 11, 40
band, CsparseMatrix-method (band), 9
band, ddenseMatrix-method (band), 9
band, denseMatrix-method (band), 9
band, matrix-method (band), 9
band, RsparseMatrix-method (band), 9
band-methods (band), 9
bandSparse, 9, 10, 12, 40, 91, 135
bdiag, 11, 11, 91, 135, 145
BunchKaufman (BunchKaufman-methods), 13
BunchKaufman, dspMatrix-method
  (BunchKaufman-methods), 13
BunchKaufman, dsyMatrix-method
  (BunchKaufman-methods), 13
BunchKaufman-class (Cholesky-class), 24
BunchKaufman-methods, 13
c, 4
c.abIndex (abIsEq), 6
CAex, 14
cBind, 15, 138
cbind, 15, 138
cbind2, ANY, Matrix-method
  (Matrix-class), 92
cbind2, atomicVector, ddiMatrix-method
  (diagonalMatrix-class), 40
cbind2, atomicVector, ldiMatrix-method
  (diagonalMatrix-class), 40
cbind2, atomicVector, Matrix-method
  (Matrix-class), 92
cbind2, ddiMatrix, atomicVector-method
  (diagonalMatrix-class), 40
cbind2, ddiMatrix, matrix-method
  (diagonalMatrix-class), 40
cbind2, denseMatrix, denseMatrix-method
  (denseMatrix-class), 33
cbind2, denseMatrix, matrix-method
  (denseMatrix-class), 33
cbind2, denseMatrix, numeric-method
  (denseMatrix-class), 33
cbind2, denseMatrix, sparseMatrix-method
  (sparseMatrix-class), 137
cbind2, diagonalMatrix, sparseMatrix-method
  (diagonalMatrix-class), 40
cbind2, ldiMatrix, atomicVector-method
  (diagonalMatrix-class), 40
cbind2, ldiMatrix, matrix-method
  (diagonalMatrix-class), 40
cbind2, Matrix, ANY-method
  (Matrix-class), 92
cbind2, Matrix, atomicVector-method
  (Matrix-class), 92
cbind2, matrix, ddiMatrix-method
  (diagonalMatrix-class), 40
cbind2, matrix, denseMatrix-method
  (denseMatrix-class), 33
cbind2, matrix, ldiMatrix-method
  (diagonalMatrix-class), 40
cbind2, Matrix, Matrix-method
  (Matrix-class), 92
cbind2, Matrix, missing-method
  (Matrix-class), 92
cbind2, Matrix, NULL-method
  (Matrix-class), 92
cbind2, matrix, sparseMatrix-method
  (sparseMatrix-class), 137
cbind2, NULL, Matrix-method
  (Matrix-class), 92
cbind2, numeric, denseMatrix-method
  (denseMatrix-class), 33
cbind2, numeric, sparseMatrix-method
  (sparseMatrix-class), 137
cbind2, sparseMatrix, denseMatrix-method
  (sparseMatrix-class), 137
cbind2, sparseMatrix, diagonalMatrix-method
  (diagonalMatrix-class), 40
cbind2, sparseMatrix, matrix-method
  (sparseMatrix-class), 137
cbind2, sparseMatrix, numeric-method
  (sparseMatrix-class), 137
cbind2, sparseMatrix, sparseMatrix-method
  (sparseMatrix-class), 137
character, 4, 43, 71, 92, 102, 110
CHMfactor, 22, 23, 25, 97, 127, 128, 152
CHMfactor-class, 16
CHMsimpl, 23
CHMsimpl-class (CHMfactor-class), 16
CHMsuper, 23
CHMsuper-class (CHMfactor-class), 16
chol, 13, 19, 20–25, 36, 45, 48, 89
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chol,ddenseMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,ddiMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,dgeMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,dpoMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,dpdMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,dsCMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,dspMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,ldiMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,lsCMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,MMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol,nsCMatrix-method(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol-methods(chol)</td>
<td>19</td>
</tr>
<tr>
<td>chol2inv</td>
<td>21</td>
</tr>
<tr>
<td>chol2inv,ANY-method(chol2inv-methods)</td>
<td>21</td>
</tr>
<tr>
<td>chol2inv,CHMfactor-method(chol2inv-methods)</td>
<td>21</td>
</tr>
<tr>
<td>chol2inv,denseMatrix-method(chol2inv-methods)</td>
<td>21</td>
</tr>
<tr>
<td>chol2inv,diagonalMatrix-method(diagonalMatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>chol2inv,dtrMatrix-method(chol2inv-methods)</td>
<td>21</td>
</tr>
<tr>
<td>chol2inv,sparseMatrix-method(chol2inv-methods)</td>
<td>21</td>
</tr>
<tr>
<td>Cholesky</td>
<td>13, 16–20, 22, 22, 25, 48, 59, 97, 138, 152</td>
</tr>
<tr>
<td>Cholesky,dsCMatrix-method(Cholesky)</td>
<td>22</td>
</tr>
<tr>
<td>Cholesky,dMatrix-method(Cholesky)</td>
<td>22</td>
</tr>
<tr>
<td>Cholesky,nsCMatrix-method(Cholesky)</td>
<td>22</td>
</tr>
<tr>
<td>Cholesky,sparseMatrix-method(Cholesky)</td>
<td>22</td>
</tr>
<tr>
<td>Cholesky-class</td>
<td>24</td>
</tr>
<tr>
<td>CholeskyFactorization-class(MatrixFactorization-class)</td>
<td>96</td>
</tr>
<tr>
<td>class</td>
<td>4, 8, 11, 15, 42, 75–77, 99, 120, 134, 141, 155</td>
</tr>
<tr>
<td>coerce</td>
<td>148</td>
</tr>
<tr>
<td>coerce,abIndex,integer-method(abIndex-class)</td>
<td>4</td>
</tr>
<tr>
<td>coerce,abIndex,numeric-method(abIndex-class)</td>
<td>4</td>
</tr>
<tr>
<td>coerce,abIndex,seqMat-method(abIndex-class)</td>
<td>4</td>
</tr>
<tr>
<td>coerce,abIndex,vector-method(abIndex-class)</td>
<td>4</td>
</tr>
<tr>
<td>coerce,ANY,denseMatrix-method(denseMatrix-class)</td>
<td>33</td>
</tr>
<tr>
<td>coerce,ANY,nsparseVector-method(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce,ANY,sparseMatrix-method(sparseMatrix-class)</td>
<td>137</td>
</tr>
<tr>
<td>coerce,ANY,sparseVector-method(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce,atomicVector,dsparseVector-method(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce,atomicVector,sparseVector-method(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce,BunchKaufman,LMatrix-method(Cholesky-class)</td>
<td>24</td>
</tr>
<tr>
<td>coerce,CHMfactor,Matrix-method(CHMfactor-class)</td>
<td>16</td>
</tr>
<tr>
<td>coerce,CHMfactor,pMatrix-method(CHMfactor-class)</td>
<td>16</td>
</tr>
<tr>
<td>coerce,CHMfactor,sparseMatrix-method(CHMfactor-class)</td>
<td>16</td>
</tr>
<tr>
<td>coerce,CHMfactor,triangularMatrix-method(CHMfactor-class)</td>
<td>16</td>
</tr>
<tr>
<td>coerce,Cholesky,LMatrix-method(Cholesky-class)</td>
<td>24</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,denseMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,LMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,lsparseMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,matrix.coo-method(SparseM-conversions)</td>
<td>133</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,matrix.csc-method(SparseM-conversions)</td>
<td>133</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,matrix.csr-method(SparseM-conversions)</td>
<td>133</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,mMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,nsparseMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,sparseVector-method(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,symmetricMatrix-method(symmetricMatrix-class)</td>
<td>146</td>
</tr>
<tr>
<td>coerce,CsparseMatrix,TsparseMatrix-method(CsparseMatrix-class)</td>
<td>30</td>
</tr>
</tbody>
</table>
coerce, ddenseMatrix, dgeMatrix-method 
  (ddenseMatrix-class), 31
coerce, ddenseMatrix, matrix-method 
  (ddenseMatrix-class), 31
coerce, ddiMatrix, CsparseMatrix-method 
  (diagonalMatrix-class), 40
coerce, ddiMatrix, dddenseMatrix-method 
  (diagonalMatrix-class), 40
coerce, ddiMatrix, dgeMatrix-method 
  (diagonalMatrix-class), 40
coerce, ddiMatrix, dspMatrix-method 
  (diagonalMatrix-class), 40
coerce, ddiMatrix, dgtMatrix-method 
  (diagonalMatrix-class), 40
coerce, ddiMatrix, dgtMatrix-method 
  (diagonalMatrix-class), 40
coerce, denseMatrix, CsparseMatrix-method 
  (denseMatrix-class), 33
coerce, denseMatrix, generalMatrix-method 
  (denseMatrix-class), 33
coerce, denseMatrix, RsparseMatrix-method 
  (RsparseMatrix-class), 123
coerce, denseMatrix, sparseMatrix-method 
  (denseMatrix-class), 33
coerce, denseMatrix, symmetricMatrix-method 
  (symmetricMatrix-class), 146
coerce, denseMatrix, TsparseMatrix-method 
  (denseMatrix-class), 33
coerce, dgCMatrix, dgeMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dgTMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dscMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dtCMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dtCMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, lgCMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, matrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, matrix.csc-method 
  (SparseM-conversions), 133
coerce, dgCMatrix, ngCMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, vector-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dgCMatrix-method 
  (dgCMatrix-class), 34
coerce, dgCMatrix, dgTMatrix-method 
  (dgTMatrix-class), 37
coerce, dgCMatrix, dscMatrix-method 
  (dsCMatrix-class), 47
coerce, dgCMatrix, dspMatrix-method 
  (dsyMatrix-class), 51
coerce, dgCMatrix, dssMatrix-method 
  (dsyMatrix-class), 51
coerce, dgCMatrix, dtTMatrix-method 
  (dtTMatrix-class), 57
coerce, dgCMatrix, lgeMatrix-method 
  (dgCMatrix-class), 35
coerce, dgCMatrix, matrix-method 
  (dgCMatrix-class), 35
coerce, dgCMatrix, triangularMatrix-method 
  (dgCMatrix-class), 35
coerce, dgRMatrix, matrix.csr-method 
  (SparseM-conversions), 133
coerce, dgTMatrix, dgeMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, dgCMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, dgCMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, dssMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, dtCMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, dtCMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, graphNEL-method 
  (graph-sparseMatrix), 66
coerce, dgTMatrix, matrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, matrix.coo-method 
  (SparseM-conversions), 133
coerce, dgTMatrix, symmetricMatrix-method 
  (dgTMatrix-class), 37
coerce, dgTMatrix, triangularMatrix-method 
  (dgTMatrix-class), 37
coerce, diagonalMatrix, denseMatrix-method 
  (diagonalMatrix-class), 40
coerce, diagonalMatrix, generalMatrix-method 
  (diagonalMatrix-class), 40
coerce, dtCMatrix, dgeMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, dgTMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, dsCMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, dtRMatrix-method
  (dsCMatrix-class), 47
coerce, dtCMatrix, dtrMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, dtTMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, mtxMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, ntcMatrix-method
  (dtCMatrix-class), 52
coerce, dtCMatrix, TspMatrix-method
  (dtCMatrix-class), 52
coerce, dtpMatrix, dtrMatrix-method
  (dtpMatrix-class), 54
coerce, dtpMatrix, dtTMatrix-method
  (dtpMatrix-class), 54
coerce, dtpMatrix, ltpMatrix-method
  (dtpMatrix-class), 54
coerce, dtrMatrix, CspMatrix-method
  (dtrMatrix-class), 57
coerce, dtrMatrix, dtpMatrix-method
  (dtrMatrix-class), 57
coerce, dtrMatrix, ltrMatrix-method
  (dtrMatrix-class), 57
coerce, dtrMatrix, mtxMatrix-method
  (dtrMatrix-class), 57
coerce, dtrMatrix, sparseMatrix-method
  (dtrMatrix-class), 57
coerce, dtMatrix, dgeMatrix-method
  (dtCMatrix-class), 52
coerce, dtMatrix, dgTMatrix-method
  (dtCMatrix-class), 52
coerce, dtMatrix, dtCMatrix-method
  (dtCMatrix-class), 52
coerce, dtMatrix, dtrMatrix-method
  (dtCMatrix-class), 52
coerce, dtMatrix, generalMatrix-method
  (dtCMatrix-class), 52
coerce, dtrMatrix, CspMatrix-method
  (dtrMatrix-class), 57
coerce, factor, dcmMatrix-method
  (dcmMatrix-class), 34
coerce, factor, spMatrix-method
  (spMatrix-class), 137
coerce, graph, CspMatrix-method
  (graph-sparseMatrix), 66
coerce, graph, Matrix-method
  (graph-sparseMatrix), 66
coerce, graph, spMatrix-method
  (graph-sparseMatrix), 66
coerce, graphAM, spMatrix-method
  (graph-sparseMatrix), 66
coerce, graphNEL, CspMatrix-method
  (graph-sparseMatrix), 66
coerce, graphNEL, dcmMatrix-method
  (dcmMatrix-class), 47
coerce, graphNEL, TspMatrix-method
  (graph-sparseMatrix), 66
coerce, indMatrix, CspMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, dMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, dspMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, lMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, lspMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, matrix-method
  (indMatrix-class), 70
coerce, indMatrix, nMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, nspMatrix-method
  (indMatrix-class), 70
coerce, indMatrix, TspMatrix-method
  (indMatrix-class), 70
coerce, integer, indMatrix-method
  (indMatrix-class), 70
coerce, integer, pMatrix-method
  (pMatrix-class), 110
coerce, ldenseMatrix, matrix-method
  (ldenseMatrix-class), 81
coerce,lsTMatrix,dsTMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,lgCMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,lgTMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,lsCMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,lsymatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,lgsyMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,ltTMatrix-method
(lsparseMatrix-classes), 84
coerce,lsTMatrix,ltcMatrix-method
(lsparseMatrix-classes), 84
coerce,lsyMatrix,dsyMatrix-method
(lsyMatrix-class), 85
coerce,lsyMatrix,ldgeMatrix-method
(lsyMatrix-class), 85
coerce,lsyMatrix,lspMatrix-method
(lsyMatrix-class), 85
coerce,ltCMatrix,dMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,dtCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,lgCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,ltTMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,ltcMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,ltTMatrix-method
(lsparseMatrix-classes), 84
coerce,ltCMatrix,ltTMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce,ltTMatrix,ltCMatrix-method
(lsparseMatrix-classes), 84
coerce, matrix, dtpMatrix-method
  (dtpMatrix-class), 54
coerce, matrix, dtrMatrix-method
  (dtrMatrix-class), 57
coerce, matrix, dtMatrix-method
  (dtMatrix-class), 52
coerce, matrix, indMatrix-method
  (indMatrix-class), 70
coerce, Matrix, integer-method
  (Matrix-class), 92
coerce, matrix, ldenseMatrix-method
  (ldenseMatrix-class), 81
coerce, matrix, lgeMatrix-method
  (lgeMatrix-class), 83
coerce, matrix, lgtMatrix-method
  (lgtMatrix-class), 84
coerce, matrix, lMatrix-method
  (lMatrix-class), 43
coerce, Matrix, logical-method
  (Matrix-class), 92
coerce, matrix, lsCMatrix-method
  (lsCMatrix-class), 84
coerce, matrix, lspMatrix-method
  (lspMatrix-class), 85
coerce, matrix, lsyMatrix-method
  (lsyMatrix-class), 85
coerce, matrix, ltCMatrix-method
  (ltCMatrix-class), 84
coerce, matrix, ltpMatrix-method
  (ltpMatrix-class), 86
coerce, matrix, ltrMatrix-method
  (ltrMatrix-class), 86
coerce, matrix, ltMatrix-method
  (ltMatrix-class), 84
coerce, Matrix, matrix-method
  (Matrix-class), 92
coerce, matrix, Matrix-method
  (Matrix-class), 92
coerce, matrix, ndenseMatrix-method
  (ndenseMatrix-class), 97
coerce, matrix, ngCMatrix-method
  (ngCMatrix-class), 105
coerce, matrix, ngeMatrix-method
  (ngeMatrix-class), 101
coerce, matrix, ngTMatrix-method
  (ngTMatrix-class), 105
coerce, matrix, nMatrix-method
  (nMatrix-class), 102
coerce, matrix, nspMatrix-method
  (nspMatrix-class), 107
coerce, matrix, nsyMatrix-method
  (nsyMatrix-class), 107
coerce, matrix, ntCMatrix-method
  (ntCMatrix-class), 105
coerce, matrix, ntpMatrix-method
  (ntpMatrix-class), 108
coerce, matrix, ntrMatrix-method
  (ntrMatrix-class), 108
coerce, matrix, ntTMatrix-method
  (ntTMatrix-class), 105
coerce, Matrix, numeric-method
  (Matrix-class), 92
coerce, matrix, pMatrix-method
  (pMatrix-class), 110
coerce, matrix, R sparseMatrix-method
  (R sparseMatrix-class), 123
coerce, Matrix, sparseMatrix-method
  (Matrix-class), 92
coerce, matrix, symmetricMatrix-method
  (symmetricMatrix-class), 146
coerce, matrix, triangularMatrix-method
  (triangularMatrix-class), 148
coerce, Matrix, sparseMatrix-method
  (sparseMatrix-class), 149
coerce, Matrix, vector-method
  (Matrix-class), 92
coerce, matrix, .coo, C sparseMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .coo, dgCMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .coo, dgTMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .coo, Matrix-method
  (SparseM-conversions), 133
coerce, matrix, .coo, TsparseMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .csc, C sparseMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .csc, dgCMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .csc, dgTMatrix-method
  (SparseM-conversions), 133
coerce, matrix, .csc, Matrix-method
  (SparseM-conversions), 133
coerce, matrix.csc, TsparsesMatrix-method
(SparseM-conversions), 133
coerce, matrix.csr, CsparsesMatrix-method
(SparseM-conversions), 133
coerce, matrix.csr, dgCMatrix-method
(SparseM-conversions), 133
coerce, matrix.csr, dgRMatrix-method
(SparseM-conversions), 133
coerce, matrix.csr, Matrix-method
(SparseM-conversions), 133
coerce, matrix.csr, RsparseMatrix-method
(SparseM-conversions), 133
coerce, matrix.csr, TsparseMatrix-method
(SparseM-conversions), 133
coerce, ndenseMatrix, CsparsesMatrix-method
(ndenseMatrix-class), 97
coerce, ndenseMatrix, ldenseMatrix-method
(ndenseMatrix-class), 97
coerce, ndenseMatrix, matrix-method
(ndenseMatrix-class), 97
coerce, ndenseMatrix, sparseMatrix-method
(ndenseMatrix-class), 97
coerce, ndenseMatrix, TsparsesMatrix-method
(ndenseMatrix-class), 97
coerce, ngCMatrix, dgCMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, dMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, dsparseMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, lgCMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, LMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, lpsparseMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, matrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, ngeMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, ngeMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, symmetricMatrix-method
(nsparseMatrix-classes), 105
coerce, ngCMatrix, triangularMatrix-method
(nsparseMatrix-classes), 105
coerce, nMatrix, dMatrix-method
(nMatrix-class), 102
coerce, nMatrix, indMatrix-method
(indMatrix-class), 70
coerce, nMatrix, LMatrix-method
(nMatrix-class), 102
coerce.nMatrix, matrix-method  
  (nMatrix-class), 102
coerce.nMatrix, pMatrix-method  
  (pMatrix-class), 110
coerce.nsCMatrix, dMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, dsCMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, dsTMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, matrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, lsCMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, lsTMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsCMatrix, ngCMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsparseMatrix, dspMatrix-method  
  (sparseVector-class), 142
coerce.nsparseVector, dspVector-method  
  (sparseVector-class), 142
coerce.nsparseVector, lsparseVector-method  
  (sparseVector-class), 142
coerce.nsparseVector, nzsparseVector-method  
  (sparseVector-class), 142
coerce.nspMatrix, dspMatrix-method  
  (nspMatrix-class), 107
coerce.nspMatrix, lspMatrix-method  
  (ndenseMatrix-class), 97
coerce.nspMatrix, ngeMatrix-method  
  (nspMatrix-class), 107
coerce.nspMatrix, nsyMatrix-method  
  (nspMatrix-class), 107
coerce.nsTMatrix, matrix-method  
  (nsparseMatrix-classes), 105
coerce.nsTMatrix, ngTMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsTMatrix, nsCMatrix-method  
  (nsparseMatrix-classes), 105
coerce.nsTMatrix, nsyMatrix-method  
  (nsMatrix-class), 107
coerce.nsrMatrix, dtrMatrix-method  
  (ntrMatrix-class), 108
coerce.nsrMatrix, ntrMatrix-method  
  (ntrMatrix-class), 108
coerce.ntMatrix, dtrMatrix-method  
  (ndenseMatrix-class), 97
coerce.ntMatrix, ngeMatrix-method  
  (ntrMatrix-class), 108
coerce.ntMatrix, ntpMatrix-method  
  (ntrMatrix-class), 108
coerce, ntTMatrix, dtTMatrix-method  
(ntsparseMatrix-classes), 105
coerce, ntTMatrix, generalMatrix-method  
(ntsparseMatrix-classes), 105
coerce, ntTMatrix, matrix-method  
(ntsparseMatrix-classes), 105
coerce, ntTMatrix, ngCMatrix-method  
(ntsparseMatrix-classes), 105
coerce, ntTMatrix, ngTMatrix-method  
(ntsparseMatrix-classes), 105
coerce, numeric, abIndex-method  
(abIndex-class), 4
coerce, numeric, CsparseMatrix-method  
(CsparseMatrix-class), 30
coerce, numeric, indMatrix-method  
(indMatrix-class), 70
coerce, numeric, pMatrix-method  
(pMatrix-class), 110
coerce, numeric, seqMat-method  
(abIndex-class), 4
coerce, numeric, TsparseMatrix-method  
(TsparseMatrix-class), 149
coerce, numLike, dgeMatrix-method  
(dgeMatrix-class), 35
coerce, pBunchKaufman, lMatrix-method  
(Cholesky-class), 24
coerce, pCholesky, lMatrix-method  
(Cholesky-class), 24
coerce, pMatrix, CsparseMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, dMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, dSParseMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, lMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, matrix-method  
(pMatrix-class), 110
coerce, pMatrix, ngeMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, ngTMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, nMatrix-method  
(pMatrix-class), 110
coerce, pMatrix, nspMatrix-method  
(pMatrix-class), 110
coerce, RsparseMatrix, CsparseMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, dgeMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, dMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, dSParseMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, generalMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, lMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, lSParseMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, matrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, nMatrix-method  
(RsparseMatrix-class), 123
coerce, RsparseMatrix, nSParseMatrix-method  
(RsparseMatrix-class), 123
coerce, seqMat, abIndex-method  
(abIndex-class), 4
coerce, seqMat, numeric-method  
(abIndex-class), 4
coerce, sparseMatrix, generalMatrix-method  
(sparseMatrix-class), 137
coerce, sparseMatrix, graph-method  
(graph-sparseMatrix), 66
coerce, sparseMatrix, graphNEL-method  
(graph-sparseMatrix), 66
coerce, sparseMatrix, indMatrix-method  
(indMatrix-class), 70
coerce, sparseMatrix, pMatrix-method  
(pMatrix-class), 110
coerce, sparseMatrix, dSParseMatrix-method  
(RsparseMatrix-class), 123
coerce, sparseMatrix, sparseVector-method  
(sparseVector-class), 142
coerce, sparseMatrix, symmetricMatrix-method  
(sparseMatrix-class), 137
<table>
<thead>
<tr>
<th>Function</th>
<th>Method Type</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>coerce sparseMatrix, triangularMatrix</td>
<td>method (sparseMatrix-class)</td>
<td>137</td>
</tr>
<tr>
<td>coerce sparseVector, CsparseMatrix</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, integer</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, logical</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, Matrix</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, numeric</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, sparseMatrix</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, TsparseMatrix</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce sparseVector, vector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce table, sparseMatrix</td>
<td>method (triangularMatrix-class)</td>
<td>137</td>
</tr>
<tr>
<td>coerce triangularMatrix, symmetricMatrix</td>
<td>method (triangularMatrix-class)</td>
<td>148</td>
</tr>
<tr>
<td>coerce TsparseMatrix, CsparseMatrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, graphNEL</td>
<td>method (graph sparseMatrix)</td>
<td>66</td>
</tr>
<tr>
<td>coerce TsparseMatrix, LMatrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, LsparseMatrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, matrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, nMatrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, nsparseMatrix</td>
<td>method (TsparseMatrix-class)</td>
<td>149</td>
</tr>
<tr>
<td>coerce TsparseMatrix, sparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce xsparseVector, dsparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce xsparseVector, isparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce xsparseVector, lSparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce xsparseVector, nsparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>coerce xsparseVector, zSparseVector</td>
<td>method (sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>colMeans</td>
<td>(colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, CsparseMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, denseMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, ddenseMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, denseMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, dgCMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, dgeMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, diagonalMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, LgCMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, LuigiMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, LuigiMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>colMeans, RsparseMatrix</td>
<td>method (colSums, colSums)</td>
<td>26</td>
</tr>
<tr>
<td>compare CsparseMatrix, CsparseMatrix</td>
<td>method (CsparseMatrix-class)</td>
<td>30</td>
</tr>
<tr>
<td>compare dMatrix, logical</td>
<td>method (dMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>compare dMatrix, numeric</td>
<td>method (dMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>compare lgeMatrix, lgeMatrix</td>
<td>method (lgeMatrix-class)</td>
<td></td>
</tr>
</tbody>
</table>
crossprod, dtpMatrix, matrix-method
(matrix-products), 94

crossprod, dtrMatrix, dtrMatrix-method
(matrix-products), 94

crossprod, dtrMatrix, denseMatrix-method
(matrix-products), 94

crossprod, denseMatrix, matrix-method
(matrix-products), 94

crossprod, matrix, ANY-method
(matrix-products), 94

crossprod, matrix, CsparseMatrix-method
(matrix-products), 94

crossprod, matrix, dgeMatrix-method
(matrix-products), 94

crossprod, matrix, diagonalMatrix-method
(matrix-products), 94

crossprod, matrix, mMatrix-method
(matrix-products), 94

crossprod, matrix, sparseVector-method
(matrix-products), 94

crossprod, sparseMatrix, diagonalMatrix-method
(matrix-products), 94

crossprod, sparseMatrix, mMatrix-method
(matrix-products), 94

crossprod, sparseVector, mMatrix-method
(matrix-products), 94

crossprod, sparseVector, sparseVector-method
(matrix-products), 94

crossprod, symmetricMatrix, ANY-method
(matrix-products), 94

crossprod, numLike, CsparseMatrix-method
(matrix-products), 94

crossprod, numLike, diagonalMatrix-method
(matrix-products), 94

crossprod, numLike, sparseVector-method
(matrix-products), 94

crossprod, mMatrix, sparseVector-method
(matrix-products), 94

crossprod, ndenseMatrix, nsparseMatrix-method
(matrix-products), 94

crossprod, numLike, sparseVector-method
(matrix-products), 94

crossprod, ndenseMatrix, nsparseMatrix-method
(matrix-products), 94

crossprod, denseMatrix, missing-method
(matrix-products), 94

crossprod, denseMatrix, numLike-method
(matrix-products), 94

crossprod, denseMatrix, sparseVector-method
(matrix-products), 94

crossprod, denseMatrix, symmetricMatrix-method
(matrix-products), 94

crossprod, denseMatrix, symmetricMatrix-method
(matrix-products), 94

crossprod, denseMatrix, symmetricMatrix-method
(matrix-products), 94
crossprod, symmetricMatrix, Matrix-method
(matrix-products), 94
crossprod, symmetricMatrix, missing-method
(matrix-products), 94
crossprod, TsparseMatrix, ANY-method
(matrix-products), 94
crossprod, TsparseMatrix, Matrix-method
(matrix-products), 94
crossprod, TsparseMatrix, missing-method
(matrix-products), 94
crossprod, TsparseMatrix, TsparseMatrix-method
(matrix-products), 94
crossprod-methods, 34
crossprod-methods (matrix-products), 94
CsparseMatrix, 10–12, 16, 34, 39, 46–48, 53, 78, 124, 130, 134, 135, 145, 149, 150
CsparseMatrix-class, 30
cumsum, 41
data.frame, 61, 138
dCHMsimpl, 152
dCHMsimpl-class (CHMfactor-class), 16
dCHMsuper-class (CHMfactor-class), 16
ddenseMatrix, 33, 34, 58
ddenseMatrix-class, 31
ddiMatrix, 39, 41, 82
ddiMatrix-class, 32
denseLU, 88
denseLU-class (LU-class), 89
denseMatrix, 42, 75, 81, 88, 97, 117, 128, 148, 150
denseMatrix-class, 33
det, 93
det (Matrix-class), 92
determinant, 25, 45, 55, 138
determinant, CHMfactor, logical-method
(CHMfactor-class), 16
determinant, CHMfactor, missing-method
(CHMfactor-class), 16
determinant, ddenseMatrix, logical-method
(ddenseMatrix-class), 31
determinant, ddenseMatrix, missing-method
(ddenseMatrix-class), 31
determinant, dgCMatrix, logical-method
(sparseMatrix-class), 137
determinant, dgeMatrix, logical-method
(dgeMatrix-class), 35
determinant, dgeMatrix, missing-method
(dgeMatrix-class), 35
determinant, diagonalMatrix, logical-method
(diagonalMatrix-class), 40
determinant, dpoMatrix, logical-method
(dpoMatrix-class), 44
determinant, dpoMatrix, missing-method
(dpoMatrix-class), 44
determinant, dppMatrix, logical-method
(dppMatrix-class), 44
determinant, dppMatrix, missing-method
(dppMatrix-class), 44
determinant, dsCMatrix, logical-method
(dsCMatrix-class), 47
determinant, dsCMatrix, missing-method
(dsCMatrix-class), 47
determinant, dsparseMatrix, logical-method
(sparseMatrix-class), 137
determinant, dtCMatrix, logical-method
(sparseMatrix-class), 137
determinant, dtpMatrix, logical-method
(dtpMatrix-class), 54
determinant, dtpMatrix, missing-method
(dtpMatrix-class), 54
determinant, dtrMatrix, logical-method
(dtrMatrix-class), 57
determinant, dtrMatrix, missing-method
(dtrMatrix-class), 57
determinant, indMatrix, logical-method
(indMatrix-class), 70
determinant, Matrix, logical-method
(Matrix-class), 92
determinant, Matrix, missing-method
(Matrix-class), 92
determinant, pMatrix, logical-method
(pMatrix-class), 110
determinant, sparseMatrix, logical-method
(sparseMatrix-class), 137
determinant, sparseMatrix, missing-method
(sparseMatrix-class), 137
dgCMatrix-class, 34
dgeMatrix, 31, 32, 44, 45, 48, 51–53, 57, 88, 90, 94, 139, 140
dgeMatrix-class, 35
dgRMatrix, 49, 124
dgRMatrix-class, 36
dgTMatrix, 48, 49, 51, 53, 57, 66, 69, 149, 150
INDEX

183
dgTMatrix-class, 37
diag, 11, 40, 71, 97, 98
diag, CsparseMatrix-method
  (CsparseMatrix-class), 30
diag, ddenseMatrix-method
  (ddenseMatrix-class), 31
diag, dgCMatrix-method
  (dgCMatrix-class), 34
diag, dgeMatrix-method
  (dgeMatrix-class), 35
diag, dgRMatrix-method
  (dgRMatrix-class), 36
diag, diagonalMatrix-method
  (diagonalMatrix-class), 40
diag, dspMatrix-method
  (dsyMatrix-class), 51
diag, dsyMatrix-method
  (dsyMatrix-class), 51
diag, dtpMatrix-method
  (dtpMatrix-class), 54
diag, dtrMatrix-method
  (dtrMatrix-class), 57
diag, ldenseMatrix-method
  (ldenseMatrix-class), 81
diag, lgeMatrix-method
  (lgeMatrix-class), 83
diag, lspMatrix-method
  (lsyMatrix-class), 85
diag, lsyMatrix-method
  (lsyMatrix-class), 85
diag, ltpMatrix-method
  (ltrMatrix-class), 86
diag, ltrMatrix-method
  (ltrMatrix-class), 86
diag, ngeMatrix-method
  (ngeMatrix-class), 101
diag, nspMatrix-method
  (nsyMatrix-class), 107
diag, nsyMatrix-method
  (nsyMatrix-class), 107
diag, ntpMatrix-method
  (ntRMatrix-class), 108
diag, ntrMatrix-method
  (ntRMatrix-class), 108
diagU2U(diagU2N), 42
Diagonal, 12, 32, 33, 39, 41, 82, 91, 135, 145
diagonalMatrix, 12, 33, 39, 40, 66, 77, 82, 91
diagonalMatrix-class, 40
diagU2N, 42
diff, 93, 121
diff, Matrix-method (Matrix-class), 92
dim, 93, 113, 122
dim, dgCMatrix-method (dgCMatrix-class),
  34
dim, dgeMatrix-method (dgeMatrix-class),
  35
dim, dgRMatrix-method (dgRMatrix-class),
  36
dim, Matrix-method (Matrix-class), 92
dim, MatrixFactorization-method
  (MatrixFactorization-class), 96
dim<-, denseMatrix-method
  (denseMatrix-class), 33
dim<-, Matrix-method (Matrix-class), 92
dim<-, sparseMatrix-method
  (sparseMatrix-class), 137
dim<-, sparseVector-method
  (sparseVector-class), 142
dimnames, 32, 40, 64, 75, 76, 78, 82, 91, 93,
  134
dimnames, dgeMatrix-method
forceSymmetric, dsTMatrix, character-method (forceSymmetric), 63
forceSymmetric, dsTMatrix, missing-method (forceSymmetric), 63
forceSymmetric, dsyMatrix, character-method (forceSymmetric), 63
forceSymmetric, dsyMatrix, missing-method (forceSymmetric), 63
forceSymmetric, lsCMatrix, character-method (forceSymmetric), 63
forceSymmetric, lsCMatrix, missing-method (forceSymmetric), 63
forceSymmetric, lspMatrix, character-method (forceSymmetric), 63
forceSymmetric, lspMatrix, missing-method (forceSymmetric), 63
forceSymmetric, lsRMatrix, character-method (forceSymmetric), 63
forceSymmetric, lsRMatrix, missing-method (forceSymmetric), 63
forceSymmetric, lsTMatrix, character-method (forceSymmetric), 63
forceSymmetric, lsTMatrix, missing-method (forceSymmetric), 63
forceSymmetric, lsyMatrix, character-method (forceSymmetric), 63
forceSymmetric, lsyMatrix, missing-method (forceSymmetric), 63
forceSymmetric, matrix, ANY-method (forceSymmetric), 63
forceSymmetric, Matrix, missing-method (forceSymmetric), 63
forceSymmetric, nsCMatrix, character-method (forceSymmetric), 63
forceSymmetric, nsCMatrix, missing-method (forceSymmetric), 63
forceSymmetric, nspMatrix, character-method (forceSymmetric), 63
forceSymmetric, nspMatrix, missing-method (forceSymmetric), 63
forceSymmetric, nsRMatrix, character-method (forceSymmetric), 63
forceSymmetric, nsRMatrix, missing-method (forceSymmetric), 63
forceSymmetric, nsTMatrix, character-method (forceSymmetric), 63
forceSymmetric, nsTMatrix, missing-method (forceSymmetric), 63
forceSymmetric, nsyMatrix, character-method (forceSymmetric), 63
forceSymmetric, nsyMatrix, missing-method (forceSymmetric), 63
forceSymmetric, sparseMatrix, ANY-method (forceSymmetric), 63
format, ANY-method (image-methods), 68
format, sparseMatrix-method (sparseMatrix-class), 137
formatSparseM, ANY-method (image-methods), 68
formatSpMatrix, ANY-method (image-methods), 68
formatSpMatrix (printSpMatrix), 112
forwardsolve, ANY-method (image-methods), 128
function, ANY-method (image-methods), 78, 122
gamma, dgCMatrix-method (dgCMatrix-class), 34
gamma, dMatrix-method (dMatrix-class), 43
generalMatrix, ANY-method (image-methods), 71, 135, 145, 147
generalMatrix-class, 65
generalMatrix-class, ANY-method (image-methods), 65
get.adjacency, ANY-method (image-methods), 66
getClassDef, ANY-method (image-methods), 113
getOption, ANY-method (image-methods), 113
graph, ANY-method (image-methods), 66
graph, ANY-method (image-methods), 66
graph, ANY-method (image-methods), 66
graph, ANY-method (image-methods), 66
graph2T (graph-sparseMatrix), ANY-method (image-methods), 66
grey, ANY-method (image-methods), 68
grid.raster, ANY-method (image-methods), 68
grid.rect, ANY-method (image-methods), 69
head, ANY-method (image-methods), 142
head, ANY-method (image-methods), 142
head, ANY-method (image-methods), 92
head, ANY-method (image-methods), 92
head, ANY-method (image-methods), 92
head, ANY-method (image-methods), 142
head, ANY-method (image-methods), 142
Hilbert, ANY-method (image-methods), 67
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, ANY-method (image-methods), 68
image, l sparseMatrix-method
   (image-methods), 68
image, Matrix-method (image-methods), 68
image, nsparseMatrix-method
   (image-methods), 68
image-methods, 68
iMatrix-class (Unused-classes), 151
index, 155
index-class, 70
indMatrix, 110, 111
indMatrix-class, 70
integer, 8, 30, 47, 53, 103
invisible, 113
invPerm, 73, 110, 111
is, 68
is.atomic, 8
is.finite, 74, 75
is.finite, abIndex-method
   (abIndex-class), 4
is.finite, ddenseMatrix-method
   (is.na-methods), 74
is.finite, dgeMatrix-method
   (is.na-methods), 74
is.finite, diagonalMatrix-method
   (is.na-methods), 74
is.finite, dsparseMatrix-method
   (is.na-methods), 74
is.finite, indMatrix-method
   (is.na-methods), 74
is.finite, lMatrix-method
   (is.na-methods), 74
is.finite, nMatrix-method
   (is.na-methods), 74
is.finite, n sparseVector-method
   (sparseVector-class), 142
is.finite, sparseVector-method
   (sparseVector-class), 142
is.infinite, LMatrix-method
   (is.na-methods), 74
is.infinite, nMatrix-method
   (is.na-methods), 74
is.infinite, n sparseVector-method
   (sparseVector-class), 142
is.infinite, sparseVector-method
   (sparseVector-class), 142
is.na, 74, 75
is.na, abIndex-method (abIndex-class), 4
is.na, denseMatrix-method
   (is.na-methods), 74
is.na, indMatrix-method (is.na-methods), 74
is.na, nsparseMatrix-method
   (nsparseMatrix-classes), 105
is.na, nsparseVector-method
   (sparseVector-class), 142
is.na, sparseMatrix-method
   (is.na-methods), 74
is.na, sparseVector-method
   (sparseVector-class), 142
is.na-methods, 74
is.null, 75, 76
is.null.DN, 75
isDiagonal, 41
isDiagonal (isTriangular), 77
isDiagonal, denseMatrix-method
   (isTriangular), 77
isDiagonal, diagonalMatrix-method
   (isTriangular), 77
isDiagonal, matrix-method
   (isTriangular), 77
isDiagonal, sparseMatrix-method
   (isTriangular), 77
isDiagonal, symmetricMatrix-method
   (isTriangular), 77
isDiagonal, triangularMatrix-method
   (isTriangular), 77
isDiagonal-methods (isTriangular), 77
isLDL (CHMfactor-class), 16
isSparseVector-class
   (sparseVector-class), 142
isSymmetric, 77, 98, 127, 146–148
isSymmetric, denseMatrix-method
   (isSymmetric-methods), 76
isSymmetric, dgCMatrix-method
   (dgCMatrix-class), 34
isSymmetric, diagonalMatrix-method (isSymmetric-methods), 76
isSymmetric, sparseMatrix-method (isSymmetric-methods), 76
isSymmetric, symmetricMatrix-method (isSymmetric-methods), 76
isSymmetric, triangularMatrix-method (isSymmetric-methods), 76
isSymmetric-methods, 76, 146
isTriangular, 77, 148
isTriangular, BunchKaufman-method (isTriangular), 77
isTriangular, Cholesky-method (isTriangular), 77
isTriangular, CsparseMatrix-method (isTriangular), 77
isTriangular, denseMatrix-method (isTriangular), 77
isTriangular, diagonalMatrix-method (isTriangular), 77
isTriangular, dtCMatrix-method (isTriangular), 77
isTriangular, dtPMatrix-method (isTriangular), 77
isTriangular, dtRMatrix-method (isTriangular), 77
isTriangular, dtTMatrix-method (isTriangular), 77
isTriangular, ltcMatrix-method (isTriangular), 77
isTriangular, ltpMatrix-method (isTriangular), 77
isTriangular, ltrMatrix-method (isTriangular), 77
isTriangular, ltrTMatrix-method (isTriangular), 77
isTriangular, ltTMatrix-method (isTriangular), 77
isTriangular, matrix-method (isTriangular), 77
isTriangular, ntcMatrix-method (isTriangular), 77
isTriangular, ntpMatrix-method (isTriangular), 77
isTriangular, ntrMatrix-method (isTriangular), 77
isTriangular, ntTMatrix-method (isTriangular), 77
isTriangular, pbunchKaufman-method (isTriangular), 77
isTriangular, pCholesky-method (isTriangular), 77
isTriangular, triangularMatrix-method (isTriangular), 77
isTriangular, TsparseMatrix-method (isTriangular), 77
isTriangular-methods (isTriangular), 77

kappa, 93, 119
KhatriRao, 78
KNex, 79
kronecker, 12, 31, 34, 78–80, 94
kronecker, ANY, diagonalMatrix-method (kronecker-methods), 80
kronecker, ANY, Matrix-method (kronecker-methods), 80
kronecker, ANY, sparseMatrix-method (kronecker-methods), 80
kronecker, dgTMatrix, dgTMatrix-method (kronecker-methods), 80
kronecker, dtMatrix, dtMatrix-method (kronecker-methods), 80
kronecker, diagonalMatrix, ANY-method (kronecker-methods), 80
kronecker, diagonalMatrix, Matrix-method (kronecker-methods), 80
kronecker, dsparseMatrix, dsparseMatrix-method (kronecker-methods), 80
kronecker, dtMatrix, dgTMatrix-method (kronecker-methods), 80
kronecker, dtMatrix, dtMatrix-method (kronecker-methods), 80
kronecker, dtTMatrix, dtMatrix-method (kronecker-methods), 80
kronecker, indMatrix, indMatrix-method (kronecker-methods), 80
kronecker, Matrix, ANY-method (kronecker-methods), 80
kronecker, Matrix, diagonalMatrix-method (kronecker-methods), 80
kronecker, sparseMatrix, ANY-method (kronecker-methods), 80
kronecker, sparseMatrix, TsparseMatrix-method (kronecker-methods), 80
kronecker, TsparseMatrix, sparseMatrix-method (kronecker-methods), 80
kronecker, TsparseMatrix, TsparseMatrix-method (kronecker-methods), 80
kronecker-methods, 80
ldenseMatrix, 33, 86, 87
ldenseMatrix-class, 81
ldiMatrix, 39, 41
ldiMatrix-class, 82
length, 104, 142
length, abIndex-method (abIndex-class), 4
length, Matrix-method (Matrix-class), 92
length, sparseVector-method (sparseVector-class), 142
levelplot, 34, 37, 38, 68, 69, 93
lgamma, dgCMatrix-method (dgCMatrix-class), 34
lgamma, dMatrix-method (dMatrix-class), 43
lgCMatrix, 83
lgCMatrix-class (lsparseMatrix-classes), 84
lgeMatrix, 81, 86, 87
lgeMatrix-class, 83
lgRMatrix-class (lsparseMatrix-classes), 84
lgTMatrix-class (lsparseMatrix-classes), 84
list, 12, 29, 40, 69, 93, 121, 124, 130
lMatrix, 81, 82, 85, 87, 91, 97, 103, 106
lMatrix-class (dMatrix-class), 43
log, dgCMatrix-method (dgCMatrix-class), 34
log, dMatrix-method (dMatrix-class), 43
Logic, ANY, Matrix-method (Matrix-class), 92
Logic, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class), 30
Logic, dMatrix, logical-method (dMatrix-class), 43
Logic, dMatrix, numeric-method (dMatrix-class), 43
Logic, dMatrix, sparseVector-method (sparseVector-class), 142
Logic, ldenseMatrix, lsparseMatrix-method (ldenseMatrix-class), 81
Logic, lgCMatrix, lgCMatrix-method (lsparseMatrix-classes), 84
Logic, lgeMatrix, lgeMatrix-method (lgeMatrix-class), 83
Logic, lgTMatrix, lgTMatrix-method (lsparseMatrix-classes), 84
Logic, lMatrix, logical-method (dMatrix-class), 43
Logic, lMatrix, numeric-method (dMatrix-class), 43
Logic, lMatrix, sparseVector-method (sparseVector-class), 142
Logic, ltCMatrix, ltCMatrix-method (lsparseMatrix-classes), 84
Logic, Matrix, ANY-method (Matrix-class), 92
Logic, Matrix, logical-method (Matrix-class), 92
Logic, Matrix, nMatrix-method (nMatrix-class), 102
Logic, ngeMatrix, ngeMatrix-method (ngeMatrix-class), 101
Logic, nMatrix, logical-method (nMatrix-class), 102
Logic, nMatrix, Matrix-method (nMatrix-class), 102
Logic, nMatrix, numeric-method (nMatrix-class), 102
Logic, nMatrix, sparseVector-method (sparseVector-class), 142
Logic, nsparseVector, nsparseVector-method (sparseVector-class), 142
INDEX

Logic, numeric, dMatrix-method
(dMatrix-class), 43

Logic, numeric, lMatrix-method
(dMatrix-class), 43

Logic, numeric, nMatrix-method
(nMatrix-class), 102

Logic, sparseVector, dMatrix-method
(sparseVector-class), 142

Logic, sparseVector, lMatrix-method
(sparseVector-class), 142

Logic, sparseVector, nMatrix-method
(sparseVector-class), 142

Logic, triangularMatrix, diagonalMatrix-method
(diagonalMatrix-class), 40

logical, 17, 44, 45, 71, 75–77, 83, 91, 103, 143, 155

logm, 59, 60

lsCMatrix, 17, 39, 146
lsCMatrix-class
(lsparseMatrix-classes), 84

lsCMatrix-class
(lsparseMatrix-classes), 84

lsCMatrix-classes, 84

lsCMatrix-class
(sparseVector-class), 142

lsPMatrix-class (lsyMatrix-class), 85

lsRMatrix-class
(lsparseMatrix-classes), 84

lsTMatrix-class
(lsparseMatrix-classes), 84

lsyMatrix, 83

lsyMatrix-class, 85

ltCMatrix, 148

ltCMatrix-class
(lsparseMatrix-classes), 84

ltPMatrix-class (ltrMatrix-class), 86

ltrMatrix, 83

ltrMatrix-class
(lsparseMatrix-classes), 84

ltrMatrix-class, 86

ltrTMatrix-class
(lsparseMatrix-classes), 84

LU, 89, 97, 132

lu, 13, 28, 35, 59, 87, 88–90, 128, 132, 138

lu, ddenseMatrix-method
(ddenseMatrix-class), 31

lu, dgCMatrix-method (lu), 87

lu, dgeMatrix-method (lu), 87

lu, dsparseMatrix-method
(dsparseMatrix-class), 49

lu, dtCMatrix-method (lu), 87

lu, matrix-method (lu), 87

lu, sparseMatrix-method
(sparseMatrix-class), 137

LU-class, 89

Math, CsparseMatrix-method
(CsparseMatrix-class), 30

Math, ddenseMatrix-method
(ddenseMatrix-class), 31

Math, denseMatrix-method
(denseMatrix-class), 33

Math, dgCMatrix-method
(dgCMatrix-class), 34

Math, dgeMatrix-method
(dgeMatrix-class), 35

Math, diagonalMatrix-method
(diagonalMatrix-class), 40

Math, sparseMatrix-method
(sparseMatrix-class), 137

Math, sparseVector-method
(sparseVector-class), 142

Math2, dMatrix-method (dMatrix-class), 43

Math2, dsparseVector-method
(sparseVector-class), 142

Math2, Matrix-method (Matrix-class), 92

Math2, sparseVector-method
(sparseVector-class), 142


matrix, 7, 25, 41, 63, 64, 71, 75, 76, 91, 93, 104, 111, 124, 155

Matrix-class, 92

matrix-products, 94

matrix.csr, 134

MatrixFactorization, 23, 25, 90, 124, 126, 128, 132

MatrixFactorization-class, 96

max, 44

mean, Matrix-method (Matrix-class), 92
mean, sparseMatrix-method
(sparseMatrix-class), 137
mean, sparseVector-method
(sparseVector-class), 142
Methods, 15
min, 44
missing, 39
mode, 71, 111
model.frame, 130
model.Matrix, 130, 131
model.matrix, 130, 131
modelMatrix, 130

NA, 22, 41, 44, 74, 75, 77, 80, 84, 88, 102, 103,
138, 143, 147, 155

nb2listw, 153
nCHMsimpl-class (CHMfactor-class), 16
nCHMspuper-class (CHMfactor-class), 16
ndenseMatrix-class, 97
nearcor, 100
nearPD, 98
new, 31, 134
ngCMatrix, 102
ngCMatrix-class
(nsparseMatrix-classes), 105
ngEMatrix, 97, 108, 109
ngEMatrix-class, 101
ngRMMatrix-class
(nsparseMatrix-classes), 105
ngTMatrix, 66, 71, 72, 111
ngTMMatrix-class
(nsparseMatrix-classes), 105
ngMMatrix, 10, 44, 74, 75, 101
ngMMatrix-class, 102
nnzero, 56, 103, 123
nnzero, ANY-method (nnzero), 103
nnzero, CHMfactor-method (nnzero), 103
nnzero, denseMatrix-method (nnzero), 103
nnzero, diagonalMatrix-method (nnzero), 103
nnzero, indMatrix-method (nnzero), 103
nnzero, sparseMatrix-method (nnzero), 103
norm, 28, 29, 52, 99, 104, 118, 119, 138
norm, ANY, missing-method (norm), 104
norm, ddenseMatrix, character-method
(ddenseMatrix-class), 31
norm, ddenseMatrix, missing-method
(ddenseMatrix-class), 31

norm, dgeMatrix, character-method
(dgeMatrix-class), 35
norm, dgeMatrix, missing-method
(dgeMatrix-class), 35
norm, diagonalMatrix, character-method
(diagonalMatrix-class), 40
norm, dspMatrix, character-method
(dspMatrix-class), 51
norm, dspMatrix, missing-method
(dspMatrix-class), 51
norm, dsyMatrix, character-method
(dsyMatrix-class), 51
norm, dsyMatrix, missing-method
(dsyMatrix-class), 51
norm, dtpMatrix, character-method
(dtpMatrix-class), 54
norm, dtpMatrix, missing-method
(dtpMatrix-class), 54
norm, dtrMatrix, character-method
(dtrMatrix-class), 57
norm, dtrMatrix, missing-method
(dtrMatrix-class), 57
norm, ldenseMatrix, character-method
(ldenseMatrix-class), 81
norm, Matrix, character-method (norm), 104
norm, matrix, character-method (norm), 104
norm, ndenseMatrix, character-method
(ndenseMatrix-class), 97
norm, sparseMatrix, character-method
(sparseMatrix-class), 137
nsCMatrix-class
(nsparseMatrix-classes), 105
nsparseMatrix, 39, 102, 103, 134, 138
nsparseMatrix-class
(nsparseMatrix-classes), 105
nsparseMatrix-classes, 105
nsparseVector-class
(sparseVector-class), 142
nsPMatrix-class (nsyMatrix-class), 107
nsRMatrix-class
(nsparseMatrix-classes), 105
nsTMatrix, 66
nsTMatrix-class
(nsparseMatrix-classes), 105
nsyMatrix, 102
nsyMatrix-class, 107
ntCMatrix-class
(nsparseMatrix-classes), 105
INDEX

ntPMatrix-class (ntMatrix-class), 108
ntMatrix, 102
ntRMatrix-class
  (nsparseMatrix-classes), 105
ntMatrix-class, 108
ntTMatrix-class
  (nsparseMatrix-classes), 105
NULL, 61, 64, 75, 91, 113, 134
numeric, 4, 5, 45, 79, 103, 124
onenormest, 105
onenormest (condest), 28
Ops, 5, 41, 58, 143
Ops, abIndex, abIndex-method
  (abIndex-class), 4
Ops, abIndex, ANY-method (abIndex-class), 4
Ops, ANY, numeric-method
  (abIndex-class), 4
Ops, ANY, abIndex-method (abIndex-class), 4
Ops, ANY, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, ANY, ldiMatrix-method
  (diagonalMatrix-class), 40
Ops, ANY, Matrix-method (Matrix-class), 92
Ops, ANY, sparseVector-method
  (sparseVector-class), 142
Ops, atomicVector, sparseVector-method
  (sparseVector-class), 142
Ops, ddiMatrix, ANY-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, diagonalMatrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, dMatrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, ldiMatrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, logical-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, Matrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, numeric-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, sparseMatrix-method
  (diagonalMatrix-class), 40
Ops, ddiMatrix, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, diagonalMatrix, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, diagonalMatrix, diagonalMatrix-method
  (diagonalMatrix-class), 40
Ops, diagonalMatrix, sparseMatrix-method
  (sparseMatrix-class), 137
Ops, diagonalMatrix, triangularMatrix-method
  (diagonalMatrix-class), 40
Ops, dMatrix, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, dMatrix, dMatrix-method
  (dMatrix-class), 43
Ops, dMatrix, ldiMatrix-method
  (diagonalMatrix-class), 40
Ops, dMatrix, lMatrix-method
  (dMatrix-class), 43
Ops, dMatrix, nmatrix-method
  (dMatrix-class), 43
Ops, dpoMatrix, logical-method
  (dpoMatrix-class), 44
Ops, dpoMatrix, numeric-method
  (dpoMatrix-class), 44
Ops, dppMatrix, logical-method
  (dpoMatrix-class), 44
Ops, dppMatrix, numeric-method
  (dpoMatrix-class), 44
Ops, dsparseMatrix, nsparseMatrix-method
  (nsparseMatrix-classes), 105
Ops, ldenseMatrix, ldenseMatrix-method
  (ldenseMatrix-class), 81
Ops, ldiMatrix, ANY-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, ddiMatrix-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, dMatrix-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, ldiMatrix-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, logical-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, Matrix-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, numeric-method
  (diagonalMatrix-class), 40
Ops, ldiMatrix, sparseMatrix-method
  (diagonalMatrix-class), 40
Ops, lMatrix, dMatrix-method
  (dMatrix-class), 43
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ops,lMatrix,lMatrix-method</td>
<td>(dMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>Ops,lMatrix,nMatrix-method</td>
<td>(nMatrix-class)</td>
<td>102</td>
</tr>
<tr>
<td>Ops,lMatrix,numeric-method</td>
<td>(dMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>Ops,logical,ddiMatrix-method</td>
<td>(diagonalmatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>Ops,logical,dpoMatrix-method</td>
<td>(dpoMatrix-class)</td>
<td>44</td>
</tr>
<tr>
<td>Ops,logical,dpdMatrix-method</td>
<td>(dpoMatrix-class)</td>
<td>44</td>
</tr>
<tr>
<td>Ops,logical,ldiMatrix-method</td>
<td>(diagonalmatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>Ops,logical,Matrix-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,lsparseMatrix,lsparseMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>84</td>
</tr>
<tr>
<td>Ops,lsparseMatrix,nsparseMatrix-method</td>
<td>(nsparseMatrix-classes)</td>
<td>105</td>
</tr>
<tr>
<td>Ops,Matrix,ANY-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,Matrix,ddiMatrix-method</td>
<td>(diagonalmatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>Ops,Matrix,ldiMatrix-method</td>
<td>(diagonalmatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>Ops,Matrix,logical-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,Matrix,Matrix-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,Matrix,matrix-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,matrix,Matrix-method</td>
<td>(Matrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Ops,Matrix,sparseVector-method</td>
<td>(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>Ops,ndenseMatrix,ndenseMatrix-method</td>
<td>(ndenseMatrix-class)</td>
<td>97</td>
</tr>
<tr>
<td>Ops,nMatrix,dMatrix-method</td>
<td>(dMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>Ops,nMatrix,lMatrix-method</td>
<td>(nMatrix-class)</td>
<td>102</td>
</tr>
<tr>
<td>Ops,nMatrix,numeric-method</td>
<td>(nMatrix-class)</td>
<td>102</td>
</tr>
<tr>
<td>Ops,nsparseMatrix,sparseMatrix-method</td>
<td>(nsparseMatrix-classes)</td>
<td>105</td>
</tr>
<tr>
<td>Options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>order,73</td>
<td></td>
<td></td>
</tr>
<tr>
<td>outer,93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pack (unpack),150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pack,Matrix-method (unpack),150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pack,sparsesMatrix-method (unpack),150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pack,triangularMatrix-method (unpack),150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>panel.levelplot.raster,68</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

pBunchKaufman-class (Cholesky-class), 24
pCholesky-class (Cholesky-class), 24
plot.default, 68
pMatrix, 71–73
pMatrix-class, 110
posdefify, 98–100
print, 41, 65, 93, 112, 113, 137, 138
print, diagonalMatrix-method
(diagonalMatrix-class), 40
print, sparseMatrix-method
(sparseMatrix-class), 137
print.default, 64, 113
printSpMatrix, 65, 112, 137
printSpMatrix2 (printSpMatrix), 112
prod, 44
prod, ddiMatrix-method
(diagonalMatrix-class), 40
prod, ldiMatrix-method
(diagonalMatrix-class), 40
qr, 89, 114–117, 139, 140
qr (qr-methods), 114
qr, dgCMatrix-method (qr-methods), 114
qr, sparseMatrix-method (qr-methods), 114
qr-methods, 114
qr.coef, 128, 140
qr.coef, sparseQR, ddenseMatrix-method
(sparseQR-class), 139
qr.coef, sparseQR, Matrix-method
(sparseQR-class), 139
qr.coef, sparseQR, matrix-method
(sparseQR-class), 139
qr.coef, sparseQR, numeric-method
(sparseQR-class), 139
qr.fitted, 140
qr.fitted, sparseQR, ddenseMatrix-method
(sparseQR-class), 139
qr.fitted, sparseQR, Matrix-method
(sparseQR-class), 139
qr.fitted, sparseQR, matrix-method
(sparseQR-class), 139
qr.fitted, sparseQR, numeric-method
(sparseQR-class), 139
qr.Q, 140
qr.Q (sparseQR-class), 139
qr.Q, sparseQR-method (sparseQR-class), 139
qr.qty, 140
qr.qty, sparseQR, ddenseMatrix-method
(sparseQR-class), 139
qr.qty, sparseQR, Matrix-method
(sparseQR-class), 139
qr.qty, sparseQR, matrix-method
(sparseQR-class), 139
qr.qty, sparseQR, numeric-method
(sparseQR-class), 139
qr.resid, 140
qr.resid, sparseQR, ddenseMatrix-method
(sparseQR-class), 139
qr.resid, sparseQR, Matrix-method
(sparseQR-class), 139
qr.resid, sparseQR, matrix-method
(sparseQR-class), 139
qr.resid, sparseQR, numeric-method
(sparseQR-class), 139
qr.R (qr-methods), 114
range, 44
rankMatrix, 115
rBind (cBind), 15
rbind, 15, 138
rbind2, 15
rbind2, ANY, Matrix-method
(Matrix-class), 92
rbind2, atomicVector, ddiMatrix-method
(diagonalMatrix-class), 40
rbind2, atomicVector, ldiMatrix-method
(diagonalMatrix-class), 40
rbind2, atomicVector, Matrix-method
(Matrix-class), 92
rbind2, ddiMatrix, atomicVector-method
(diagonalMatrix-class), 40
rbind2, ddiMatrix, matrix-method
(diagonalMatrix-class), 40
rbind2, denseMatrix, denseMatrix-method
denseMatrix-class), 33
rbind2, denseMatrix, matrix-method
denseMatrix-class), 33
rbind2, denseMatrix, numeric-method
denseMatrix-class), 33
rbind2, denseMatrix, sparseMatrix-method
(sparseMatrix-class), 137
rbind2, diagonalMatrix, sparseMatrix-method
diagonalMatrix-class), 40
rbind2, indMatrix, indMatrix-method
indMatrix-class), 70
rbind2, ldiiMatrix, atomicVector-method
diagonalMatrix-class), 40
rbind2, ldiiMatrix, matrix-method
diagonalMatrix-class), 40
rbind2, Matrix, ANY-method
Matrix-class), 92
rbind2, Matrix, atomicVector-method
Matrix-class), 92
rbind2, matrix, ddiiMatrix-method
diagonalMatrix-class), 40
rbind2, matrix, denseMatrix-method
denseMatrix-class), 33
rbind2, matrix, ldiiMatrix-method
diagonalMatrix-class), 40
rbind2, Matrix, Matrix-method
Matrix-class), 92
rbind2, Matrix, missing-method
Matrix-class), 92
rbind2, Matrix, NULL-method
Matrix-class), 92
rbind2, matrix, sparseMatrix-method
(sparseMatrix-class), 137
rbind2, NULL, Matrix-method
Matrix-class), 92
rbind2, numeric, denseMatrix-method
denseMatrix-class), 33
rbind2, numeric, sparseMatrix-method
(sparseMatrix-class), 137
rbind2, sparseMatrix, denseMatrix-method
(sparseMatrix-class), 137
rbind2, sparseMatrix, diagonalMatrix-method
diagonalMatrix-class), 40
rbind2, sparseMatrix, matrix-method
(sparseMatrix-class), 137
rbind2, sparseMatrix, numeric-method
(sparseMatrix-class), 137
rbind2, sparseMatrix, sparseMatrix-method
(sparseMatrix-class), 137
rbind2, sparseMatrix, sparseMatrix-method
(sparseMatrix-class), 137
rcond, 29, 36, 45, 52, 118
rcond, ANY, missing-method (rcond), 118
rcond, ddenseMatrix, character-method
denseMatrix-class), 31
rcond, ddenseMatrix, missing-method
denseMatrix-class), 31
rcond, denseMatrix, character-method
denseMatrix-class), 33
rcond, dgeMatrix, character-method
dgeMatrix-class), 35
rcond, dgeMatrix, missing-method
dgeMatrix-class), 35
rcond, dpoMatrix, character-method
dpoMatrix-class), 44
rcond, dpoMatrix, missing-method
dpoMatrix-class), 44
rcond, dppMatrix, character-method
dpoMatrix-class), 44
rcond, dppMatrix, missing-method
dpoMatrix-class), 44
rcond, dspMatrix, character-method
dsyMatrix-class), 51
rcond, dspMatrix, missing-method
dsyMatrix-class), 51
rcond, dsyMatrix, character-method
dsyMatrix-class), 51
rcond, dsyMatrix, missing-method
dsyMatrix-class), 51
rcond, dtpMatrix, character-method
dtpMatrix-class), 54
rcond, dtpMatrix, missing-method
dtpMatrix-class), 54
rcond, dtrMatrix, character-method
dtrMatrix-class), 57
rcond, dtrMatrix, missing-method
dtrMatrix-class), 57
rcond, ldenseMatrix, character-method
(rcond), 118
rcond, Matrix, character-method (rcond), 118
rcond, ndenseMatrix, character-method
(rcond), 118
rcond, sparseMatrix, character-method
(rcond), 118
read.gal, 153
readHB (externalFormats), 60
readMM(externalFormats), 60
rep.sparseVector-method
(sparseVector-class), 142
rep.int, 120
rep2abI, 6, 120
replValue-class, 121
rle, 4–6, 121, 122
rleDiff-class, 121
round, 43
round,dgCMatrix,numeric-method
(dgCMatrix-class), 34
rowMeans, 93
rowMeans(colSums), 26
rowMeans,CsparseMatrix-method
(colSums), 26
rowMeans,ddenseMatrix-method(colSums), 26
rowMeans,denseMatrix-method
(colSums), 26
rowMeans,dgCMatrix-method
(dgeMatrix-class), 35
rowMeans,diagonalMatrix-method
(colSums), 26
rowMeans,lgCMatrix-method
(indMatrix-class), 70
rowMeans,ngCMatrix-method
(colSums), 26
rowMeans,RsparseMatrix-method
(colSums), 26
rowMeans,TsparseMatrix-method
(colSums), 26
rowSums,lgCMatrix-method(colSums), 26
rowSums,ngCMatrix-method(colSums), 26
rowSums,RsparseMatrix-method(colSums), 26
rowSums,TsparseMatrix-method(colSums), 26
RsparseMatrix, 37, 50
rsparsematrix, 122
RsparseMatrix-class, 123
sample.int, 122
Schur, 60, 124, 124, 125, 126, 148
Schur,dgeMatrix,logical-method(Schur),
124
Schur,dgeMatrix,missing-method(Schur),
124
Schur,diagonalMatrix,logical-method
(Schur), 124
Schur,diagonalMatrix,missing-method
(Schur), 124
Schur,dsyMatrix,ANY-method(Schur), 124
Schur,generalMatrix,logical-method
(Schur), 124
Schur,generalMatrix,missing-method
(Schur), 124
Schur,matrix,logical-method(Schur), 124
Schur,matrix,missing-method(Schur), 124
Schur,symmetricMatrix,logical-method
(Schur), 124
Schur,symmetricMatrix,missing-method
(Schur), 124
Schur,triangularMatrix,logical-method
(Schur), 124
Schur,triangularMatrix,missing-method
(Schur), 124
Schur-class, 125
seq, 6
seqMat-class(abIndex-class), 4
set.seed, 28
setClassUnion, 70, 109, 121, 142
show, 5, 65, 93, 96, 112, 113, 121, 137, 142
show,abIndex-method(abIndex-class), 4
show,BunchKaufman-method
(Cholesky-class), 24
show,ddenseMatrix-method
(ddenseMatrix-class), 31
show,denseMatrix-method
(denseMatrix-class), 33
INDEX

solve, dppMatrix, dgeMatrix-method
  (solve-methods), 126
solve, dppMatrix, integer-method
  (solve-methods), 126
solve, dppMatrix, matrix-method
  (solve-methods), 126
solve, dppMatrix, missing-method
  (solve-methods), 126
solve, dsCMatrix, ddenseMatrix-method
  (solve-methods), 126
solve, dsCMatrix, denseMatrix-method
  (solve-methods), 126
solve, dsCMatrix, dsparseMatrix-method
  (solve-methods), 126
solve, dsCMatrix, matrix-method
  (solve-methods), 126
solve, dsCMatrix, missing-method
  (solve-methods), 126
solve, dsCMatrix, numeric-method
  (solve-methods), 126
solve, dspMatrix, ddenseMatrix-method
  (solve-methods), 126
solve, dspMatrix, matrix-method
  (solve-methods), 126
solve, dspMatrix, missing-method
  (solve-methods), 126
solve, dspMatrix, numeric-method
  (solve-methods), 126
solve, dsysMatrix, ddenseMatrix-method
  (solve-methods), 126
solve, dsysMatrix, denseMatrix-method
  (solve-methods), 126
solve, dsysMatrix, matrix-method
  (solve-methods), 126
solve, dsysMatrix, missing-method
  (solve-methods), 126
solve, dtCMatrix, CsparseMatrix-method
  (solve-methods), 126
solve, dtCMatrix, dgeMatrix-method
  (solve-methods), 126
solve, dtCMatrix, matrix-method
  (solve-methods), 126
solve, dtCMatrix, missing-method
  (solve-methods), 126
solve, dtCMatrix, numeric-method
  (solve-methods), 126
solve, dtpMatrix, ddenseMatrix-method
  (solve-methods), 126
solve, dtpMatrix, matrix-method
  (solve-methods), 126
solve, dtpMatrix, missing-method
  (solve-methods), 126
solve, dtrMatrix, ddenseMatrix-method
  (solve-methods), 126
solve, dtrMatrix, dMatrix-method
  (solve-methods), 126
solve, dtrMatrix, Matrix-method
  (solve-methods), 126
solve, dtrMatrix, missing-method
  (solve-methods), 126
solve, Matrix, ANY-method
  (solve-methods), 126
solve, Matrix, diagonalMatrix-method
  (solve-methods), 126
solve, Matrix, matrix-method
  (solve-methods), 126
solve, matrix, Matrix-method
  (solve-methods), 126
solve, Matrix, missing-method
  (solve-methods), 126
solve, Matrix, numeric-method
  (solve-methods), 126
solve, Matrix, pMatrix-method
  (solve-methods), 126
solve, Matrix, sparseVector-method
  (solve-methods), 126
solve, MatrixFactorization, ANY-method
  (solve-methods), 126
solve, MatrixFactorization, missing-method
  (solve-methods), 126
solve, MatrixFactorization, numeric-method
  (solve-methods), 126
solve, pMatrix, Matrix-method
  (solve-methods), 126
solve, pMatrix, matrix-method
  (solve-methods), 126
solve, pMatrix, missing-method
  (solve-methods), 126
solve, sparseQR, ANY-method
  (solve-methods), 126
solve, TsparseMatrix, ANY-method
  (solve-methods), 126
solve, TsparseMatrix, missing-method
  (solve-methods), 126
solve-methods, 126
sort.list, 73
sparse.model.matrix, 129, 135, 138
sparselu, 89–90, 128
sparselu-class, 132
SparseM-coerce-methods
  (SparseM-conversions), 133
SparseM-conversions, 133
SparseM-ontology, 134
sparseMatrix, 9, 11, 15, 16, 30, 31, 33, 34,
  37, 40, 46, 47, 60, 64, 71, 75, 82, 88,
  91, 102, 113, 114, 122–124, 127,
  128, 130, 131, 134, 138, 141, 142,
  145, 149, 150, 152
sparseMatrix-class, 137
sparseqr, 97, 115
sparseqr-class, 139
sparseVector, 7, 26, 27, 75, 94, 141, 143
sparseVector-class, 142
spMatrix, 37, 38, 46, 114, 144, 149
sqrtm, 59
stop, 88, 155
subassign-methods (<=methods), 155
substring, 113
sum, 44
sum, ddiMatrix-method
  (diagonalMatrix-class), 40
sum, idMatrix-method
  (diagonalMatrix-class), 40
Summary, 143
Summary, abIndex-method (abIndex-class), 4
Summary, ddenseMatrix-method
  (ddenseMatrix-class), 31
Summary, ddiMatrix-method
  (diagonalMatrix-class), 40
summary, diagonalMatrix-method
  (diagonalMatrix-class), 40
Summary, dsparseMatrix-method
  (dsparseMatrix-class), 49
Summary, indMatrix-method
  (indMatrix-class), 70
Summary, ldenseMatrix-method
  (ldenseMatrix-class), 81
Summary, idMatrix-method
  (diagonalMatrix-class), 40
Summary, lMMatrix-method (dMatrix-class), 43
Summary, Matrix-method (Matrix-class), 92
Summary, ndenseMatrix-method
  (ndenseMatrix-class), 97
summary, nMatrix-method (nMatrix-class),
  102
Summary, nspvector-method
  (spvector-class), 142
Summary, pMatrix-method (pMatrix-class), 110
summary, sparseMatrix-method
  (sparseMatrix-class), 137
Summary, sparseVector-method
  (sparseVector-class), 142
svd, 93, 116, 117
symmetricMatrix, 10, 39, 48, 50, 63, 66, 76,
  86, 91, 94, 103, 104, 122, 128, 147,
  148, 150
symmetricMatrix-class, 146
symmpart, 63, 76, 98, 147
symmpart, ddenseMatrix-method
  (symmpart), 147
symmpart, denseMatrix-method (symmpart),
  147
symmpart, diagonalMatrix-method
  (symmpart), 147
symmpart, matrix-method (symmpart), 147
symmpart, matrix-method (symmpart), 147
symmpart, symmetricMatrix-method
  (symmpart), 147
symmpart-methods (symmpart), 147
t, 52, 83, 86, 87, 94, 101, 108–110, 116, 130
  t, CsparseMatrix-method
  (CsparseMatrix-class), 30
t, ddenseMatrix-method
  (ddenseMatrix-class), 31
t, dgCMatrix-method (dgCMatrix-class), 34
t, dgeMatrix-method (dgeMatrix-class), 35
t, dgRMatrix-method (dgRMatrix-class), 36
t, diagonalMatrix-method
  (diagonalMatrix-class), 40
t, dpmatrix-method (dpoMatrix-class), 44
t, dsCMatrix-method (dsCMatrix-class), 47
t, dspMatrix-method (dsyMatrix-class), 51
t, dtCMatrix-method (dtCMatrix-class), 52
t, dtpMatrix-method (dtpMatrix-class), 54
t, dtrMatrix-method (dtrMatrix-class), 57
t, dtTMatrix-method (dtCMatrix-class), 52
t, indMatrix-method (indMatrix-class), 70
<table>
<thead>
<tr>
<th>Method</th>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>t.lgcMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.lgeMatrix-method</td>
<td>lgeMatrix-class</td>
<td>83</td>
</tr>
<tr>
<td>t.lgtMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.lscMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.lspMatrix-method</td>
<td>lsyMatrix-class</td>
<td>85</td>
</tr>
<tr>
<td>t.lstMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.lysMatrix-method</td>
<td>lsyMatrix-class</td>
<td>85</td>
</tr>
<tr>
<td>t.ltpMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.ltrMatrix-method</td>
<td>ltrMatrix-class</td>
<td>86</td>
</tr>
<tr>
<td>t.lltMatrix-method</td>
<td>lsparseMatrix-classes</td>
<td>84</td>
</tr>
<tr>
<td>t.ltsMatrix-method</td>
<td>tsparseMatrix-class</td>
<td>149</td>
</tr>
<tr>
<td>t.matrix-method</td>
<td>matrix-class</td>
<td>92</td>
</tr>
<tr>
<td>tLngCMatrix-method</td>
<td>nsparseMatrix-classes</td>
<td>105</td>
</tr>
<tr>
<td>tLngEMatrix-method</td>
<td>ngematrix-class</td>
<td>101</td>
</tr>
<tr>
<td>tLngTMatrix-method</td>
<td>nsparseMatrix-classes</td>
<td>105</td>
</tr>
<tr>
<td>t.LnsyMatrix-method</td>
<td>nsyMatrix-class</td>
<td>107</td>
</tr>
<tr>
<td>t.LntMatrix-method</td>
<td>nsparseMatrix-classes</td>
<td>105</td>
</tr>
<tr>
<td>t.LntyMatrix-method</td>
<td>nsyMatrix-class</td>
<td>107</td>
</tr>
<tr>
<td>t.LntyMatrix-method</td>
<td>ntrMatrix-class</td>
<td>108</td>
</tr>
<tr>
<td>t.LnttMatrix-method</td>
<td>nsparseMatrix-classes</td>
<td>105</td>
</tr>
<tr>
<td>t.LtMatrix-method</td>
<td>pmatrix-class</td>
<td>110</td>
</tr>
<tr>
<td>t.LsbMatrix-method</td>
<td>RsparseMatrix-class</td>
<td>123</td>
</tr>
<tr>
<td>t.Lsymatrix-method</td>
<td>lsymatrix-class</td>
<td>85</td>
</tr>
<tr>
<td>t.Ltmatrix-method</td>
<td>ltmatrix-class</td>
<td>86</td>
</tr>
<tr>
<td>t.Ltrmatrix-method</td>
<td>ntrmatrix-class</td>
<td>108</td>
</tr>
<tr>
<td>t.Ltsmatrix-method</td>
<td>tsparsematrix-class</td>
<td>149</td>
</tr>
<tr>
<td>t.L2graph</td>
<td>(sparseMatrix-class)</td>
<td>66</td>
</tr>
<tr>
<td>tail.Matrix-method (Matrix-class)</td>
<td>(sparseVector-class)</td>
<td>142</td>
</tr>
<tr>
<td>tcrossprod</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ANY,Matrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ANY,symmetricMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ANY,TsparseMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod CsparseMatrix,CsparseMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod CsparseMatrix,diagonalMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod CsparseMatrix,matrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod CsparseMatrix,missing-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod CsparseMatrix,numLike-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,CsparseMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,dsCMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,dtMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,lsCMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,missing-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod ddenseMatrix,nsCMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod dgeMatrix,dgeMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod dgeMatrix,diagonalMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod dgeMatrix,matrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod dgeMatrix,missing-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod dgeMatrix,numLike-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
<tr>
<td>tcrossprod diagonalMatrix,CsparseMatrix-method</td>
<td>(matrix-products)</td>
<td>94</td>
</tr>
</tbody>
</table>
tcrossprod, diagonalMatrix, diagonalMatrix-method
tcrossprod, matrix, nsCMatrix-method
tcrossprod, diagonalMatrix, matrix-method
tcrossprod, diagonalMatrix, missing-method
tcrossprod, diagonalMatrix, sparseMatrix-method
tcrossprod, matrix, pMatrix-method
tcrossprod, dtrMatrix, dtrMatrix-method
tcrossprod, indMatrix, indMatrix-method
tcrossprod, indMatrix, missing-method
tcrossprod, lgCMatrix, missing-method
tcrossprod, lgeMatrix, diagonalMatrix-method
tcrossprod, lgtMatrix, missing-method
tcrossprod, lsparseMatrix, missing-method
tcrossprod, Matrix, ANY-method
tcrossprod, matrix, CsparseMatrix-method
tcrossprod, matrix, dgeMatrix-method
tcrossprod, matrix, diagonalMatrix-method
tcrossprod, matrix, dsCMatrix-method
tcrossprod, matrix, dtrMatrix-method
tcrossprod, Matrix, indMatrix-method
tcrossprod, matrix, indMatrix-method
tcrossprod, matrix, lsCMatrix-method
tcrossprod, Matrix, Matrix-method
tcrossprod, matrix, matrix-method
tcrossprod, matrix, matrix-method
tcrossprod, Matrix, missing-method
tcrossprod, TsparseMatrix, ANY-method

tcrossprod, Matrix, numLike-method
tcrossprod, Matrix, pMatrix-method
tcrossprod, Matrix, symmetricMatrix-method
tcrossprod, Matrix, TsparseMatrix-method
tcrossprod, mMatrix, sparseVector-method
tcrossprod, ngCMatrix, missing-method
tcrossprod, ngTMatrix, missing-method
tcrossprod, nsparseMatrix, missing-method
tcrossprod, numLike, CsparseMatrix-method
tcrossprod, numLike, dgeMatrix-method
tcrossprod, numLike, Matrix-method
tcrossprod, numLike, sparseMatrix-method
tcrossprod, pMatrix, missing-method
tcrossprod, pMatrix, pMatrix-method
tcrossprod, sparseMatrix, diagonalMatrix-method
tcrossprod, sparseMatrix, sparseVector-method
tcrossprod, sparseVector, missing-method
tcrossprod, sparseVector, mMatrix-method
tcrossprod, sparseVector, numLike-method
tcrossprod, sparseVector, sparseMatrix-method
tcrossprod, sparseVector, sparseVector-method
tcrossprod, TsparseMatrix, ANY-method
INDEX

tcrossprod,TsparseMatrix,Matrix-method (matrix-products), 94
tcrossprod,TsparseMatrix,missing-method (matrix-products), 94
tcrossprod,TsparseMatrix,TsparseMatrix-methodTriu dtRMatrix-method (band), 9
tcrossprod-methods (matrix-products), 94
toeplitz, 143
toeplitz (sparseVector-class), 142
toeplitz,sparseVector-method (sparseVector-class), 142
triangularMatrix, 9, 21, 42, 43, 53, 55–58, 63, 66, 77, 87, 91, 95, 108, 146, 150
triangularMatrix-class, 148
tril (band), 9
tril,CsparseMatrix-method (band), 9
tril,ddenseMatrix-method (band), 9
tril,denseMatrix-method (band), 9
tril,diagonalMatrix-method (band), 9
tril,dsCMatrix-method (band), 9
tril,dtCMatrix-method (band), 9
tril,dtMatrix-method (band), 9
tril,ntCMatrix-method (band), 9
tril,ntMatrix-method (band), 9
tril,ntRMatrix-method (band), 9
tril,nttmatrix-method (band), 9
tril,nttpmatrix-method (band), 9
tril,nttrmatrix-method (band), 9
tril,rsparsematrix-method (band), 9
triu,diagonalMatrix-method (band), 9
triu,dCMatrix-method (band), 9
triu,dtCMatrix-method (band), 9
triu,dtMatrix-method (band), 9
triu,dtRMatrix-method (band), 9
triu,dtTMatrix-method (band), 9
triu,ltCMatrix-method (band), 9
triu,ltMatrix-method (band), 9
triu,ltRMatrix-method (band), 9
triu,lttmatrix-method (band), 9
triu,ltttmatrix-method (band), 9
triu,lttpmatrix-method (band), 9
triu,lttrmatrix-method (band), 9
triu,ltTMatrix-method (band), 9
triu,matrix-method (band), 9
triu,ncMatrix-method (band), 9
triu,ntCMatrix-method (band), 9
triu,ntMatrix-method (band), 9
triu,ntRMatrix-method (band), 9
triu,nttmatrix-method (band), 9
triu,rsparsematrix-method (band), 9
triu,TsparseMatrix-method (band), 9
triu-methods (band), 9
TRUE, 39, 44, 75, 140
TsparseMatrix, 10, 12, 38, 48, 66, 130, 134, 135, 144, 145, 150
TsparseMatrix-class, 149
type, 155

unpack, 150
unpack, dspMatrix-method (unpack), 150
unpack, dtMatrix-method (unpack), 150
unpack, sparseMatrix-method (unpack), 150
unpack, symmetricMatrix-method (unpack), 150

unpack, triangularMatrix-method (unpack), 150

Unused-classes, 151
update, 18
update,CHMfactor-method (CHMfactor-class), 16
updown, 152
updown, ANY, ANY, ANY-method (updown), 152
validObject, 31

warning, 88
which, 41, 44, 81, 85, 97, 106
which, ldenseMatrix-method
   (ldenseMatrix-class), 81
which, ldiMatrix-method
   (diagonalMatrix-class), 40
which, lgtMatrix-method
   (lsparseMatrix-classes), 84
which, lsparseMatrix-method
   (lsparseMatrix-classes), 84
which, lsparseVector-method
   (lsparseMatrix-classes), 84
which, lstMatrix-method
   (lsparseMatrix-classes), 84
which, lttMatrix-method
   (lsparseMatrix-classes), 84
which, ndenseMatrix-method
   (ndenseMatrix-class), 97
which, ngtMatrix-method
   (nsparseMatrix-classes), 105
which, nsparseMatrix-method
   (nsparseMatrix-classes), 105
which, nsparseVector-method
   (nsparseMatrix-classes), 105
which, nstMatrix-method
   (nsparseMatrix-classes), 105
which, ntTMatrix-method
   (nsparseMatrix-classes), 105
writeMM, 138
writeMM(externalFormats), 60
writeMM, CsparseMatrix-method
   (externalFormats), 60
writeMM, sparseMatrix-method
   (externalFormats), 60

xsparsesVector-class
   (sparseVector-class), 142
xtabs, 135, 138

zapsmall, 46
zapsmall, dMatrix-method
   (dMatrix-class), 43
zMatrix-class (Unused-classes), 151
zsparsesVector-class
   (sparseVector-class), 142