Package ‘Matrix’

November 11, 2022

Version 1.5-3
Date 2022-11-09
Priority recommended
Title Sparse and Dense Matrix Classes and Methods
Contact Matrix-authors@R-project.org
Maintainer Martin Maechler <mmaechler+Matrix@gmail.com>
Description A rich hierarchy of matrix classes, including triangular, symmetric, and diagonal matrices, both dense and sparse and with pattern, logical and numeric entries. Numerous methods for and operations on these matrices, using 'LAPACK' and 'SuiteSparse' libraries.
Depends R (>= 3.5.0), methods
Imports graphics, grid, lattice, stats, utils
Suggests expm, MASS
Enhances MatrixModels, graph, SparseM, sfsmisc, igraph, maptools, sp, spdep
EnhancesNote line 2: for ```Rd xrefs```
Encoding UTF-8
LazyData no
LazyDataNote not possible, since we use data/*.R *and* our classes
BuildResaveData no
License GPL (>= 2) | file LICENCE
BugReports https://R-forge.R-project.org/tracker/?atid=294&group_id=61
NeedsCompilation yes
Author Douglas Bates [aut],
  Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>),
  Mikael Jagan [aut] (<https://orcid.org/0000-0002-3542-2938>),
  Timothy A. Davis [ctb] (SuiteSparse and 'cs' C libraries, notably
CHOLMOD and AMD, collaborators listed in
dir(pattern=``^[A-Z]+\.[.]txt$'', full.names=TRUE, system.file(``doc``,
``SuiteSparse'', package=``Matrix''))),
Jens Oehlschlägel [ctb] (initial nearPD()),
Jason Riedy [ctb] (condest() and onenormest() for octave, Copyright:
Regents of the University of California),
R Core Team [ctb] (base R matrix implementation)

Repository  CRAN
Date/Publication  2022-11-11 08:30:10 UTC

R topics documented:

abolIndex-class ................................................. 5
abIseq ........................................................... 6
all-methods ....................................................... 7
all.equal-methods ............................................... 8
atomicVector-class ........................................... 8
band ............................................................. 9
bandSparse ...................................................... 10
bdiaq ........................................................... 12
BunchKaufman-methods ....................................... 14
CAex ............................................................ 15
cBind .......................................................... 16
CHMfactor-class ............................................... 18
chol ............................................................ 21
chol2inv-methods .............................................. 23
Cholesky ......................................................... 24
Cholesky-class .................................................. 26
colSums ........................................................ 28
compMatrix-class .............................................. 29
condest .......................................................... 30
CsparseMatrix-class .......................................... 32
ddenseMatrix-class ........................................... 34
ddiMatrix-class ................................................ 35
denseMatrix-class ............................................ 36
dgCMatrix-class ................................................ 36
dgEMatrix-class ................................................ 37
dgRMatrix-class ............................................... 39
dgTMatrix-class ............................................... 40
Diagonal ........................................................ 41
diagonalMatrix-class ........................................ 43
diagU2N ........................................................ 44
dimScale ......................................................... 46
dMatrix-class .................................................. 47
dmperm ........................................................ 48
dpoMatrix-class ............................................... 50
drop0 ............................................................ 52
Topics documented:

- dsCMatrix-class .................................................. 53
- dsparseMatrix-class ............................................. 55
- dsRMatrix-class ................................................... 56
- dsyMatrix-class ................................................... 57
- dtCMatrix-class ................................................... 59
- dtpMatrix-class ................................................... 61
- dtRMatrix-class ................................................... 62
- dtrMatrix-class ................................................... 63
- expand .............................................................. 65
- expm ................................................................. 66
- externalFormats .................................................... 67
- facmul ............................................................... 68
- forceSymmetric ..................................................... 70
- formatSparseM ..................................................... 71
- generalMatrix-class ............................................. 72
- graph-sparseMatrix .............................................. 73
- Hilbert ............................................................. 74
- image-methods ..................................................... 75
- index-class ........................................................ 77
- indMatrix-class ................................................... 78
- invPerm ............................................................. 80
- is.na-methods ....................................................... 81
- is.null.DN .......................................................... 83
- isSymmetric-methods ............................................. 84
- isTriangular ......................................................... 86
- KhatriRao .......................................................... 87
- KNex ................................................................. 89
- kronecker-methods ................................................. 90
- ldenseMatrix-class .............................................. 91
- ldiMatrix-class ................................................... 92
- lgeMatrix-class ................................................... 92
- lsparseMatrix-classes .......................................... 93
- lsyMatrix-class ................................................... 95
- ltrMatrix-class ................................................... 96
- lu .................................................................. 97
- LU-class ............................................................ 100
- mat2triplet ........................................................ 101
- Matrix ............................................................. 102
- Matrix-class ...................................................... 104
- matrix-products .................................................. 106
- MatrixClass ........................................................ 108
- MatrixFactorization-class .................................... 109
- ndenseMatrix-class ............................................. 110
- nearPD .............................................................. 111
- ngeMatrix-class .................................................. 114
- nMatrix-class ..................................................... 115
- nnzero .............................................................. 116
- norm ................................................................. 118
### R topics documented:

- `nsparseMatrix-class` .................................................. 119
- `nsyMatrix-class` ......................................................... 121
- `ntrMatrix-class` ......................................................... 122
- `number-class` ............................................................. 123
- `packedMatrix-class` ..................................................... 123
- `pMatrix-class` ............................................................ 125
- `printSpMatrix` ............................................................ 127
- `qr-methods` ............................................................... 130
- `rankMatrix` ............................................................... 132
- `rcond` ..................................................................... 135
- `rep2abI` .................................................................. 137
- `replValue-class` .......................................................... 138
- `rleDiff-class` ............................................................. 138
- `rsparsematrix` ............................................................. 139
- `RsparseMatrix-class` ..................................................... 140
- `Schur` .................................................................. 141
- `Schur-class` ............................................................... 143
- `solve-methods` ............................................................. 144
- `sparse.model.matrix` ..................................................... 147
- `sparseLU-class` ............................................................ 149
- `SparseM-conversions` ..................................................... 151
- `sparseMatrix` .............................................................. 151
- `sparseMatrix-class` ..................................................... 155
- `sparseQR-class` ........................................................... 157
- `sparseVector` .............................................................. 159
- `sparseVector-class` ..................................................... 160
- `spMatrix` ................................................................. 163
- `symmetricMatrix-class` ................................................ 164
- `symmpart` ................................................................. 166
- `triangularMatrix-class` ................................................. 167
- `TsparseMatrix-class` .................................................... 168
- `uniqTsparse` .............................................................. 169
- `unpack` .................................................................. 171
- `unpackedMatrix-class` .................................................. 173
- `Unused-classes` ........................................................... 174
- `updown` ................................................................. 174
- `USCounties` .............................................................. 175
- `wrlld_1deg` ............................................................... 176
- `[-methods` ............................................................... 178
- `[<-methods` ............................................................. 178
- `%&%-%-methods` .......................................................... 179

### Index

182
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"): ...

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

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

ccoeerce 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

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

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

abIseq

Sequence Generation of "abIndex", Abstract Index Vectors

Description
Generation of abstract index vectors, i.e., objects of class "abIndex".
abIseq() is designed to work entirely like seq, but producing "abIndex" vectors.
abIseq1() 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

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

## 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 \texttt{R} objects; here, when the first is an \texttt{"abIndex"} vector, these arguments will be concatenated to a new \texttt{"abIndex"} object.

**Value**

An abstract index vector, i.e., object of class \texttt{"abIndex"}.

**See Also**

the class \texttt{abIndex} documentation; \texttt{rep2abI()} for another constructor; \texttt{rle} \texttt{(base)}.

**Examples**

```r
test <- c(1, 2, 3)
abTest <- abIndex(test)
all(abTest >= 1) # TRUE
any(abTest < 0 ) # FALSE
```

---

**Description**

The basic \texttt{R} functions \texttt{all} and \texttt{any} now have methods for \texttt{Matrix} objects and should behave as for \texttt{matrix} ones.

**Methods**

- \texttt{all} signature(x = "Matrix", \ldots, na.rm = FALSE):...
- \texttt{any} signature(x = "Matrix", \ldots, na.rm = FALSE):...
- \texttt{all} signature(x = "ldenseMatrix", \ldots, na.rm = FALSE):...
- \texttt{all} signature(x = "lsparseMatrix", \ldots, 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
```
Methods for function `all.equal()` (from R package `base`) are defined for all `Matrix` classes.

Methods

- `target = "Matrix", current = "Matrix"`
- `target = "ANY", current = "Matrix"`
- `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

```r
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[lu.@p + 1L, lu.@q + 1L], "CsparseMatrix"),
                    lu.@L %*% 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))))
```

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.
**band**

**Description**

Return the matrix obtained by setting to zero elements below a diagonal (`triu`), above a diagonal (`tril`), or outside of a general band (`band`).

**Usage**

```r
band(x, k1, k2, ...)  
triu(x, k = 0, ...)  
tril(x, k = 0, ...)  
```

**Arguments**

- `x`: a matrix-like object
- `k, k1, k2`: integers specifying the diagonals that are not set to zero. These are interpreted relative to the main diagonal, which is `k=0`. Positive and negative values of `k` indicate diagonals above and below the main diagonal, respectively.
- `...`: optional arguments passed methods (currently unused by package Matrix).

**Details**

`triu(x, k)` is equivalent to `band(x, k, dim(x)[2])`. Similarly, `tril(x, k)` is equivalent to `band(x, -dim(x)[1], k).`

**Value**

An object of a suitable matrix class, inheriting from `triangularMatrix` where appropriate. It inherits from `sparseMatrix` if and only if `x` does.
Methods

- **x = "CsparseMatrix"** method for compressed, sparse, column-oriented matrices.
- **x = "RsparseMatrix"** method for compressed, sparse, row-oriented matrices.
- **x = "TsparseMatrix"** method for sparse matrices in triplet format.
- **x = "diagonalMatrix"** method for diagonal matrices.
- **x = "denseMatrix"** method for dense matrices in packed or unpacked format.
- **x = "matrix"** method for traditional matrices of implicit class `matrix`.

See Also

`bandSparse` for the 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), ncol = 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), ncol = 5)) # not square
triu(m65) # result not "dtrMatrix" unless square
(sm5 <- crossprod(m65)) # symmetric
band(sm5, -1, 1) # "dsyMatrix": symmetric band preserves symmetry property
as(band(sm5, -1, 1), "sparseMatrix") # often preferable
(sm <- round(crossprod(triu(mm/2)))) # sparse symmetric ("dsC*")
band(sm, -1, 1) # remains "dsC", however
band(sm, -2, 1) # -> "dgC"
```

---

**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

bandSparse(n, m = n, k, diagonals, symmetric = FALSE, repr = "C", giveCsparse = (repr == "C"))

Arguments

- **n, m**: the matrix dimension \((n, m) = (nrow, ncol)\).
- **k**: integer vector of “diagonal numbers”, with identical meaning as in band(*, k), i.e., relative to the main diagonal, which is \(k=0\).
- **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 \(d \leftarrow length(k)\) and \(n' \geq min(n, m)\).
  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).
- **repr**: character string, one of "C", "T", or "R", specifying the sparse representation to be used for the result, i.e., one from the super classes CsparseMatrix, TsparseMatrix, or RsparseMatrix.
- **giveCsparse**: (deprecated, replaced with repr): logical indicating if the result should be a CsparseMatrix or a TsparseMatrix, where the default was TRUE, and now is determined from repr; very often Csparse matrices are 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), "nsparseMatrix"))

n <- 1e4
bk <- c(0:5, 7:11)
bMat <- matrix(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 xor a matrix for specifying the diagonals:
stopifnot(identical(B, bandSparse(n, k = bk, diag = bLis)),
         identical(Bs, bandSparse(n, k = bk, diag = bLis, symmetric=TRUE))
         , inherits(B, "dtCMatrix") # triangular!)

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.
lst 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() inherits from class CsparseMatrix, whereas .bdiag() returns a TsparseMatrix.

**Note**

This function has been written and is efficient for the case of relatively few block matrices which are typically sparse themselves.

It is currently inefficient for the case of many small dense block matrices. For the case of many dense $k \times k$ matrices, the bdiag_m() function in the ‘Examples’ is an order of magnitude faster.
**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**

```r
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") )

## random (diagonal-)block-triangular matrices:
rblockTri <- function(nb, max.ni, lambda = 3) {
  .bdiag(replicate(nb, {
    n <- sample.int(max.ni, 1)
    tril(Matrix(rpois(n*n, lambda=lambda), n,n)) )))
}
(T4 <- rblockTri(4, 10, lambda = 1))
image(T1 <- rblockTri(12, 20))
```

```r
# Fast version of Matrix :: .bdiag() -- for the case of many (k x k) matrices:
# @param lmat list(<mat1>, <mat2>, ......, <mat_N>) where each mat_j is a k x k 'matrix'
# @return a sparse (N*k x N*k) matrix of class 'dgCMatrix'

bdiag_m <- function(lmat) {
  ## Copyright (C) 2016 Martin Maechler, ETH Zurich
  if(!length(lmat)) return(new("dgCMatrix"))
  stopifnot(is(list(lmat), is.matrix(lmat[[1]])],
    (k <- (d <- dim(lmat[[1]])[1]) == d[2], # k x k
     all(vapply(lmat, dim, integer(2)) == k)) # all of them
  N <- length(lmat)
  if(N * k > .Machine$integer.max)
```
stop("resulting matrix too large; would be M x M, with M=", N*k)
M <- as.integer(N * k)
## result: an M x M matrix
new("dgCMatrix", Dim = c(M,M),
    ## 'i' maybe there's a faster way (w/o matrix indexing), but elegant?
    i = as.vector(matrix(0L:(M-1L), nrow=k)[, rep(seq_len(N), each=k)]),
    p = k * 0L:M,
    x = as.double(unlist(lmat, recursive=FALSE, use.names=FALSE))
)

l12 <- replicate(12, matrix(rpois(16, lambda = 6.4), 4,4), simplify=FALSE)
dim(T12 <- bdiag_m(l12))# 48 x 48
T12[1:20, 1:20]

BunchKaufman-methods

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 $2 \times 2$.

This is generalization of a pivoting $LDL'$ Cholesky decomposition.

Usage

## S4 method for signature 'dsyMatrix'
BunchKaufman(x, ...)
## S4 method for signature 'dspMatrix'
BunchKaufman(x, ...)
## S4 method for signature 'matrix'
BunchKaufman(x, uplo = NULL, ...)

Arguments

x a symmetric square matrix.
uplo optional string, "U" or "L" indicating which "triangle" half of x should determine the result. The default is "U" unless x has a uplo slot which is the case for those inheriting from class symmetricMatrix, where x@uplo will be used.
... potentially further arguments passed to methods.

Details

FIXME: We really need an expand() method in order to work with the result!

Value

an object of class BunchKaufman, which can also be used as a (triangular) matrix directly. Somewhat amazingly, it inherits its uplo slot from x.
Methods

Currently, only methods for dense numeric symmetric matrices are implemented. To compute the Bunch-Kaufman decomposition, the methods use either one of two Lapack routines:

- \texttt{x = "dspMatrix"} routine \texttt{dsptrf()}, whereas
- \texttt{x = "dsyMatrix"}, and
- \texttt{x = "matrix"} use \texttt{dsytrf}().

References

The original LAPACK source code, including documentation: https://netlib.org/lapack/double/dsytrf.f and https://netlib.org/lapack/double/dsptrf.f

See Also

The resulting class, \texttt{BunchKaufman}. Related decompositions are the LU, \texttt{lu}, and the Cholesky, \texttt{chol} (and for sparse matrices, \texttt{Cholesky}).

Examples

data(CAex)
dim(CAex)
isSymmetric(CAex)# TRUE

\begin{verbatim}
CAs <- as(CAex, "symmetricMatrix")
if(FALSE) # no method defined yet for *sparse* :
bk. <- BunchKaufman(CAs)
## does apply to *dense* symmetric matrices:
bkCA <- BunchKaufman(as(CAs, "denseMatrix"))
bkCA
pkCA <- pack(bkCA)
stopifnot(is(bkCA, "triangularMatrix"),
          is(pkCA, "triangularMatrix"),
          is(pkCA, "packedMatrix"))
\end{verbatim}

image(bkCA)# shows how sparse it is, too
str(R.CA <- as(bkCA, "sparseMatrix"))
## an upper triangular 72x72 matrix with only 144 non-zero entries
stopifnot(is(R.CA, "triangularMatrix"), is(R.CA, "CsparseMatrix"))

\begin{verbatim}
CAex
\end{verbatim}

\textit{Albers’ example Matrix with “Difficult” Eigen Factorization}

Description

An example of a sparse matrix for which \texttt{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 \texttt{dgCMatrix}.

Details

Historical note (2006-03-30): In earlier versions of \texttt{R}, \texttt{eigen(CAex)} fell into an infinite loop whereas \texttt{eigen(CAex, EISPACK=TRUE)} had 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, 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"))

---

\texttt{cBind}

\texttt{cbind()} and \texttt{rbind()} recursively built on \texttt{cbind2/rbind2}

Description

The base functions \texttt{cbind} and \texttt{rbind} are defined for an arbitrary number of arguments and hence have the first formal argument \ldots. Now, when S4 objects are found among the arguments, base \texttt{cbind()} and \texttt{rbind()} internally “dispatch” recursively, calling \texttt{cbind2} or \texttt{rbind2} respectively, where these have methods defined and so should dispatch appropriately.

\texttt{cbind2()} and \texttt{rbind2()} are from the \texttt{methods} package, i.e., standard \texttt{R}, and have been provided for binding together two matrices, where in \texttt{Matrix}, we have defined methods for these and the \texttt{Matrix} matrices.

Usage

\texttt{## cbind(..., deparse.level = 1)}
\texttt{## rbind(..., deparse.level = 1)}

\texttt{## and e.g.,}
\texttt{## S4 method for signature 'denseMatrix,sparseMatrix'}
\texttt{cbind2(x,y, sparse = NA, \ldots)}
## S4 method for signature 'sparseMatrix,denseMatrix'
cbind2(x, y, sparse = NA, ...)
## S4 method for signature 'denseMatrix,sparseMatrix'
rbind2(x, y, sparse = NA, ...)
## S4 method for signature 'sparseMatrix,denseMatrix'
rbind2(x, y, sparse = NA, ...)

**Arguments**

..., x, y    matrix-like R objects to be bound together, see `cbind` and `rbind`.
sparse    option logical indicating if the result should be sparse, i.e., formally inheriting from "sparseMatrix". The default, NA, decides from the “sparsity” of x and y, see e.g., the R code in selectMethod(cbind2, c("sparseMatrix","denseMatrix")).
deparse.level    integer determining under which circumstances column and row names are built from the actual arguments’ ‘expression’, see `cbind`.

**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()`).

**Historical Remark**

Before R version 3.2.0 (April 2015), we have needed a substitute for S4-enabled versions of `cbind` and `rbind`, and provided cBind and rBind with identical syntax and semantic in order to bind together multiple matrices ("matrix" or "Matrix" and vectors. With R version 3.2.0 and newer, cBind and rBind are deprecated and produce a deprecation warning (via `.Deprecated`), and your code should start using `cbind()` and `rbind()` instead.

**Author(s)**

Martin Maechler

**See Also**

cbind2, cbind. Documentation in base R’s methods package.

Our class definition help pages mentioning cbind2() and rbind2() methods: “denseMatrix”, “diagonalMatrix”, “indMatrix”.

**Examples**

(a <- matrix(c(2:1,1:2), 2,2))

(M1 <- cbind(0, rbind(a, 7))) # a traditional matrix

D <- Diagonal(2)
(M2 <- cbind(4, a, D, -1, D, 0)) # a sparse Matrix
stopifnot(validObject(M2), inherits(M2, "sparseMatrix"),
  dim(M2) == c(2,9))

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","LDLt","LD","DLt","L","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 dsCMatrix or lsCMatrix 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(x = "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(x = "CHMfactor"): Plot the image of the lower triangular factor, L, from the decomposition. This method is equivalent to image(as(x, "sparseMatrix")) so the comments in the above description of the coerce method apply here too.

solve signature(a = "CHMfactor", b = "ddenseMatrix"), system= *:
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 $Ax = b$ where A is the sparse, positive-definite matrix that was factored to produce a. Analogously, system = "L" returns the solution $x$ of $Lx = b$. Similarly, for all system codes but "P" and "Pt" where, e.g., $x <- solve(a, b, system="P")$ is equivalent to $x <- P %*% b$. See also solve-methods.
determinant signature(x = "CHMfactor", logarithm = "logical") returns the determinant (or the logarithm of the determinant, if logarithm = 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 \( \text{logarithm} = \text{TRUE} \)) of the positive-definite matrix that was decomposed.

update signature(object = "CHMfactor"), parent. The update method requires an additional argument parent, which is 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 \( A := MM' (= \text{tcrossprod(parent))} \) is used for \( A \). Further it provides an optional argument \( \text{mult} \), 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 \( \text{mult} \). 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.

See Also

Cholesky, also for examples; class dgCMatrix.

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 = TRUE),
  j = sample(m, nnz, replace = TRUE),
  x = round(rnorm(nnz),1))
XX <- crossprod(M1) ## = M1 = M M' 
XX <- crossprod(M1)
CX <- Cholesky(XX)
isLDL(CX)
str(CX) # a "dCHMsimpl" object
r <- expand(CX)
L.P <- with(r, crossprod(L,P)) # == L'P
PLLP <- crossprod(L.P) # == (L'P)' L'P == P'LL'P = XX = M M'
b <- sample(m)
stopifnot(all.equal(PLLP, XX),
  all(as.vector(solve(CX, b, system="P" )) == r$P %*% b),
  all(as.vector(solve(CX, b, system="Pt")) == t(r$P) %*% b ) )

u1 <- update(CX, XX, mult=pi)
u2 <- update(CX, t(M1), mult=pi) # with the original M, where XX = M M'
stopifnot(all.equal(u1,u2, tol=1e-14))
```

## [ See help(Cholesky) for more examples ]

### [-------------------]
The Cholesky Decomposition - 'Matrix' S4 Generic and Methods

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

Usage
chol(x, ...)
## S4 method for signature 'dsyMatrix'
chol(x, ...)
## S4 method for signature 'dspMatrix'
chol(x, ...)
## S4 method for signature 'dsCMatrix'
chol(x, pivot = FALSE, ...)
## S4 method for signature 'dsRMMatrix'
chol(x, pivot = FALSE, cache = TRUE, ...)
## S4 method for signature 'dsTMMatrix'
chol(x, pivot = FALSE, cache = TRUE, ...)

Arguments
x a (sparse or dense) square matrix, here inheriting from class Matrix; if x is not symmetric positive definite, then 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 certain classes inheriting from compMatrix.
... 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 = "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 of x, via LAPACK routine `dpptrf`.
- **chol signature(x = "dsyMatrix")**: works via "dpoMatrix", see class `dpoMatrix`.
- **chol signature(x = "dspMatrix")**: works via "dppMatrix", see class `dppMatrix`.
- **chol signature(x = "dsCMatrix")**: Returns (and stores) the Cholesky decomposition of x. If pivot is TRUE, then 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.
- **chol signature(x = "dsRMatrix")**: works via "dsCMatrix", see class `dsCMatrix`.
- **chol signature(x = "dsTMatrix")**: works via "dsCMatrix", see class `dsCMatrix`.

References


See Also

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

Examples

```r
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")))
c2 <- chol(M2)
C2 <- chol(M2, pivot=TRUE)

## For the experts, check the caching of the factorizations:
ff <- M2@factors["spdCholesky"]
FF <- M2@factors["sPdCholesky"]
L1 <- as(ff, "Matrix")# pivot=FALSE: no perm.
L2 <- as(FF, "Matrix"); P2 <- as(FF, "pMatrix")
stopifnot(identical(t(L1), c2),
  all.equal(t(L2), C2, tolerance=0),#-- why not identical()? all.equal(M2, tcrossprod(L1)),
   # M = LL'
  all.equal(M2, crossprod(crossprod(L2, P2)))# M = P'L L'P
)

---

**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 \((R)\) 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**

(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)
Cholesky Decomposition of a Sparse Matrix

Description

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

Usage

Cholesky(A, perm = TRUE, LDL = !super, super = FALSE, Imult = 0, ...)

Arguments

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

Details

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

The method for class 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 chol(.).

Note that if perm=TRUE (default), the decomposition is
\[ A = P' \tilde{L} \tilde{D} \tilde{L}' P = P' L L' P, \]
where L can be extracted by as(*, "Matrix"), P by as(*, "pMatrix") and both by expand(*), see the class CHMfactor documentation.

Note that consequently, you cannot easily get the “traditional” cholesky factor R, from this decomposition, as
\[ R' R = A = P' L L' P = P' \tilde{R} \tilde{R} P = (\tilde{R} P)' (\tilde{R} P), \]
but \( \tilde{R} P \) is not triangular even though \( \tilde{R} \) is.
Cholesky

Value

an object inheriting from either "CHMsuper", 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


See Also

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

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

Examples

data(KNex)
mtm <- with(KNex, crossprod(mm))
str(mtm@factors) # empty list()
(C1 <- Cholesky(mtm)) # uses show(<MatrixFactorization>)
str(mtm@factors) # 'sPDCholesky' (simpl)
(Cm <- Cholesky(mtm, super = TRUE))
c(C1 = isLDL(C1), Cm = isLDL(Cm))
str(mtm@factors) # 'sPDCholesky' *and* 'SPdCholesky'
str(cm1 <- as(C1, "sparseMatrix"))
str(cmat <- as(Cm, "sparseMatrix")) # hmm: super is *less* sparse here
cm1[1:20, 1:20]
b <- matrix(c(rep(0, 711), 1), ncol = 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(ckbind(format(100 * sizes / sizes[1], digits=4)))
## Visualize the sparseness:

dq <- function(ch) paste(""",ch,""", sep="")  # dQuote(<UTF-8>) gives bad plots
image(mtm, main=paste("crossprod(mtm) : Sparse", dq(class(mtm))))
image(cm1, main= paste("as(Cholesky(crossprod(mm)),\"sparseMatrix\"):",
  dq(class(cm1))))

## Smaller example, with same matrix as in help(chol):

(mm <- Matrix(toeplitz(c(10, 0, 1, 0, 3)), sparse = TRUE)) # 5 x 5
(opts <- expand.grid(perm = c(TRUE,FALSE), LDL = c(TRUE,FALSE), super = c(FALSE,TRUE)))
rr <- lapply(seq_len(nrow(opts)), function(i)
  do.call(Cholesky, c(list(A = mm), opts[i,])))
nn <- do.call(expand.grid, c(attr(opts, "out.attrs")$dimnames,
  stringsAsFactors=FALSE,KEEP.OUT.ATTRS=FALSE))
names(rr) <- apply(nn, 1, function(r)
  paste(sub("(=.).*","\1", r), collapse=","))
str(rr, max.level=1)

str(re <- lapply(rr, expand), max.level=2)  # each has a 'P' and a 'L' matrix

R0 <- chol(mm, pivot=FALSE)
R1 <- chol(mm, pivot=TRUE)
stopifnot(all.equal(t(R1), re[[1]]$L),
  all.equal(t(R0), re[[2]]$L),
  identical(as(1:5, "pMatrix"), re[[2]]$P), # no pivoting
  TRUE)

# Version of the underlying SuiteSparse library by Tim Davis :
.SuiteSparse_version()

---

**Cholesky-class**

Cholesky and Bunch-Kaufman Decompositions

### 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* × *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* <- *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 <- 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 <- pack(Matrix(diag(5) + 1))) # 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 objects, for sparseMatrix the result may optionally be sparse (sparseVector), too. Row or column names are kept respectively as for base matrices and colSums methods, when the result is numeric vector.

Usage

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

## S4 method for signature 'CsparseMatrix'
colSums(x, na.rm = FALSE, dims = 1L, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
rowSums(x, na.rm = FALSE, dims = 1L, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
colMeans(x, na.rm = FALSE, dims = 1L, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
rowMeans(x, na.rm = FALSE, dims = 1L, 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. Only applicable when x is inheriting from a sparseMatrix class.
Value

returns a numeric vector if sparseResult is FALSE as per default. Otherwise, returns a sparseVector.
dimnames(x) are only kept (as names(v)) when the resulting v is numeric, since sparseVectors do not have names.

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)

## dimnames(x) --> names( <value> ) :
dimnames(M) <- list(paste0("r", 1:7), paste0("V",1:8))
M
colSums(M)
rowMeans(M)

## Assertions :
stopifnot(all.equal(colSums(M),
  setNames(c(1,1,6,6,6,6,3,2), colnames(M))),
  all.equal(rowMeans(M), structure(c(1,1,4,8,12,3,2) / 8,
    .Names = paste0("r", 1:7))))
```

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 automatcally whenever a matrix factorization is computed.

Dim, Dimnames: inherited from the Matrix class, see there.

Extends

Class "Matrix", directly.

Methods

dimnames<- signature(x = "compMatrix", value = "list"): set the dimnames to a list of length 2, see dimnames<-. The factors slot is currently reset to empty, as the factorization dimnames would have to be adapted, too.

See Also

The matrix factorization classes "MatrixFactorization" and their generators, lu(), qr(), chol() and Cholesky(), BunchKaufman(), Schur().

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, norm(A,"1"), through onenormest(.).

Usage

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

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 \( A, A \times x \) and \( A^T \times x \) 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 \( \text{norm}(A, "1") \); 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 \times x, A^T \times x**
  - when \( A \) is missing, these two must be given as functions which compute \( A \times x \), or \( t(A) \times x \), respectively.

- **n**
  - \( = \text{nrow}(A) \), 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 \times x = , A^T \times 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()`.

**Value**

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

- **est**
  - a number > 0, the estimated (1-norm) condition number \( \hat{\kappa} \); when \( r := \text{rcond}(A) \), \( 1/\hat{\kappa} \approx r \).

- **v**
  - the maximal \( A \times x \) column, scaled to \( \text{norm}(v) = 1 \). Consequently, \( \text{norm}(Av) = \text{norm}(A)/\text{est} \); when \( \text{est} \) 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 \( A \times x \) 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$v))  # || v ||_1 == 1
## Prove that || A v || = || A || / est (as ||v|| = 1):
stopifnot(all.equal(norm(mtm %*% ce$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
## the maximal column:
which(one$v == 1)  # mostly 4, rarely 1, depending on random seed
CsparseMatrix-class

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:

signature(e1 = "CsparseMatrix", e2 = "numeric"): ...

Arith  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"): ...

c coerce signature(from = "TsparseMatrix", to = "CsparseMatrix"): ...

c 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.
ddenseMatrix-class

Virtual Class "ddenseMatrix" of Numeric Dense Matrices

Description
This is the virtual class of all dense numeric (i.e., double, hence “ddense”) S4 matrices.
Its most important subclass is the dgeMatrix class.

Extends
Class "dMatrix" directly; class "Matrix", by the above.

Slots
the same slots at its subclass dgeMatrix, see there.

Methods
Most methods are implemented via as(*, "generalMatrix") and are mainly used as “fallbacks”
when the subclass doesn’t need its own specialized method.
Use showMethods(class = "ddenseMatrix", where = "package:Matrix") for an overview.

See Also
The virtual classes Matrix, dMatrix, and dsparseMatrix.

Examples
showClass("ddenseMatrix")

showMethods(class = "ddenseMatrix", where = "package:Matrix")
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.

Extends

Class "diagonalMatrix", directly. Class "dMatrix", directly. Class "sparseMatrix", indirectly, see showClass("ddiMatrix").

Methods

%*% signature(x = "ddiMatrix", y = "ddiMatrix"): ...

See Also

Class diagonalMatrix and function Diagonal.

Examples

(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)))
denseMatrix-class

Virtual Class "denseMatrix" of All Dense Matrices

Description

This is the virtual class of all dense (S4) matrices. It partitions into two subclasses packedMatrix and unpackedMatrix. Alternatively into the (currently) three subclasses ddenseMatrix, ldenseMatrix, and ndenseMatrix.

denseMatrix is (hence) the direct superclass of these \((2 + 3 = 5)\) classes.

Extends

class "Matrix" directly.

Slots

exactly those of its superclass "Matrix", i.e., "Dim" and "Dimnames".

Methods

Use showMethods(class = "denseMatrix", where = "package:Matrix") for an overview of methods.

Extraction ("[" methods, see [-methods.

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

See Also

  Classes dsCMat, dtCMat, lu

Examples

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

- **x**: Object of class "numeric" - the numeric values contained in the matrix, in column-major order.
- **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**.
- **factors**: Object of class "list" - a list of factorizations of the matrix.

Methods

The are group methods (see, e.g., **Arith**)

- **Arith** signature(e1 = "dgeMatrix", e2 = "dgeMatrix"): ...
- **Arith** signature(e1 = "dgeMatrix", e2 = "numeric"): ...
- **Arith** signature(e1 = "numeric", e2 = "dgeMatrix"): ...
- **Math** signature(x = "dgeMatrix"): ...
- **Math2** signature(x = "dgeMatrix", digits = "numeric"): ...

matrix products `%*%`, **crossprod()** and **tcrossprod()**, several **solve** methods, and other matrix methods available:

- **Schur** signature(x = "dgeMatrix", vectors = "logical"): ...
- **Schur** signature(x = "dgeMatrix", vectors = "missing"): ...
- **chol** signature(x = "dgeMatrix"): see **chol**.
- **coerce** signature(from = "dgeMatrix", to = "lgeMatrix"): ...
- **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**.
Description

The 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.

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

Objects from the Class

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

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 `spMatrix()` or `sparseMatrix(*, repr = "T")`.

**Slots**

- **i**: integer row indices of non-zero entries in 0-base, i.e., must be in `0:(nrow(.)-1)`.
- **j**: integer 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**: numeric vector - 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 "TsparseMatrix" classes) implicitly add `x_k`'s that belong to identical `(i_k, j_k)` pairs.

However this means that a matrix typically can be stored in more than one possible "TsparseMatrix" representations. Use `uniqTsparse()` in order to ensure uniqueness of the internal representation of such a matrix.
Diagonal

Construct a Diagonal Matrix

Description

Construct a formally diagonal Matrix, i.e., an object inheriting from virtual class diagonalMatrix (or, if desired, a mathematically diagonal CsparseMatrix).

Usage

Diagonal(n, x = NULL, names = FALSE)

.sparseDiagonal(n, x = NULL, uplo = "U", shape = "t", unitri = TRUE, kind, cols)
  .trDiagonal(n, x = NULL, uplo = "U", unitri = TRUE, kind)
  .symDiagonal(n, x = NULL, uplo = "U", kind)

Arguments

n
  integer indicating the dimension of the (square) matrix. If missing, then length(x) is used.

x
  numeric or logical vector listing values for the diagonal entries, to be recycled as necessary. If NULL (the default), then the result is a unit diagonal matrix. 
  .sparseDiagonal() and friends ignore non-NULL x when kind = "n".

See Also

Class dgCMatrix or the superclasses dsparseMatrix and TsparseMatrix; uniqTsparse.
names either logical TRUE or FALSE or then a character vector of length n. If true and names(x) is not NULL, use that as both row and column names for the resulting matrix. When a character vector, use it for both dimnames.

uplo one of c("U", "L"), specifying the uplo slot of the result if the result is formally triangular of symmetric.

shape one of c("t", "s", "g"), indicating if the result should be formally triangular, symmetric, or “general”. The result will inherit from virtual class triangularMatrix, symmetricMatrix, or generalMatrix, respectively.

unitri logical indicating if a formally triangular result with ones on the diagonal should be formally unit triangular, i.e., with diag slot equal to "U" rather than "N".

kind one of c("d", "l", "n"), indicating the “mode” of the result: numeric, logical, or pattern. The result will inherit from virtual class dsparseMatrix, lsparseMatrix, or nsparseMatrix, respectively. Values other than "n" are ignored when x is non-NULL; in that case the mode is determined by typeof(x).

cols optional integer vector with values in 0:(n-1), indexing columns of the specified diagonal matrix. If specified, then the result is (mathematically) D[, cols+1] rather than D, where D = Diagonal(n, x), and it is always “general” (i.e., shape is ignored).

Value Diagonal() returns an object inheriting from virtual class diagonalMatrix.

.sparseDiagonal() returns a CsparseMatrix representation of Diagonal(n, x) or, if cols is given, of Diagonal(n, x)[, cols+1]. The precise class of the result depends on shape and kind.

.trDiagonal() and .symDiagonal() are simple wrappers, for .sparseDiagonal(shape = "t") and .sparseDiagonal(shape = "s"), respectively.

.sparseDiagonal() exists primarily to leverage efficient C-level methods available for CsparseMatrix.

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) #-> "IdiMatrix"

## Use Diagonal() + kronecker() for "repeated-block" matrices:
M1 <- Matrix(0:0.5, 2, 3)
\textit{diagonalMatrix-class} 

\textit{Class \textquote{\texttt{diagonalMatrix}} of Diagonal Matrices}

\textbf{Description}

Class \textquote{\texttt{diagonalMatrix}} is the virtual class of all diagonal matrices.

\textbf{Objects from the Class}

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

\textbf{Slots}

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

\textbf{Extends}

Class \textquote{\texttt{sparseMatrix}}, directly.

\textbf{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.

- \texttt{coerce} signature\texttt{(from = \texttt{"matrix"}, to = \texttt{"diagonalMatrix"})}: ...
- \texttt{coerce} signature\texttt{(from = \texttt{"Matrix"}, to = \texttt{"diagonalMatrix"})}: ...
- \texttt{coerce} signature\texttt{(from = \texttt{"diagonalMatrix"}, to = \texttt{"generalMatrix"})}:...
- \texttt{coerce} signature\texttt{(from = \texttt{"diagonalMatrix"}, to = \texttt{"triangularMatrix"})}: ...
- \texttt{coerce} signature\texttt{(from = \texttt{"diagonalMatrix"}, to = \texttt{"nMatrix"})}: ...
- \texttt{coerce} signature\texttt{(from = \texttt{"diagonalMatrix"}, to = \texttt{"matrix"})}: ...
- \texttt{coerce} signature\texttt{(from = \texttt{"diagonalMatrix"}, to = \texttt{"sparseVector"})}:...
- \texttt{t} signature\texttt{(x = \texttt{"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 \(0/0\mapsto \text{NaN}\)), 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

```r
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:
rbind2(cbind2(I5, D5),
       cbind2(D5, I5))
```

`diagU2N` *Transform Triangular Matrices from Unit Triangular to General Triangular and Back*
Description
Transform a triangular matrix x, i.e., of class \textit{triangularMatrix}, from (internally!) unit triangular ("unitriangular") to "general" triangular (\texttt{diagU2N(x)}) or back (\texttt{diagN2U(x)}). Note that the latter, \texttt{diagN2U(x)}, also sets the diagonal to one in cases where \texttt{diag(x)} was not all one.

\texttt{diagU2N(x)} and \texttt{diagN2U(x)} assume \textit{without} checking that \texttt{x} is a \texttt{triangularMatrix} with suitable diag slot ("U" and "N", respectively), hence they should be used with care.

Usage
\begin{verbatim}
\texttt{diagU2N(x, cl = getClassDef(class(x)), checkDense = FALSE)}
\texttt{diagN2U(x, cl = getClassDef(class(x)), checkDense = FALSE)}
\end{verbatim}

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

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

Value
a triangular matrix of the same \texttt{class} but with a different diag slot. For \texttt{diagU2N} (semantically) with identical entries as \texttt{x}, whereas in \texttt{diagN2U(x)}, the off-diagonal entries are unchanged and the diagonal is set to all 1 even if it was not previously.

Note
Such internal storage details should rarely be of relevance to the user. Hence, these functions really are rather \textit{internal} utilities.

See Also
"\texttt{triangularMatrix}", "\texttt{dtCMatrix}".

Examples
\begin{verbatim}
(T <- Diagonal(7) + triu(Matrix(rpois(49, 1/4), 7,7), k = 1))
(uT <- diagN2U(T)) # "unitriangular"
(t.u <- diagN2U(10*T))# changes the diagonal!
stopifnot(all(T == uT), diag(t.u) == 1,
    identical(T, diagU2N(uT)))
T[upper.tri(T)] <- 5 # still "dtC"
\end{verbatim}
T <- diagN2U(as(T, "triangularMatrix"))
.DTO <- as(T, "denseMatrix") # (unitriangular)
.DTO.n <- diagU2N(.DTO, checkDense = TRUE)
.S.T.n <- diagU2N(.DTO)
stopifnot(is(.DTO.n, "denseMatrix"), is(S.T.n, "sparseMatrix"),
  .DTO@diag == "U", .DTO.n@diag == "N", .S.T.n@diag == "N",
  all(.DTO == .DTO.n), all(.DTO == .S.T.n))

---

**dimScale**

Scale the Rows and Columns of a Matrix

**Description**

dimScale, rowScale, and colScale implement \(D_1 \%\% x \%\% D_2, D \%\% x, \text{ and } x \%\% D\) for diagonal matrices \(D_1, D_2, \text{ and } D\) with diagonal entries \(d_1, d_2, \text{ and } d\), respectively. Unlike the explicit products, these functions preserve dimnames(x) and symmetry where appropriate.

**Usage**

dimScale(x, d1 = sqrt(1/diag(x, names = FALSE)), d2 = d1)
rowScale(x, d)
colScale(x, d)

**Arguments**

- **x**: a matrix, possibly inheriting from virtual class Matrix.
- **d1, d2, d**: numeric vectors giving factors by which to scale the rows or columns of \(x\); they are recycled as necessary.

**Details**

dimScale(x) (with \(d_1\) and \(d_2\) unset) is only roughly equivalent to cov2cor(x). cov2cor sets the diagonal entries of the result to 1 (exactly); dimScale does not.

**Value**

The result of scaling \(x\), currently always inheriting from virtual class dMatrix.

It inherits from triangularMatrix if and only if \(x\) does. In the special case of dimScale(x, d1, d2) with identical \(d_1\) and \(d_2\), it inherits from symmetricMatrix if and only if \(x\) does.

**Author(s)**

Mikael Jagan

**See Also**

cov2cor
Examples

n <- 6L
(x <- forceSymmetric(matrix(1, n, n)))
dimnames(x) <- rep.int(list(letters[seq_len(n)]), 2L)

(d <- seq_len(n))
(D <- Diagonal(x = d))

(scx <- dimScale(x, d)) # symmetry and 'dimnames' kept
(mmx <- D %*% x %*% D) # symmetry and 'dimnames' lost
stopifnot(identical(unname(as(scx, "generalMatrix")), mmx))

rowScale(x, d)
colScale(x, d)

---

**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().

Compare signature(e1 = "numeric", e2 = "dMatrix"):

Compare signature(e1 = "dMatrix", e2 = "numeric"):

Compare signature(e1 = "dMatrix", e2 = "dMatrix"):

Summary signature(x = "dMatrix"): The "Summary" group contains the seven functions max(), min(), range(), prod(), sum(), any(), and all().
The following methods are also defined for all double matrices:

- **coerce** signature(from = "dMatrix", to = "matrix"): ...
- **expm** signature(x = "dMatrix"): computes the “Matrix Exponential”, see expm.
- **zapsmall** signature(x = "dMatrix"): ...

The following methods are defined for all logical matrices:

- **which** signature(x = "lsparseMatrix") and many other subclasses of "lMatrix": as the base function which(x, arr.ind) returns the indices of the TRUE entries in x; if arr.ind is true, as a 2-column matrix of row and column indices. Since Matrix version 1.2-9, if useNames is true, as by default, with dimnames, the same as base:::which.

See Also

The nonzero-pattern matrix class nMatrix, which can be used to store non-NA logical matrices even more compactly.

The numeric matrix classes dgeMatrix, dgCMatrix, and Matrix.

drop0(x, tol=1e-10) is sometimes preferable to (and more efficient than) zapsmall(x, digits=10).

Examples

```r
showClass("dMatrix")
set.seed(101)
round(Matrix(rnorm(28), 4,7), 2)
M <- Matrix(rlnorm(56, sd=10), 4,14)
(M. <- zapsmall(M))
table(as.logical(M. == 0))
```

```
###

**dmperm**

**Dulmage-Mendelsohn Permutation / Decomposition**

**Description**

For any $n \times m$ (typically) sparse matrix $x$ compute the Dulmage-Mendelsohn row and columns permutations which at first splits the $n$ rows and $m$ columns into coarse partitions each; and then a finer one, reordering rows and columns such that the permutated matrix is “as upper triangular” as possible.

**Usage**

dmperm(x, nAns = 6L, seed = 0L)
```
Arguments

x  a typically sparse matrix; internally coerced to either "dgCMatrix" or "dtCMatrix".

nAns an integer specifying the length of the resulting list. Must be 2, 4, or 6.

seed an integer code in -1,0,1; determining the (initial) permutation; by default, seed = 0, no (or the identity) permutation; seed = -1 uses the "reverse" permutation k:1; for seed = 1, it is a random permutation (using R’s RNG, seed, etc).

Details

See the book section by Tim Davis; page 122–127, in the References.

Value

a named list with (by default) 6 components,

p integer vector with the permutation p, of length nrow(x).

q integer vector with the permutation q, of length ncol(x).

r integer vector of length nb+1, where block k is rows r[k] to r[k+1]-1 in A[p,q].

s integer vector of length nb+1, where block k is cols s[k] to s[k+1]-1 in A[p,q].

rr5 integer vector of length 5, defining the coarse row decomposition.

cc5 integer vector of length 5, defining the coarse column decomposition.

Author(s)

Martin Maechler, with a lot of “encouragement” by Mauricio Vargas.

References


See Also

Schur, the class of permutation matrices; "pMatrix".

Examples

set.seed(17)
(S9 <- rsparsematrix(9, 9, nnz = 10, symmetric=TRUE)) # dsCMatrix
str(dm9 <- dmperm(S9))
(S9p <- with(dm9, S9[p, q]))
## looks good, but *not* quite upper triangular; these, too:
str(dm9.0 <- dmperm(S9, seed=-1)) # non-random too.
str(dm9.1 <- dmperm(S9, seed= 1)) # a random one
## The last two permutations differ, but have the same effect!
(S9p0 <- with(dm9.0, S9[p, q])) # .. hmm ..
dpoMatrix-class
Positive Semi-definite Dense (Packed \ Non-packed) Numeric Matrices

Description

• 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 of correlation matrices extends "dpoMatrix" with a slot sd, which allows to restore the original covariance matrix.

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"
**dpoMatrix-class**

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 "I" 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 any more. The only exceptions, currently, are multiplications, divisions or additions with **positive** length(.) == 1 numbers (or **logicals**).

**Note**
Currently the validity methods for these classes such as getValidity(getClass("dpoMatrix")) for efficiency reasons only check the diagonal entries of the matrix – they may not be negative. This is only necessary but not sufficient for a symmetric matrix to be positive semi-definite.

A more reliable (but often more expensive) check for positive semi-definiteness would look at the signs of diag(BunchKaufman(.)) (with some tolerance for very small negative values), and for (strict) positive definiteness at something like !inherits(tryCatch(chol(.), error=identity), "error"). Indeed, when **coercing** to these classes, a version of Cholesky() or chol() is typically used, e.g., see selectMethod("coerce", c(from="dsyMatrix", to="dpoMatrix"))

**See Also**
Classes **dsyMatrix** and **dgeMatrix**; further, **Matrix**, **rcond**, **chol**, **solve**, **crossprod**.

**Examples**

```r
h6 <- Hilbert(6)
rcond(h6)
str(h6)
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`; `nnzero`
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(10L, 1L, 3L, 10L, 1L, 10L, 1L, 10L, 1L, 10L),
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.

\texttt{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 \texttt{symmetricMatrix} \texttt{dsparseMatrix} directly; \texttt{dsMatrix} further directly extends \texttt{CsparseMatrix}, where \texttt{dsTMatrix} does \texttt{TsparseMatrix}.

**Methods**

\begin{itemize}
  \item \texttt{solve} \texttt{signature(a = "dsCMatrix", b = ".\ldots."): x <- solve(a,b) solves Ax = b for x; see \texttt{solve-methods}.}
  \item \texttt{chol} \texttt{signature(x = "dsCMatrix", pivot = "logical"): Returns (and stores) the Cholesky decomposition of x, see \texttt{chol}.}
  \item \texttt{Cholesky} \texttt{signature(A = "dsCMatrix",\ldots): Computes more flexibly Cholesky decompositions, see \texttt{Cholesky}.}
  \item \texttt{determinant} \texttt{signature(x = "dsCMatrix", logarithm = "missing"): Evaluate the determinant of x on the logarithm scale. This creates and stores the Cholesky factorization.}
  \item \texttt{determinant} \texttt{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.}
  \item \texttt{t} \texttt{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 \texttt{uplo} slot is swapped, and the row and column indices are interchanged.}
  \item \texttt{t} \texttt{signature(x = "dsTMatrix"): Transpose. The \texttt{uplo} slot is swapped from "U" to "L" or vice versa, as for a "dsCMatrix", see above.}
  \item \texttt{coerce} \texttt{signature(from = "dsCMatrix", to = "dgTMatrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsCMatrix", to = "dgeMatrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsCMatrix", to = "matrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsTMatrix", to = "dgeMatrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsTMatrix", to = "dsCMatrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsTMatrix", to = "dsyMatrix")}
  \item \texttt{coerce} \texttt{signature(from = "dsTMatrix", to = "matrix")}
\end{itemize}

**See Also**

Classes \texttt{dgCMatrix}, \texttt{dgTMatrix}, \texttt{dgeMatrix} and those mentioned above.

**Examples**

\begin{verbatim}
mm <- Matrix(toeplitz(c(10, 0, 1, 0, 3)), sparse = TRUE)
mm # automatically dsCMatrix
str(mm)
mt <- as(as(mm, "generalMatrix"), "TsparseMatrix")
\end{verbatim}
dsparseMatrix-class

Virtual Class "dsparseMatrix" of Numeric Sparse Matrices

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")
dsRMatrix-class

Symmetric Sparse Compressed Row Matrices

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 ".RMatrix" 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

See Also

the classes dgCMatrix, dgTMatrix, and dgeMatrix.

Examples

```r
(m0 <- new("dsRMatrix"))
m2 <- new("dsRMatrix", Dim = c(2L, 2L),
       x = c(3, 1), j = c(1L, 1L), p = 0:2)
m2
stopifnot(colSums(as(m2, "TsparseMatrix")) == 3:4)
str(m2)
(ds2 <- forceSymmetric(diag(2))) # dsy*
dR <- as(ds2, "RsparseMatrix")
dR # dsRMatrix
```

dsyMatrix-class

Symmetric Dense (Packed or Unpacked) 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, see pack(). 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", ...)` or `new("dspMatrix", ...),` respectively.

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 = "..."): \( x \leftarrow solve(a,b) \) solves \( Ax = b \) for \( x \); see \( solve\)-methods.

- **t** 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

- The positive (Semi-)definite dense (packed or non-packed numeric matrix classes \( \text{dpoMatrix}, \text{dppMatrix} \) and \( \text{corMatrix} \),

- Classes \( \text{dgeMatrix} \) and \( \text{Matrix} \); solve, norm, rcond, t

Examples

```r
## 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

## 4x4 example
m <- matrix(0,4,4); m[upper.tri(m)] <- 1:6
(sym <- m+t(m)+diag(11:14, 4))
(S1 <- pack(sym))
(S2 <- t(S1))
stopifnot(all(S1 == S2)) # equal "seen as matrix", but differ internally :
str(S1)
S2@x
```
**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 coercions such as `as(x, "triangularMatrix")`.

**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

showClass("dtCMatrix")
showClass("dtTMatrix")
t1 <- new("dtTMatrix", 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, "CsparseMatrix"))
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("dtCMatrix", i= c(1L, 0:3), p=c(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  # should equal the identity matrix
i5 <- iu2 # should match I5
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)))
  )
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 `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"!).
- `Dim`, `Dimnames`: The dimension (a length-2 "integer") and corresponding names (or NULL), inherited from the `Matrix`, see there.

Extends

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

Methods

- `%*%` signature(x = "dtpMatrix", y = "dgeMatrix"): Matrix multiplication; ditto for several other signature combinations, see `showMethods("%*%", class = "dtpMatrix")`.
- `coerce` signature(from = "dtpMatrix", to = "dtrMatrix")
- `coerce` signature(from = "dtpMatrix", to = "matrix")
- `determinant` signature(x = "dtpMatrix", logarithm = "logical"): the determinant(x) trivially is prod(diag(x)), but computed on log scale to prevent over- and underflow.
- `diag` signature(x = "dtpMatrix"): ...
- `norm` signature(x = "dtpMatrix", type = "character"): ...
- `rcond` signature(x = "dtpMatrix", norm = "character"): ...
- `solve` signature(a = "dtpMatrix", b = "...")): efficiently using internal backsolve or forwardsolve, see `solve-methods`.
- `t` signature(x = "dtpMatrix"): t(x) remains a "dtpMatrix", lower triangular if x is upper triangular, and vice versa.
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.
**dtrMatrix-class**

**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
```

---

**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, see `pack()`.

**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

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

(T <- as(M, "triangularMatrix")) # formally upper triangular
(T2 <- as(t(M), "triangularMatrix"))
stopifnot(T@uplo == "U", T2@uplo == "L", identical(T2, t(T)))

m <- matrix(0,4,4); m[upper.tri(m)] <- 1:6
(t1 <- Matrix(m+diag(,4)))
str(t1p <- pack(t1))
(t1pu <- diagN2U(t1p))
stopifnot(exprs = {
  inherits(t1, "dtrMatrix"); validObject(t1)
  inherits(t1p, "dtpMatrix"); validObject(t1p)
  inherits(t1pu,"dtCMatrix"); validObject(t1pu)
  t1pu@x == 1:6
  all(t1pu == t1p)
  identical((t1pu - t1)@x, numeric())# sparse all-0
})
```
`expand`  

Expand a (Matrix) Decomposition into Factors

**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.

**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**

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

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 Padé’ 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

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


See Also

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

\[(m1 \leftarrow \text{Matrix}(c(1,0,1), \text{ncol} = 2))\]
\[(e1 \leftarrow \exp(m1)) ; e \leftarrow \exp(1)\]
\[\text{stopifnot(all.equal}(e1@x, c(e,0,e,e), \text{tolerance} = 1e-15)\]
\[(m2 \leftarrow \text{Matrix}(c(-49, -64, 24, 31), \text{ncol} = 2))\]
\[(e2 \leftarrow \exp(m2))\]
\[(m3 \leftarrow \text{Matrix}(\text{cbind}(0, \text{rbind}(6*\text{diag}(3), 0)))) \# \text{sparse!}\]
\[(e3 \leftarrow \exp(m3)) \# \text{upper triangular}\]

---

**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.

Note that these formats do not know anything about `dimnames`, hence these are dropped by `writeMM()`.

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.
facmul

Multiplication by Decomposition Factors

Description

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

References

https://math.nist.gov/MatrixMarket/
https://sparse.tamu.edu/

Examples

```
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")))
str(jgl009 <- readMM(system.file("external/jgl009.mtx", 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("https://www.cise.ufl.edu/research/sparse/RB/Boeing/msc00726.tar.gz"))))
## End(Not run)
```

```
data(KNex)
## Store as MatrixMarket (".mtx") file, here inside temporary dir./folder:
(MMfile <- file.path(tempdir(), "mmMM.mtx"))
writeMM(KNex$mm, file=MMfile)
file.info(MMfile)[,c("size", "ctime")]
```

```
data(CAex)
s.CA <- summary(CAex)
s.CA # shows (i, j, x) [columns of a data frame]
message("writing to ", outf <- tempfile())
write.table(s.CA, file = outf, row.names=FALSE)
## and read it back -- showing off sparseMatrix():
str(dd <- read.table(outf, header=TRUE))
## has columns (i, j, x) -> we can use via do.call() as arguments to sparseMatrix():
mm <- do.call(sparseMatrix, dd)
stopifnot(all.equal(mm, CAex, tolerance=1e-15))
```
Usage

facmul(x, factor, y, transpose, left, ...)

Arguments

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

Value

the product of the selected factor (or its transpose) and y

NOTE

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

References

more.

Examples

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)
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 \texttt{format} and \texttt{print} methods for sparse matrices and can be applied as well to standard \texttt{R} matrices. Note that all arguments but the first are optional.

\texttt{formatSparseM()} is the main “workhorse” of \texttt{formatSpMatrix}, the format method for sparse matrices.

.. \texttt{formatSparseSimple()} is a simple helper function, also dealing with (short/empty) column names construction.

Usage

\begin{verbatim}
formatSparseM(x, zero.print = ".", align = c("fancy", "right"),
  m = as(x,"matrix"), asLogical=NULL, uniDiag=NULL,
  digits=NULL, cx, iN0, dn = dimnames(m))

.formatSparseSimple(m, asLogical=FALSE, digits=NULL,
  col.names, note.dropping.colnames = TRUE,
  dn=dimnames(m))
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} an \texttt{R} object inheriting from class \texttt{sparseMatrix}.
  \item \texttt{zero.print} character which should be used for \textit{structural} zeroes. The default "." may occasionally be replaced by " " (blank); using "0" would look almost like \texttt{print()}ing of non-sparse matrices.
  \item \texttt{align} a string specifying how the \texttt{zero.print} codes should be aligned, see \texttt{formatSpMatrix}.
  \item \texttt{m} (optional) a (standard \texttt{R}) \texttt{matrix} version of \texttt{x}.
  \item \texttt{asLogical} should the matrix be formatted as a logical matrix (or rather as a numeric one); mostly for \texttt{formatSparseM()}.
  \item \texttt{uniDiag} logical indicating if the diagonal entries of a sparse unit triangular or unit-diagonal matrix should be formatted as "I" instead of "1" (to emphasize that the 1’s are “structural”).
  \item \texttt{digits} significant digits to use for printing, see \texttt{print.default}.
  \item \texttt{cx} (optional) character matrix; a formatted version of \texttt{x}, still with strings such as "0.00" for the zeros.
  \item \texttt{iN0} (optional) integer vector, specifying the location of the \textit{non}-zeroes of \texttt{x}.
  \item \texttt{col.names}, \texttt{note.dropping.colnames} see \texttt{formatSpMatrix}.
  \item \texttt{dn} \texttt{dimnames} to be used; a list (of length two) with row and column names (or \texttt{NULL}).
\end{itemize}
Value

a character matrix like `cx`, where the zeros have been replaced with (padded versions of) `zero.print`. 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))
f <- formatSparseM(m)
oquote(f)
## nice, but this is nicer (with "units" vertically aligned):
print(f, quote=FALSE, right=TRUE)
## and "the same" as :
Matrix(m)

## align = "right" is cheaper -- the "." are not aligned:
oquote(f2 <- formatSparseM(m,align="r"))
stopifnot(f2 == f | 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.

---

**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,

\[
\text{as(from, Class)}
\]

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**

\[
\text{graph2T(from, use.weights = )}
\]
\[
\text{T2graph(from, need.uniq = is_not_uniqT(from), edgemode = NULL)}
\]

**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)))) { ## super careful .. for "checking reasons"
  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" (nsTMatrix)

  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 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.

The `Matrix` package image methods are based on `levelplot()` from package `lattice`; hence these methods return an “object” of class “`trellis`”, producing a graphic when (auto-) `print()`ed.

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, border.col = 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 \( \text{abs}(x) \) 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.

border.col  color for the border of each rectangle. NA means no border is drawn. When NULL as by default, border.col <- if(lwd < .01) NA else NULL is used. Consider using an opaque color instead of NULL which corresponds to grid::get.gpar("col").

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

Value

as all lattice graphics functions, image(<Matrix>) returns a "trellis" object, effectively the result of levelplot().

Methods

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

See Also

levelplot, and print.trellis from package lattice.

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)") -> imgC; imgC
stopifnot(!is.null(leg <- imgC$legend), is.list(imgC$right)) # failed for 2 days ..
image(CAex, useAbs=TRUE, main = "image(CAex, useAbs=TRUE)"

cCA <- Cholesky(crossprod(CAex), Imult = .01)
## See ?print.trellis --- place two image() plots side by side:
print(image(cCA, main="Cholesky(crossprod(CAex), Imult = .01)")),
   split=c(x=1,y=1,nx=2, ny=1), more=TRUE)
print(image(cCA, useAbs=TRUE),
   split=c(x=2,y=1,nx=2,ny=1))
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)
## These 3 are all equivalent :
(I1 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE))
I2 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE, border.col=NA)
I3 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE, lwd=2, border.col=NA)
stopifnot(all.equal(I1, I2, check.environment=FALSE),
  all.equal(I2, I3, check.environment=FALSE))
## using an opaque border color
image(USCounties, c(1,100), c(1,100), useAbs=FALSE, lwd=3, border.col = adjustcolor("skyblue", 1/2))

if(interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA"))) {
## 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 :
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)

caption(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

\(-\text{methods}\), and
\text{Subassign-methods}, also for examples.

Examples

showClass("index")

---

\begin{tabular}{ll}
\textbf{indMatrix-class} & \textbf{Index Matrices} \\
\end{tabular}

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

Objects from the Class

Objects can be created by calls of the form \text{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 1:\texttt{Dim}[2].

\texttt{Dim}: \textbf{integer} vector of length two. In some applications, the matrix will be skinny, i.e., with at least as many rows as columns.
**Dimnames:** a list of length two where each component is either NULL or a character vector of length equal to the corresponding Dim element.

**Extends**

Class "sparseMatrix" and "generalMatrix", directly.

**Methods**

```
%*% signature(x = "matrix", y = "indMatrix") and other signatures (use showMethods("%*%", class="indMatrix")); ...
```

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

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

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

```
coerce signature(from = "indMatrix", to = "matrix"): coercion to a traditional FALSE/TRUE matrix of mode logical.
```

```
coerce signature(from = "indMatrix", to = "ngTMatrix"): coercion to sparse logical matrix of class ngTMatrix.
```

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

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

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

```
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.
```

**Author(s)**

Fabian Scheipl, Uni Muenchen, building on existing "pMatrix", after a nice hike’s conversation with Martin Maechler; diverse tweaks by the latter. The crossprod(x,y) and kronecker(x,y) methods when both arguments are "indMatrix" have been made considerably faster thanks to a suggestion by Boris Vaillant.

**See Also**

The permutation matrices pMatrix are special index matrices. The “pattern” matrices, nMatrix and its subclasses.
Examples

```r
p1 <- as(c(2,3,1), "pMatrix")
(sml <- as(rep(c(2,3,1), e=3), "indMatrix"))
stopifnot(all(sml == p1[rep(1:3, each=3),]))

## row-indexing of a <pMatrix> turns it into an <indMatrix>:
class(p1[rep(1:3, each=3),])

set.seed(12) # so we know '10' is in sample
## random index matrix for 30 observations and 10 unique values:
(s10 <- as(sample(10, 30, 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(1:18, 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")

s10[1:7, 1:4] # gives an "ngTMatrix" (most economic!)
s10[1:4, ] # preserves "indMatrix"-class

I1 <- as(c(5:1,6:4,7:3), "indMatrix")
I2 <- as(7:1, "pMatrix")
(I12 <- rbind(I1, I2))
stopifnot(is(I12, "indMatrix"),
identical(I12, rbind(I1, I2)),
colSums(I12) == c(2L,2:4,4:2))
```

---

**invPerm**

**Inverse Permutation Vector**

**Description**

From a permutation vector `p`, compute its *inverse* permutation vector.

**Usage**

```r
invPerm(p, zero.p = FALSE, zero.res = FALSE)
```
Arguments

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

Value

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

Author(s)

Martin Maechler

See Also

the class of permutation matrices, `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 generic functions `is.na()`, `is.nan()`, `is.finite()`, `is.infinite()`, and `anyNA()`, for objects inheriting from virtual class `Matrix` or `sparseVector`. 
Usage

## S4 method for signature 'dsparseMatrix'
is.na(x)

## S4 method for signature 'dsparseMatrix'
is.nan(x)

## S4 method for signature 'dsparseMatrix'
is.finite(x)

## S4 method for signature 'dsparseMatrix'
is.infinite(x)

## S4 method for signature 'dsparseMatrix'
anyNA(x)

## ...  
## and for other classes

Arguments

x an R object, here a sparse or dense matrix or vector.

Value

For is.*(), an nMatrix or nsparseVector matching the dimensions of x and specifying the positions in x of (some subset of) NA, NaN, Inf, and -Inf. For anyNA(), TRUE if x contains NA or NaN and FALSE otherwise.

See Also

NA, NaN, Inf

Examples

(M <- Matrix(1:6, nrow = 4, ncol = 3,      
dimnames = list(letters[1:4], LETTERS[1:3])))
stopifnot(!anyNA(M), !any(is.na(M)))

M[2:3, 2] <- NA
(inM <- is.na(M))
stopifnot(anyNA(M), sum(inM) == 2)

(A <- spMatrix(nrow = 10, ncol = 20,      
i = c(1, 3:8), j = c(2, 9, 6:10), x = 7 * (1:7)))
stopifnot(!anyNA(A), !any(is.na(A)))

(inA <- is.na(A))
stopifnot(anyNA(A), sum(inA) == 1 + 1 + 5)
is.null.DN

Are the Dimnames dn NULL-like?

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 dim-
names 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

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)))

hasNoDimnames <- function(.) is.null.DN(dimnames(.))

stopifnot(exprs = {
  hasNoDimnames(m)
  hasNoDimnames(m1); hasNoDimnames(m2)
  hasNoDimnames(m3); hasNoDimnames(m4)
  hasNoDimnames(Matrix(m) -> M)
  hasNoDimnames(as(M, "sparseMatrix"))
})

---

**isSymmetric-methods**  
*Methods for Function 'isSymmetric' in Package 'Matrix'*

**Description**

`isSymmetric` tests whether its argument is a symmetric square matrix, by default tolerating some numerical fuzz and requiring symmetric [dD]imnames in addition to symmetry in the mathematical sense. `isSymmetric` is a generic function in *base*, which has a method for traditional matrices of implicit class "matrix". Methods are defined here for various proper and virtual classes in *Matrix*, so that `isSymmetric` works for all objects inheriting from virtual class "Matrix".

**Usage**

```
## S4 method for signature 'symmetricMatrix'
isSymmetric(object, ...)
## S4 method for signature 'triangularMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'diagonalMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'indMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'dgeMatrix'
isSymmetric(object, tol = 100 * .Machine$double.eps, tol1 = 8 * tol, checkDN = TRUE, ...)
## S4 method for signature 'lgeMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'ngeMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'dgCMatrix'
isSymmetric(object, tol = 100 * .Machine$double.eps, checkDN = TRUE, ...)
## S4 method for signature 'lgCMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'ngCMatrix'
isSymmetric(object, checkDN = TRUE, ...)
```
isSymmetric-methods

Arguments

- object: a "Matrix".
- tol, tol1: numerical tolerances allowing approximate symmetry of numeric (rather than logical) matrices. See also isSymmetric.matrix.
- checkDN: a logical indicating whether symmetry of the Dimnames slot of object should be checked.
- ...: further arguments passed to methods (typically methods for all.equal).

Details

The Dimnames slot of object, say dn, is considered to be symmetric if and only if

- \( \text{dn}[1] \) and \( \text{dn}[2] \) are identical or one is NULL; and
- \( \text{ndn} \leftarrow \text{names(dn)} \) is NULL or \( \text{ndn}[1] \) and \( \text{ndn}[2] \) are identical or one is the empty string "".

Hence list(a=nms, a=nms) is considered to be symmetric, and so too are list(a=nms, NULL) and list(NULL, a=nms).

Note that this definition is looser than that employed by isSymmetric.matrix, which requires \( \text{dn}[1] \) and \( \text{dn}[2] \) to be identical, where \( \text{dn} \) is the dimnames attribute of a traditional matrix.

Value

A logical, either TRUE or FALSE (never NA).

See Also

forceSymmetric; symmpart and skewpart: virtual class "symmetricMatrix" and its subclasses.

Examples

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, "generalMatrix")) # still symmetric, even if not "formally"
isSymmetric(triu(M)) # FALSE

## Look at implementations:
showMethods("isSymmetric", includeDefs = TRUE) # includes S3 generic from base
isTriangular

Test whether a Matrix is Triangular or Diagonal

Description

isTriangular and isDiagonal test whether their argument is a triangular or diagonal matrix, respectively. Unlike the analogous isSymmetric, these two functions are generically from Matrix rather than base. Hence Matrix defines methods for traditional matrices of implicit class "matrix" in addition to matrices inheriting from virtual class "Matrix".

By our definition, triangular and diagonal matrices are square, i.e., they have the same number of rows and columns.

Usage

isTriangular(object, upper = NA, ...)

isDiagonal(object)

Arguments

object

an R object, typically a matrix.

upper

a logical, either TRUE or FALSE, in which case TRUE is returned only for upper or lower triangular object; or otherwise NA (the default), in which case TRUE is returned for any triangular object.

... further arguments passed to methods (currently unused by Matrix).

Value

A logical, either TRUE or FALSE (never NA).

If object is triangular and upper is NA, then isTriangular returns TRUE with an attribute kind, either "U" or "L", indicating that object is upper or lower triangular, respectively. Users should not rely on how kind is determined for diagonal matrices, which are both upper and lower triangular.

See Also

isSymmetric; virtual classes "triangularMatrix" and "diagonalMatrix" and their subclasses.

Examples

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, "generalMatrix")) # 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

KhatriRao(X, Y = X, FUN = "*", sparseY = TRUE, make.dimnames = FALSE)

Arguments

- **X, Y**: matrices of with the same number of columns.
- **FUN**: the (name of the) function to be used for the column-wise Kronecker products, see kronecker, defaulting to the usual multiplication.
- **sparseY**: logical specifying if Y should be coerced and treated as sparseMatrix. Set this to FALSE, e.g., to distinguish structural zeros from zero entries.
- **make.dimnames**: logical indicating if the result should inherit dimnames from X and Y in a simple way.

Value

A "CsparseMatrix", say R, the Khatri-Rao product of X (n × k) and Y (m × k), is of dimension (n · m) × k, where the j-th column, R[,j] is the kronecker product kronecker(X[,j], Y[,j]).

Note

The current implementation is efficient for large sparse matrices.

Author(s)

Original by Michael Cysouw, Univ. Marburg; minor tweaks, bug fixes etc, by Martin Maechler.
References


See Also

kronecker.

Examples

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

mm <- as(m, "nsparseMatrix")
nd <- as(d, "nsparseMatrix")
KhatriRao(mm,nd, make.dimnames=TRUE)
KhatriRao(nd,mm, make.dimnames=TRUE)

stopifnot(dim(KhatriRao(m,d)) == c(nrow(m)*nrow(d), ncol(d)))
## border cases / checks:
zm <- mm; zm[] <- FALSE # all FALSE matrix
stopifnot(all(K1 <- KhatriRao(nd, zm) == 0), identical(dim(K1), c(12L, 4L)),
  all(K2 <- KhatriRao(zm, nd) == 0), identical(dim(K2), c(12L, 4L)))

d0 <- d; d0[] <- 0; m0 <- Matrix(d0[-1,])
stopifnot(all(K3 <- KhatriRao(d0, m0) == 0), identical(dim(K3), dim(Kdm)),
  all(K4 <- KhatriRao(m0, d0) == 0), identical(dim(K4), dim(Kmd)),
  all(KhatriRao(d0, d0) == 0), all(KhatriRao(m0, d0) == 0),
  all(KhatriRao(d0, m0) == 0), all(KhatriRao(m0, m0) == 0),
  identical(dimnames(KhatriRao(m0, d0, make.dimnames=TRUE)), dimnames(Kmd)))

## a matrix with "structural" and non-structural zeros:
m01 <- new("dgCMatrix", i = c(0L, 2L, 0L, 1L), p = c(0L, 0L, 0L, 2L, 4L),
  Dim = 3:4, x = c(1, 0, 1, 0))
```

D4 <- Diagonal(4, x=1:4) # "as" d
DU <- Diagonal(4)# unit-diagonal: uplo="U"
(K5 <- KhatriRao( d, m01))
K5d <- KhatriRao( d, m01, sparseY=FALSE)
K5Dd <- KhatriRao(D4, m01, sparseY=FALSE)
K5Ud <- KhatriRao(DU, m01, sparseY=FALSE)
(K6 <- KhatriRao(diag(3), t(m01)))
K6D <- KhatriRao(Diagonal(3), t(m01))
K6d <- KhatriRao(diag(3), t(m01), sparseY=FALSE)
K6Dd <- KhatriRao(Diagonal(3), t(m01), sparseY=FALSE)
stopifnot(exprs = {
  all(K5 == K5d)
  identical(cbind(c(7L, 10L), c(3L, 4L)),
            which(K5 != 0, arr.ind = TRUE, useNames=FALSE))
  identical(K5d, K5Dd)
  identical(K6, K6D)
  all(K6 == K6d)
  identical(cbind(3:4, 1L),
            which(K6 != 0, arr.ind = TRUE, useNames=FALSE))
  identical(K6d, K6Dd)
})

---

**KNex**

**Koenker-Ng Example Sparse Model Matrix and Response Vector**

**Description**

A model matrix `mm` and corresponding response vector `y` used in an example by Koenker and Ng. The matrix `mm` 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**

```r
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))))
```
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 \(0 \times NA\) as \(0\), not as \(NA\) as usually in \(R\) (and as used for the base function \texttt{kronecker}).

Methods

\texttt{kronecker} signature(X = "Matrix", Y = "ANY")
\texttt{kronecker} signature(X = "ANY", Y = "Matrix")
\texttt{kronecker} signature(X = "diagonalMatrix", Y = "ANY")
\texttt{kronecker} signature(X = "sparseMatrix", Y = "ANY")
\texttt{kronecker} signature(X = "TsparseMatrix", Y = "TsparseMatrix")
\texttt{kronecker} signature(X = "dgTMatrix", Y = "dgTMatrix")
\texttt{kronecker} signature(X = "dtTMatrix", Y = "dtTMatrix")
\texttt{kronecker} signature(X = "indMatrix", Y = "indMatrix")

Examples

\( (t1 \leftarrow \text{spMatrix}(5, 4, x = c(3, 2, -7, 11), i = 1:4, j = 4:1)) \) \# 5 x 4
\( (t2 \leftarrow \text{kronecker}(\text{Diagonal}(3, 2:4), t1)) \) \# 15 x 12

\# should also work with special-cased logical matrices
\( l3 \leftarrow \text{upper.tri(matrix}(, 3, 3)) \)
\( M \leftarrow \text{Matrix}(l3) \)
\( (N \leftarrow \text{as}(M, "nspMatrix")) \) \# "ntMatrix" (upper triangular)
\( N2 \leftarrow \text{as}(N, "generalMatrix") \) \# (lost "t"riangularity)
\( MM \leftarrow \text{kronecker}(M, M) \)
\( NN \leftarrow \text{kronecker}(N, N) \) \# "dtTMatrix" i.e. did keep
\( NN2 \leftarrow \text{kronecker}(N2, N2) \)
\text{stopifnot}(\text{identical}(NN, MM),
\text{is}(NN2, "sparseMatrix"), \text{all}(NN2 == NN),
\text{is}(NN, "triangularMatrix"))
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")
**ldiMatrix-class**  
*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**

```r
(lM <- Diagonal(x = c(TRUE,FALSE,FALSE)))
str(lM)#> gory details (slots)
crossprod(lM) # numeric
(nM <- as(lM, "nMatrix"))# -> sparse (not formally \`diagonal\')
crossprod(nM) # logical sparse
```

---

**lgeMatrix-class**  
*Class "lgeMatrix" of General Dense Logical Matrices*

**Description**

This is the class of general dense *logical* matrices.
lsparseMatrix-classes

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".

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, "CsparseMatrix")

Sparse logical matrices

Description

The lsparseMatrix class is a virtual class of sparse matrices with TRUE/FALSE or NA entries. Only the positions of the elements that are TRUE are stored.

These can be stored in the “triplet” form (class TsparseMatrix, subclasses lgTMatrix, lsTMatrix, and ltTMatrix) or in compressed column-oriented form (class CsparseMatrix, subclasses lgCMatrix, lsCMatrix, and ltCMatrix) or—rarely—in compressed row-oriented form (class RsparseMatrix, subclasses lgRMatrix, lsRMatrix, and ltRMatrix). The second letter in the name of these non-virtual classes indicates general, symmetric, or triangular.
Details

Note that triplet stored (\text{TsparseMatrix}) matrices such as \text{lTMatrix} may contain duplicated pairs of indices \((i, j)\) as for the corresponding numeric class \text{dgTMatrix} where for such pairs, the corresponding \text{x} slot entries are added. For logical matrices, the \text{x} entries corresponding to duplicated index pairs \((i, j)\) are "added" as well if the addition is defined as logical or, i.e., "TRUE + TRUE |-> TRUE" and "TRUE + FALSE |-> TRUE". Note the use of \text{uniqTsparse()} for getting an internally unique representation without duplicated \((i, j)\) entries.

Objects from the Class

Objects can be created by calls of the form \text{new("lgCMatrix", ...)} and so on. More frequently objects are created by coercion of a numeric sparse matrix to the logical form, e.g. in an expression \text{x != 0}.

The logical form is also used 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

\text{x}: Object of class "logical", i.e., either TRUE, NA, or FALSE.

\text{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.

\text{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.

\text{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.

\text{i}: Object of class "integer" of length \text{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.

\text{j}: Object of class "integer" of length \text{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 row-oriented forms only.

\text{Dim}: Object of class "integer" - the dimensions of the matrix.

Methods

\text{coerce} \text{signature(from = "dgCMatrix", to = "lgCMatrix")}

\text{t} \text{signature(x = "lgCMatrix")}: returns the transpose of \text{x}

\text{which} \text{signature(x = "lsparseMatrix"), semantically equivalent to base function which(x, arr.ind); for details, see the \text{lMatrix} class documentation.}
lsyMatrix-class

Symmetric Dense Logical Matrices

Description

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

Objects from the Class

Objects can be created by calls of the form new("lsyMatrix", ...).
ltrMatrix-class

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.

classes: 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

```r
(M2 <- Matrix(c(TRUE, NA, FALSE, FALSE), 2, 2)) # logical dense (ltr)
str(M2)
# can
(sM <- M2 | t(M2)) # "lge"
as(sM, "symmetricMatrix")
str(sM <- as(sM, "packedMatrix")) # 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)), "ldenseMatrix"))
str(lutp <- pack(lutr)) # packed matrix: only 10 = 4*(4+1)/2 entries
!lutp # the logical negation (is *not* logical triangular !)
## but this one is:
stopifnot(all.equal(lutp, pack(!lutp)))
```

---

**lu**  
(Generalized) Triangular Decomposition of a Matrix

Description

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

\texttt{lu}(x, ...) \\
  \hspace{1em} \texttt{\# S4 method for signature 'matrix'} \\
\texttt{lu}(x, ...) \\
  \hspace{1em} \texttt{\# S4 method for signature 'dgeMatrix'} \\
\texttt{lu}(x, warnSing = \texttt{TRUE}, ...) \\
  \hspace{1em} \texttt{\# S4 method for signature 'dgCMatrix'} \\
\texttt{lu}(x, errSing = \texttt{TRUE}, order = \texttt{TRUE}, tol = 1, \\
  \hspace{2em} keep.dimnames = \texttt{TRUE}, ...) \\
\texttt{lu}(x, ...) \\
  \hspace{1em} \texttt{\# S4 method for signature 'dsyMatrix'} \\
\texttt{lu}(x, cache = \texttt{TRUE}, ...) \\
  \hspace{1em} \texttt{\# S4 method for signature 'dsCMatrix'} \\
\texttt{lu}(x, cache = \texttt{TRUE}, ...)

Arguments

- \texttt{x} a dense or sparse matrix, in the latter case of square dimension. No missing values or IEEE special values are allowed.
- \texttt{warnSing} (when \texttt{x} is a "denseMatrix") logical specifying if a \texttt{warning} should be signalled when \texttt{x} is singular.
- \texttt{errSing} (when \texttt{x} is a "sparseMatrix") logical specifying if an error (see \texttt{stop}) should be signalled when \texttt{x} is singular. When \texttt{x} is singular, \texttt{lu(x, errSing=FALSE)} returns \texttt{NA} instead of an LU decomposition. No warning is signalled and the user should be careful in that case.
- \texttt{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.
- \texttt{tol} positive number indicating the pivoting tolerance used in \texttt{cs_lu}. Do only change with much care.
- \texttt{keep.dimnames} logical indicating that \texttt{dimnames} should be propagated to the result, i.e., "kept". This was hardcoded to \texttt{FALSE} in upto \texttt{Matrix} version 1.2-0. Setting to \texttt{FALSE} may gain some performance.
- \texttt{cache} logical indicating if the result should be cached in \texttt{x@factors}; note that this argument is experimental and only available for certain classes inheriting from \texttt{compMatrix}.
- \texttt{...} further arguments passed to or from other methods.

Details

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

The method for class \texttt{dgeMatrix} (and all dense, non-triangular matrices) is based on LAPACK’s \texttt{dgetrf} subroutine. It returns a decomposition also for singular and non-square matrices.

The method for class \texttt{dgCMatrix} (and all sparse, non-triangular matrices) is based on functions from the CSparse library. It signals an error (or returns \texttt{NA}, when \texttt{errSing = FALSE}; see above) when the decomposition algorithm fails, as when \texttt{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.

Note

Because the underlying algorithm differ entirely, in the dense case (class denseLU), the decomposition is

\[ A = PLU, \]

where as in the sparse case (class sparseLU), it is

\[ A = P'LUQ. \]

References


See Also

Class definitions denseLU 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
pm <- pm[pmLU@p + 1L, pmLU@q + 1L]
pLUU <- drop0(pmLU%*%pmLU@U) # L %*% U -- dropping extra zeros
## equal up to "rounding"
ppm[1:14, 1:5]
pLUU[1:14, 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 \( \text{dtrMatrix} \)).

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 \( \text{new("denseLU", ...)} \). More commonly the objects are created explicitly from calls of the form \( \text{lu(mm)} \) where \( mm \) is an object that inherits from the "dgeMatrix" class or as a side-effect of other functions applied to "dgeMatrix" objects.

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 \( \text{expand()} \) method, see below.

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

- \( \text{Dim} \): the dimension of the original matrix; inherited from class \( \text{MatrixFactorization} \).

Methods

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

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

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

Examples

```r
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
elu$L %*% elu$U
(m2 <- with(elu, P %*% L %*% U)) # the same as 'mm'
stopifnot(all.equal(as(mm, "matrix"),
                    as(m2, "matrix")))
```

Description

From an R object coercible to "TsparseMatrix", typically a (sparse) matrix, produce its triplet representation which may collapse to a “Duplet” in the case of binary aka pattern, such as "nMatrix" objects.

Usage

```r
mat2triplet(x, uniqT = FALSE)
```

Arguments

- `x` any R object for which as(x, "TsparseMatrix") works; typically a matrix of one of the Matrix package matrices.
- `uniqT` logical indicating if the triplet representation should be ‘unique’ in the sense of uniqTsparse().

Value

A list, typically with three components,

- `i` vector of row indices for all non-zero entries of x
- `i` vector of columns indices for all non-zero entries of x
- `x` vector of all non-zero entries of x; exists only when as(x, "TsparseMatrix") is not a "nsparseMatrix".

Note that the order of the entries is determined by the coercion to "TsparseMatrix" and hence typically with increasing j (and increasing i within ties of j).
Note

The mat2triplet() utility was created to be a more efficient and more predictable substitute for summary(<sparseMatrix>). UseRs have wrongly expected the latter to return a data frame with columns i and j which however is wrong for a "diagonalMatrix".

See Also

The summary() method for "sparseMatrix", summary,sparseMatrix-method. mat2triplet() is conceptually the inverse function of spMatrix and (one case of) sparseMatrix.

Examples

if(FALSE) ## The function is defined (don't redefine here!), simply as
mat2triplet <- function(x, uniqT = FALSE) {
  T <- as(x, "TsparseMatrix")
  if(uniqT && anyDuplicatedT(T)) T <- .uniqTsparse(T)
  if(is(T, "nsparseMatrix"))
    list(i = T@i + 1L, j = T@j + 1L)
  else list(i = T@i + 1L, j = T@j + 1L, x = T@x)
}

i <- c(1,3:8); j <- c(2,9,6:10); x <- 7 * (1:7)
(Ax <- sparseMatrix(i, j, x = x)) ## 8 x 10 "dgCMatrix"
str(trA <- mat2triplet(Ax)) ## 8 x 10 "dgCMatrix"
stopifnot(i == sort(trA$i), sort(j) == trA$j, x == sort(trA$x))

D <- Diagonal(x=4:2)
summary(D)
str(mat2triplet(D))

Matrix

Construct a Classed Matrix

Description

Construct a Matrix of a class that inherits from 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
Matrix

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.

doDiag
logical indicating if a diagonalMatrix object should be returned when the resulting matrix is diagonal (mathematically). As class diagonalMatrix extends sparseMatrix, this is a natural default for all values of sparse. Otherwise, if doDiag is false, a dense or sparse (depending on sparse) 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'

## 4 cases - 3 different results :
Matrix(0, 2, 2)  # diagonal !
Matrix(0, 2, 2, sparse=FALSE)# (ditto)
Matrix(0, 2, 2, doDiag=FALSE)# -> sparse symm. "dsCMatrix"
Matrix(0, 2, 2, sparse=FALSE, doDiag=FALSE)# -> dense symm. "dsyMatrix"
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) determi-
nant 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

```r
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) # (ditto)
Matrix(diag(4) >= 0) # -> "lsyMatrix" (of all 'TRUE')

## triangular
l3 <- upper.tri(matrix(,3,3))
(M <- Matrix(l3)) # -> "ltCMatrix"
Matrix(! l3) # -> "ltrMatrix"
as(l3, "CsparseMatrix") # "lgCMatrix"

Matrix(1:9, nrow=3,
dimnames = list(c("a", "b", "c"), c("A", "B", "C")))
(I3 <- Matrix(diag(3)))# identity, i.e., unit "diagonalMatrix"
str(I3) # note 'diag = "U"' and 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
forceSymmetric(A) # also symmetric, w/ symm. dimnames
stopifnot(is(As, "symmetricMatrix"),
is(Matrix(0, 3,3), "sparseMatrix"),
is(Matrix(FALSE, 1,1), "sparseMatrix"))
```
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. For printing sparse matrices, see `printSpMatrix`.

**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.

**coerce** signature(from = "ANY", to = "Matrix"): This relies on a correct `as.matrix()` method for from.

There are many more methods that (conceptually should) work for all "Matrix" objects, e.g., `colSums, rowMeans`. Even base functions may work automatically (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, eigen, svd` or `kappa` all do work via coercion 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
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))

matrix-products Matrix (Cross) Products (of Transpose)

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 implemented 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).

Boolean matrix products are computed via either %&% or boolArith = TRUE.

Usage

## S4 method for signature 'CsparseMatrix,diagonalMatrix'
x % * % y

## S4 method for signature 'dgeMatrix,missing'
crossprod(x, y = NULL, boolArith = NA, ...)

## S4 method for signature 'CsparseMatrix,diagonalMatrix'
crossprod(x, y = NULL, boolArith = NA, ...)
    ## .... and for many more signatures
## S4 method for signature 'CsparseMatrix,ddenseMatrix'
tcrossprod(x, y = NULL, boolArith = NA, ...)
## S4 method for signature 'TsparseMatrix,missing'
tcrossprod(x, y = NULL, boolArith = NA, ...)
  ## .... and for many more signatures

**Arguments**

- **x**: a matrix-like object
- **y**: a matrix-like object, or for `[t]crossprod()` NULL (by default); the latter case is formally equivalent to `y = x`.
- **boolArith**: logical, i.e., NA, TRUE, or FALSE. If true the result is (coerced to) a pattern matrix, i.e., "nMatrix", unless there are NA entries and the result will be a "lMatrix". If false the result is (coerced to) numeric. When NA, currently the default, the result is a pattern matrix when `x` and `y` are "nsparseMatrix" and numeric otherwise.
- ... potentially more arguments passed to and from methods.

**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. `boolArith = TRUE` for regular ("non cross") matrix products, `%*%` cannot be specified. Instead, we provide the `%&%` operator for boolean matrix products.

**Value**

A `Matrix` object, in the one argument case of an appropriate symmetric matrix class, i.e., inheriting from `symmetricMatrix`.

**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.
MatrixClass

Note

boolArith = TRUE, FALSE or NA has been newly introduced for Matrix 1.2.0 (March 2015). Its implementation has still not been tested extensively. Notably the behaviour for sparse matrices with x slots containing extra zeros had not been documented previously, see the \%\% help page.

Currently, boolArith = TRUE is implemented via CsparseMatrix coercions which may be quite inefficient for dense matrices. Contributions for efficiency improvements are welcome.

See Also
tcrossprod in R’s base, and crossprod and \%\%\%. Matrix package \%\% for boolean matrix product methods.

Examples

## A random sparse "incidence" matrix :

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

## tcrossprod() is very fast:

```r
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]
```

MatrixClass

The Matrix (Super-) Class of a Class

Description

Return the (maybe super-)class of class cl from package Matrix, returning character(0) if there is none.

Usage

MatrixClass(cl, cld = getClassDef(cl), ...Matrix = TRUE, 
dropVirtual = TRUE, ...)
MatrixFactorization-class

Arguments

- cl: string, class name
- cld: its class definition
- ...Matrix: logical indicating if the result must be of pattern "]dlniz]..Matrix" where the first letter "]dlniz]" denotes the content kind.
- dropVirtual: logical indicating if virtual classes are included or not.
- ... further arguments are passed to .selectSuperClasses().

Value

a character string

Author(s)

Martin Maechler, 24 Mar 2009

See Also

Matrix, the mother of all Matrix classes.

Examples

mkA <- setClass("A", contains="dgCMatrix")
(A <- mkA())
stopifnot(identical(
  MatrixClass("A"),
  "dgCMatrix"))

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"): ...
nearPD

**Description**

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

**Usage**

```r
nearPD(x, corr = FALSE, keepDiag = FALSE, base.matrix = 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 $n \times n$ 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.
- `base.matrix` logical indicating if the resulting `mat` component should be a base matrix or (by default) a `Matrix` of class `dpoMatrix`.
- `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.

**See Also**

Class `ngeMatrix` and the other subclasses.

**Examples**

```r
showClass("ndenseMatrix")

as(diag(3) > 0, "ndenseMatrix") # -> "nge"
```
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 eigenvalues of the approximating matrix.

ensureSymmetry logical; by default, \( \text{symmpart}(x) \) is used whenever \( \text{isSymmetric}(x) \) is not true. The user can explicitly set this to TRUE or FALSE, saving the symmetry test. \textit{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, \( \lambda_1 \). Eigenvalues \( \lambda_k \) are treated as if zero when \( \lambda_k/\lambda_1 \leq \text{eig.tol} \).

conv.tol convergence tolerance for Higham algorithm.

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 \((\text{norm}(\ast , \text{type}))\) 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 \texttt{posdefify}. The algorithm of Knol and ten Berge (1989) (not implemented here) is more general in that it allows constraints to (1) fix some rows (and columns) of the matrix and (2) force the smallest eigenvalue to have a certain value.

Note that setting \texttt{corr = TRUE} just sets \( \text{diag}(.) \leftarrow 1 \) within the algorithm.

Higham (2002) uses Dykstra’s correction, but the version by Jens Oehlschlaegel did not use it (accidentally), and still gave reasonable results; this simplification, now only used if \texttt{doDykstra = FALSE}, was active in \texttt{nearPD()} up to Matrix version 0.999375-40.

### Value

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

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

### Author(s)

Jens Oehlschlaegel donated a first version. Subsequent changes by the Matrix package authors.
nearPD

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
n.A[c("mat", "normF")]

n.A.m <- nearPD(A, corr=TRUE, do2eigen=FALSE, base.matrix=TRUE)$mat

stopifnot(exprs = {
  all.equal(n.A$mat[1,2], 0.760689917)
  all.equal(n.A$normF, 0.52779033, tolerance=1e-9)
  all.equal(n.A.m, unname(as.matrix(n.A$mat)), tolerance = 1e-15)# seen rel.d.= 1.46e-16
})

set.seed(27)
m <- matrix(round(rnorm(25),2), 5, 5)
m <- m + t(m)
diag(m) <- pmax(0, diag(m)) + 1
(m <- round(cov2cor(m), 2))

str(near.m <- nearPD(m, trace = TRUE))
round(near.m$mat, 2)

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
## But indeed, the 'corr = TRUE' constraint did ensure a better solution;
## cov2cor() does not just fix it up equivalently :
norm(pr - cov2cor(ncr$mat)) # = 0.09994
norm(pr - ncr.1$mat) # = 0.08746 / 0.08805
### 3) a real data example from a 'systemfit' model (3 eq.):
(load(system.file("external", "symW.rda", package="Matrix"))) # "symW"
dim(symW) # 24 x 24
class(symW)# "dsCMatrix": sparse symmetric
if(dev.interactive()) image(symW)
EV <- eigen(symW, only=TRUE)$values
summary(EV) ## looking more closely (EV sorted decreasingly):
tail(EV)# all 6 are negative
EV2 <- eigen(sWpos <- nearPD(symW)$mat, only=TRUE)$values
stopifnot(EV2 > 0)
if(require("sfsmisc")) {
    plot(pmax(1e-3,EV), EV2, type="o", log="xy", xaxt="n",yaxt="n")
    eaxis(1); eaxis(2)
} else plot(pmax(1e-3,EV), EV2, type="o", log="xy")
abline(0,1, col="red3",lty=2)

ngeMatrix-class

Class "ngeMatrix" of General Dense Nonzero-pattern Matrices

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.
nMatrix-class

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 \texttt{as(.)}); use, e.g., \texttt{showMethods(class="ngeMatrix")} for details.

See Also

Non-general logical dense matrix classes such as \texttt{ntrMatrix}, or \texttt{nsyMatrix}; sparse logical classes such as \texttt{ngCMatrix}.

Examples

\begin{verbatim}
showClass("ngeMatrix")
## "lgeMatrix" is really more relevant
\end{verbatim}

---

nMatrix-class  

\texttt{Class "nMatrix" of Non-zero Pattern Matrices}

Description

The \texttt{nMatrix} class is the virtual “mother” class of all \texttt{non-zero pattern} (or simply \texttt{pattern}) matrices in the \texttt{Matrix} package.

Slots

Common to all matrix object in the package:

- \texttt{Dim}: Object of class "integer" - the dimensions of the matrix - must be an integer vector with exactly two non-negative values.
- \texttt{Dimnames}: list of length two; each component containing NULL or a \texttt{character} vector length equal the corresponding \texttt{Dim} element.

Methods

There is a bunch of coercion methods (for \texttt{as(.)}), e.g.,

- \texttt{coerce} signature(from = "matrix", to = "nMatrix"): Note that these coercions (must) coerce \texttt{NA}s to non-zero, hence conceptually TRUE. This is particularly important when \texttt{sparseMatrix} objects are coerced to "nMatrix" and hence to \texttt{nsparseMatrix}.
- \texttt{coerce} signature(from = "dMatrix", to = "nMatrix"), and
- \texttt{coerce} signature(from = "lMatrix", to = "nMatrix"): For dense matrices with \texttt{NA}s, these coercions are valid since \texttt{Matrix} version 1.2.0 (still with a \texttt{warning} or a \texttt{message} if "Matrix.warn", or "Matrix.verbose" options are set.)
coerce signature(from = "nMatrix", to = "matrix"): ...
coerce signature(from = "nMatrix", to = "dMatrix"): ...
coerce signature(from = "nMatrix", to = "lMatrix"): ...

Additional methods contain group methods, such as

**Ops** signature(e1 = "nMatrix", e2 = "...."), ...
**Arith** signature(e1 = "nMatrix", e2 = "...."), ...
**Compare** signature(e1 = "nMatrix", e2 = "...."), ...
**Logic** signature(e1 = "nMatrix", e2 = "...."), ...
**Summary** signature(x = "nMatrix", "...."), ...

See Also

The classes `lMatrix`, `nsparseMatrix`, and the mother class, `Matrix`.

Examples

gClass("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 "ngTMatrix"

nnzero(x, na.counted = NA)

---

**nnzero**

The Number of Non-Zero Values of a Matrix

Description

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

Usage

`nnzero(x, na.counted = NA)`
### Arguments

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

For sparse matrices, you may often want to use `na.counted = TRUE`.

### Value

the number of non zero entries in `x` (typically `integer`).

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

### Methods

- `signature(x = "ANY")` the default method for non-`Matrix` class objects, simply counts the number 0's in `x`, counting NA's depending on the `na.counted` argument, see above.
- `signature(x = "denseMatrix")` conceptually the same as for traditional `matrix` objects, care has to be taken for "`symmetricMatrix`" objects.
- `signature(x = "diagonalMatrix"), and signature(x = "indMatrix")` fast simple methods for these special "`sparseMatrix`" classes.
- `signature(x = "sparseMatrix")` typically, the most interesting method, also carefully taking "`symmetricMatrix" objects into account.

### See Also

The `Matrix` class also has a `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`.

- `drop0`, `zapsmall`.

### Examples

```r
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, "TsparseMatrix"))

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)
```
norm

Description

Computes a matrix norm of \( x \), using Lapack for dense matrices. The norm can be the one ("O", or "1") norm, the infinity ("I") norm, the Frobenius ("F") norm, the maximum modulus ("M") among elements of a matrix, or the spectral norm or 2-norm ("2"), as determined by the value of type.

Usage

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

Arguments

- \( x \) a real or complex matrix.
- \( \text{type} \) A character indicating the type of norm desired.
  - "O", "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);
  - "M" or "m" specifies the maximum modulus of all the elements in \( x \); and
  - "2" specifies the "spectral norm" or 2-norm, which is the largest singular value (\( \text{svd} \)) of \( x \).
  
  The default is "O". Only the first character of type[1] is used.
  
  \( \ldots \) further arguments passed to or from other methods.

Details

For dense matrices, the methods eventually call the Lapack functions \( \text{dlange} \), \( \text{dlansy} \), \( \text{dlantr} \), \( \text{zlange} \), \( \text{zlansy} \), and \( \text{zlantr} \).

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.

The norm() function from R’s base package.

Examples

x <- Hilbert(9)
norm(x) # = "O" = "1"
stopifnot(identical(norm(x), norm(x, "1")))
norm(x, "I") # the same, because 'x' is symmetric

allnorms <- function(d) vapply(c("1","I","F","M","2"), norm, x = d, double(1))
allnorms(x)
allnorms(Hilbert(10))

i <- c(1,3:8); j <- c(2,9,6:10); x <- 7 * (1:7)
A <- sparseMatrix(i, j, x = x) ## 8 x 10 "dgCMatrix"
(sA <- sparseMatrix(i, j, x = x, symmetric = TRUE)) ## 10 x 10 "dsCMatrix"
(tA <- sparseMatrix(i, j, x = x, triangular= TRUE)) ## 10 x 10 "dtCMatrix"
(allnorms(A) -> nA)
allnorms(sA)
allnorms(tA)
stopifnot(all.equal(nA, allnorms(as(A, "matrix"))),
  all.equal(nA, allnorms(tA))) # because tA == rbind(A, 0, 0)
A. <- A; A.[1,3] <- NA
stopifnot(is.na(allnorms(A.))) # gave error

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 (TsparseMatrix, subclasses ngTMatrix, nsTMatrix, and ntTMatrix which really contain pairs, not triplets) or in compressed column-oriented form (class CsparseMatrix, subclasses ngCMatrix, nsCMatrix, and ntCMatrix) or—rarely—in compressed row-oriented form (class RsparseMatrix, subclasses 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 row-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
mmm <- !mm # no longer sparse
# consistency check:
stopifnot(xor(as(m, "matrix"),
    as(mmm, "matrix")))
# low-level way of adding "non-structural zeros" :
```
nsyMatrix-class

Symmetric Dense Nonzero-Pattern Matrices

Description

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.

Methods

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

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

Both extend Class "triangularMatrix", directly, and class "denseMatrix", "lMatrix" and others, indirectly, use 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

ngeMatrix, Matrix, t

Examples

(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, "symmetricMatrix")) # -> "nsy"
str (sM <- as(sM, "packedMatrix")) # -> "nsp": packed symmetric
See Also

Classes `ngeMatrix`, `Matrix`; function `t`

Examples

```r
showClass("ntrMatrix")

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

---

**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**

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

---

**packedMatrix-class**  
*Virtual Class "packedMatrix" of Packed Dense Matrices*

### Description

Class "packedMatrix" is the virtual class of dense symmetric or triangular matrices in "packed" format, storing only the `choose(n+1,2) == n*(n+1)/2` elements of the upper or lower triangle of an `n`-by-`n` matrix. It is used to define common methods for efficient subsetting, transposing, etc. of its proper subclasses: currently `[dln]spMatrix` (packed symmetric), `[dln]tpMatrix` (packed triangular), and subclasses of these, such as "dppMatrix","pCholesky", and "pBunchKaufman".

### Slots

- `uplo`: "character"; either "U", for upper triangular, and "L", for lower.
- `Dim`, `Dimnames`: as all `Matrix` objects.
Extends


Methods

pack signature(x = "packedMatrix"): ...
unpack signature(x = "packedMatrix"): ...
isSymmetric signature(object = "packedMatrix"): ...
isTriangular signature(object = "packedMatrix"): ...
isDiagonal signature(object = "packedMatrix"): ...
t signature(x = "packedMatrix"): ...
diag signature(x = "packedMatrix"): ...
diag<- signature(x = "packedMatrix"): ...
...
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):

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

where $p$ is the “permutation vector” corresponding to the permutation matrix $P$ (see first note), and $i(p)$ is short for $\text{invPerm}(p)$.
Also one could argue that these are really only two cases if you take into account that inversion (solve) and transposition (t) are the same for permutation matrices \( P \).

Objects from the Class

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

Slots

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

Extends

Class "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

$$P \leftarrow \text{as}(p, \text{"pMatrix"})$$
$$p \leftarrow P@\text{perm}$$

see also the ‘Examples’.
“Row-indexing” a permutation matrix typically returns an "indMatrix". See "indMatrix" for all other subsetting/indexing and subassignment ($A[\ldots] \leftarrow v$) operations.

See Also

`invPerm(p)` computes the inverse permutation of an integer (index) vector $p$.

Examples

```r
(pm1 <- as(as.integer(c(2,3,1)), "pMatrix"))
t(pm1) # is the same as
solve(pm1)
pml %*% t(pm1) # check that the transpose is the inverse
stopifnot(all(diag(3) == as(pm1 %*% t(pm1), "matrix")),
      is.logical(as(pm1, "matrix")))
```

```r
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
pm1 %*% mm
try(as(as.integer(c(3,3,1)), "pMatrix"))# Error: not a permutation
as(pm1, "TsparseMatrix")
p10[1:7, 1:4] # gives an "ngTMatrix" (most economic!)
```

```r
## 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

```r
formatSpMatrix(x, digits = NULL, maxp = 1e9,
   cl = getClassDef(class(x)), zero.print = ".",
   col.names, note.dropping.colnames = TRUE, uniDiag = TRUE,
   align = c("fancy", "right"))
```

```r
printSpMatrix(x, digits = NULL, maxp = max(100L, getOption("max.print")),
   cl = getClassDef(class(x)),
   zero.print = ".", col.names, note.dropping.colnames = TRUE,
   uniDiag = TRUE, col.trailer = "",
   align = c("fancy", "right"))
```

```r
printSpMatrix2(x, digits = NULL, maxp = max(100L, getOption("max.print")),
   zero.print = ".", col.names, note.dropping.colnames = TRUE,
   uniDiag = TRUE, suppressCols = NULL, col.trailer = if(suppCols) "......" else "",
   align = c("fancy", "right"),
   width = getOption("width"), fitWidth = TRUE)
```

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. Typically should not be smaller than around 1000; values smaller than 100 are silently "rounded up" to 100.
- **cl**: 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.
- **uniDiag**: logical indicating if the diagonal entries of a sparse unit triangular or unit-diagonal matrix should be formatted as "1" instead of "1" (to emphasize that the 1’s are "structural").
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).

width number, a positive integer, indicating the approximately desired (line) width of
the output, see also fitWidth.

fitWidth logical indicating if some effort should be made to match the desired width or
temporarily enlarge that if deemed necessary.

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

## Possibilities with column names printing:
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")

descriptions

QR Decomposition – S4 Methods and Generic

Description

The Matrix package provides methods for the QR decomposition of special classes of matrices. There is a generic function which uses \textit{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, row.names = TRUE)

Arguments

\begin{itemize}
\item \textbf{x} \hspace{1cm} a numeric or complex matrix whose QR decomposition is to be computed. Logical matrices are coerced to numeric.
\item \textbf{qr} \hspace{1cm} a QR decomposition of the type computed by \textit{qr}.
\item \textbf{complete} \hspace{1cm} logical indicating whether the \textbf{R} matrix is to be completed by binding zero-value rows beneath the square upper triangle.
\item \textbf{backPermute} \hspace{1cm} logical indicating if the rows of the \textbf{R} matrix should be back permuted such that qrR(\textit{\textbullet})’s result can be used directly to reconstruct the original matrix \textit{X}.
\item \textbf{row.names} \hspace{1cm} logical indicating if \texttt{rownames} should propagated to the result.
\item \textbf{...} \hspace{1cm} further arguments passed to or from other methods
\end{itemize}

Methods

\begin{itemize}
\item \texttt{x = "dgCMatrix"} \hspace{1cm} QR decomposition of a general sparse double-precision matrix with nrow(x) >= ncol(x). Returns an object of class "\texttt{sparseQR}".
\item \texttt{x = "sparseMatrix"} \hspace{1cm} works via "\texttt{dgCMatrix}".
\end{itemize}
See Also

qr; then, the class documentations, mainly sparseQR, and also dgCMatrix.

Examples

```r
#--------------- example of pivoting -- from base' qraux.Rd --------------
X <- 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))
rownames(X) <- paste0("r", seq_len(nrow(X)))
dnX <- dimnames(X)
bX <- X # [b]ase version of X
X <- as(bX, "sparseMatrix")
X # is singular, columns "b2" and "c3" are "extra"
stopifnot(identical(dimnames(X), dnX))# some versions changed X's dimnames!
c(rankMatrix(X)) # = 4 (not 6)
m <- function(.) as(.,. "matrix")

#----- regular case -----------------------------------------------
Xr <- X[ , -c(3,6)] # the "regular" (non-singular) version of X
stopifnot(rankMatrix(Xr) == ncol(Xr))
Y <- cbind(y <- setNames(1:6, paste0("y", 1:6)))

## regular case:
qXr <- qr(Xr)
qxr <- qr(m(Xr))
qcfXy <- qr.coef (qXr, y) # vector
qcfXY <- qr.coef (qXr, Y) # 4x1 dgeMatrix
cf <- c(int=6, b1=-3, c1=-2, c2=-1)
doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA"))
tolE <- if(doExtras) 1e-15 else 1e-13
stopifnot(exprs = {
  all.equal(qr.coef(qxr, y), cf, tol=tolE)
  all.equal(qr.coef(qXR, Y), m(cf), tol=tolE)
  all.equal(qcfXY, cf, tol=tolE)
  all.equal(qcfXY, m(cf), tol=tolE)
  all.equal(y, qr.fitted(qxr, y), tol=2*tolE)
  all.equal(y, qr.fitted(qXR, y), tol=2*tolE)
  all.equal(m(qr.fitted(qXR, Y)), qr.fitted(qxr, Y), tol=tolE)
  all.equal(m(qr.resid (qXR, y)), qr.resid (qxr, y), tol=tolE)
  all.equal(m(qr.resid (qXR, Y)), qr.resid (qxr, Y), tol=tolE)
})

#----- rank-deficient ("singular") case ---------------------------
(qX <- qr(X)) # both @p and @q are non-trivial permutations
qx <- qr(m(X)); str(qx) # @pivot is non-trivial, too
drop0(R. <- qr.R(qX), tol=tolE) # columns *permuted*: c3 b1 ..
Q. <- qr.Q(qX)
```
rankMatrix

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.

(*) The rank of a \( n \times m \) matrix \( A \), \( rk(A) \), is the maximal number of linearly independent columns (or rows); hence \( rk(A) \leq \min(n,m) \).

Usage

```r
rankMatrix(x, tol = NULL, method = c("tolNorm2", "qr.R", "qrLlPACK", "qr", 
"useGrad", "maybeGrad"),
  sval = svd(x, 0, 0)$d, warn.t = TRUE, warn.qr = TRUE)
```

```r
qr2rankMatrix(qr, tol = NULL, isBqr = is.qr(qr), do.warn = 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, \( \max(\text{dim}(x)) \times \text{Machine}\$\text{double.e}\text{ps} \) 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 \( \geq tol \times \max(sval) \);
"qrLINPACK": for a dense matrix, this is the rank of \texttt{qr(x, tol, LAPACK=FALSE)} (which is \texttt{qr(...)$rank});
This ("qr", dense) version used to be \textit{the} recommended way to compute a matrix rank for a while in the past.
For sparse \texttt{x}, this is equivalent to "qr.R".

"qr.R": this is the rank of triangular matrix \( R \), where \texttt{qr()} uses LAPACK or a "sparseQR" method (see \texttt{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 \(|d_i| \geq \text{tol} \cdot \max(|d_i|)\).

"qr": is for back compatibility; for dense \texttt{x}, it corresponds to "qrLINPACK", whereas for sparse \texttt{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 \texttt{x}, as in that case, when \( \text{sval} \) is not specified, the default, computing \texttt{svd()} currently coerces \texttt{x} to a dense matrix.

"useGrad": considering the “gradient” of the (decreasing) singular values, the index of the \textit{smallest} gap.

"maybeGrad": choosing method "useGrad" only when that seems \textit{reasonable}; otherwise using "tolNorm2".

\begin{itemize}
  \item \texttt{sval} \texttt{numeric vector of non-increasing singular values of x; typically unspecified and computed from x when needed, i.e., unless method = "qr".}
  \item \texttt{warn.t} \texttt{logical indicating if \texttt{rankMatrix()} should warn when it needs \texttt{t(x)} instead of \texttt{x}. Currently, for method = "qr" only, gives a warning by default because the caller often could have passed \texttt{t(x)} directly, more efficiently.}
  \item \texttt{warn.qr} \texttt{in the QR cases (i.e., if method starts with "qr"), \texttt{rankMatrix()} calls \texttt{qr2rankMarix(, do.warn = warn.qr)}, see below.}
  \item \texttt{qr} \texttt{an R object resulting from \texttt{qr(x, ...), i.e., typically inheriting from class "qr" or "sparseQR".}}
  \item \texttt{isBqr} \texttt{logical indicating if qr is resulting from base \texttt{qr}(). (Otherwise, it is typically from \texttt{Matrix} package sparse \texttt{qr}.)}
  \item \texttt{do.warn} \texttt{logical; if true, warn about non-finite (or in the sparseQR case negative) diagonal entries in the R matrix of the QR decomposition. Do not change lightly!}
\end{itemize}

Details
\texttt{qr2rankMatrix()} is typically called from \texttt{rankMatrix()} for the "qr"* methods, but can be used directly - much more efficiently in case the qr-decomposition is available anyway.

Value
If \texttt{x} is a matrix of all \( \emptyset \) (or of zero dimension), the rank is zero; otherwise, typically a positive integer in \( 1:\text{min(dim(x))} \) with attributes detailing the method used.

There are rare cases where the sparse \( QR \) decomposition “fails” in so far as the diagonal entries of \( R \), the \( d_i \) (see above), end with non-finite, typically \texttt{NaN} entries. Then, a warning is signalled (unless \texttt{warn.qr / do.warn} is not true) and \texttt{NA} (specifically, \texttt{NA_integer_}) is returned.
Note

For large sparse matrices \( x \), unless you can specify \( sval \) yourself, currently \( \text{method} = \text{"qr"} \) may be the only feasible one, as the others need \( sval \) and call \( \text{svd()} \) which currently coerces \( x \) to a \text{denseMatrix} which may be very slow or impossible, depending on the matrix dimensions.

Note that in the case of sparse \( x \), \( \text{method} = \text{"qr"} \), all non-strictly zero diagonal entries \( d \), where counted, up to including \text{Matrix} version 1.1-0, i.e., that method implicitly used \( \text{tol} = 0 \), see also the \text{set.seed(42)} example below.

Author(s)

Martin Maechler; for the "*Grad" methods building on suggestions by Ravi Varadhan.

See Also

\text{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.R  qrLINPACK  qr  useGrad  maybeGrad
##  11  11  12  12  11  11
## The meaning of \textquote{tol}' for method="qrLINPACK" and \textquote{dense*} \( x \) is not entirely "scale free"
rMQL <- function(ex, M) rankMatrix(M, method="qrLINPACK",tol = 10^-ex)
rMQR <- function(ex, M) rankMatrix(M, method="qr.R", tol = 10^-ex)
sapply(5:15, rMQL, M = H12) # result is platform dependent
##  7  7  8  10  10  11  11  12  12  12  {x86_64}
sapply(5:15, rMQL, M = 1000 * H12) # not identical unfortunately
##  7  7  8  10  10  11  12  12  12  12  sapply(5:15, rMQR, M = H12)
##  5  6  7  8  8  9  9 10 10 11 11
sapply(5:15, rMQR, 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 \textquote{useGrad} has 14.
sapply(meths, function(.m.) rankMatrix(M15, method = .m., tol = 1e-7)) # all 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 (2013)
```
rcond <- system.time(r2 <- rankMatrix(L, method = "qr")) # considerably faster!

## 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,

## S4 method for signature 'sparseMatrix,character'
rcond(x, norm, useInv=FALSE, ...)

Arguments

x

an \texttt{R} object that inherits from the \texttt{Matrix} class.

norm

character indicating the type of norm to be used in the estimate. The
default is "\texttt{O}" for the 1-norm ("\texttt{O}" is equivalent to "\texttt{1}").
For sparse matrices, when \texttt{useInv=TRUE}, \texttt{norm} can be any
of the kinds allowed for \texttt{norm}; otherwise, the other possible
value is "\texttt{I}" for the infinity norm, see also \texttt{norm}.

useInv

logical (or "\texttt{Matrix}" containing \texttt{solve}(\texttt{x}))

...
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

\(\text{norm, kappa()}\) from package base computes an approximate condition number of a “traditional” matrix, even non-square ones, with respect to the \(p = 2\) (Euclidean) \(\text{norm.solve}\).

\(\text{condest}\), a newer approximate estimate of the (1-norm) condition number, particularly efficient for large sparse matrices.

Examples

\(x \leftarrow \text{Matrix(rnorm(9), 3, 3)}\)
\(rcond(x)\)

## Typically "the same" (with more computational effort):
\(1 / (\text{norm(x)} \times \text{norm(solve(x))})\)
\(rcond(\text{Hilbert(9)})\)  # should be about 9.1e-13

## For non-square matrices:
\(rcond(x1 \leftarrow \text{cbind(1,1:10)})\) # 0.05278
\(rcond(x2 \leftarrow \text{cbind(x1, 2:11)})\) # 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
rcond(Matrix(cbind(0, diag(2:-1))))

## Large sparse example --------
m <- Matrix(c(3,0:2), 2,2)
M <- bdiag(kronecker(Diagonal(2), m), kronecker(m,m))
36*(IM <- solve(M)) # still sparse
MM <- kronecker(Diagonal(10), kronecker(Diagonal(5), kronecker(m,M)))
dim(M3 <- kronecker(bdiag(M,M,MM)) # 12'800 ^ 2
if(interactive()) ## takes about 2 seconds if you have >= 8 GB RAM
  system.time(r <- rcond(M3))
## whereas this is *fast* even though it computes solve(M3)
  system.time(r. <- rcond(M3, useInv=TRUE))
if(interactive()) ## the values are not the same
  c(r, r.) # 0.05555 0.013888
## for all 4 norms available for sparseMatrix :
cbind(rr <- sapply(c("1","I","F","M"),
  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**

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
(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").
rlc: 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. The default has rounded gaussian non-zero entries, and rand.x = NULL generates random pattern matrices, i.e. inheriting from nsparseMatrix.

Usage

rsparsematrix(nrow, ncol, density, nnz = round(density * maxE),
              symmetric = FALSE,
              rand.x = function(n) signif(rnorm(n), 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
  NULL or 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 repr.
Details

The algorithm first samples “encoded” \((i, j)\)s without replacement, via one dimensional indices, if not symmetric \(\text{sample.int}(\text{nrow} \times \text{ncol}, \text{nnz})\), then—if \(\text{rand.x}\) is not \(\text{NULL}\)—gets \(x \leftarrow \text{rand.x}(\text{nnz})\) and calls \(\text{sparseMatrix}(i=i, j=j, x=x, ..)\). When \(\text{rand.x}=\text{NULL}\), \(\text{sparseMatrix}(i=i, j=j, ..)\) will return a pattern matrix (i.e., inheriting from \(\text{nsparseMatrix}\)).

Value

a \(\text{sparseMatrix}\), say \(M\) of dimension \((\text{nrow}, \text{ncol})\), i.e., with \(\text{dim}(M) == c(\text{nrow}, \text{ncol})\), 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

set.seed(17)# to be reproducible
\[
M \leftarrow \text{rsparsematrix}(8, 12, \text{nnz} = 30) \quad \# \text{small example, not very sparse}
\]
\[
M
\]
\[
M1 \leftarrow \text{rsparsematrix}(1000, 20, \text{nnz} = 123, \text{rand.x} = \text{runif})
\]
\[
\text{summary}(M1)
\]

## a random *symmetric* Matrix
\[
(S9 \leftarrow \text{rsparsematrix}(9, 9, \text{nnz} = 10, \text{symmetric}=\text{TRUE})) \quad \# \text{dsCMatrix}
\]
\[
\text{nnzero}(S9)\# == 20: \text{as 'nnz' only counts one "triangle"
}

## a random pattern aka boolean Matrix (no 'x' slot):
\[
(n7 \leftarrow \text{rsparsematrix}(5, 12, \text{nnz} = 10, \text{rand.x} = \text{NULL})
\]

## a \([T]\)riplet representation \(\text{sparseMatrix}\):
\[
T2 \leftarrow \text{rsparsematrix}(40, 12, \text{nnz} = 99, \text{repr} = "T")
\]
\[
\text{head}(T2)
\]

RsparseMatrix-class

Class "RsparseMatrix" of Sparse Matrices in Row-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 \(\text{showClass("RsparseMatrix")}\) for its subclasses.

Slots

\(j\): 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.

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

\(\text{Dim, Dimnames}\): inherited from the superclass, see \(\text{sparseMatrix}\).
**Schur**

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.
Schur

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 \( A = Q T^t(Q) \), 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

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))))
)
Class "Schur" of Schur Matrix Factorizations

Description

Class "Schur" is the class of Schur matrix factorizations. These are a generalization of eigenvalue (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()
Methods for generic function `solve`, for solving linear systems of equations. These solve for \( X \) in

\[ AX = B \]

where \( A \) is a square matrix and \( X \) and \( B \) are matrices with compatible dimensions. The usual \( \mathcal{R} \) syntax is

\[ x <- \text{solve}(a, b, ...) \]

where \( b \) may also be a vector, in which case it is treated as a 1-column matrix. Methods support \( a \) inheriting from virtual classes \texttt{Matrix} and \texttt{MatrixFactorization} and \( b \) inheriting from virtual classes \texttt{Matrix} and \texttt{sparseVector}.

**Usage**

```r
## solve(a, b, ...) # the two-argument version, almost always preferred to
## solve(a, ...) # the *rarely needed* one-argument version

## S4 method for signature 'dgCMatrix,missing'
solve(a, b, sparse = NA, ...)

## S4 method for signature 'dgCMatrix,matrix'
solve(a, b, sparse = FALSE, ...)

## S4 method for signature 'dgCMatrix,denseMatrix'
solve(a, b, sparse = FALSE, ...)

## S4 method for signature 'dgCMatrix,sparseMatrix'
solve(a, b, sparse = NA, tol = .Machine$double.eps, ...)

## S4 method for signature 'CHMfactor,denseMatrix'
solve(a, b, system = c("A", "LDLt", "LD", "DLt", "L", "Lt", "D", "P", "Pt"), ...)
```

**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 \( Ax = b \).
- \texttt{sparse}: only when \( a \) is a \texttt{sparseMatrix}: logical specifying if the result should also (formally) be sparse.
- \texttt{tol}: only when \( a \) is a \texttt{sparseMatrix} and \texttt{sparse} is \texttt{TRUE}: an error is signaled if the ratio \( \min(d)/\max(d) \), where \( d = \text{abs(diag(U))} \) and \( A = L U \), is \textless{} than \texttt{tol}, indicating near-singular \( A \).
- \texttt{system}: only when \( a \) is a \texttt{CHMfactor}: character string indicating the kind of linear system to be solved, see below. Note that the default, "A", does \textit{not} solve the triangular system (but "L" does).
- \texttt{...}: potentially further arguments to the methods.
Methods

signature(a = "ANY", b = "ANY") is simply the base package's S3 generic solve.

signature(a = "CHMfactor", b = "..."), system= * 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 sparse, positive-definite matrix that was factored to produce \(a\). Analogously, 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 \(x <- P \%*% b\).

If \(b\) is a sparseMatrix, system is used as above the corresponding sparse CHOLMOD algorithm is called.

signature(a = "ddenseMatrix", b = "...") (for all \(b\)) work via as(a, "generalMatrix"), using its methods, see below.

signature(a = "denseLU", b = "missing") basically computes uses triangular forward- and back-solve.

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 = "dsparseMatrix") , 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 (\(b = \text{"ddenseMatrix"}\)) 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 dsptri, dsptrs, dsytrs and dsytri.

signature(a = "dtCMatrix", b = "CsparseMatrix"),
signature(a = "dtCMatrix", 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, "CsparseMatrix")
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 = "TsparseMatrix", b = "ANY") all work via as(a, "CsparseMatrix").

See Also

solve, lu, and class documentations CHMfactor, sparseLU, and MatrixFactorization.

Examples

## 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(ia <- 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")
  ## Fails 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)
idad <- solve(a)
ias <- solve(a, sparse=TRUE)
stopifnot(all.equal(as(ias,"denseMatrix"), iad, tolerance=1e-14))
I. <- iad %x% a ; image(I.)
I0 <- drop0(zapsmall(I.)); image(I0)
.I <- a %x% iad
.I0 <- drop0(zapsmall(.
1))
stopifnot( all.equal(as(I0, "diagonalMatrix"), Diagonal(n)),
  all.equal(as(.10,"diagonalMatrix"), Diagonal(n)) )
Description

Construct a sparse model or “design” matrix, from a formula and data frame (sparse.model.matrix) or a single factor (fac2sparse).

The fac2[Ss]parse() functions are utilities, also used internally in the principal user level function sparse.model.matrix().

Usage

sparse.model.matrix(object, data = environment(object),
contrasts.arg = NULL, xlev = NULL, transpose = FALSE,
drop.unused.levels = FALSE, row.names = TRUE,
sep = "", verbose = FALSE, ...)

fac2sparse(from, to = c("d", "l", "n"),
drop.unused.levels = TRUE, repr = c("C", "R", "T"), giveCsparse)

fac2Sparse(from, to = c("d", "l", "n"),
drop.unused.levels = TRUE, repr = c("C", "R", "T"), giveCsparse,
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.
sep character string passed to paste() when constructing column names from the variable name and its levels.
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 deprecated, replaced with repr; logical indicating if the result must be a CsparseMatrix.

repr character string, one of "C", "T", or "R", specifying the sparse representation to be used for the result, i.e., one from the super classes CsparseMatrix, TsparseMatrix, or RsparseMatrix.

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 repr = "C" as per default; a TsparseMatrix or RsparseMatrix, 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.

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) ==
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 $P'LUQ$.

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/pores_1.mtx", 
    package = "Matrix")), "CsparseMatrix")

## with dimnames(.) - to see that they propagate to L, U :
A <- as(readMM(system.file("external/pores_1.mtx", 
    package = "Matrix")), "CsparseMatrix")

## with dimnames(.) - to see that they propagate to L, U :
dimnames(A) <- list(paste0("r", seq_len(nrow(A))), 
    paste0("C", seq_len(ncol(A))))

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$P, as(luA@p +1L, "pMatrix")),
    identical(xA$Q, as(luA@q +1L, "pMatrix")))

P.LUQ <- with(xA, t(P) %*% L %*% U %*% Q)
stopifnot(all.equal(unname(A), unname(P.LUQ), tolerance = 1e-12))

## permute rows and columns of original matrix
pA <- A[luA@p + 1L, luA@q + 1L] 
P.AQ. <- with(xA, P %*% A %*% Q)
stopifnot(all.equal(unname(pA), unname(P.AQ.), tolerance = 1e-12))

pLU <- drop0(luA@L %*% luA@U) # L %*% U -- dropping extra zeros
stopifnot(all.equal(pA, pLU, tolerance = 1e-12))
```
**SparseM-conversions**  
*Sparse Matrix Coercion from and to those from package SparseM*

**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. All these work via the usual `as(. , "<class>")` coercion,

\[\text{as(from, Class)}\]

**Methods**

- `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 = "Matrix", 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"` 

**See Also**

The documentation in CRAN package **SparseM**, such as **SparseM.ontology**, and one important class, **matrix.csr**.

**sparseMatrix**  
*General Sparse Matrix Construction from Nonzero Entries*

**Description**

User-friendly construction of sparse matrices (inheriting from virtual class **CsparseMatrix**, **RsparseMatrix**, or **TsparseMatrix**) from the positions and values of their nonzero entries. This interface is recommended over direct construction via calls such as `new("..[CRT]Matrix", ...)`. 

...
sparseMatrix

Usage

sparseMatrix(i, j, p, x, dims, dimnames,
    symmetric = FALSE, triangular = FALSE, index1 = TRUE,
    repr = c("C", "R", "T"), giveCsparse,
    check = TRUE, use.last.ij = FALSE)

Arguments

i, j
 integer vectors of equal length specifying the positions (row and column indices) of the nonzero (or non-TRUE) entries of the matrix. Note that, when x is non-missing, the \( x_k \) corresponding to repeated pairs \((i_k, j_k)\) are added, for consistency with the definition of class \(\text{TsparseMatrix}\), unless use.last.ij is TRUE, in which case only the last such \( x_k \) is used.

p
 integer 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 \( i \), \( j \), and \( p \) must be missing.

x
 optional, typically nonzero values for the matrix entries. If specified, then the length must equal that of \( i \) (or \( j \)) or equal 1, in which case \( x \) is recycled as necessary. If missing, then the result is a nonzero pattern matrix, i.e., inheriting from class \(\text{nsparseMatrix}\).

dims
 optional length-2 integer vector of matrix dimensions. If missing, then \(!\text{index1}+\text{c}(\max(i),\max(j))\) is used.

dimnames
 optional list of \text{dimnames}; if missing, then \text{NULL} ones are used.

symmetric
 logical indicating if the resulting matrix should be symmetric. In that case, \((i, j, p)\) should specify only one triangle (upper or lower).

triangular
 logical indicating if the resulting matrix should be triangular. In that case, \((i, j, p)\) should specify only one triangle (upper or lower).

index1
 logical. If TRUE (the default), then \( i \) and \( j \) are interpreted as 1-based indices, following the R convention. That is, counting of rows and columns starts at 1. If FALSE, then they are interpreted as 0-based indices.

repr
 character string, one of "C", "R", and "T", specifying the representation of the sparse matrix result, i.e., specifying one of the virtual classes \(\text{CsparseMatrix}\), \(\text{RsparseMatrix}\), and \(\text{TsparseMatrix}\).

giveCsparse
 (deprecated, replaced by repr) logical indicating if the result should inherit from \(\text{CsparseMatrix}\) or \(\text{TsparseMatrix}\). Note that operations involving \(\text{CsparseMatrix}\) are very often (but not always) more efficient.

check
 logical indicating whether to check that the result is formally valid before returning. Do not set to FALSE unless you know what you are doing!

use.last.ij
 logical indicating if, in the case of repeated (duplicated) pairs \((i_k, j_k)\), only the last pair should be used. FALSE (the default) is consistent with the definition of class \(\text{TsparseMatrix}\).

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, \( \text{rep(seq_along(dp),dp)} \) where \( dp \leftarrow \text{diff}(p) \), is used as the (1-based) row or column indices.

You cannot set both singular and triangular to true; rather use \( \text{Diagonal()} \) (or its alternatives, see there).

The values of i, j, p and index1 are used to create 1-based index vectors i and j from which a \( \text{TsparseMatrix} \) 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 \( \text{TsparseMatrix} \) class, unless use.last.ij is set to true.

By default, when \( \text{repr = "C"} \), the \( \text{CsparseMatrix} \) derived from this triplet form is returned, where \( \text{repr = "R"} \) now allows to directly get an \( \text{RsparseMatrix} \) and \( \text{repr = "T"} \) leaves the result as \( \text{TsparseMatrix} \).

The reason for returning a \( \text{CsparseMatrix} \) 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 \( \text{CsparseMatrix} \) is a unique representation of the sparse matrix.

Value

A sparse matrix, by default in compressed sparse column format and (formally) without symmetric or triangular structure, i.e., by default inheriting from both \( \text{CsparseMatrix} \) and \( \text{generalMatrix} \).

Note

You \textit{do} need to use \texttt{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

\( \text{Matrix}(*, \text{sparse=TRUE}) \) for the constructor of such matrices from a \textit{dense} matrix. That is easier in small sample, but much less efficient (or impossible) for large matrices, where something like \texttt{sparsesMatrix()} is needed. Further \( \text{bdia} \) and \( \text{Diagonal} \) for (block-)diagonal and \( \text{bandSparse} \) for banded sparse matrix constructors.

Random sparse matrices via \( \text{rsparsematrix}() \).

The standard \texttt{R xtabs}(*, sparse=TRUE), for sparse tables and \texttt{sparses.model.matrix}() for building sparse model matrices.

Consider \( \text{CsparseMatrix} \) 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)) ## 8 x 10 "dgCMatrix"
summary(A)
```
str(A) # note that *internally* 0-based row indices are used

(sA <- sparseMatrix(i, j, x = x, symmetric = TRUE)) # 10 x 10 "dsCMatrix"
(tA <- sparseMatrix(i, j, x = x, triangular= TRUE)) # 10 x 10 "dtCMatrix"
stopifnot( all(sA == tA + t(tA)) ,
          identical(sA, as(tA + t(tA), "symmetricMatrix")))

## 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))
(A1 <- sparseMatrix(i[perm], j[perm], x = x[perm]))
stopifnot(identical(A, A1))

## 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*nx* 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))
stopifnot(isSymmetric(sy),

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", uniqT=FALSE): Returns an object of S3 class "sparseSummary" which is basically a data.frame with columns (i,j,x) (or just (i,j) for nsparseMatrix 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.

cbind2 (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 dgeMatrix 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(.)

## factor -> t(<sparse design matrix> ):
(fact <- gl(5, 3, 30, labels = LETTERS[1:5]))
(Xt <- as(fact, "sparseMatrix")) # indicator rows
## missing values --> all-0 columns:
f.mis <- fact
i.mis <- c(3:5, 17)
is.na(f.mis) <- i.mis
Xt != (X. <- as(f.mis, "sparseMatrix")) # differ only in columns 3:5,17
stopifnot(all(X.[,i.mis] == 0), all(Xt[,-i.mis] == X.[,-i.mis]))

---

### Description

Objects class "sparseQR" represent a QR decomposition of a sparse \(m \times n\) ("long": \(m \geq n\)) rectangular matrix \(A\), typically resulting from `qr()`, see ‘Details’ notably about row and column permutations for pivoting.

### Details

For a sparse \(m \times n\) ("long": \(m \geq n\)) rectangular matrix \(A\), the sparse QR decomposition is either of the form \(PA = QR\) with a (row) permutation matrix \(P\), (encoded in the \(p\) slot of the result) if the \(q\) slot is of length 0, or of the form \(PAP^* = QR\) with an extra (column) permutation matrix \(P^*\) (encoded in the \(q\) slot). Note that the row permutation \(PA\) in \(R\) is simply \(A[\ell+1, ]\) where \(p\) is the \(p\)-slot, 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 \(R\), and \(AP^*\) in \(R\) is simply \(A[\ell, q+1]\).

\(R\) is an \(m \times n\) matrix that is zero below the main diagonal, i.e., upper triangular \((m \times m)\) with \(m - n\) extra zero rows.

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. Note however that `qr.Q()` returns the row permuted matrix \(Q^* := P^{-1}Q = PQ\) as permutation matrices are orthogonal; and \(Q^*\) is orthogonal itself because \(Q\) and \(P\) are. This is useful because then, as in the dense matrix and `base R` matrix `qr` case, we have the mathematical identity

\[
PA = Q^* R,
\]

in \(R\) as

\[
A[p+1,] == qr.Q(*) \mod R.
\]

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 "dgCMatrix". 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.
- **R**: Object of class "dgCMatrix": An upper triangular matrix of the same dimension as $X$.
- **q**: Object of class "integer": Permutation applied from the right, i.e., to the columns of the original matrix. 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 `options()` either "Matrix.quiet.qr.R" or (the more general) either "Matrix.quiet" to `TRUE`.
- `qr.Q` signature(qr = "sparseQR"): compute the orthogonal $Q$ matrix of the QR decomposition.
- `qr.coef` signature(qr = "sparseQR", y = "ddenseMatrix"): ...
- `qr.coef` signature(qr = "sparseQR", y = "matrix"): ...
- `qr.fitted` signature(qr = "sparseQR", y = "ddenseMatrix"): ...
- `qr.fitted` signature(qr = "sparseQR", y = "matrix"): ...
- `qr.qty` signature(qr = "sparseQR", y = "ddenseMatrix"): ...
- `qr.qty` signature(qr = "sparseQR", y = "matrix"): ...
- `qr.qy` signature(qr = "sparseQR", y = "ddenseMatrix"): ...
- `qr.qy` signature(qr = "sparseQR", y = "matrix"): ...
- `qr.resid` signature(qr = "sparseQR", y = "ddenseMatrix"): ...
- `qr.resid` signature(qr = "sparseQR", y = "matrix"): ...
- `solve` signature(a = "sparseQR", b = "ANY"): For `solve(a,b)`, simply uses `qr.coef(a,b)`.

See Also

- `qr`, `qr.Q`, `qr.R`, `qr.fitted`, `qr.resid`, `qr.coef`, `qr.qty`, `qr.qy`,

Permutation matrices in the Matrix package: `pMatrix`; `dgCMatrix`, `dgeMatrix`.
sparseVector

Examples

```r
data(KNex)
mm <- KNex$mm
y <- KNex$y
y. <- as(y, "CsparseMatrix")
str(qrm <- qr(mm))
qc <- qr.coef(qrm, y); qc. <- qr.coef(qrm, y.) # 2nd failed in Matrix <= 1.1-0
qf <- qr.fitted(qrm, y); qf. <- qr.fitted(qrm, y.)
qs <- qr.resid(qrm, y); qs. <- qr.resid(qrm, y.)
stopifnot(all.equal(qc, as.numeric(qc.), tolerance=1e-12),
          all.equal(qf, as.numeric(qf.), tolerance=1e-12),
          all.equal(qs, as.numeric(qs.), tolerance=1e-12),
          all.equal(qf+qs, y, tolerance=1e-12))
```

sparseVector

Sparse Vector Construction from Nonzero Entries

Description

User friendly construction of sparse vectors, i.e., objects inheriting from class `sparseVector`, from indices and values of its non-zero entries.

Usage

```r
sparseVector(x, i, length)
```

Arguments

- **x**: vector of the non zero entries; may be missing in which case a "nsparseVector" will be returned.
- **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,-3:1,0,0,2,0,5,0,0)
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)))

(ns <- sparseVector(i= c(7, 3, 2), length = 10))
stopifnot(identical(ns,
    new("nsparseVector", length = 10, i = c(2, 3, 7))))
```

---

**sparseVector-class**  
**Sparse Vector Classes**

Description

Sparse Vector Classes: The virtual mother class "sparseVector" has the five actual daughter classes "dsparseVector", "isparseVector", "lsparseVector", "nsparseVector", and "zsparseVector", where we've mainly implemented methods for the d*, l* and n* 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 `setClassUnion`) 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.
toeplitz signature(x = "sparseVector"): as toeplitz(x), produce the n x n Toeplitz matrix from x, where n = length(x).
rep signature(x = "sparseVector") repeat x, with the same argument list (x, times, length.out, each,...) as the default method for rep().
which signature(x = "nsparseVector") and
which signature(x = "lsparseVector") return the indices of the non-zero entries (which is trivial for sparse vectors).
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"): return logical or "nsparseVector" of the same length as x, indicating if/where x is NA (or NaN), finite or infinite, entirely analogously to the corresponding base R functions.
c.sparseVector() is an S3 method for all "sparseVector"s, but automatic dispatch only happens for the first argument, so it is useful also as regular R function, see the examples.

See Also
sparseVector() for friendly construction of sparse vectors (apart from as(*, "sparseVector")).

Examples

getClass("sparseVector")
getClass("dsparseVector")
getClass("xsparseVector")# those with an 'x' slot

sx <- c(0,0,3, 3.2, 0,0,0,-3:1,0,0,2,0,0,5,0,0)
(ss <- as(sx, "sparseVector"))
ix <- as.integer(round(sx))
(is <- as(ix, "sparseVector")) ## an "isparseVector" (!)
(ns <- sparseVector(i = c(7, 3, 2), length = 10)) # "nsparseVector"
## rep() works too:
(ri <- rep(is, length.out = 25))

## Using `dim<-` as in base R:
(r <- ss)
dim(r) <- c(4, 5) # becomes a sparse Matrix:
(r
## or coercion (as as.matrix() in base R):
as(ss, "Matrix")
stopifnot(all(ss == print(as(ss, "CsparseMatrix")))

## 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(sN, "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(v@i, 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(9, 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))))

## c() method for "sparseVector" - also available as regular function
(c1 <- c(x20, 0, 0, 0, -10*x20))
(c2 <- c(ns, is, FALSE))
(c3 <- c(ns, !ns, TRUE, NA, FALSE))
(c4 <- c(ns, rev(ns)))
## here, c() would produce a list (not dispatching to c.sparseVector())
(c5 <- c.sparseVector(0, 0, x20))

## checking (consistency)
.v <- as.vector
.s <- function(v) as(v, "sparseVector")
stopifnot(
  all.equal(c1, .s(c(.v(x20), 0, 0, 0, -10*.v(x20))), tol=0),
  all.equal(c2, .s(c(.v(ns), .v(is), FALSE)), tol=0),
  all.equal(c3, .s(c(.v(ns), !.v(ns), TRUE, NA, FALSE)), tol=0),
  all.equal(c4, .s(c(.v(ns), rev(.v(ns))))),
  all.equal(c5, .s(c(0, 0, .v(x20))), tol=0)
)

spMatrix

Sparse Matrix Constructor From Triplet

Description

User friendly construction of a sparse matrix (inheriting from class TsparseMatrix) from the triplet representation.

This is much less flexible than spMatrix() and hence somewhat deprecated.

Usage

spMatrix(nrow, ncol, i = integer(), j = integer(), x = double())

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.

Value

A sparse matrix in triplet form, as an R object inheriting from both TsparseMatrix and generalMatrix.

The matrix $M$ will have $M[i[k], j[k]] = x[k]$, for $k = 1, 2, ..., n$, where $n = \text{length}(i)$ and $M[i', j'] = 0$ for all other pairs $(i', j')$. 
See Also

Matrix(*, sparse=TRUE) for the more usual constructor of such matrices. Then, sparseMatrix is more general and flexible than spMatrix() and by default returns a CsparseMatrix which is often slightly more desirable. Further, bdiag and Diagonal for (block-)diagonal matrix constructors.

Consider TsparseMatrix and similar class definition help files.

Examples

## 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"

## A simplified predecessor of Matrix' rsparsematrix() function:

rSpMatrix <- function(nrow, ncol, nnz,
                      rand.x = function(n) round(rnorm(nnz), 2))
{
  ## Purpose: random sparse matrix
  ## --------------------------------------------------------------
  ## Arguments: (nrow,ncol): dimension
  ## nnz : number of non-zero entries
  ## 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)
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 with the “actual” subclasses.

The main use is in methods (and C functions) that can deal with all symmetric matrices, and in as(*, "symmetricMatrix").

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. See below, about storing only one of the two Dimnames components.
- factors: a list of matrix factorizations, also from the Matrix class.

Extends

Class "Matrix", directly.

Methods

- coerce signature(from = "ddiMatrix", to = "symmetricMatrix"): and many other coercion methods, some of which are particularly optimized.
- dimnames signature(object = "symmetricMatrix"): returns symmetric dimnames, even when the Dimnames slot only has row or column names. This allows to save storage for large (typically sparse) symmetric matrices.
- isSymmetric signature(object = "symmetricMatrix"): returns TRUE trivially. There’s a C function symmetricMatrix_validate() called by the internal validity checking functions, and also from getValidity(getClass("symmetricMatrix")).

Validity and dimnames

The validity checks do not require a symmetric Dimnames slot, so it can be list(NULL, <character>), e.g., for efficiency. However, dimnames() and other functions and methods should behave as if the dimnames were symmetric, i.e., with both list components identical.

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

```r
## An example about the symmetric Dimnames:
sy <- sparseMatrix(i= c(2,4,3:5), j= c(4,7:5,5), x = 1:5, dims = c(7,7),
                    symmetric=TRUE, dimnames = list(NULL, letters[1:7]))
sy # shows symmetrical dimnames
```
symmpart  

Symmetric Part and Skew(symmetric) Part of a Matrix

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 \(x == \text{symmpart}(x) + \text{skewpart}(x)\) for all square matrices – apart from extraneous NA values in the RHS.

Usage

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).
triangularMatrix-class

See Also

isSymmetric.

Examples

```r
m <- Matrix(1:4, 2,2)
sympart(m)
skewpart(m)

stopifnot(all(m == symmpart(m) + skewpart(m)))

dn <- dimnames(m) <- list(row = c("r1", "r2"), col = c("var.1", "var.2"))
sympart(m)
skewpart(m)

stopifnot(all(m == symmpart(m) + skewpart(m)))

colnames(m) <- NULL

stopifnot(all(m == symmpart(m) + skewpart(m)))

dimnames(m) <- unname(dn)

stopifnot(all(m == symmpart(m) + skewpart(m)))

## investigate the current methods:
showMethods(skewpart, include = TRUE)
```

triangularMatrix-class

Virtual Class of Triangular Matrices in Package Matrix

Description

The virtual class of triangular matrices, "triangularMatrix", the package Matrix contains square (nrow == ncol) numeric and logical, dense and sparse matrices, e.g., see the examples. A main use of the virtual class is 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, such that 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.
TsparseMatrix-class

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".

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")

\---

TsparseMatrix-class  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".
uniqTsparse

Methods

Extraction ("[") methods, see \([-\text{methods}\].

Note

Most operations with sparse matrices are performed using the compressed, column-oriented or \(\text{CsparseMatrix}\) 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 \(\text{CsparseMatrix}\) for further operations.

Note that all \(\text{new(.)}\), \(\text{spMatrix}\) and \(\text{sparsesMatrix(x, repr="T")}\) constructors for "TsparseMatrix" classes implicitly add (i.e., “sum up”) \(x_k\)'s that belong to identical \((i_k, j_k)\) pairs, see, the example below, or also "\text{dgTMatrix}\".

For convenience, methods for some operations such as \(\%\%\%\text{and crossprod}\) are defined for \(\text{TsparseMatrix}\) objects. These methods simply coerce the \(\text{TsparseMatrix}\) object to a \(\text{CsparseMatrix}\) object then perform the operation.

See Also

its superclass, \(\text{sparseMatrix}\), and the \(\text{dgTMatrix}\) class, for the links to other classes.

Examples

\[
\begin{align*}
\text{showClass("TsparseMatrix")}
\#\# \text{or just the subclasses' names}
\text{names(getClass("TsparseMatrix")@subclasses)}
\end{align*}
\]

\[
\begin{align*}
\text{T3} & \gets \text{spMatrix}(3,4, i=\text{c}(1,3:1), j=\text{c}(2,4:2), x=1:4) \\
\text{T3} & \# \text{only 3 non-zero entries, 5 = 1+4 !}
\end{align*}
\]

uniqTsparse  \hspace{1cm} \text{Unique (Sorted) TsparseMatrix Representations}

Description

Detect or “unify” (or “standardize”) non-unique \(\text{TsparseMatrix}\) matrices, producing unique \((i, j, x)\) triplets which are \textit{sorted}, first in \(j\), then in \(i\) (in the sense of \(\text{order}(j,i)\)).

Note that \(\text{new(.)}, \text{spMatrix}\) or \(\text{sparsesMatrix}\) constructors for "\text{dgTMatrix}\" (and other "\text{TsparseMatrix}\" classes) implicitly add \(x_k\)'s that belong to identical \((i_k, j_k)\) pairs.

\(\text{anyDuplicatedT()}\) reports the index of the first duplicated pair, or 0 if there is none.

\(\text{uniqTsparse(x)}\) replaces duplicated index pairs \((i, j)\) and their corresponding \(x\) slot entries by the triple \((i, j, sx)\) where \(sx = \text{sum}(x [\text{all pairs matching } (i, j)])\), and for logical \(x\), addition is replaced by logical \(\text{or}\).
uniqTsparse(x, class.x = c(class(x)))
anyDuplicatedT(x, di = dim(x))

Arguments

x a sparse matrix stored in triplet form, i.e., inheriting from class TsparseMatrix.
class.x optional character string specifying class(x).
di the matrix dimension of x, dim(x).

Value

uniqTsparse(x) returns a TsparseMatrix “like x”, of the same class and with the same elements,
just internally possibly changed to “unique” (i, j, x) triplets in sorted order.

anyDuplicatedT(x) returns an integer as anyDuplicated, the index of the first duplicated entry
(from the (i, j) pairs) if there is one, and 0 otherwise.

See Also

TsparseMatrix, for uniqueness, notably dgTMatrix.

Examples

example("dgTMatrix-class", echo=FALSE)
## -> 'T2' with (i,j,x) slots of length 5 each
T2u <- uniqTsparse(T2)
stopifnot(## They "are" the same (and print the same):
       all.equal(T2, T2u, tol=0),
       ## but not internally:
       anyDuplicatedT(T2) == 2,
       anyDuplicatedT(T2u) == 0,
       length(T2 @x) == 5,
       length(T2u@x) == 3)

## is 'x' a "uniq Tsparse" Matrix ? [requires x to be TsparseMatrix!]
non_uniqT <- function(x, di = dim(x))
  is.unsorted(x@j) || anyDuplicatedT(x, di)
non_uniqT(T2) # TRUE
non_uniqT(T2u) # FALSE

T3 <- T2u
T3[1, c(1,3)] <- 10; T3[2, c(1,5)] <- 20
T3u <- uniqTsparse(T3)
str(T3u) # sorted in 'j', and within j, sorted in i
stopifnot(!non_uniqT(T3u))

## Logical l.TMatrix and n.TMatrix :
(L2 <- T2 > 0)
validObject(L2u <- uniqTsparse(L2))
(N2 <- as(L2, "nMatrix"))
validObject(N2u <- uniqTsparse(N2))
stopifnot(N2u@i == L2u@i, N2@i == L2@i, N2u@j == L2u@j, N2@j == L2@j, N2@i == L2@i, N2@j == L2@j)
# now with a nasty NA [partly failed in Matrix 1.1-5]:
L.0N <- L.1N <- L2
L.0N@x[1:2] <- c(FALSE, NA)
L.1N@x[1:2] <- c(TRUE, NA)
validObject(L.0N)
validObject(L.1N)
(m.0N <- as.matrix(L.0N))
(m.1N <- as.matrix(L.1N))
stopifnot(identical(10L, which(is.na(m.0N))), !anyNA(m.1N))
symnum(m.0N)
symnum(m.1N)

unpack

Representation of Packed and Unpacked Dense Matrices

Description

pack() coerces dense symmetric and dense triangular matrices from unpacked format (storing the full matrix) to packed format (storing only one of the upper and lower triangles). unpack() performs the reverse coercion. The two formats are formalized by the virtual classes "packedMatrix" and "unpackedMatrix".

Usage

pack(x, ...)
## S4 method for signature 'dgeMatrix'
pack(x, symmetric = NA, upperTri = NA, ...)
## S4 method for signature 'lgeMatrix'
pack(x, symmetric = NA, upperTri = NA, ...)
## S4 method for signature 'ngeMatrix'
pack(x, symmetric = NA, upperTri = NA, ...)
## S4 method for signature 'matrix'
pack(x, symmetric = NA, upperTri = NA, ...)

unpack(x, ...)

Arguments

x

A dense symmetric or dense triangular matrix.

For pack(): typically an "unpackedMatrix" or a standard "matrix", though "packedMatrix" are allowed and returned unchanged.

For unpack(): typically a "packedMatrix", though "unpackedMatrix" are allowed and returned unchanged.

symmetric

logical (including NA) optionally indicating whether x is symmetric (or triangular).
upperTri  (for triangular x only) logical (including NA) indicating whether x is upper (or lower) triangular.

... further arguments passed to or from other methods.

Details

pack(x) checks matrices x not inheriting from one of the virtual classes "symmetricMatrix" "triangularMatrix" for symmetry (via isSymmetric()) then for upper and lower triangularity (via isTriangular()) in order to identify a suitable coercion. Setting one or both of symmetric and upperTri to TRUE or FALSE rather than NA allows skipping of irrelevant tests for large matrices known to be symmetric or (upper or lower) triangular.

Users should not assume that pack() and unpack() are inverse operations. Specifically, y <- unpack(pack(x)) may not reproduce an "unpackedMatrix" x in the sense of identical(). See the examples.

Value

For pack(): a "packedMatrix" giving the condensed representation of x.

For unpack(): an "unpackedMatrix" giving the full storage representation of x.

Examples

showMethods("pack")
(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("unpack")
(cp4 <- chol(Hilbert(4))) # is triangular
 tp4 <- pack(cp4) # [t]riangular [p]acked
str(tp4)
(unpack(tp4))
stopifnot(identical(tp4, pack(unpack(tp4))))

z1 <- new("dsyMatrix", Dim = c(2L, 2L), x = as.double(1:4), uplo = "U")
z2 <- unpack(pack(z1))
stopifnot(!identical(z1, z2), # _not_ identical
 all(z1 == z2)) # but mathematically equal
cbind(z1@x, z2@x) # (unused!) lower triangle is "lost" in translation
unpackedMatrix-class

Virtual Class "unpackedMatrix" of Unpacked Dense Matrices

Description

Class "unpackedMatrix" is the virtual class of dense matrices in "unpacked" format, storing all m*n elements of an m-by-n matrix. It is used to define common methods for efficient subsetting, transposing, etc. of its proper subclasses: currently "[dln]geMatrix" (unpacked general), "[dln]syMatrix" (unpacked symmetric), "[dln]trMatrix" (unpacked triangular), and subclasses of these, such as "dpoMatrix", "Cholesky", and "BunchKaufman".

Slots

Dim, Dimnames: as all Matrix objects.

Extends


Methods

pack signature(x = "unpackedMatrix"): ...
unpack signature(x = "unpackedMatrix"): ...
isSymmetric signature(object = "unpackedMatrix"): ...
isTriangular signature(object = "unpackedMatrix"): ...
isDiagonal signature(object = "unpackedMatrix"): ...
t signature(x = "unpackedMatrix"): ...
diag signature(x = "unpackedMatrix"): ...
diag< signature(x = "unpackedMatrix"): ...

Author(s)

Mikael Jagan

See Also

pack and unpack; its virtual "complement" "packedMatrix"; its proper subclasses "dsyMatrix", "ltrMatrix", etc.

Examples

showClass("unpackedMatrix")
showMethods(classes = "unpackedMatrix")
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

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 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
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 dsCMatrix with 9101 non-zero entries.

Details

The data were read into 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 (in early 2008).
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))

________

wrld_1deg  World 1-degree grid contiguity matrix

Description

This matrix represents the distance-based contiguities of 15260 one-degree grid cells of land areas. The representation is as a row standardised spatial weights matrix transformed to a symmetric matrix (see Ord (1975), p. 125).

Usage

data(wrld_1deg)

Format

A $15260^2$ symmetric sparse matrix of class *dsCMatrix* with 55973 non-zero entries.
**wrld_1deg**

**Details**

The data were created into \( \mathbb{R} \) using the coordinates of a ‘SpatialPixels’ object containing approximately one-degree grid cells for land areas only (world excluding Antarctica), using package spdep’s `dnearneigh` with a cutoff distance of \( \sqrt{2} \), and row-standardised and transformed to symmetry using `nb2listw` and `similar.listw`. This spatial weights object was converted to a `dsTMatrix` using `as_dsTMatrix_listw` and then coerced (column-compressed).

**Source**

The shoreline data was read into \( \mathbb{R} \) using `Rgshhs` from the GSHHS coarse shoreline database distributed with the maptools package, omitting Antarctica. A matching approximately one-degree grid was generated using `Sobj_SpatialGrid`, and the grids on land were found using the appropriate `over` method for the `SpatialPolygons` and `SpatialGrid` objects, yielding a ‘SpatialPixels’ one containing only the grid cells with centres on land.

**References**


**Examples**

```r
data(wrld_1deg)
(n <- ncol(wrld_1deg))
IM <- .symDiagonal(n)
doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA"))
nn <- if(doExtras) 20 else 3
set.seed(1)
rho <- runif(nn, 0, 1)
system.time(MJ <- sapply(rho,
  function(x) determinant(IM - x * wrld_1deg,
    logarithm = TRUE)$modulus))
nWC <- -wrld_1deg
C1 <- Cholesky(nWC, Imult = 2)
## Note that det(<CHMfactor>) = det(L) = sqrt(det(A))
## ===> log det(A) = log( det(L)^2 ) = 2 * log det(L) :
system.time(MJ1 <- n * log(rho) +
  sapply(rho, function(x) c(2* determinant(update(C1, nWC, 1/x))$modulus))
)
stopifnot(all.equal(MJ, MJ1))
C2 <- Cholesky(nWC, super = TRUE, Imult = 2)
system.time(MJ2 <- n * log(rho) +
  sapply(rho, function(x) c(2* determinant(update(C2, nWC, 1/x))$modulus))
)
stopifnot(all.equal(MJ, MJ2),
  all.equal(MJ, MJ3),
  all.equal(MJ, MJ4))
```
Methods for "[" - Extraction or Subsetting in Package 'Matrix'

Description

Methods for ", i.e., extraction or subsetting mostly of matrices, in package Matrix.

Methods

There are more than these:

\[
x = \text{"Matrix"}, i = \text{"missing"}, j = \text{"missing"}, \text{drop} = \text{"ANY"} \quad ... \\
x = \text{"Matrix"}, i = \text{"numeric"}, j = \text{"missing"}, \text{drop} = \text{"missing"} \quad ... \\
x = \text{"Matrix"}, i = \text{"missing"}, j = \text{"numeric"}, \text{drop} = \text{"missing"} \quad ... \\
x = \text{"dsparseMatrix"}, i = \text{"missing"}, j = \text{"numeric"}, \text{drop} = \text{"logical"} \quad ... \\
x = \text{"dsparseMatrix"}, i = \text{"numeric"}, j = \text{"missing"}, \text{drop} = \text{"logical"} \quad ... \\
x = \text{"dsparseMatrix"}, i = \text{"numeric"}, j = \text{"numeric"}, \text{drop} = \text{"logical"} \quad ...
\]

See Also

\<-methods for subassignment to "Matrix" objects. Extract about the standard extraction.

Examples

\[
\text{str(m <- Matrix(round(rnorm(7*4),2), nrow = 7))}
\text{stopifnot(identical(m, m[]))}
\text{m[2, 3]} \quad \text{# simple number}
\text{m[2, 3:4]} \quad \text{# simple numeric of length 2}
\text{m[2, 3:4, drop=FALSE]} \quad \text{# sub matrix of class 'dgeMatrix'}
\text{#\# rows or columns only:}
\text{m[1,]} \quad \text{# first row, as simple numeric vector}
\text{m[,1:2]} \quad \text{# sub matrix of first two columns}
\text{showMethods("[", inherited = FALSE)}
\]

Methods for ":-assigning to Subsets for 'Matrix'

Description

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:

- \texttt{x = "Matrix", i = "missing", j = "missing", value= "ANY"} is currently a simple fallback method implementation which ensures “readable” error messages.
- \texttt{x = "Matrix", i = "ANY", j = "ANY", value= "ANY"} currently gives an error
- \texttt{x = "denseMatrix", i = "index", j = "missing", value= "numeric"} ...
- \texttt{x = "denseMatrix", i = "index", j = "index", value= "numeric"} ...
- \texttt{x = "denseMatrix", i = "missing", j = "index", value= "numeric"} ...

See Also

[-methods for subsetting "Matrix" objects; the \texttt{index} class; \texttt{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 # \texttt{<-} 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")
```

Description

For boolean or “pattern” matrices, i.e., \texttt{R} objects of class \texttt{nMatrix}, it is natural to allow matrix products using boolean instead of numerical arithmetic.

In package \texttt{Matrix}, we use the binary operator \texttt{\&\&} (aka “infix”) function for this and provide methods for all our matrices and the traditional \texttt{R} matrices (see \texttt{matrix}).
Value

A pattern matrix, i.e., inheriting from "nMatrix", or an "ldiMatrix" in case of a diagonal matrix.

Methods

We provide methods for both the “traditional” (R base) matrices and numeric vectors and conceptually all matrices and sparseVectors in package Matrix.

signature(x = "ANY", y = "ANY")
signature(x = "ANY", y = "Matrix")
signature(x = "Matrix", y = "ANY")
signature(x = "mMatrix", y = "mMatrix")
signature(x = "nMatrix", y = "nMatrix")
signature(x = "nMatrix", y = "nsparseMatrix")
signature(x = "nsparseMatrix", y = "nMatrix")
signature(x = "nsparseMatrix", y = "nsparseMatrix")
signature(x = "mMatrix", y = "sparseVector")
signature(x = "sparseVector", y = "sparseVector")

Note

These boolean arithmetic matrix products had been newly introduced for Matrix 1.2.0 (March 2015). Its implementation has still not been tested extensively.

Originally, it was left unspecified how non-structural zeros, i.e., 0’s as part of the M@x slot should be treated for numeric ("dMatrix") and logical ("lMatrix") sparse matrices. We now specify that boolean matrix products should behave as if applied to drop0(M), i.e., as if dropping such zeros from the matrix before using it.

Equivalently, for all matrices M, boolean arithmetic should work as if applied to M != 0 (or M != FALSE).

The current implementation ends up coercing both x and y to (virtual) class nsparseMatrix which may be quite inefficient for dense matrices. A future implementation may well return a matrix with different class, but the “same” content, i.e., the same matrix entries \( m_{ij} \).

See Also

\%\%\%, crossprod(), or tcrossprod(), for (regular) matrix product methods.

Examples

```r
set.seed(7)
L <- Matrix(rnorm(20) > 1, 4,5)
(N <- as(L, "nMatrix"))
L. <- L; L.[1:2,1] <- TRUE; L.@x[1:2] <- FALSE; L. # has "zeros" to drop0()
D <- Matrix(round(rnorm(30)), 5,6) # -> values in -1:1 (for this seed)
L %&% D
```
stopifnot(identical(L %% D, N %% D),
  all(L %% D == as((L %*% abs(D)) > 0, "sparseMatrix"))

## cross products, possibly with boolArith = TRUE:
crossprod(N)    # -> sparse patter'n' (TRUE/FALSE : boolean arithmetic)
crossprod(N +0) # -> numeric Matrix (with same "pattern")
stopifnot(all(crossprod(N) == t(N) %% N),
  identical(crossprod(N), crossprod(N +0, boolArith=TRUE)),
  identical(crossprod(L), crossprod(N , boolArith=FALSE)))
crossprod(D, boolArith = TRUE) # pattern: "nsCMatrix"
crossprod(L, boolArith = TRUE) # ditto
crossprod(L, boolArith = FALSE) # numeric: "dsCMatrix"
Index

!, Matrix-method (Matrix-class), 104
!, ldiMatrix-method (ldiMatrix-class), 92
!, lgeMatrix-method (lgeMatrix-class), 92
!, lspMatrix-method (lsyMatrix-class), 95
!, lsparseMatrix-method (lsparseMatrix-classes), 93
!, lsysMatrix-method (lsyMatrix-class), 95
!, ltpMatrix-method (ltrMatrix-class), 96
!, ltrMatrix-method (ltrMatrix-class), 96
!, ngeMatrix-method (ngeMatrix-class), 114
!, nspMatrix-method (nsyMatrix-class), 121
!, nspMatrix-method (nsyMatrix-class), 119
!, nMatrix-method (nsyMatrix-class), 121
!, ntpMatrix-method (ntrMatrix-class), 122
!, ntrMatrix-method (ntrMatrix-class), 122
!, sparseVector-method (sparseVector-class), 160

* Choleski
  Cholesky, 24

* IO
  externalFormats, 67

* algebra
  band, 9
  bandSparse, 10
  CHMfactor-class, 18
  chol, 21
  chol2inv-methods, 23
  Cholesky, 24
  Cholesky-class, 26
  colSums, 28
  dgCMatrix-class, 36
  dgeMatrix-class, 37
  dgRMatrix-class, 39
  dgTMatrix-class, 40
  Diagonal, 41
  dMatrix-class, 47
  dmperm, 48
  dpoMatrix-class, 50
  dsCMatrix-class, 53
  dsRMatrix-class, 56
  dtCMatrix-class, 59
  dtRMatrix-class, 62
  expand, 65
  expm, 66
  externalFormats, 67
  facmul, 68
  Hilbert, 74
  lsparseMatrix-classes, 93
  lu, 97
  LU-class, 100
  Matrix, 102
  Matrix-class, 104
  matrix-products, 106
  nearPD, 111
  nMatrix-class, 115
  norm, 118
  nsparseMatrix-classes, 119
  qr-methods, 130
  rankMatrix, 132
  rcond, 135
  Schur, 141
  sparseQR-class, 157
  unpack, 171

* arithmetic
  invPerm, 80

* arithPerm
  all.equal-methods, 8
  colSums, 28
  symmpart, 166

* array
  [-methods, 178
  [<--methods, 178
INDEX

bandSparse, 10
bdiag, 12
cBind, 16
chol, 21
Cholesky, 24
colSums, 28
Diagonal, 41
drop0, 52
externalFormats, 67
facmul, 68
forceSymmetric, 70
Hilbert, 74
KhatriRao, 87
kronecker-methods, 90
lu, 97
Matrix, 102
nearPD, 111
qr-methods, 130
rankMatrix, 132
rcond, 135
rsparsematrix, 139
sparseMatrix, 151
sparseQR-class, 157
sparseVector, 159
spMatrix, 163
symmpart, 166
unpack, 171

* attribute
nnzero, 116

* boolean matrix products
%&%-methods, 179

* classes
abIndex-class, 5
abIseq, 6
atomicVector-class, 8
CMfactor-class, 18
Cholesky-class, 26
compMatrix-class, 29
cSparseMatrix-class, 32
ddenseMatrix-class, 34
ddiMatrix-class, 35
denseMatrix-class, 36
dgCMatrix-class, 36
dgeMatrix-class, 37
dgRMatrix-class, 39
dgTMatrix-class, 40
diagonalMatrix-class, 43
diagU2N, 44
dMatrix-class, 47
dpoMatrix-class, 50
dsymMatrix-class, 53
dsparseMatrix-class, 55
dsRMatrix-class, 56
dsyMatrix-class, 57
dtCMatrix-class, 59
dtpMatrix-class, 61
dtrRMatrix-class, 62
dtrMatrix-class, 63
generalMatrix-class, 72
index-class, 77
indMatrix-class, 78
ldenseMatrix-class, 91
ldiMatrix-class, 92
lgeMatrix-class, 92
lsparseMatrix-classes, 93
lsyMatrix-class, 95
ltreMatrix-class, 96
LU-class, 100
mat2triplet, 101
Matrix-class, 104
MatrixClass, 108
MatrixFactorization-class, 109
ndenseMatrix-class, 110
ngeMatrix-class, 114
nMatrix-class, 115
nsparseMatrix-classes, 119
nsyMatrix-class, 121
ntMatrix-class, 122
number-class, 123
packedMatrix-class, 123
pMatrix-class, 125
replValue-class, 138
rleDiff-class, 138
RsparseMatrix-class, 140
Schur-class, 143
sparseLU-class, 149
sparseMatrix-class, 155
sparseQR-class, 157
sparseVector-class, 160
symmetricMatrix-class, 164
triangularMatrix-class, 167
TsparseMatrix-class, 168
uniqTsparse, 169
unpackedMatrix-class, 173
Unused-classes, 174

* datasets
<table>
<thead>
<tr>
<th>Package</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAex</td>
<td>15</td>
</tr>
<tr>
<td>KNex</td>
<td>89</td>
</tr>
<tr>
<td>USCounties</td>
<td>175</td>
</tr>
<tr>
<td>wrld_1deg</td>
<td>176</td>
</tr>
<tr>
<td>*distribution</td>
<td>rsparsematrix, 139</td>
</tr>
<tr>
<td>*graph</td>
<td>graph-sparseMatrix, 73</td>
</tr>
<tr>
<td>*hplot</td>
<td>image-methods, 75</td>
</tr>
<tr>
<td>*manip</td>
<td>abIseq, 6, cBind, 16, mat2triplet, 101, rep2abI, 137</td>
</tr>
<tr>
<td>*math</td>
<td>expm, 66</td>
</tr>
<tr>
<td>*methods</td>
<td>[-methods, 178, [&lt;-methods, 178, &amp;%&amp;%-methods, 179, all-methods, 7, all.equal-methods, 8, band, 9, BunchKaufman-methods, 14, chol2inv-methods, 23, image-methods, 75, is.na-methods, 81, isSymmetric-methods, 84, isTriangular, 86, KhatriRao, 87, khroncker-methods, 90, matrix-products, 106, qr-methods, 130, solve-methods, 144, SparseM-conversions, 151, updown, 174</td>
</tr>
<tr>
<td>*models</td>
<td>sparse.model.matrix, 147</td>
</tr>
<tr>
<td>*print</td>
<td>formatSparseM, 71, printSpMatrix, 127</td>
</tr>
<tr>
<td>Operation</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td><code>%*%</code></td>
<td>General matrix product</td>
</tr>
<tr>
<td><code>&lt;-</code></td>
<td>Method assignment</td>
</tr>
<tr>
<td>ANY, Matrix, Matrix-method</td>
<td></td>
</tr>
<tr>
<td>ANY, TsparseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>CsparseMatrix, CsparseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>CsparseMatrix, ddenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>CsparseMatrix, diagonalMatrix-method</td>
<td></td>
</tr>
<tr>
<td>CsparseMatrix, matrix-method</td>
<td></td>
</tr>
<tr>
<td>CsparseMatrix, numLike-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, ANY-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, TsparseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, indMatrix-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, matrix-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, numLike-method</td>
<td></td>
</tr>
<tr>
<td>Matrix, pMatrix-method</td>
<td></td>
</tr>
<tr>
<td>RsparseMatrix, diagonalMatrix-method</td>
<td></td>
</tr>
<tr>
<td>RsparseMatrix, mMatrix-method</td>
<td></td>
</tr>
<tr>
<td>TsparseMatrix, ANY-method</td>
<td></td>
</tr>
<tr>
<td>TsparseMatrix, Matrix-method</td>
<td></td>
</tr>
<tr>
<td>TsparseMatrix, TsparseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dMatrix, lMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dMatrix, mMatrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, CsparseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, ddenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, dtrMatrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, ldenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, matrix-method</td>
<td></td>
</tr>
<tr>
<td>ddenseMatrix, ndenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>denseMatrix, diagonalMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dgeMatrix, dgeMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dgeMatrix, dtpMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dgeMatrix, matrix-method</td>
<td></td>
</tr>
<tr>
<td>dspMatrix, ddenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dspMatrix, matrix-method</td>
<td></td>
</tr>
<tr>
<td>dsyMatrix, ddenseMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dsyMatrix, dtrMatrix-method</td>
<td></td>
</tr>
<tr>
<td>dtrMatrix, dtrMatrix-method</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Index of matrix-product methods:

- **%*%,dtrMatrix,matrix-method** (matrix-products), 106
- **%*%,indMatrix,Matrix-method** (matrix-products), 106
- **%*%,indMatrix,indMatrix-method** (matrix-products), 106
- **%*%,indMatrix,matrix-method** (matrix-products), 106
- **%*%,indMatrix,pMatrix-method** (matrix-products), 106
- **%*%,lMatrix,dMatrix-method** (matrix-products), 106
- **%*%,lMatrix,lMatrix-method** (matrix-products), 106
- **%*%,lMatrix,nMatrix-method** (matrix-products), 106
- **%*%,ldenseMatrix,ddenseMatrix-method** (matrix-products), 106
- **%*%,ldenseMatrix,ldenseMatrix-method** (matrix-products), 106
- **%*%,ldenseMatrix,lsparseMatrix-method** (matrix-products), 106
- **%*%,ldenseMatrix,matrix-method** (matrix-products), 106
- **%*%,ldenseMatrix,ndenseMatrix-method** (matrix-products), 106
- **%*%,lsparseMatrix,ldenseMatrix-method** (matrix-products), 106
- **%*%,lsparseMatrix,lsparseMatrix-method** (matrix-products), 106
- **%*%,mMatrix,RsparseMatrix-method** (matrix-products), 106
- **%*%,mMatrix,sparseVector-method** (matrix-products), 106
- **%*%,matrix,CsparseMatrix-method** (matrix-products), 106
- **%*%,matrix,Matrix-method** (matrix-products), 106
- **%*%,matrix,ddenseMatrix-method** (matrix-products), 106
- **%*%,matrix,dgeMatrix-method** (matrix-products), 106
- **%*%,matrix,diagonalMatrix-method** (matrix-products), 106
- **%*%,matrix,dsyMatrix-method** (matrix-products), 106
- **%*%,matrix,dtpMatrix-method** (matrix-products), 106
- **%*%,matrix,drtrMatrix-method** (matrix-products), 106
- **%*%,matrix,indMatrix-method** (matrix-products), 106
- **%*%,matrix,ldenseMatrix-method** (matrix-products), 106
- **%*%,matrix,ndenseMatrix-method** (matrix-products), 106
- **%*%,matrix,pMatrix-method** (matrix-products), 106
- **%*%,matrix,sparseMatrix-method** (matrix-products), 106
- **%*%,nMatrix,dMatrix-method** (matrix-products), 106
- **%*%,nMatrix,lMatrix-method** (matrix-products), 106
- **%*%,nMatrix,nMatrix-method** (matrix-products), 106
- **%*%,ndenseMatrix,ddenseMatrix-method** (matrix-products), 106
- **%*%,ndenseMatrix,ldenseMatrix-method** (matrix-products), 106
- **%*%,ndenseMatrix,matrix-method** (matrix-products), 106
- **%*%,ndenseMatrix,ndenseMatrix-method** (matrix-products), 106
- **%*%,ndenseMatrix,nsparseMatrix-method** (matrix-products), 106
- **%*%,nsparseMatrix,ndenseMatrix-method** (matrix-products), 106
- **%*%,nsparseMatrix,nsparseMatrix-method** (matrix-products), 106
- **%*%,numLike,CsparseMatrix-method** (matrix-products), 106
- **%*%,numLike,Matrix-method** (matrix-products), 106
- **%*%,numLike,sparseVector-method** (matrix-products), 106
- **%*%,pMatrix,pMatrix-method** (matrix-products), 106
- **%*%,sparseMatrix,matrix-method** (matrix-products), 106
- **%*%,sparseVector,mMatrix-method** (matrix-products), 106
- **%*%,sparseVector,numLike-method** (matrix-products), 106
- **%*%,sparseVector,sparseVector-method** (matrix-products), 106
%*%-methods (matrix-products), 106
%/, ddiMatrix, Matrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ddenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ldenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ndenseMatrix-method
(ddiMatrix-class), 35
%/, ldiMatrix, Matrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ddenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ldenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ndenseMatrix-method
(ldiMatrix-class), 92
%/, ddiMatrix, Matrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ddenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ldenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ndenseMatrix-method
(ddiMatrix-class), 35
%/, ldiMatrix, Matrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ddenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ldenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ndenseMatrix-method
(ldiMatrix-class), 92
%/, ddiMatrix, Matrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ddenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ldenseMatrix-method
(ddiMatrix-class), 35
%/, ddiMatrix, ndenseMatrix-method
(ddiMatrix-class), 35
%/, ldiMatrix, Matrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ddenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ldenseMatrix-method
(ldiMatrix-class), 92
%/, ldiMatrix, ndenseMatrix-method
(ldiMatrix-class), 92
%/, (%%/)-methods), 179
%/, ANY, ANY-method (%%/)-methods), 179
%/, ANY, Matrix-method (%%/)-methods), 179
%/, ANY, matrix-method (%%/)-methods), 179
%/, CsparseMatrix, RsparseMatrix-method
(%%/)-methods), 179
%/, CsparseMatrix, TsparseMatrix-method
(%%/)-methods), 179
%/, CsparseMatrix, diagonalMatrix-method
(%%/)-methods), 179
%/, CsparseMatrix, mMatrix-method
(%%/)-methods), 179
%/, Matrix, ANY-method (%%/)-methods), 179
%/, Matrix, Matrix-method (%%/)-methods), 179
%/, Matrix, indMatrix-method
(%%/)-methods), 179
%/, Matrix, pMatrix-method
(%%/)-methods), 179
%/, RsparseMatrix, CsparseMatrix-method
(%%/)-methods), 179
%/, RsparseMatrix, RsparseMatrix-method
(%%/)-methods), 179
%/, RsparseMatrix, TsparseMatrix-method
(%%/)-methods), 179
%/, TsparseMatrix, CsparseMatrix-method
(%%/)-methods), 179
%/, TsparseMatrix, RsparseMatrix-method
(%%/)-methods), 179
%/, TsparseMatrix, TsparseMatrix-method
(%%/)-methods), 179
%/, denseMatrix, denseMatrix-method
(%%/)-methods), 179
%/, denseMatrix, diagonalMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, CsparseMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, RsparseMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, TsparseMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, denseMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, diagonalMatrix-method
(%%/)-methods), 179
%/, diagonalMatrix, matrix-method
(%%/)-methods), 179
%/, indMatrix, Matrix-method
(%%/)-methods), 179
%/, indMatrix, indMatrix-method
(%%/)-methods), 179
%/, indMatrix, matrix-method
(%%/)-methods), 179
%/, indMatrix, pMatrix-method
(%%/)-methods), 179
INDEX

%&%, mMatrix, CsparseMatrix-method
  (%&%-methods), 179
%&%, mMatrix, RsparseMatrix-method
  (%&%-methods), 179
%&%, mMatrix, TsparseMatrix-method
  (%&%-methods), 179
%&%, mMatrix, sparseMatrix-method
  (%&%-methods), 179
%&%, mMatrix, sparseVector-method
  (%&%-methods), 179
%&%, matrix, ANY-method (%&%-methods), 179
%&%, matrix, diagonalMatrix-method
  (%&%-methods), 179
%&%, matrix, indMatrix-method
  (%&%-methods), 179
%&%, matrix, matrix-method (%&%-methods), 179
%&%, matrix, pMatrix-method
  (%&%-methods), 179
%&%, nCsparsesMatrix, nCsparsesMatrix-method
  (%&%-methods), 179
%&%, nCsparsesMatrix, nsparseMatrix-method
  (%&%-methods), 179
%&%, nMatrix, nMatrix-method
  (%&%-methods), 179
%&%, nMatrix, nsparseMatrix-method
  (%&%-methods), 179
%&%, nsparseMatrix, nCsparsesMatrix-method
  (%&%-methods), 179
%&%, nsparseMatrix, nMatrix-method
  (%&%-methods), 179
%&%, nsparseMatrix, nsparseMatrix-method
  (%&%-methods), 179
%&%, numLike, sparseVector-method
  (%&%-methods), 179
%&%, sparseMatrix, mMatrix-method
  (%&%-methods), 179
%&%, sparseMatrix, sparseMatrix-method
  (%&%-methods), 179
%&%, sparseVector, mMatrix-method
  (%&%-methods), 179
%&%, sparseVector, numLike-method
  (%&%-methods), 179
%&%, sparseVector, sparseVector-method
  (%&%-methods), 179
&%, Matrix, ddiMatrix-method
  (Matrix-class), 104
&%, dddenseMatrix, dddiMatrix-method
  (dddenseMatrix-class), 34
&%, dddenseMatrix, ldiMatrix-method
  (dddenseMatrix-class), 34
&%, ddiMatrix, Matrix-method
  (ddiMatrix-class), 35
&%, ddiMatrix, dddenseMatrix-method
  (ddiMatrix-class), 35
&%, ddiMatrix, ldaenseMatrix-method
  (ddiMatrix-class), 35
&%, ldaenseMatrix, ddiMatrix-method
  (ldaenseMatrix-class), 91
&%, ldaenseMatrix, ldiMatrix-method
  (ldaenseMatrix-class), 91
&%, ldiMatrix, Matrix-method
  (ldiMatrix-class), 92
&%, ldiMatrix, dddenseMatrix-method
  (ldiMatrix-class), 92
&%, ldiMatrix, ldaenseMatrix-method
  (ldiMatrix-class), 92
&%, ldaenseMatrix, ddiMatrix-method
  (ldaenseMatrix-class), 110
&%, ldaenseMatrix, ldiMatrix-method
  (ldaenseMatrix-class), 110
%&%, 33, 38, 108, 180
%&%, 106–108
%&%-methods, 179
^, Matrix, ddiMatrix-method
  (Matrix-class), 104
^, Matrix, ldiMatrix-method
  (Matrix-class), 104
^, dddenseMatrix, ddiMatrix-method
  (dddenseMatrix-class), 34
^, dddenseMatrix, ldiMatrix-method
  (dddenseMatrix-class), 34
^, ldaenseMatrix, ddiMatrix-method
  (ldaenseMatrix-class), 91
^, ldaenseMatrix, ldiMatrix-method
  (ldaenseMatrix-class), 91
^, ndenseMatrix, ddiMatrix-method
  (ndenseMatrix-class), 110
^, ndenseMatrix, ldiMatrix-method
  (ndenseMatrix-class), 110
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbreviate</td>
<td>128</td>
</tr>
<tr>
<td>abIndex</td>
<td>6, 7, 137, 139</td>
</tr>
<tr>
<td>abIndex-class</td>
<td>5</td>
</tr>
<tr>
<td>abIseq</td>
<td>5, 6, 137</td>
</tr>
<tr>
<td>abIseq(1) (abIseq)</td>
<td>6</td>
</tr>
<tr>
<td>abs</td>
<td>76</td>
</tr>
<tr>
<td>all, ddiMatrix-method (all-methods)</td>
<td>7</td>
</tr>
<tr>
<td>all, ldiMatrix-method (all-methods)</td>
<td>7</td>
</tr>
<tr>
<td>all, lMatrix-method (all-methods)</td>
<td>7</td>
</tr>
<tr>
<td>all, nsparseMatrix-method (all-methods)</td>
<td>7</td>
</tr>
<tr>
<td>any-methods</td>
<td>7</td>
</tr>
<tr>
<td>apply</td>
<td>105</td>
</tr>
<tr>
<td>Arith</td>
<td>38, 47</td>
</tr>
<tr>
<td>Arith, abIndex, abIndex-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>Arith, abIndex, numLike-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>Arith, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class)</td>
<td>32</td>
</tr>
<tr>
<td>Arith, CsparseMatrix, numeric-method (CsparseMatrix-class)</td>
<td>32</td>
</tr>
<tr>
<td>Arith, ddiMatrix, logical-method (ddenseMatrix-class)</td>
<td>34</td>
</tr>
<tr>
<td>Arith, ddiMatrix, numeric-method (ddenseMatrix-class)</td>
<td>34</td>
</tr>
<tr>
<td>Arith, ddiMatrix, sparseVector-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Arith, ddiMatrix, numeric-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Arith, dgCMatrix, dgCMatrix-method</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

(dgCMatrix-class), 36
Arith,dgCMatrix,logical-method
(dgCMatrix-class), 36
Arith,dgCMatrix,numeric-method
(dgCMatrix-class), 36
Arith,dgeMatrix,dgeMatrix-method
(dgeMatrix-class), 37
Arith,dgeMatrix,logical-method
(dgeMatrix-class), 37
Arith,dgeMatrix,numeric-method
(dgeMatrix-class), 37
Arith,dpoMatrix,logical-method
(dpoMatrix-class), 50
Arith,dpoMatrix,numeric-method
(dpoMatrix-class), 50
Arith,dppMatrix,logical-method
(dpoMatrix-class), 50
Arith,dppMatrix,numeric-method
(dpoMatrix-class), 50
Arith,dsCMatrix,dsCMatrix-method
(dsCMatrix-class), 53
Arith,dsparseMatrix,logical-method
(dsparseMatrix-class), 55
Arith,dsparseMatrix,nsparseMatrix-method
(nsparseMatrix-classes), 119
Arith,dsparseMatrix,numeric-method
(dsparseMatrix-class), 55
Arith,dsparseVector,dsparseVector-method
(sparseVector-class), 160
Arith,dtCMatrix,dtCMatrix-method
(dtCMatrix-class), 59
Arith,ldiMatrix,logical-method
(ldiMatrix-class), 92
Arith,ldiMatrix,numeric-method
(ldiMatrix-class), 92
Arith,lgCMatrix,lgCMatrix-method
(lsparseMatrix-classes), 93
Arith,lgeMatrix,lgeMatrix-method
(lsparseMatrix-classes), 93
Arith,lgTMatrix,lgTMatrix-method
(lsparseMatrix-classes), 93
Arith,lMatrix,logical-method
(dMatrix-class), 47
Arith,lMatrix,numeric-method
(dMatrix-class), 47
Arith,logical,ddenseMatrix-method
(ddenseMatrix-class), 34
Arith,logical,ddiMatrix-method
(ddiMatrix-class), 35
Arith,logical,dgCMatrix-method
(dgCMatrix-class), 36
Arith,logical,dgeMatrix-method
(dgeMatrix-class), 36
Arith,logical,dpoMatrix-method
(dpoMatrix-class), 50
Arith,logical,dppMatrix-method
(dpoMatrix-class), 50
Arith,logical,dsparseMatrix-method
(dsparseMatrix-class), 55
Arith,logical,lgeMatrix-method
(lsparseMatrix-classes), 93
Arith,logical,lgCMatrix-method
(lsparseMatrix-classes), 93
Arith,logical,lgTMatrix-method
(lsparseMatrix-classes), 93
Arith,logical,lMatrix-method
(dMatrix-class), 47
Arith,logical,lMatrix-method
(dMatrix-class), 47
Arith,logical,nMatrix-method
(nMatrix-class), 115
Arith,logical,nMatrix-method
(nMatrix-class), 115
Arith,logical,nsparseMatrix-method
(nsparseMatrix-classes), 119
Arith,logical,nsparseMatrix-method
(nsparseMatrix-classes), 119
Arith,logical,numeric-method
(nMatrix-class), 115
Arith,logical,numeric-method
(nMatrix-class), 115
Arith,nsparseMatrix,dsparseMatrix-method
(nsparseMatrix-classes), 119
Arith,nsparseMatrix,nsparseMatrix-method
(nsparseMatrix-classes), 119
Arith,numeric,CsparseMatrix-method
(CsparseMatrix-class), 32
Arith,numeric,ddenseMatrix-method
(ddenseMatrix-class), 34
Arith,numeric,ddiMatrix-method
(ddiMatrix-class), 35
Arith,numeric,dgCMatrix-method
(dgCMatrix-class), 36
Arith, numeric, dgeMatrix-method
(dgeMatrix-class), 37
Arith, numeric, dpoMatrix-method
(dpoMatrix-class), 50
Arith, numeric, dppMatrix-method
(dpoMatrix-class), 50
Arith, numeric, dsparseMatrix-method
(dsparseMatrix-class), 55
Arith, numeric, ldiMatrix-method
(ldiMatrix-class), 92
Arith, numeric, lMatrix-method
(lMatrix-class), 47
Arith, numeric, nMatrix-method
(nMatrix-class), 115
Arith, numLike, abIndex-method
(abIndex-class), 5
Arith, sparseVector, ddenseMatrix-method
(sparseVector-class), 160
Arith, sparseVector, dgeMatrix-method
(sparseVector-class), 160
Arith, sparseVector, sparseVector-method
(sparseVector-class), 160
Arith, triangularMatrix, diagonalMatrix-method
(triangularMatrix-class), 167
as, 73, 93, 96, 97, 115, 121, 122, 151
as.array, 105
as.array, Matrix-method (Matrix-class), 104
as.integer, abIndex-method
(abIndex-class), 5
as.logical, denseMatrix-method
(denseMatrix-class), 36
as.logical, diagonalMatrix-method
(diagonalMatrix-class), 43
as.logical, dsparseMatrix-method
(d sparseMatrix-class), 55
as.logical, indMatrix-method
(indMatrix-class), 78
as.logical, ldiMatrix-method
(ldiMatrix-class), 92
as.logical, lsparseMatrix-method
(lsparseMatrix-classes), 93
as.logical, Matrix-method
(Matrix-class), 104
as.logical, ndenseMatrix-method
(ndenseMatrix-class), 110
as.logical, nspars eMatrix-method
(nspars eMatrix-classes), 119
as.logical, sparseVector-method
(sparseVector-class), 160
as.matrix, 105
as.matrix, Matrix-method (Matrix-class), 104
as.numeric, abIndex-method
(abIndex-class), 5
as.numeric, ddiMatrix-method
(ddiMatrix-class), 35
as.numeric, denseMatrix-method
(denseMatrix-class), 36
as.numeric, diagonalMatrix-method
(diagonalMatrix-class), 43
as.numeric, dsparseMatrix-method
(d sparseMatrix-class), 55
as.numeric, indMatrix-method
(indMatrix-class), 78
as.numeric, lsparseMatrix-method
(lsparseMatrix-classes), 93
as.numeric, Matrix-method
(Matrix-class), 104
as.numeric, ndenseMatrix-method
(ndenseMatrix-class), 110
as.numeric, nspars eMatrix-method
(nspars eMatrix-classes), 119
as.numeric, sparseVector-method
(sparseVector-class), 160
as.vector, 8
as.vector, abIndex-method
(abIndex-class), 5
as.vector, CsparseMatrix-method
(CsparseMatrix-class), 32
as.vector, denseMatrix-method
(denseMatrix-class), 36
as.vector, dgeMatrix-method
(dgeMatrix-class), 37
as.vector, diagonalMatrix-method
(diagonalMatrix-class), 43
as.vector, indMatrix-method
(indMatrix-class), 78
as.vector, ldiMatrix-method
(ldiMatrix-class), 92
as.vector, lsparseMatrix-method
(lsparseMatrix-classes), 93
as.vector, Matrix-method
(Matrix-class), 104
as.vector, ndenseMatrix-method
(ndenseMatrix-class), 110
as.vector, ngeMatrix-method
INDEX

(ngeMatrix-class), 114
as.vector, RsparseMatrix-method (RsparseMatrix-class), 140
as.vector, sparseVector-method (sparseVector-class), 160
as.vector, TsparseMatrix-method (TsparseMatrix-class), 168
as_dsTMatrix_listw, 175, 177
atomicVector-class, 8
attribute, 85, 86
backsolve, 146
band, 9, 11, 42
band, CsparseMatrix-method (band), 9
band, denseMatrix-method (band), 9
band, diagonalMatrix-method (band), 9
band, indMatrix-method (band), 9
band, matrix-method (band), 9
band, RsparseMatrix-method (band), 9
band, TsparseMatrix-method (band), 9
band-methods (band), 9
bandSparse, 10, 10, 13, 42, 103, 153
bdiag, 11, 12, 103, 153, 164
BunchKaufman, 14, 15, 26, 30, 51, 173
BunchKaufman (BunchKaufman-methods), 14
BunchKaufman, dspMatrix-method (BunchKaufman-methods), 14
BunchKaufman, dsysMatrix-method (BunchKaufman-methods), 14
BunchKaufman, matrix-method (BunchKaufman-methods), 14
BunchKaufman-class (Cholesky-class), 26
BunchKaufman-methods, 14
c, 5
c.asIndex (abIseq), 6
c.sparseVector (sparseVector-class), 160
CAex, 15
cBind, 16
cbind, 16, 17, 156
cbind2, 16, 17
cbind2, ANY, matrix-method (Matrix-class), 104
cbind2, atomicVector, ddiMatrix-method (atomicVector-class), 8
cbind2, atomicVector, ldiMatrix-method (atomicVector-class), 8
cbind2, atomicVector, matrix-method (atomicVector-class), 8
cbind2, ddiMatrix, atomicVector-method (ddiMatrix-class), 35
cbind2, ddiMatrix, matrix-method (ddiMatrix-class), 35
cbind2, denseMatrix, denseMatrix-method (denseMatrix-class), 36
cbind2, denseMatrix, matrix-method (denseMatrix-class), 36
cbind2, denseMatrix, numeric-method (denseMatrix-class), 36
cbind2, denseMatrix, sparseMatrix-method (cBind), 16
cbind2, diagonalMatrix, sparseMatrix-method (diagonalMatrix-class), 43
cbind2, ldiMatrix, atomicVector-method (ldiMatrix-class), 92
cbind2, ldiMatrix, matrix-method (ldiMatrix-class), 92
cbind2, Matrix, ANY-method (Matrix-class), 104
cbind2, Matrix, atomicVector-method (Matrix-class), 104
cbind2, matrix, ddiMatrix-method (ddiMatrix-class), 35
cbind2, matrix, denseMatrix-method (denseMatrix-class), 36
cbind2, matrix, ldiMatrix-method (ldiMatrix-class), 92
cbind2, Matrix, matrix-method (Matrix-class), 104
cbind2, Matrix, missing-method (Matrix-class), 104
cbind2, Matrix, NULL-method (Matrix-class), 104
cbind2, matrix, sparseMatrix-method (sparseMatrix-class), 155
cbind2, NULL, Matrix-method (Matrix-class), 104
cbind2, numeric, denseMatrix-method (denseMatrix-class), 36
cbind2, sparseMatrix, denseMatrix-method (cBind), 16
cbind2, sparseMatrix, diagonalMatrix-method (sparseMatrix-class), 155
cbind2, sparseMatrix, matrix-method (sparseMatrix-class), 155
cbind2, sparseMatrix, sparseMatrix-method (sparseMatrix-class), 155

CHMfactor, 24, 25, 27, 110, 144–146, 174

CHMfactor-class, 18

CHMsimpl, 25

CHMsimpl-class (CHMfactor-class), 18

CHMsuper, 25

CHMsuper-class (CHMfactor-class), 18

chol, 15, 21, 22–27, 30, 38, 51, 54, 99

chol, dgCMatrix-method (chol), 21

chol, dgeMatrix-method (chol), 21

chol, dgRMatrix-method (chol), 21

chol, dgTMatrix-method (chol), 21

chol, diagonalMatrix-method (chol), 21

chol, dsCMatrix-method (chol), 21

chol, dspMatrix-method (chol), 21

chol, dsRMatrix-method (chol), 21

chol, dsTMatrix-method (chol), 21

chol, generalMatrix-method (chol), 21

chol, symmetricMatrix-method (chol), 21

chol, triangularMatrix-method (chol), 21

chol-methods (chol), 21

chol2inv, 23

chol2inv, ANY-method (chol2inv-methods), 23

chol2inv, CHMfactor-method
(chol2inv-methods), 23

chol2inv, denseMatrix-method
(chol2inv-methods), 23

chol2inv, diagonalMatrix-method
(chol2inv-methods), 23

chol2inv, dtrMatrix-method
(chol2inv-methods), 23

chol2inv, sparseMatrix-method
(chol2inv-methods), 23

chol2inv-methods, 23

Cholesky, 15, 18, 20–22, 24, 27, 30, 51, 54, 63, 110, 156, 173, 175

Cholesky, denseMatrix-method (Cholesky), 24

Cholesky, dgeMatrix-method (Cholesky), 24

Cholesky, dgeMatrix-method (Cholesky), 24

Cholesky, nspMatrix-method
(Cholesky), 24

Cholesky, sparseMatrix-method
(Cholesky), 24

Cholesky-class, 26

CholeskyFactorization-class

chol-methods, 21

coerce, 168

coerce, abIndex, integer-method
(abIndex-class), 5

coerce, abIndex, numeric-method
(abIndex-class), 5

coerce, abIndex, seqMat-method
(abIndex-class), 5

coerce, abIndex, vector-method
(abIndex-class), 5

coerce, ANY, denseMatrix-method
(denseMatrix-class), 36

coerce, ANY, Matrix-method
(Matrix-class), 104

coerce, ANY, nspMatrix-method
(sparseMatrix-class), 160

coerce, ANY, sparseMatrix-method
(sparseMatrix-class), 155

coerce, ANY, sparseVector-method
(sparseVector-class), 160

coerce, atomicVector, dsparseVector-method
(atomicVector-class), 8

coerce, atomicVector, sparseVector-method
(atomicVector-class), 8

coerce, CHMfactor, CsparseMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, dMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, dsparseMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, Matrix-method
(CHMfactor-class), 18

coerce, CHMfactor, pMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, RsparseMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, sparseMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, triangularMatrix-method
(CHMfactor-class), 18

coerce, CHMfactor, TsparseMatrix-method
(CHMfactor-class), 18

coerce, CsparseMatrix, denseMatrix-method
(CsparseMatrix-class), 32

coerce, CsparseMatrix, generalMatrix-method

coerce, CsparseMatrix, generalMatrix-method

coerce, CHMfactor-class

coerce, CHMfactor-class

coerce, CHMfactor-class

INDEX
coerce, CsparseMatrix, matrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, packedMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, RsparseMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, sparseVector-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, TsparseMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, unpackedMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, vector-method (CsparseMatrix-class), 32
coerce, ddenseMatrix, dgeMatrix-method (ddenseMatrix-class), 34
coerce, ddiMatrix, dgCMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dgeMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dRMATRIX-method (ddiMatrix-class), 35
coerce, ddiMatrix, dTMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dsCMATRIX-method (ddiMatrix-class), 35
coerce, ddiMatrix, dsRMATRIX-method (ddiMatrix-class), 35
coerce, ddiMatrix, dsTMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dsyMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dtCMATRIX-method (ddiMatrix-class), 35
coerce, ddiMatrix, dtRMATRIX-method (ddiMatrix-class), 35
coerce, ddiMatrix, dtTMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dtrMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, dTMatrix-method (ddiMatrix-class), 35
coerce, ddiMatrix, ldiMatrix-method (ddiMatrix-class), 35
coerce, denseMatrix, CsparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, ddenseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, dMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, dsparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, generalMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, ldenseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, lMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, lSparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, matrix-method (denseMatrix-class), 36
coerce, denseMatrix, ndenseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, nMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, nSparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, packedMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, RsparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, sparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, TsparseMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, unpackedMatrix-method (denseMatrix-class), 36
coerce, denseMatrix, vector-method (denseMatrix-class), 36
coerce, dgCMatrix, dgeMatrix-method (dgCMatrix-class), 36
coerce, dgCMatrix, dRMATRIX-method (dgCMatrix-class), 36
coerce, dgCMatrix, dTMatrix-method (dgCMatrix-class), 36
coerce, dgCMatrix, dtrMatrix-method (dgCMatrix-class), 36
coerce, dgCMatrix, dsCMATRIX-method (dgCMatrix-class), 36
coerce, dgCMatrix, dsTMatrix-method (dgCMatrix-class), 36
coerce, dgCMatrix, dtCMATRIX-method (dgCMatrix-class), 36
coerce, dgCMatrix, dtRMATRIX-method (dgCMatrix-class), 36
coerce, dgCMatrix, dtTMatrix-method (dgCMatrix-class), 36
coerce, dgCMatrix, ldgCMATRIX-method (dgCMatrix-class), 36
coerce, dgCMatrix, matrix.csc-method (dgCMatrix-class), 36
coerce, dgCMatrix, ngCMatrix-method
  (dgCMatrix-class), 36
coerce, dgeMatrix, dgCMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dgRMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dgTMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dspMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dsTMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dsyMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dtpMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, dtrMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, lgeMatrix-method
  (dgeMatrix-class), 37
coerce, dgeMatrix, vector-method
  (dgeMatrix-class), 37
coerce, dgRMatrix, dgCMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, dgTMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, dgeMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, dtpMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, dtrMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, lgeMatrix-method
  (dgRMatrix-class), 39
coerce, dgRMatrix, matrix.csr-method
  (SparseM-conversions), 151
coerce, dgRMatrix, ngRMatrix-method
  (dgRMatrix-class), 39
coerce, diagonalMatrix, CsparseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, dddenseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, denseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, lndenseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, ndenseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, packedMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, ndenseMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, nMatrix-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, sparseVector-method
  (diagonalMatrix-class), 43
coerce, diagonalMatrix, symmetricMatrix-method
  (diagonalMatrix-class), 43
(diagonalMatrix-class), 43
coerce, diagonalMatrix, triangularMatrix-method
(diagonalMatrix-class), 43
coerce, diagonalMatrix, TsparseMatrix-method
(diagonalMatrix-class), 43
coerce, diagonalMatrix, unpackedMatrix-method
(diagonalMatrix-class), 43
coerce, diagonalMatrix, vector-method
(diagonalMatrix-class), 43
coerce, dpoMatrix, corMatrix-method
(dpoMatrix-class), 50
coerce, dpoMatrix, dppMatrix-method
(dpoMatrix-class), 50
code, dppMatrix, corMatrix-method
(dpoMatrix-class), 50
code, dppMatrix, dpoMatrix-method
(dpoMatrix-class), 50
code, dsCMatrix, dgCMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dgeMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dsCMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dspMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dsRMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dsTMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dsyMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, dtCMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, lsCMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, nsCMatrix-method
(dsCMatrix-class), 53
code, dsCMatrix, RsparseMatrix-method
(dsCMatrix-class), 53
code, dsparseMatrix, lMatrix-method
(dsparseMatrix-class), 55
code, dsparseMatrix, lsparseMatrix-method
(dsparseMatrix-class), 55
code, dsparseMatrix, mMatrix-method
(dsparseMatrix-class), 55
code, dsparseMatrix, nsparseMatrix-method
(dsparseMatrix-class), 55
code, dspMatrix, dgeMatrix-method
(dspMatrix-class), 55
coerce, dsCMatrix, dsyMatrix-method (dsCMatrix-class), 53
coerce, dsCMatrix, dtMMatrix-method (dsCMatrix-class), 53
coerce, dsCMatrix, lMMatrix-method (dsCMatrix-class), 53
coerce, dsCMatrix, nMMatrix-method (dsCMatrix-class), 53
coerce, dsCMatrix, nsyMatrix-method (dsCMatrix-class), 53
coerce, dsyMatrix, corMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dgeMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dpoMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dppMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dsRMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dsTMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dtMMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dtrMatrix-method (dsyMatrix-class), 57
coerce, dsyMatrix, dtRMatrix-method (dsyMatrix-class), 57
coerce, dtCMatrix, dsCMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, dtCMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, dtpMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, dtrMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, dtRMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ltrMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ltmatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ltmatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ntcmatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 59
coerce, dtCMatrix, dtpMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, dtrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, dtrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, dtrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, dtrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ltrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ltrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ltrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
coerce, dtCMatrix, ntrMatrix-method (dtCMatrix-class), 61
INDEX

(dtRMatrix-class), 62
coerce, dtrMatrix, ntrMatrix-method
(dtRMatrix-class), 62
coerce, dtMMatrix, dgCMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dgeMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, dtMMatrix, dtTMatrix-method
(dtCMatrix-class), 59
coerce, factor, sparseMatrix-method
(sparseMatrix-class), 155
coerce, graph, CsparseMatrix-method
(graph-sparseMatrix), 73
coerce, graph, Matrix-method
(graph-sparseMatrix), 73
coerce, graph, RsparseMatrix-method
(graph-sparseMatrix), 73
coerce, graph, sparseMatrix-method
(graph-sparseMatrix), 73
coerce, graph, TsparseMatrix-method
(graph-sparseMatrix), 73
coerce, graphAM, TsparseMatrix-method
(graph-sparseMatrix), 73
coerce, graphNEL, TsparseMatrix-method
(graph-sparseMatrix), 73
coerce, indMatrix, CsparseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, dgeMatrix-method
(indMatrix-class), 78
coerce, indMatrix, denseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, diagonalMatrix-method
(indMatrix-class), 78
coerce, indMatrix, dMatrix-method
(indMatrix-class), 78
coerce, indMatrix, dsparseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, generalMatrix-method
(indMatrix-class), 78
coerce, indMatrix, ldenseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, LMatrix-method
(indMatrix-class), 78
coerce, indMatrix, lMatrix-method
(indMatrix-class), 78
coerce, indMatrix, lgeMatrix-method
(indMatrix-class), 78
coerce, indMatrix, lgRMatrix-method
(indMatrix-class), 78
coerce, indMatrix, lgTMatrix-method
(indMatrix-class), 78
coerce, indMatrix, lsparseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, matrix-method
(indMatrix-class), 78
coerce, indMatrix, ndenseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, ngeMatrix-method
(indMatrix-class), 78
coerce, indMatrix, packedMatrix-method
(indMatrix-class), 78
coerce, indMatrix, pMatrix-method
(indMatrix-class), 78
coerce, indMatrix, RsparseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, TsparseMatrix-method
(indMatrix-class), 78
coerce, indMatrix, unpackedMatrix-method
(indMatrix-class), 78
coerce, integer, indMatrix-method
(indMatrix-class), 78
coerce, integer, pMatrix-method
(pMatrix-class), 125
coerce, ldiMatrix, ddiMatrix-method
(ldiMatrix-class), 92
coerce, ldiMatrix, lgCMatrix-method
(ldiMatrix-class), 92
coerce, ldiMatrix, lgeMatrix-method
(ldiMatrix-class), 92
coerce, ldiMatrix, lgRMatrix-method
(ldiMatrix-class), 92
INDEX

(coerce,lsCMatrix,lgCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,lgTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,lspMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,lsRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,lsTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,lsyMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,ltCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,nsCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,RsparseMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,dsparseMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,nMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,nsCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,RsparseMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,dsparseMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,nMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsCMatrix,nsCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,CsparseMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,dsRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,lgRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,lsCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,lspMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,lsTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,lsyMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,ltRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsRMatrix,nsRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,dsTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lgCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lgTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lsCMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lspMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lsRMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,lsyMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,ltTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsTMatrix,nsTMatrix-method (lsparseMatrix-classes), 93)
(coerce,lsyMatrix,dsyMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,lgeMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,lsCMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,lspMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,lsRMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,ltRMatrix-method (lsyMatrix-class), 95)
(coerce,lsyMatrix,nsyMatrix-method (lsyMatrix-class), 95)
(coerce,ltCMatrix,dtCMatrix-method (ltCMatrix-class), 95)
coerce,ltCMatrix,lgCMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,lgTMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,lsCMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,ltpMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,ltCMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,ltRMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,ltTMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,lsTMatrix-method (lsparseMatrix-classes), 93
coerce,ltCMatrix,ltTMatrix-method (lsparseMatrix-classes), 93
coerce,ltRMatrix,dtRMatrix-method (lsparseMatrix-classes), 93
coerce,matrix,corMatrix-method (Matrix-class), 104
coerce,Matrix,complex-method (Matrix-class), 104
coerce,Matrix,corMatrix-method (Matrix-class), 104
coerce,matrix,corMatrix-method (dpoMatrix-class), 50
coerce,matrix,CsparseMatrix-method (Matrix-class), 104
coerce,matrix,ddenseMatrix-method (ddenseMatrix-class), 32
coerce,Matrix,denseMatrix-method (Matrix-class), 104
INDEX

(Matrix-class), 104
coerce, matrix, denseMatrix-method
   (denseMatrix-class), 36
c coerce, matrix, dgCMatrix-method
   (dgCMatrix-class), 36
c coercion, matrix, dgeMatrix-method
   (dgeMatrix-class), 37
c coercion, matrix, dgRMatrix-method
   (dgRMatrix-class), 39
c coercion, matrix, dgTMatrix-method
   (dgTMatrix-class), 40
c coercion, Matrix, diagonalMatrix-method
   (Matrix-class), 104
c coercion, matrix, diagonalMatrix-method
   (diagonalMatrix-class), 43
c coercion, matrix, dMatrix-method
   (dMatrix-class), 47
c coercion, Matrix, dpoMatrix-method
   (Matrix-class), 104
c coercion, matrix, dpoMatrix-method
   (dpoMatrix-class), 50
c coercion, Matrix, dppMatrix-method
   (Matrix-class), 104
c coercion, matrix, dppMatrix-method
   (dppMatrix-class), 50
c coercion, matrix, dsCMatrix-method
   (dsCMatrix-class), 53
c coercion, matrix, dsparseMatrix-method
   (dsparseMatrix-class), 55
c coercion, matrix, dsysMatrix-method
   (dsyMatrix-class), 57
c coercion, matrix, dsRMatrix-method
   (dsRMatrix-class), 56
c coercion, matrix, dsTMatrix-method
   (dsTMatrix-class), 53
c coercion, matrix, dsyMatrix-method
   (dsyMatrix-class), 57
c coercion, matrix, dtCMatrix-method
   (dtCMatrix-class), 59
c coercion, matrix, dtpMatrix-method
   (dtpMatrix-class), 61
c coercion, matrix, dtRMatrix-method
   (dtRMatrix-class), 62
c coercion, matrix, dtRMatrix-method
   (dtRMatrix-class), 63
c coercion, matrix, dtTMatrix-method
   (dtTMatrix-class), 59
c coercion, matrix, generalMatrix-method
   (generalMatrix-class), 72
c coercion, Matrix, graph-method
   (graph-class), 73
c coercion, Matrix, graphNEL-method
   (graphNEL-class), 73
c coercion, Matrix, indMatrix-method
   (Matrix-class), 104
c coercion, matrix, indMatrix-method
   (indMatrix-class), 78
c coercion, Matrix, integer-method
   (Matrix-class), 104
c coercion, matrix, ldenseMatrix-method
   (ldenseMatrix-class), 91
c coercion, matrix, lgeMatrix-method
   (lgeMatrix-class), 92
c coercion, matrix, lgsRMatrix-method
   (lgsRMatrix-class), 93
c coercion, matrix, lgsTMatrix-method
   (lgsTMatrix-class), 93
c coercion, matrix, lMatrix-method
   (dMatrix-class), 47
c coercion, Matrix, logical-method
   (Matrix-class), 104
c coercion, matrix, lscMatrix-method
   (lscMatrix-class), 93
c coercion, matrix, lsparseMatrix-method
   (lsparseMatrix-class), 93
c coercion, matrix, lssMatrix-method
   (lssMatrix-class), 95
c coercion, matrix, lstMatrix-method
   (lstMatrix-class), 93
c coercion, matrix, lstMatrix-method
   (lstMatrix-class), 93
c coercion, matrix, ltrMatrix-method
   (ltrMatrix-class), 96
c coercion, matrix, ltrMatrix-method
   (ltrMatrix-class), 96
c coercion, matrix, ltTMatrix-method
   (ltTMatrix-class), 93
c coercion, matrix, ltTMatrix-method
   (ltTMatrix-class), 93
c coercion, Matrix, matrix-method
INDEX

(Matrix-class), 104
coerce, matrix, Matrix-method (Matrix-class), 104
coop, Matrix, matrix.coo-method (SparseM-conversions), 151
coop, Matrix, matrix.csc-method (SparseM-conversions), 151
coop, Matrix, matrix.csr-method (SparseM-conversions), 151
coop, matrix, ndenseMatrix-method (ndenseMatrix-class), 110
coop, matrix, ngeMatrix-method (ngeMatrix-class), 114
coop, matrix, ngRMMatrix-method (nsparseMatrix-classes), 119
coop, matrix, nsyMatrix-method (nsyMatrix-class), 121
coop, matrix, nsyMatrix-method (nsyMatrix-class), 121
coop, matrix, ntCMatrix-method (nsparseMatrix-classes), 119
coop, matrix, ntRMatrix-method (nsparseMatrix-classes), 119
coop, matrix, nMatrix-method (nMatrix-class), 115
coop, matrix, packedMatrix-method (packedMatrix-class), 123
coop, Matrix, pMatrix-method (Matrix-class), 125
coop, matrix, RsparseMatrix-method (RsparseMatrix-class), 140
coop, Matrix, sparseMatrix-method (Matrix-class), 104
coop, matrix, sparseMatrix-method (sparseMatrix-class), 155
coop, Matrix, symmetricMatrix-method (Matrix-class), 104
coop, matrix, symmetricMatrix-method (symmetricMatrix-class), 164
coop, Matrix, triangularMatrix-method (Matrix-class), 104
coop, matrix, triangularMatrix-method (triangularMatrix-class), 167
coop, matrix, Tmatrix-method (TsparseMatrix-class), 168
coop, matrix, unpackedMatrix-method (unpackedMatrix-class), 173
coop, Matrix, vector-method (Matrix-class), 104
coop, matrix.coo, C sparseMatrix-method (SparseM-conversions), 151
coop, matrix.coo, dgCMatrix-method (SparseM-conversions), 151
coop, matrix.coo, dgTMatrix-method (SparseM-conversions), 151
coop, matrix.coo, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.coo, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.coo, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.csc, C sparseMatrix-method (SparseM-conversions), 151
coop, matrix.csc, dgCMatrix-method (SparseM-conversions), 151
coop, matrix.csc, Matrix-method (SparseM-conversions), 151
coop, matrix.csc, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.csc, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.csc, sparseMatrix-method (SparseM-conversions), 151
coop, matrix.csc, T sparseMatrix-method (SparseM-conversions), 151
INDEX

coerce, nsysMatrix, nspMatrix-method  
(nsysMatrix-class), 121
coerce, nsysMatrix, nsRMMatrix-method  
(nsysMatrix-class), 121
coerce, nsysMatrix, nstMatrix-method  
(nsysMatrix-class), 121
coerce, nsysMatrix, ntrMatrix-method  
(nsysMatrix-class), 121
coerce, nsysMatrix, dtCMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ltCMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ngCMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ntRMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ntrMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ntCMatrix-method  
(nsparseMatrix-classes), 119
coerce, nsysMatrix, ngeMatrix-method  
(ntrMatrix-class), 122
coerce, nsysMatrix, nspMatrix-method  
(ntrMatrix-class), 122
coerce, nsysMatrix, ngRMatrix-method  
(ntrMatrix-class), 122
coerce, nsysMatrix, nsRMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntRMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntTMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, nsysMatrix, ntrMatrix-method  
(nspMatrix-class), 119
coerce, numeric, abIndex-method  
(abIndex-class), 5
coerce, numeric, indMatrix-method
<table>
<thead>
<tr>
<th>Function Call</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>coerce, numeric, pMatrix-method (pMatrix-class)</td>
<td>125</td>
</tr>
<tr>
<td>coerce, numeric, seqMat-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>coerce, numLike, CsparseMatrix-method (CsparseMatrix-class)</td>
<td>32</td>
</tr>
<tr>
<td>coerce, numLike, ddenseMatrix-method (ddenseMatrix-class)</td>
<td>34</td>
</tr>
<tr>
<td>coerce, numLike, denseMatrix-method (denseMatrix-class)</td>
<td>36</td>
</tr>
<tr>
<td>coerce, numLike, dgCMatrix-method (dgCMatrix-class)</td>
<td>36</td>
</tr>
<tr>
<td>coerce, numLike, dgeMatrix-method (dgeMatrix-class)</td>
<td>37</td>
</tr>
<tr>
<td>coerce, numLike, dgRMatrix-method (dgRMatrix-class)</td>
<td>39</td>
</tr>
<tr>
<td>coerce, numLike, dgTMatrix-method (dgTMatrix-class)</td>
<td>40</td>
</tr>
<tr>
<td>coerce, numLike, dMatrix-method (dMatrix-class)</td>
<td>47</td>
</tr>
<tr>
<td>coerce, numLike, dsparseMatrix-method (dsparseMatrix-class)</td>
<td>55</td>
</tr>
<tr>
<td>coerce, numLike, generalMatrix-method (generalMatrix-class)</td>
<td>72</td>
</tr>
<tr>
<td>coerce, numLike, ldenseMatrix-method (ldenseMatrix-class)</td>
<td>91</td>
</tr>
<tr>
<td>coerce, numLike, lgCMatrix-method (lsparseMatrix-classes)</td>
<td>93</td>
</tr>
<tr>
<td>coerce, numLike, lgeMatrix-method (lgeMatrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>coerce, numLike, lgRMatrix-method (lsparseMatrix-classes)</td>
<td>93</td>
</tr>
<tr>
<td>coerce, numLike, lgTMatrix-method (lsparseMatrix-classes)</td>
<td>93</td>
</tr>
<tr>
<td>coerce, numLike, lMatrix-method (dMatrix-class)</td>
<td>47</td>
</tr>
<tr>
<td>coerce, numLike, lsparseMatrix-method (lsparseMatrix-classes)</td>
<td>93</td>
</tr>
<tr>
<td>coerce, numLike, ndenseMatrix-method (ndenseMatrix-class)</td>
<td>110</td>
</tr>
<tr>
<td>coerce, numLike, ngCMatrix-method (nsparseMatrix-classes)</td>
<td>119</td>
</tr>
<tr>
<td>coerce, numLike, ngeMatrix-method (ngeMatrix-class)</td>
<td>114</td>
</tr>
<tr>
<td>coerce, numLike, ngRMatrix-method (nsparseMatrix-classes)</td>
<td>119</td>
</tr>
<tr>
<td>coerce, numLike, ngTMatrix-method (nsparseMatrix-classes)</td>
<td>119</td>
</tr>
<tr>
<td>coerce, numLike, TsparseMatrix-method (TsparseMatrix-class)</td>
<td>168</td>
</tr>
<tr>
<td>coerce, numLike, unpackedMatrix-method (unpackedMatrix-class)</td>
<td>173</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, CsparseMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, denseMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, dgeMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, generalMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, matrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, packedMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, TsparseMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, unpackedMatrix-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, RsparseMatrix, vector-method (RsparseMatrix-class)</td>
<td>140</td>
</tr>
<tr>
<td>coerce, seqMat, abIndex-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>coerce, seqMat, numeric-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>coerce, sparseMatrix, indMatrix-method (sparseMatrix-class)</td>
<td>155</td>
</tr>
<tr>
<td>coerce, sparseMatrix, sparseVector-method (sparseMatrix-class)</td>
<td>155</td>
</tr>
<tr>
<td>coerce, sparseVector, CsparseMatrix-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>coerce, sparseVector, dsparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>coerce, sparseVector, integer-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>coerce, sparseVector, isparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>coerce, sparseVector, logical-method</td>
<td></td>
</tr>
</tbody>
</table>
INDEX (matrix-products), 106

crossprod,indMatrix,matrix-method (matrix-products), 106
crossprod,indMatrix,missing-method (matrix-products), 106
crossprod,ldenseMatrix,ddenseMatrix-method (matrix-products), 106
crossprod,ldenseMatrix,ldenseMatrix-method (matrix-products), 106
crossprod,ldenseMatrix,lsparseMatrix-method (matrix-products), 106
crossprod,ldenseMatrix,matrix-method (matrix-products), 106
crossprod,ldenseMatrix,missing-method (matrix-products), 106
crossprod,ldenseMatrix,ndenseMatrix-method (matrix-products), 106
crossprod,lsparseMatrix,ldenseMatrix-method (matrix-products), 106
crossprod,lsparseMatrix,lsparseMatrix-method (matrix-products), 106
crossprod,Matrix,ANY-method (matrix-products), 106
crossprod,Matrix,CsparseMatrix-method (matrix-products), 106
crossprod,Matrix,dgeMatrix-method (matrix-products), 106
crossprod,Matrix,diagonalMatrix-method (matrix-products), 106
crossprod,Matrix,dtrMatrix-method (matrix-products), 106
crossprod,Matrix,indMatrix-method (matrix-products), 106
crossprod,Matrix,indMatrix-method (matrix-products), 106
crossprod,Matrix,Matrix-method (matrix-products), 106
crossprod,Matrix,matrix-method (matrix-products), 106
crossprod,Matrix,missing-method (matrix-products), 106
crossprod,Matrix,numLike-method (matrix-products), 106
crossprod,Matrix,pMatrix-method (matrix-products), 106
crossprod,Matrix,pMatrix-method (matrix-products), 106
crossprod,MATRIX,RsparseMatrix-method (matrix-products), 106
crossprod,MATRIX,diagonalMatrix-method (matrix-products), 106
crossprod,MATRIX,mMatrix-method (matrix-products), 106
crossprod,MATRIX,sparseVector-method (matrix-products), 106
crossprod,pMatrix,indMatrix-method (matrix-products), 106
crossprod,pMatrix,Matrix-method (matrix-products), 106
crossprod,pMatrix,pMatrix-method (matrix-products), 106
crossprod,pMatrix,missing-method (matrix-products), 106
crossprod,pMatrix,RsparseMatrix,ANY-method (matrix-products), 106
crossprod,RsparseMatrix,diagonalMatrix-method (matrix-products), 106
crossprod,RsparseMatrix,mMatrix-method (matrix-products), 106
crossprod,RsparseMatrix,mMatrix-method (matrix-products), 106
crossprod,sparseVector,missing-method (matrix-products), 106
INDEX

(matrix-products), 106
crossprod, sparseVector, mMatrix-method (matrix-products), 106
crossprod, sparseVector, numLike-method (matrix-products), 106
crossprod, sparseVector, sparseVector-method (matrix-products), 106
crossprod, symmetricMatrix, pANY-method (matrix-products), 106
crossprod, symmetricMatrix, Matrix-method (matrix-products), 106
crossprod, symmetricMatrix, missing-method (matrix-products), 106
crossprod, symmetricMatrix, diagonalMatrix-method (matrix-products), 106
crossprod, TsparseMatrix, Matrix-method (matrix-products), 106
crossprod, TsparseMatrix, diagonalMatrix-method (matrix-products), 106
crossprod, TsparseMatrix, missing-method (matrix-products), 106
crossprod, TsparseMatrix, TsparseMatrix-method (matrix-products), 106
crossprod-methods, 37
crossprod-methods (matrix-products), 106
cSparseMatrix-class, 32
cumsum, 44
data.frame, 67, 156
dCHMsimpl, 174
dCHMsimpl-class (CHMfactor-class), 18
dCHMsuper-class (CHMfactor-class), 18
ddenseMatrix, 36, 64
ddenseMatrix-class, 34
ddiMatrix, 44, 92
ddiMatrix-class, 35
denseLU, 99
denseLU-class (LU-class), 100
denseMatrix, 17, 45, 91, 98, 110, 124, 134, 145, 146, 167, 173
denseMatrix-class, 36
det, 104
det (Matrix-class), 104
determinant, 27, 51, 61, 156
determinant, CHMfactor, logical-method (CHMfactor-class), 18
determinant, dgCMatrix, logical-method (dgCMatrix-class), 36
determinant, dgeMatrix, logical-method (dgeMatrix-class), 37
determinant, dGRMatrix, logical-method (dGRMatrix-class), 39
determinant, dGMatrix, logical-method (dGMatrix-class), 40
determinant, diagonalMatrix, logical-method (diagonalMatrix-class), 43
determinant, dpoMatrix, logical-method (dpoMatrix-class), 50
determinant, dppMatrix, logical-method (dppMatrix-class), 50
determinant, dScMatrix, logical-method (dScMatrix-class), 53
determinant, dspMatrix, logical-method (dspMatrix-class), 57
determinant, dsRMatrix, logical-method (dsRMatrix-class), 56
determinant, dsTMatrix, logical-method (dsTMatrix-class), 53
determinant, dSyMatrix, logical-method (dSyMatrix-class), 57
determinant, Matrix, logical-method (Matrix-class), 104
determinant, Matrix, missing-method (Matrix-class), 104
determinant, MatrixFactorization, missing-method (MatrixFactorization-class), 109
determinant, pMatrix, logical-method (pMatrix-class), 125
determinant, triangularMatrix, logical-method (triangularMatrix-class), 167
dgCMatrix, 16, 18, 20, 34, 39–41, 48, 49, 54, 55, 57, 60, 63, 95, 98, 105, 107, 120, 131, 149, 150, 158
dgCMatrix-class, 36
dgeMatrix, 34, 48, 51, 54, 57, 58, 60, 63, 98, 101, 105, 157, 158
dgeMatrix-class, 37
dgRMatrix, 55, 141
dgRMatrix-class, 39
dgTMatrix, 54, 55, 57, 60, 63, 73, 76, 94, 95, 168–170
dgTMatrix-class, 40
diag, 11, 42, 78, 111
diag,CsparseMatrix-method
(diag,CsparseMatrix-class), 32
diag,diagonalMatrix-method
(diag,diagonalMatrix-class), 43
diag,indMatrix-method
(diag,indMatrix-class), 78
diag,packMatrix-method
(diag,packMatrix-class), 123
diag, RsparseMatrix-method
(diag,RsparseMatrix-class), 140
diag,TsparseMatrix-method
(diag,TsparseMatrix-class), 168
diag,unpackMatrix-method
(diag,unpackMatrix-class), 173
diag<-, CsparseMatrix-method
(diag<-, CsparseMatrix-class), 32
diag<-,diagonalMatrix-method
(diag<-,diagonalMatrix-class), 43
diag<-,indMatrix-method
(diag<-,indMatrix-class), 78
diag<-,packMatrix-method
(diag<-,packMatrix-class), 123
diag<-, RsparseMatrix-method
(diag<-, RsparseMatrix-class), 140
diag<-,TsparseMatrix-method
(diag<-,TsparseMatrix-class), 168
diag<-,unpackMatrix-method
(diag<-,unpackMatrix-class), 173
diagU2N (diagU2N), 44
Diagonal, 13, 35, 41, 44, 92, 103, 153, 164
diagonalMatrix, 13, 17, 35, 41, 42, 73, 86, 92, 102, 103
diagonalMatrix-class, 43
diagU2N, 44
diff, 105, 138
diff,Matrix-method (Matrix-class), 104
dim, 105, 129, 139, 170
diag,Matrix-method (Matrix-class), 104
dim,MatrixFactorization-method
(diag,MatrixFactorization-class), 109
dim<-, denseMatrix-method
(dim<-, denseMatrix-class), 36
dim<-, sparseMatrix-method
(dim<-, sparseMatrix-class), 155
dim<-, sparseVector-method
(dim<-, sparseVector-class), 160
dimnames, 29, 35, 43, 48, 67, 71, 83, 87, 92,
98, 103, 105, 152, 165
dimnames,Matrix-method (Matrix-class), 104
dimnames,symmetricMatrix-method
(dimnames,symmetricMatrix-class), 164
dimnames<-,compMatrix,list-method
(dimnames<-, compMatrix-class), 29
dimnames<-,compMatrix,NULL-method
(dimnames<-, compMatrix-class), 29
dimnames<-,Matrix,list-method
(dimnames<-, Matrix-class), 104
dimnames<-,Matrix,NULL-method
(dimnames<-, Matrix-class), 104
dimScale, 46
dMatrix, 34, 35, 46, 66, 180
dMatrix-class, 47
dmperm, 48
dnearneigh, 177
double, 138, 148
dpoMatrix, 22, 26, 27, 58, 74, 111, 112, 173
dpoMatrix-class, 50
dppMatrix, 22, 58, 123
dppMatrix-class (dpoMatrix-class), 50
drop, abIndex-method (abIndex-class), 5
drop,Matrix-method (Matrix-class), 104
drop0, 19, 48, 52, 117, 159, 180
dsCMatrix, 18, 20, 22, 24, 25, 37, 175, 176
dsCMatrix-class, 53
dspMatrix, 18, 34, 41, 42, 54, 56
dspMatrix-class, 55
dspMatrix-class (dsyMatrix-class), 57
dSRMatrix, 56
dsTMatrix, 177
dsTMatrix-class (dsCMatrix-class), 53
dsyMatrix, 26, 38, 51, 165, 173
dsyMatrix-class, 57
dtCMatrix, 37, 45, 49, 149
dtCMatrix-class, 59
dtpMatrix, 64
dtpMatrix-class, 61
dtrMatrix, 27, 38, 60, 62, 100, 168
dtrMatrix-class, 62
dtrMatrix-class, 63
dtTMatrix-class (dtCMatrix-class), 59
eigen, 15, 16, 105
eigen, Matrix, ANY, logical-method
  (Matrix-class), 104
eigen, Matrix, ANY, missing-method
  (Matrix-class), 104
expand, 24, 25, 65, 99, 101
expand, CHMfactor-method (expand), 65
expand, denseLU-method (expand), 65
expand, MatrixFactorization-method (expand), 65
expand, sparseLU-method (expand), 65
expm, 48, 66, 66
expm, ddiMatrix-method (expm), 66
expm, dgeMatrix-method (expm), 66
expm, dMatrix-method (expm), 66
expm, dsparseMatrix-method (expm), 66
expm, dspMatrix-method (expm), 66
expm, dsyMatrix-method (expm), 66
expm, dtpMatrix-method (expm), 66
expm, dtrMatrix-method (expm), 66
expm, Matrix-method (expm), 66
expm, matrix-method (expm), 66
extends, 103
externalFormats, 67
Extract, 178, 179
fac2Sparse (sparse.model.matrix), 147
fac2sparse, 148
fac2sparse (sparse.model.matrix), 147
facmul, 65, 68
factor, 147, 148
FALSE, 83
forceSymmetric, 70, 85
forceSymmetric, CsparseMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, CsparseMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, diagonalMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, diagonalMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, indMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, indMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, matrix, character-method
  (forceSymmetric), 70
forceSymmetric, matrix, missing-method
  (forceSymmetric), 70
forceSymmetric, packedMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, packedMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, RsparseMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, RsparseMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, TsparseMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, TsparseMatrix, missing-method
  (forceSymmetric), 70
forceSymmetric, unpackedMatrix, character-method
  (forceSymmetric), 70
forceSymmetric, unpackedMatrix, missing-method
  (forceSymmetric), 70
format, 71, 72, 127, 155
format, sparseMatrix-method
  (sparseMatrix-class), 155
formatSparseM, 71, 129
formatSpMatrix, 71, 72, 155
formatSpMatrix (printSpMatrix), 127
function, 87, 139
generalMatrix, 42, 79, 153, 163, 166
generalMatrix-class, 72
get.adjacency, 74
get.gpar, 76
getClassDef, 128
getOption, 128
getValidity, 51, 165
graph, 73
graph-sparseMatrix, 73
graph. adjacency, 74
graph2T (graph-sparseMatrix), 73
grey, 76
grid.raster, 75
grid.rect, 76
head, 161
head, Matrix-method (Matrix-class), 104
head, sparseVector-method
  (sparseVector-class), 160
Hilbert, 74
identical, 172
image, 75, 105
image, ANY-method (image-methods), 75
image, CHMfactor-method (image-methods), 75  
image, dgtMatrix-method (image-methods), 75  
image, Matrix-method (image-methods), 75  
image-methods, 75  
iMatrix-class (Unused-classes), 174  
index, 179  
index-class, 77  
indMatrix, 17, 126, 127  
indMatrix-class, 78  
Inf, 82  
initialize, Matrix-method (Matrix-class), 104  
initialize, sparseVector-method (sparseVector-class), 160  
integer, 9, 32, 40, 53, 59, 78, 117, 170  
invisible, 129  
invPerm, 80, 125, 127  
is, 75  
is.atomic, 8, 9  
is.finite, 81  
is.finite, abIndex-method (is.na-methods), 81  
is.finite, dgeMatrix-method (is.na-methods), 81  
is.finite, diagonalMatrix-method (is.na-methods), 81  
is.finite, dsparseMatrix-method (is.na-methods), 81  
is.finite, dspMatrix-method (is.na-methods), 81  
is.finite, dtpMatrix-method (is.na-methods), 81  
is.finite, dtrMatrix-method (is.na-methods), 81  
is.finite, lgeMatrix-method (is.na-methods), 81  
is.finite, lMatrix-method (is.na-methods), 81  
is.finite, ltrMatrix-method (is.na-methods), 81  
is.finite, nMatrix-method (is.na-methods), 81  
is.finite, nsparseVector-method (is.na-methods), 81  
is.finite, sparseVector-method (is.na-methods), 81  
is.finite-methods (is.na-methods), 81  
is.infinite, 81  
is.infinite, abIndex-method (is.na-methods), 81  
is.infinite, dgeMatrix-method (is.na-methods), 81  
is.infinite, diagonalMatrix-method (is.na-methods), 81  
is.infinite, dsparseMatrix-method (is.na-methods), 81  
is.infinite, dspMatrix-method (is.na-methods), 81  
is.infinite, dtpMatrix-method (is.na-methods), 81  
is.infinite, dtrMatrix-method (is.na-methods), 81  
is.infinite, indMatrix-method (is.na-methods), 81  
is.infinite, lMatrix-method (is.na-methods), 81  
is.infinite, nMatrix-method (is.na-methods), 81  
is.infinite, nsparseVector-method (is.na-methods), 81  
is.infinite, sparseVector-method (is.na-methods), 81  
is.infinite-methods (is.na-methods), 81  
is.na, 81  
is.na, abIndex-method (is.na-methods), 81  
is.na, dgeMatrix-method (is.na-methods), 81  
is.na, diagonalMatrix-method (is.na-methods), 81  
is.na, dsparseMatrix-method (is.na-methods), 81  
is.na, dspMatrix-method (is.na-methods), 81  
is.na, dtpMatrix-method (is.na-methods), 81  
is.na, dtrMatrix-method (is.na-methods), 81  
is.na, indMatrix-method (is.na-methods), 81  
is.na, matrix-method (is.na-methods), 81  
is.na, nMatrix-method (is.na-methods), 81  
is.na, nsparseVector-method (is.na-methods), 81  
is.na, sparseVector-method (is.na-methods), 81  
is.na-methods, 81  
invisible, 129  
is.finite, ltpMatrix-method (is.na-methods), 81  
is.finite, ltrMatrix-method (is.na-methods), 81  
is.finite, nMatrix-method (is.na-methods), 81  
is.finite, nsparseVector-method (is.na-methods), 81  
is.finite, sparseVector-method (is.na-methods), 81  
is.finite-methods (is.na-methods), 81  
is.infinite, 81  
is.infinite, abIndex-method (is.na-methods), 81  
is.infinite, dgeMatrix-method (is.na-methods), 81  
is.infinite, diagonalMatrix-method (is.na-methods), 81  
is.infinite, dsparseMatrix-method (is.na-methods), 81  
is.infinite, dspMatrix-method (is.na-methods), 81  
is.infinite, dtpMatrix-method (is.na-methods), 81  
is.infinite, dtrMatrix-method (is.na-methods), 81  
is.infinite, indMatrix-method (is.na-methods), 81  
is.infinite, lMatrix-method (is.na-methods), 81  
is.infinite, nMatrix-method (is.na-methods), 81  
is.infinite, nsparseVector-method (is.na-methods), 81  
is.infinite, sparseVector-method (is.na-methods), 81  
is.infinite-methods (is.na-methods), 81  
is.na, 81  
is.na, abIndex-method (is.na-methods), 81  
is.na, dgeMatrix-method (is.na-methods), 81  
is.na, diagonalMatrix-method (is.na-methods), 81  
is.na, dsparseMatrix-method (is.na-methods), 81  
is.na, dspMatrix-method (is.na-methods), 81  
is.na, dtpMatrix-method (is.na-methods), 81  
is.na, dtrMatrix-method (is.na-methods), 81  
is.na, indMatrix-method (is.na-methods), 81  
is.na, matrix-method (is.na-methods), 81  
is.na, nMatrix-method (is.na-methods), 81  
is.na, nsparseVector-method (is.na-methods), 81  
is.na, sparseVector-method (is.na-methods), 81  
is.na-methods, 81
is.na, dgeMatrix-method (is.na-methods), 81
is.na, dgeMatrix-method (is.na-methods), 81
is.na, dtpMatrix-method (is.na-methods), 81
is.na, dtpMatrix-method (is.na-methods), 81
is.na, dtrMatrix-method (is.na-methods), 81
is.na, dtrMatrix-method (is.na-methods), 81
is.na, indMatrix-method (is.na-methods), 81
is.na, indMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lMatrix-method (is.na-methods), 81
is.na, lMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lgpMatrix-method (is.na-methods), 81
is.na, lgpMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lgeMatrix-method (is.na-methods), 81
is.na, lgpMatrix-method (is.na-methods), 81
is.na, lgpMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, ltsMatrix-method (is.na-methods), 81
is.na, nMatrix-method (is.na-methods), 81
is.na, nMatrix-method (is.na-methods), 81
is.na, nMatrix-method (is.na-methods), 81
is.na, nMatrix-method (is.na-methods), 81
is.na, nsparseVector-method (is.na-methods), 81
is.na, nsparseVector-method (is.na-methods), 81
is.na, sparseVector-method (is.na-methods), 81
is.na, sparseVector-method (is.na-methods), 81
is.null, 83
is.null.DN, 83
isDiagonal, 44
isDiagonal, CsparseMatrix-method (isTriangular), 86
isDiagonal, diagonalMatrix-method (isTriangular), 86
isDiagonal, indMatrix-method (isTriangular), 86
isDiagonal, matrix-method (isTriangular), 86
isDiagonal, packedMatrix-method (isTriangular), 86
isDiagonal, sparseMatrix-method (isTriangular), 86
isDiagonal, TsparseMatrix-method (isTriangular), 86
isDiagonal, unpackedMatrix-method (isTriangular), 86
isDiagonal-methods (isTriangular), 86
isLDL (CHMfactor-class), 18
isSparseVector-class (sparseVector-class), 160
isSymmetric, 86, 112, 165, 167, 168, 172
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dgCMatrix-method (isSymmetric-methods), 84
isSymmetric, dgCMatrix-method (isSymmetric-methods), 84
isSymmetric, dtCMatrix-method (isSymmetric-methods), 84
isSymmetric, dtCMatrix-method (isSymmetric-methods), 84
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dgRMatrix-method (isSymmetric-methods), 84
isSymmetric, dgRMatrix-method (isSymmetric-methods), 84
isSymmetric, dgTMatrix-method (isSymmetric-methods), 84
isSymmetric, dgTMatrix-method (isSymmetric-methods), 84
isSymmetric, diagonalMatrix-method (isSymmetric-methods), 84
isSymmetric, diagonalMatrix-method (isSymmetric-methods), 84
isSymmetric, dtCMatrix-method (isSymmetric-methods), 84
isSymmetric, dtCMatrix-method (isSymmetric-methods), 84
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dgeMatrix-method (isSymmetric-methods), 84
isSymmetric, dtpMatrix-method (isSymmetric-methods), 84
isSymmetric, dtpMatrix-method (isSymmetric-methods), 84
isSymmetric, dtRMatrix-method (isSymmetric-methods), 84
isSymmetric, dtRMatrix-method (isSymmetric-methods), 84
isSymmetric, matrix-method (isSymmetric-methods), 84
isSymmetric, matrix-method (isSymmetric-methods), 84
isSymmetric, packedMatrix-method (isSymmetric-methods), 84
isSymmetric, packedMatrix-method (isSymmetric-methods), 84
isSymmetric, sparseMatrix-method (isSymmetric-methods), 84
isSymmetric, sparseMatrix-method (isSymmetric-methods), 84
isSymmetric, sparseVector-method (isSymmetric-methods), 84
isSymmetric, sparseVector-method (isSymmetric-methods), 84
isSymmetric, TsparseMatrix-method (isSymmetric-methods), 84
isSymmetric, TsparseMatrix-method (isSymmetric-methods), 84
isSymmetric, unpackedMatrix-method (isSymmetric-methods), 84
isSymmetric, unpackedMatrix-method (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric-methods (isSymmetric-methods), 84
isSymmetric,dtTMatrix-method
(isSymmetric-methods), 84
isSymmetric,indMatrix-method
(isSymmetric-methods), 84
isSymmetric,lgCMatrix-method
(isSymmetric-methods), 84
isSymmetric,lgeMatrix-method
(isSymmetric-methods), 84
isSymmetric,lgRMatrix-method
(isSymmetric-methods), 84
isSymmetric,lgTMatrix-method
(isSymmetric-methods), 84
isSymmetric,ngCMatrix-method
(isSymmetric-methods), 84
isSymmetric,ngEMatrix-method
(isSymmetric-methods), 84
isSymmetric,ngRMatrix-method
(isSymmetric-methods), 84
isSymmetric,ngTMatrix-method
(isSymmetric-methods), 84
isSymmetric,symmetricMatrix-method
(isSymmetric-methods), 84
isSymmetric,triangularMatrix-method
(isSymmetric-methods), 84

isSymmetric-methods, 84, 165
isSymmetric.matrix, 85
isTriangular, 86, 168, 172
isTriangular,dgCMatrix-method
(isTriangular), 86
isTriangular,dgeMatrix-method
(isTriangular), 86
isTriangular,lgRMatrix-method
(isTriangular), 86
isTriangular,lgTMatrix-method
(isTriangular), 86
isTriangular,diagonalMatrix-method
(isTriangular), 86
isTriangular,indMatrix-method
(isTriangular), 86
isTriangular,lgCMatrix-method
(isTriangular), 86
isTriangular,lgeMatrix-method
(isTriangular), 86
isTriangular,lgTMatrix-method
(isTriangular), 86
isTriangular,triangularMatrix-method
(isTriangular), 86

isTriangular-methods (isTriangular), 86
isTriangular,ngCMatrix-method
(isTriangular), 86
isTriangular,ngEMatrix-method
(isTriangular), 86
isTriangular,ngRMatrix-method
(isTriangular), 86
isTriangular,ngTMatrix-method
(isTriangular), 86
isTriangular,symmetricMatrix-method
(isTriangular), 86
isTriangular,triangularMatrix-method
(isTriangular), 86

kappa, 105, 136
KhatriRao, 87
KNex, 89
kronecker, 13, 34, 36, 79, 87, 88, 90, 105
kronecker,CsparseMatrix,CsparseMatrix-method
(kronecker-methods), 90
kronecker,CsparseMatrix,diagonalMatrix-method
(kronecker-methods), 90
kronecker,CsparseMatrix,Matrix-method
(kronecker-methods), 90
kronecker,denseMatrix,denseMatrix-method
(kronecker-methods), 90
kronecker,denseMatrix,Matrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,CsparseMatrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,diagonalMatrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,indMatrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,Matrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,RsparseMatrix-method
(kronecker-methods), 90
kronecker,diagonalMatrix,TsparseMatrix-method
(kronecker-methods), 90
kronecker,indMatrix,diagonalMatrix-method
(kronecker-methods), 90
kronecker,indMatrix,indMatrix-method
(kronecker-methods), 90
kronecker,indMatrix,Matrix-method
(kronecker-methods), 90
kronecker,Matrix,Matrix-method
(kronecker-methods), 90
INDEX

Logic,ltCMatrix,ltCMatrix-method
(lsparseMatrix-classes), 93
Logic,Matrix,ANY-method (Matrix-class), 104
Logic,nMatrix-method (Matrix-class), 104
Logic,ngeMatrix,ngeMatrix-method (ngeMatrix-class), 114
Logic,nMatrix,logical-method (Matrix-class), 115
Logic,nMatrix,Matrix-method (nMatrix-class), 115
Logic,nMatrix,nMatrix-method (nMatrix-class), 115
Logic,nMatrix,numeric-method (Matrix-class), 115
Logic,nMatrix,sparseVector-method (nMatrix-class), 115
Logic,numeric,dMatrix-method (nMatrix-class), 115
Logic,numeric,lMatrix-method (nMatrix-class), 115
Logic,numeric,nMatrix-method (nMatrix-class), 115
Logic,sparseVector,dMatrix-method (sparseVector-class), 160
Logic,sparseVector,lMatrix-method (sparseVector-class), 160
Logic,sparseVector,nMatrix-method (sparseVector-class), 160
Logic,sparseVector,sparseVector-method (sparseVector-class), 160
Logic,triangularMatrix,diagonalMatrix-method (triangularMatrix-class), 167
logical, 17, 19, 42, 48, 51, 78, 83, 85, 86, 92, 101, 103, 107, 109, 117, 133, 161, 178
logm, 66
lsCMatrix, 18, 165
lsCMatrix-class (lsparseMatrix-classes), 93
lsRMatrix-class (lsparseMatrix-classes), 93
lsTMatrix-class (lsparseMatrix-classes), 93
ltCMatrix, 168
ltCMatrix-class (lsparseMatrix-classes), 93
ltMMatrix-class (ltMMatrix-class), 95
ltMMatrix-class (lsparseMatrix-classes), 93
lttRMatrix-class (ltrMatrix-class), 96
lttRMatrix-class (lsparseMatrix-classes), 93
ltrMatrix-class (ltrMatrix-class), 96
ltrMatrix-class (lsparseMatrix-classes), 93
LU, 110, 149
lu, 15, 30, 31, 37, 65, 97, 98, 100, 101, 145, 146, 149, 150, 156
lu,denseMatrix-method (lu), 97
lu,dgCMatrix-method (lu), 97
lu,dgeMatrix-method (lu), 97
lu,dgRMatrix-method (lu), 97
lu,dgTMatrix-method (lu), 97
lu,diagonalMatrix-method (lu), 97
lu,dsCMatrix-method (lu), 97
lu,dsRMatrix-method (lu), 97
lu,dsTMatrix-method (lu), 97
lu,dsyMatrix-method (lu), 97
lu,dtCMatrix-method (lu), 97
lu,dtpMatrix-method (lu), 97
lu,dtRMatrix-method (lu), 97
lu,dtTMatrix-method (lu), 97
lu,matrix-method (lu), 97
LU-class, 100
mat2triplet, 101
Math,CsparseMatrix-method (CsparseMatrix-class), 32
Math,ddenseMatrix-method (ddenseMatrix-class), 34
Math,denseMatrix-method (denseMatrix-class), 36
Math,dgeMatrix-method (dgeMatrix-class), 37

...
INDEX

Math, diagonalMatrix-method (diagonalMatrix-class), 43
Math, sparseMatrix-method (sparseMatrix-class), 155
Math, sparseVector-method (sparseVector-class), 160
Math2, dMatrix-method (dMatrix-class), 47
Math2, dsparseMatrix-method (sparseVector-class), 160
Math2, Matrix-method (Matrix-class), 104
Math2, sparseVector-method (sparseVector-class), 160
matrix, 7, 10, 27, 44, 70, 71, 79, 83, 84, 86, 101, 103, 105, 111, 117, 126, 141, 142, 178, 179
Matrix-class, 104
matrix-products, 106
matrix.csr, 151
MatrixClass, 108
MatrixFactorization, 25, 27, 30, 100, 141, 143, 144, 146, 149
MatrixFactorization-class, 109
max, 47
mean, Matrix-method (Matrix-class), 104
mean, sparseMatrix-method (sparseMatrix-class), 155
mean, sparseVector-method (sparseVector-class), 160
message, 115
method, 84
min, 47
mode, 79, 126
model.frame, 147
model.Matrix, 148
model.matrix, 147, 148
modelMatrix, 148
NA, 17, 24, 44, 48, 82, 85, 86, 90, 94, 98, 115, 117, 156, 161, 166, 178
NA_integer_, 133
names, 29, 42
NaN, 82, 133
nb2listw, 175, 177
nCHMsimpl-class (CHMfactor-class), 18
nCHMsuper-class (CHMfactor-class), 18
ncol, 167
ndenseMatrix, 36
ndenseMatrix-class, 110
nearcor, 113
nearPD, 111
new, 33, 151
ngCMatrix, 115
ngCMMatrix-class (nsparseMatrix-classes), 119
ngeMatrix, 111, 122, 123
ngeMatrix-class, 114
ngRMatrix-class (nsparseMatrix-classes), 119
ngTMatrix, 73, 78, 79, 126
ngTMatrix-class (nsparseMatrix-classes), 119
nMatrix, 11, 48, 79, 82, 101, 107, 114, 179, 180
nMatrix-class, 115
nnzero, 52, 62, 116, 140
nnzero, ANY-method (nnzero), 116
nnzero, array-method (nnzero), 116
nnzero, CHMfactor-method (nnzero), 116
nnzero, denseMatrix-method (nnzero), 116
nnzero, diagonalMatrix-method (nnzero), 116
nnzero, indMatrix-method (nnzero), 116
nnzero, sparseMatrix-method (nnzero), 116
nnzero, vector-method (nnzero), 116
norm, 31, 32, 58, 112, 118, 119, 135, 136, 156
norm, ANY, missing-method (norm), 118
norm, denseMatrix, character-method (norm), 118
norm, dgeMatrix, character-method (norm), 118
norm, diagonalMatrix, character-method (norm), 118
norm, dspMatrix, character-method (norm), 118
norm, dsyMatrix, character-method (norm), 118
norm, dtpMatrix, character-method (norm), 118
norm, dtrMatrix, character-method (norm), 118
norm, sparseMatrix, character-method
INDEX

(norm), 118
nrow, 167
nsCMatrix-class
  (nsparseMatrix-classes), 119
nsparseMatrix-class
  (nsparseMatrix-classes), 119
nsparseMatrix-class
  (nsparseMatrix-classes), 119
nsparseMatrix-class, 119
nsparseVector, 82
nsparseVector-class
  (sparseVector-class), 160
nsyMatrix-class
  (nsyMatrix-class), 121
ntCMatrix-class
  (nsparseMatrix-classes), 119
ntpMatrix-class
  (ntrMatrix-class), 122
ntrMatrix-class
  (nsparseMatrix-classes), 119
ntrMatrix-class, 122
ntrMatrix-class
  (nsparseMatrix-classes), 119
ntTMatrix-class
  (nsparseMatrix-classes), 119
numeros, 30
Ops, 5, 44, 64, 161
Ops,abIndex,abIndex-method
  (abIndex-class), 5
Ops,abIndex,ANY-method (abIndex-class), 5
Ops,ANY,abIndex-method (abIndex-class), 5
Ops,ANY,ddiMatrix-method (ddiMatrix-class), 35
Ops,ANY,ldiMatrix-method (ldiMatrix-class), 92
Ops,ANY,Matrix-method (Matrix-class), 104
Ops,ANY,sparseVector-method
  (sparseVector-class), 160
Ops,atomicVector,sparsicVector-method
  (atomicVector-class), 8
Ops,ddiMatrix,ANY-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,ddiMatrix-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,dMatrix-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,ldiMatrix-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,logical-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,MATRIX-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,numeric-method
  (ddiMatrix-class), 35
Ops,ddiMatrix,sparseMatrix-method
  (ddiMatrix-class), 35
Ops,diagonalMAtrix,triangularMatrix-method
  (diagonalMatrix-class), 43
Ops,dMatrix,ddiMatrix-method
  (dMatrix-class), 47
Ops,dMatrix,dMatrix-method
  (dMatrix-class), 47
Ops,dMatrix,ldiMatrix-method
  (dMatrix-class), 47
Ops,dMatrix,lMatrix-method
  (dMatrix-class), 47
Ops,dMatrix,nMatrix-method
  (dMatrix-class), 47
Ops,dpoMatrix,logical-method
  (dpoMatrix-class), 50
Ops,dpoMatrix,numeric-method
  (dpoMatrix-class), 50
Ops,dppMatrix,logical-method
  (dpoMatrix-class), 50
Ops,dppMatrix,numeric-method
  (dpoMatrix-class), 50
Ops,dsparseMatrix,nsparseMatrix-method
  (dsparseMatrix-class), 55
Ops,ldenseMatrix,ldenseMatrix-method
  (ldenseMatrix-class), 91
Ops,ldiMatrix,ANY-method
  (ldiMatrix-class), 92
Ops,ldiMatrix,ddiMatrix-method
  (ldiMatrix-class), 92
Ops,ldiMatrix,dMatrix-method
   (ldiMatrix-class), 92
Ops,ldiMatrix,ldiMatrix-method
   (ldiMatrix-class), 92
Ops,ldiMatrix,logical-method
   (ldiMatrix-class), 92
Ops,ldiMatrix,Matrix-method
   (ldiMatrix-class), 92
Ops,ldiMatrix,numeric-method
   (ldiMatrix-class), 92
Ops,ldiMatrix,sparseMatrix-method
   (ldiMatrix-class), 92
Ops,lMatrix,dMatrix-method
   (dMatrix-class), 47
Ops,lMatrix,lMatrix-method
   (dMatrix-class), 47
Ops,lMatrix,nMatrix-method
   (dMatrix-class), 47
Ops,lMatrix,numeric-method
   (dMatrix-class), 47
Ops,logical,dpoMatrix-method
   (dpoMatrix-class), 50
Ops,logical,dppMatrix-method
   (dpoMatrix-class), 50
Ops,lsparseMatrix,lsparseMatrix-method
   (lsparseMatrix-classes), 93
Ops,lsparseMatrix,nsparseMatrix-method
   (lsparseMatrix-classes), 93
Ops,Matrix,ANY-method (Matrix-class), 104
Ops,Matrix,ddiMatrix-method
   (Matrix-class), 104
Ops,Matrix,ldiMatrix-method
   (Matrix-class), 104
Ops,Matrix,matrix-method
   (Matrix-class), 104
Ops,matrix,Matrix-method
   (Matrix-class), 104
Ops,Matrix,NULL-method (Matrix-class), 104
Ops,Matrix,sparseVector-method
   (Matrix-class), 104
Ops,ndenseMatrix,ndenseMatrix-method
   (ndenseMatrix-class), 110
Ops,nMatrix,dMatrix-method
   (nMatrix-class), 115
Ops,nMatrix,nMatrix-method
   (nMatrix-class), 115
Ops,nMatrix,numeric-method
   (nMatrix-class), 115
Ops,nsparseMatrix,dsparseMatrix-method
   (nsparseMatrix-classes), 119
Ops,nsparseMatrix,lsparseMatrix-method
   (nsparseMatrix-classes), 119
Ops,nsparseMatrix,sparseMatrix-method
   (nsparseMatrix-classes), 119
Ops,NULL,Matrix-method (Matrix-class), 104
Ops,numeric,dpoMatrix-method
   (dpoMatrix-class), 50
Ops,numeric,dppMatrix-method
   (dpoMatrix-class), 50
Ops,numeric,lMatrix-method
   (dMatrix-class), 47
Ops,numeric,nMatrix-method
   (nMatrix-class), 115
Ops,numeric,sparseMatrix-method
   (sparseMatrix-class), 155
Ops,sparseMatrix,ddiMatrix-method
   (sparseMatrix-class), 155
Ops,sparseMatrix,ldiMatrix-method
   (sparseMatrix-class), 155
Ops,sparseMatrix,nsparseMatrix-method
   (sparseMatrix-class), 155
Ops,sparseMatrix,numeric-method
   (sparseMatrix-class), 155
Ops,sparseMatrix,sparseMatrix-method
   (sparseMatrix-class), 155
Ops,sparseVector,ANY-method
   (sparseVector-class), 160
Ops,sparseVector,atomicVector-method
   (sparseVector-class), 160
Ops,sparseVector,Matrix-method
   (sparseVector-class), 160
Ops,sparseVector,sparseVector-method
   (sparseVector-class), 160
options, 115, 128, 129, 158, 160
order, 81, 101, 169
outer, 105
over, 177
pack, 57, 63, 125, 173
pack (unpack), 171
pack,dgeMatrix-method (unpack), 171
pack,lgeMatrix-method (unpack), 171
pack, matrix-method (unpack), 171
pack, ngeMatrix-method (unpack), 171
pack, packedMatrix-method (unpack), 171
pack, sparseMatrix-method (unpack), 171
pack, unpackedMatrix-method (unpack), 171
packedMatrix, 36, 171, 173
packedMatrix-class, 123
panel.levelplot.raster, 75
paste, 147
pBunchKaufman, 123
pBunchKaufman-class (Cholesky-class), 26
pCholesky, 123
pCholesky-class (Cholesky-class), 26
plot.default, 75
pMatrix, 49, 78, 79, 81, 158
pMatrix-class, 125
posdefify, 111–113
print, 44, 72, 75, 105, 127, 129, 155, 156
print, diagonalMatrix-method (diagonalMatrix-class), 43
print, sparseMatrix-method (sparseMatrix-class), 155
print.default, 71, 128
print.Matrix (Matrix-class), 104
print.sparseMatrix (sparseMatrix-class), 155
print.trellis, 76
printSpMatrix, 72, 105, 127, 155
prod, 47
prod, ddiMatrix-method (ddiMatrix-class), 35
prod, ldiMatrix-method (ldiMatrix-class), 92
qr, 30, 99, 130, 131, 133, 134, 157, 158
qr (qr-methods), 130
qr, denseMatrix-method (qr-methods), 130
qr, dgCMatrix-method (qr-methods), 130
qr, sparseMatrix-method (qr-methods), 130
qr-methods, 130
qr.coef, 146, 158
qr.coef, sparseQR, ddenseMatrix-method (sparseQR-class), 157
qr.coef, sparseQR, Matrix-method (sparseQR-class), 157
qr.coef, sparseQR, matrix-method (sparseQR-class), 157
qr.coef, sparseQR, numeric-method (sparseQR-class), 157
qr.fitted, 158
qr.fitted, sparseQR, ddenseMatrix-method (sparseQR-class), 157
qr.fitted, sparseQR, Matrix-method (sparseQR-class), 157
qr.fitted, sparseQR, matrix-method (sparseQR-class), 157
qr.fitted, sparseQR, numeric-method (sparseQR-class), 157
qr.Q, 158
qr.Q (sparseQR-class), 157
qr.Q, sparseQR-method (sparseQR-class), 157
qr.qty, 158
qr.qty, sparseQR, ddenseMatrix-method (sparseQR-class), 157
qr.qty, sparseQR, Matrix-method (sparseQR-class), 157
qr.qty, sparseQR, matrix-method (sparseQR-class), 157
qr.qty, sparseQR, numeric-method (sparseQR-class), 157
qr.qy, 158
qr.qy, sparseQR, ddenseMatrix-method (sparseQR-class), 157
qr.qy, sparseQR, Matrix-method (sparseQR-class), 157
qr.qy, sparseQR, matrix-method (sparseQR-class), 157
qr.qy, sparseQR, numeric-method (sparseQR-class), 157
qr.R, 158
qr.R, sparseQR-method (sparseQR-class), 157
qr.resid, 158
qr.resid, sparseQR, ddenseMatrix-method (sparseQR-class), 157
qr.resid, sparseQR, Matrix-method (sparseQR-class), 157
qr.resid, sparseQR, matrix-method (sparseQR-class), 157
qr.resid, sparseQR, numeric-method (sparseQR-class), 157
qr2rankMatrix (rankMatrix), 132
qrR (qr-methods), 130
range, 47
rankMatrix, 132
rBind (cBind), 16
rbind, 16, 17, 156
rbind2, 16
rbind2, ANY, Matrix-method
(Matrix-class), 104
rbind2, atomicVector, ddiMatrix-method
(atomicVector-class), 8
rbind2, atomicVector, ldiMatrix-method
(atomicVector-class), 8
rbind2, atomicVector, Matrix-method
(atomicVector-class), 8
rbind2, ddiMatrix, atomicVector-method
(ddiMatrix-class), 35
rbind2, ddiMatrix, matrix-method
(ddiMatrix-class), 35
rbind2, denseMatrix, denseMatrix-method
(denseMatrix-class), 36
rbind2, denseMatrix, matrix-method
(denseMatrix-class), 36
rbind2, denseMatrix, numeric-method
(denseMatrix-class), 36
rbind2, denseMatrix, sparseMatrix-method
(cBind), 16
rbind2, diagonalMatrix, sparseMatrix-method
(diagonalMatrix-class), 43
rbind2, indMatrix, indMatrix-method
(indMatrix-class), 78
rbind2, ldiMatrix, atomicVector-method
(ldiMatrix-class), 92
rbind2, ldiMatrix, matrix-method
(ldiMatrix-class), 92
rbind2, Matrix, ANY-method
(Matrix-class), 104
rbind2, Matrix, atomicVector-method
(Matrix-class), 104
rbind2, matrix, ddiMatrix-method
(ddiMatrix-class), 35
rbind2, matrix, denseMatrix-method
(denseMatrix-class), 36
rbind2, matrix, ldiMatrix-method
(ldiMatrix-class), 92
rbind2, Matrix, Matrix-method
(Matrix-class), 104
rbind2, Matrix, missing-method
(Matrix-class), 104
rbind2, Matrix, NULL-method
(Matrix-class), 104
rbind2, matrix, sparseMatrix-method
(sparseMatrix-class), 155
rbind2, NULL, Matrix-method
(Matrix-class), 104
rbind2, numeric, denseMatrix-method
(denseMatrix-class), 36
rbind2, sparseMatrix, denseMatrix-method
(cBind), 16
rbind2, sparseMatrix, diagonalMatrix-method
(sparseMatrix-class), 155
rbind2, sparseMatrix, matrix-method
(sparseMatrix-class), 155
rbind2, sparseMatrix, sparseMatrix-method
(sparseMatrix-class), 155
rcond, 32, 38, 51, 58, 135
rcond, ANY, missing-method (rcond), 135
rcond, denseMatrix, character-method
(rcond), 135
rcond, dgeMatrix, character-method
(rcond), 135
rcond, dpoMatrix, character-method
(rcond), 135
rcond, dppMatrix, character-method
(rcond), 135
rcond, dspMatrix, character-method
(rcond), 135
rcond, dtyMatrix, character-method
(rcond), 135
rcond, sparseMatrix, character-method
(rcond), 135
read.gal, 175
readHB (externalFormats), 67
readMM (externalFormats), 67
rep, Matrix-method (Matrix-class), 104
rep, sparseMatrix-method
(sparseMatrix-class), 155
rep, sparseVector-method
(sparseVector-class), 160
rep.int, 137
rep2abI, 7, 137
replValue-class, 138
Rgshhs, 177
rle, 5–7, 138, 139
rleDiff, 5
INDEX

rleDiff-class, 138
round, 47
rowMeans, 105
rowMeans(colSums), 28
rowMeans,CsparseMatrix-method
(colSums), 28
rowMeans,denseMatrix-method(colSums), 28
rowMeans,diagonalMatrix-method
(colSums), 28
rowMeans,indMatrix-method(colSums), 28
rowMeans,RsparseMatrix-method
(colSums), 28
rowMeans,TsparseMatrix-method
(colSums), 28
rownames, 130
rowScale(dimScale), 46
rowSums(colSums), 28
rowSums,CsparseMatrix-method(colSums), 28
rowSums,denseMatrix-method(colSums), 28
rowSums,diagonalMatrix-method
(colSums), 28
rowSums,indMatrix-method(colSums), 28
rowSums,RsparseMatrix-method
(colSums), 28
rowSums,TsparseMatrix-method
(colSums), 28
RsparseMatrix, 11, 39, 56, 93, 119, 148,
151–153
rsparsematrix, 139, 153
RsparseMatrix-class, 140

sample.int, 140
Schur, 30, 49, 66, 141, 141, 142, 143, 168
Schur,dgeMatrix,logical-method(Schur),
141
Schur,diagonalMatrix,logical-method
(Schur), 141
Schur,dsyMatrix,logical-method(Schur),
141
Schur,generalMatrix,logical-method
(Schur), 141
Schur,matrix,logical-method(Schur), 141
Schur,Matrix,missing-method(Schur), 141
Schur,matrix,missing-method(Schur), 141
Schur,symmetricMatrix,logical-method
(Schur), 141
Schur,triangularMatrix,logical-method
(Schur), 141
Schur-class, 143
seq, 6
seqMat-class(abIndex-class), 5
set.seed, 31
setClassUnion, 77, 123, 138, 160
show, 5, 72, 105, 110, 127, 129, 139, 155, 160
show,abIndex-method(abIndex-class), 5
show,BunchKaufman-method
(Cholesky-class), 26
show,Cholesky-method(Cholesky-class),
26
show,denseMatrix-method
denseMatrix-class), 36
show,diagonalMatrix-method
diagonalMatrix-class), 43
show,MatrixFactorization-method
MatrixFactorization-class), 109
show,pBunchKaufman-method
(Cholesky-class), 26
show,pCholesky-method(Cholesky-class),
26
show, rleDiff-method(rleDiff-class), 138
show,sparseMatrix-method
(sparseMatrix-class), 155
show,sparseVector-method
(sparseVector-class), 160
showClass, 35, 57, 96, 97, 121, 122
showMethods, 22, 24, 34, 36, 65, 93, 96–98,
115, 121, 122, 133, 166
signif, 47
similar.listw, 175, 177
skewpart, 85
skewpart(symmpart), 166
skewpart,CsparseMatrix-method
(symmpart), 166
skewpart,diagonalMatrix-method
(symmpart), 166
skewpart,indMatrix-method(symmpart),
166
skewpart,matrix-method(symmpart), 166
skewpart,packedMatrix-method
(symmpart), 166
skewpart,RsparseMatrix-method
(symmpart), 166
skewpart,TsparseMatrix-method
(symmpart), 166
skewpart, unpackedMatrix-method (symmpart), 166
skewpart-methods (symmpart), 166
slot, 85
Sobj_SpatialGrid, 177
solve, 23, 25, 33, 38, 51, 58, 126, 135, 136, 144–146, 150
solve (solve-methods), 144
solve, ANY, ANY-method (solve-methods), 144
solve, CHMfactor, denseMatrix-method (solve-methods), 144
solve, CHMfactor, matrix-method (solve-methods), 144
solve, CHMfactor, missing-method (solve-methods), 144
solve, CHMfactor, numLike-method (solve-methods), 144
solve, CSparseMatrix, ANY-method (solve-methods), 144
solve, ddiMatrix, Matrix-method (solve-methods), 144
solve, ddiMatrix, matrix-method (solve-methods), 144
solve, ddiMatrix, missing-method (solve-methods), 144
solve, ddiMatrix, numLike-method (solve-methods), 144
solve, denseLU, missing-method (solve-methods), 144
solve, dgeMatrix, ANY-method (solve-methods), 144
solve, dgeMatrix, denseMatrix-method (solve-methods), 144
solve, dgeMatrix, matrix-method (solve-methods), 144
solve, dgeMatrix, missing-method (solve-methods), 144
solve, dgeMatrix, numLike-method (solve-methods), 144
solve, dgeMatrix, sparseMatrix-method (solve-methods), 144
solve, dsyMatrix, ANY-method (solve-methods), 144
solve, dspMatrix, Matrix-method (solve-methods), 144
solve, dspMatrix, matrix-method (solve-methods), 144
solve, dsyMatrix, matrix-method (solve-methods), 144
solve, dgeMatrix, missing-method (solve-methods), 144
solve, dgeMatrix, numLike-method (solve-methods), 144
solve, dgeMatrix, sparseMatrix-method (solve-methods), 144
solve, dgeMatrix, Matrix-method (solve-methods), 144
solve, dgeMatrix, matrix-method (solve-methods), 144
solve, dgeMatrix, missing-method (solve-methods), 144
solve, dgeMatrix, numLike-method (solve-methods), 144
solve, diagonalMatrix, ANY-method (solve-methods), 144
solve, dpoMatrix, Matrix-method (solve-methods), 144
solve, dpoMatrix, matrix-method (solve-methods), 144
solve, dpoMatrix, missing-method (solve-methods), 144
solve, dpoMatrix, numLike-method (solve-methods), 144
solve, dppMatrix, Matrix-method (solve-methods), 144
solve, dppMatrix, matrix-method (solve-methods), 144
solve, dppMatrix, missing-method (solve-methods), 144
solve, dppMatrix, numLike-method (solve-methods), 144
solve, dgCMatrix, ANY-method (solve-methods), 144
solve, dgCMatrix, denseMatrix-method (solve-methods), 144
solve, dgCMatrix, matrix-method (solve-methods), 144
solve, dgCMatrix, missing-method (solve-methods), 144
solve, dgCMatrix, numLike-method (solve-methods), 144
solve, dgCMatrix, sparseMatrix-method (solve-methods), 144
solve, dgCMatrix, Matrix-method (solve-methods), 144
solve, dgCMatrix, matrix-method (solve-methods), 144
index

(solve-methods), 144
solve,dtCMatrix,denseMatrix-method
(solve-methods), 144
solve,dtCMatrix,dgCMatrix-method
(solve-methods), 144
solve,dtCMatrix,dgeMatrix-method
(solve-methods), 144
solve,dtCMatrix,dsCMatrix-method
(solve-methods), 144
solve,dtCMatrix,dspMatrix-method
(solve-methods), 144
solve,dtCMatrix,dsyMatrix-method
(solve-methods), 144
solve,dtCMatrix,dtCMatrix-method
(solve-methods), 144
solve,dtCMatrix,dtpMatrix-method
(solve-methods), 144
solve,dtCMatrix,dtrMatrix-method
(solve-methods), 144
solve,dtCMatrix,matrix-method
(solve-methods), 144
solve,dtCMatrix,missing-method
(solve-methods), 144
solve,dtCMatrix,numLike-method
(solve-methods), 144
solve,dtCMatrix,sparseMatrix-method
(solve-methods), 144
solve,dtpMatrix,Matrix-method
(solve-methods), 144
solve,dtpMatrix,matrix-method
(solve-methods), 144
solve,dtpMatrix,missing-method
(solve-methods), 144
solve,dtpMatrix,numLike-method
(solve-methods), 144
solve,dtrMatrix,Matrix-method
(solve-methods), 144
solve,dtrMatrix,matrix-method
(solve-methods), 144
solve,dtrMatrix,missing-method
(solve-methods), 144
solve,dtrMatrix,numLike-method
(solve-methods), 144
solve,pMatrix,Matrix-method
(solve-methods), 144
solve,pMatrix,matrix-method
(solve-methods), 144
solve,pMatrix,missing-method
(solve-methods), 144
solve,pMatrix,numLike-method
(solve-methods), 144
solve,RsparseMatrix,ANY-method
(solve-methods), 144
solve,sparseQR,ANY-method
(solve-methods), 144
solve,sparseQR,missing-method
(solve-methods), 144
solve,TsparseMatrix,ANY-method
(solve-methods), 144
solve-methods, 144
sort.list, 81
sparse.model.matrix, 147, 153, 156
sparseLU, 99, 101, 146
sparseLU-class, 149
SparseM-coerce-methods
(SparseM-conversions), 151
SparseM-conversions, 151
SparseM-ontology, 151
sparseMatrix, 9, 11, 17, 18, 28, 33–37, 40,
43, 52, 53, 67, 71, 79, 87, 92, 98,
102, 103, 115, 128, 129, 139–141,
144–148, 151, 156, 160, 161, 163,
164, 169, 174
sparseMatrix-class, 155
sparseQR, 110, 130, 131, 133
sparseQR-class, 157
sparseVector, 8, 28, 29, 81, 106, 144, 159,
159, 160, 161, 180
sparseVector-class, 160
spMatrix, 40, 52, 102, 129, 163, 169
sqrtm, 66
stop, 98, 178
<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subassign</td>
<td>methods ([&lt;--methods])</td>
<td></td>
</tr>
<tr>
<td>Substring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>ddiMatrix-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Sum</td>
<td>ldiMatrix-method (ldiMatrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Summary</td>
<td>abIndex-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>Summary</td>
<td>ddenseMatrix-method (ddenseMatrix-class)</td>
<td>34</td>
</tr>
<tr>
<td>Summary</td>
<td>ddiMatrix-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Summary</td>
<td>diagonalMatrix-method (diagonalMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>Summary</td>
<td>dsparseMatrix-method (dsparseMatrix-class)</td>
<td>55</td>
</tr>
<tr>
<td>Summary</td>
<td>indMatrix-method (indMatrix-class)</td>
<td>78</td>
</tr>
<tr>
<td>Summary</td>
<td>ldenseMatrix-method (ldenseMatrix-class)</td>
<td>91</td>
</tr>
<tr>
<td>Summary</td>
<td>ldiMatrix-method (ldiMatrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Summary</td>
<td>Matrix-method (Matrix-class)</td>
<td>104</td>
</tr>
<tr>
<td>Summary</td>
<td>ndenseMatrix-method (ndenseMatrix-class)</td>
<td>110</td>
</tr>
<tr>
<td>Summary</td>
<td>nMatrix-method (nMatrix-class)</td>
<td>115</td>
</tr>
<tr>
<td>Summary</td>
<td>nsparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>Summary</td>
<td>sparseMatrix-method (sparseMatrix-class)</td>
<td>160</td>
</tr>
<tr>
<td>Summary</td>
<td>sparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>SVD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD</td>
<td>Matrix-method (Matrix-class)</td>
<td>104</td>
</tr>
<tr>
<td>SVD</td>
<td>sparseVector-method (sparseVector-class)</td>
<td></td>
</tr>
<tr>
<td>Summation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summation</td>
<td>ddiMatrix-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Summation</td>
<td>ldiMatrix-method (ldiMatrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Summary</td>
<td>abIndex-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>Summary</td>
<td>ddenseMatrix-method (ddenseMatrix-class)</td>
<td>34</td>
</tr>
<tr>
<td>Summary</td>
<td>ddiMatrix-method (ddiMatrix-class)</td>
<td>35</td>
</tr>
<tr>
<td>Summary</td>
<td>diagonalMatrix-method (diagonalMatrix-class)</td>
<td>43</td>
</tr>
<tr>
<td>Summary</td>
<td>dsparseMatrix-method (dsparseMatrix-class)</td>
<td>55</td>
</tr>
<tr>
<td>Summary</td>
<td>indMatrix-method (indMatrix-class)</td>
<td>78</td>
</tr>
<tr>
<td>Summary</td>
<td>ldenseMatrix-method (ldenseMatrix-class)</td>
<td>91</td>
</tr>
<tr>
<td>Summary</td>
<td>ldiMatrix-method (ldiMatrix-class)</td>
<td>92</td>
</tr>
<tr>
<td>Summary</td>
<td>Matrix-method (Matrix-class)</td>
<td>104</td>
</tr>
<tr>
<td>Summary</td>
<td>ndenseMatrix-method (ndenseMatrix-class)</td>
<td>110</td>
</tr>
<tr>
<td>Summary</td>
<td>nMatrix-method (nMatrix-class)</td>
<td>115</td>
</tr>
<tr>
<td>Summary</td>
<td>nsparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>Summary</td>
<td>sparseMatrix-method (sparseMatrix-class)</td>
<td>160</td>
</tr>
<tr>
<td>Summary</td>
<td>sparseVector-method (sparseVector-class)</td>
<td>160</td>
</tr>
<tr>
<td>T2Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2Graph</td>
<td>graph-sparseMatrix</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>matrix-products</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>ANY, ANY-method</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>Matrix-method (Matrix-class)</td>
<td>104</td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>sparseVector-method (sparseVector-class)</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>symmetricMatrix-class</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>symmpart (symmpart)</td>
<td></td>
</tr>
<tr>
<td>T2Crossprod</td>
<td>symmetricMatrix-method (symmetricMatrix-class)</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

231

tcrossprod, ANY, TsparseMatrix-method (matrix-products), 106
tcrossprod, CsparseMatrix, CsparseMatrix-method (matrix-products), 106
tcrossprod, CsparseMatrix, ddenseMatrix-method (matrix-products), 106
tcrossprod, CsparseMatrix, diagonalMatrix-method (matrix-products), 106
tcrossprod, CsparseMatrix, missing-method (matrix-products), 106
tcrossprod, CsparseMatrix, numLike-method (matrix-products), 106
tcrossprod, ddenseMatrix, CsparseMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, ddenseMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, dsCMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, dtrMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, ldenseMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, lsCMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, missing-method (matrix-products), 106
tcrossprod, ddenseMatrix, ndenseMatrix-method (matrix-products), 106
tcrossprod, ddenseMatrix, nsCMatrix-method (matrix-products), 106
tcrossprod, denseMatrix, diagonalMatrix-method (matrix-products), 106
tcrossprod, dgeMatrix, dgeMatrix-method (matrix-products), 106
tcrossprod, dgeMatrix, matrix-method (matrix-products), 106
tcrossprod, dgeMatrix, missing-method (matrix-products), 106
tcrossprod, dgeMatrix, numLike-method (matrix-products), 106
tcrossprod, diagonalMatrix, CsparseMatrix-method (matrix-products), 106
tcrossprod, diagonalMatrix, indMatrix-method (matrix-products), 106
tcrossprod, diagonalMatrix, matrix-method (matrix-products), 106
tcrossprod, diagonalMatrix, missing-method (matrix-products), 106
tcrossprod, diagonalMatrix, TsparseMatrix-method (matrix-products), 106
INDEX

triu,denseMatrix-method (band), 9
triu,diagonalMatrix-method (band), 9
triu,indMatrix-method (band), 9
triu,RsparseMatrix-method (band), 9
triu,TsparseMatrix-method (band), 9
triu-methods (band), 9
TRUE, 48, 83, 158
TsparseMatrix-class, 168
type, 178
typeof, 42
uniqTsparse, 40, 41, 94, 101, 169
unname,Matrix,missing-method (Matrix-class), 104
unpack, 125, 171, 173
unpack,matrix-method (unpack), 171
unpack,packedMatrix-method (unpack), 171
unpack,sparseMatrix-method (unpack), 171
unpack,unpackedMatrix-method (unpack), 171
unpackedMatrix, 36, 125, 171
unpackedMatrix-class, 173
Unused-classes, 174
update, 20
update,CHMfactor-method (CHMfactor-class), 18
updown, 174
updown,ANY,ANY,ANY-method (updown), 174
updown,character,mMatrix,CHMfactor-method (updown), 174
updown,logical,mMatrix,CHMfactor-method (updown), 174
updown-methods (updown), 174
USCounties, 175
validObject, 33
warning, 98, 115
which, 44, 48, 91, 94, 111, 120
which,ldenseMatrix-method (ldenseMatrix-class), 91
which,ldiMatrix-method (ldiMatrix-class), 92
which,lgTMatrix-method (lsparseMatrix-classes), 93
which,lsparseMatrix-method (lsparseMatrix-classes), 93
which,lsparseVector-method (sparseVector-class), 160
which,lsTMatrix-method (lsparseMatrix-classes), 93
which,ltTMatrix-method (lsparseMatrix-classes), 93
which,ndenseMatrix-method (ndenseMatrix-class), 110
which,ngTMatrix-method (nsparseMatrix-classes), 119
which,nsparseMatrix-method (nsparseMatrix-classes), 119
which,nsparseVector-method (sparseVector-class), 160
which,nsTMatrix-method (nsparseMatrix-classes), 119
which,ntTMatrix-method (nsparseMatrix-classes), 119
writeMM, 156
writeMM(externalFormats), 67
writeMM,CsparseMatrix-method (externalFormats), 67
writeMM,sparseMatrix-method (externalFormats), 67
wrl_d1deg, 176
xspaseVector-class (sparseVector-class), 160
xtabs, 153, 156
zapsmall, 52, 117
zapsmall,dMatrix-method (dMatrix-class), 47
zMatrix-class (Unused-classes), 174
zsparseVector-class (sparseVector-class), 160