Package ‘Matrix’

January 6, 2021

Version 1.3-2
Date 2021-01-05
Priority recommended
Title Sparse and Dense Matrix Classes and Methods
Contact Doug and Martin <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 (>= 3.6.0)
Imports methods, graphics, grid, stats, utils, lattice
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/?group_id=61
NeedsCompilation yes
Author Douglas Bates [aut],
Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>),
Timothy A. Davis [ctb] (SuiteSparse and ‘cs’ C libraries, notably
CHOLMOD, AMD; collaborators listed in dir(pattern =
'\^[A-Z]+[.]txt$', full.names=TRUE, system.file('doc',

1
R topics documented:

'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 2021-01-06 17:40:19 UTC

R topics documented:

ablIndex-class ........................................... 4
ablseq ...................................................... 6
all-methods ................................................. 7
all.equal-methods ......................................... 7
atomicVector-class ....................................... 8
band ......................................................... 9
bandSparse ................................................. 10
bdiag ....................................................... 12
BunchKaufman-methods .................................... 14
CAex ........................................................ 15
cBind ......................................................... 16
CHMfactor-class ............................................ 17
chol ........................................................ 20
chol2inv-methods .......................................... 22
Cholesky ..................................................... 23
Cholesky-class ............................................. 26
colSums ...................................................... 27
compMatrix-class .......................................... 29
condest ....................................................... 30
CsparseMatrix-class ....................................... 32
ddenseMatrix-class ....................................... 33
ddiMatrix-class ........................................... 34
denseMatrix-class ........................................ 35
dgCMatrix-class .......................................... 36
dgMatrix-class ............................................. 37
dgRMatrix-class .......................................... 38
dgTMatrix-class .......................................... 39
Diagonal ..................................................... 41
diagonalMatrix-class ...................................... 42
diagU2N ..................................................... 44
dMatrix-class .............................................. 45
dpoMatrix-class ........................................... 46
drop0 ......................................................... 48
dsCMatrix-class ........................................... 49
dsparseMatrix-class ....................................... 51
dsRMatrix-class .......................................... 52
dsyMatrix-class .......................................... 53
R topics documented:

dtCMatrix-class ........................................... 54
dtpMatrix-class ........................................... 56
dtRMatrix-class ........................................... 58
dtrMatrix-class ........................................... 59
expand ..................................................... 60
expm ....................................................... 61
externalFormats .......................................... 62
facmul ..................................................... 64
forceSymmetric ........................................... 65
formatSparseM ............................................. 66
generalMatrix-class ...................................... 67
graph-sparseMatrix ....................................... 68
Hilbert ..................................................... 69
image-methods ............................................ 70
index-class ................................................ 72
indMatrix-class .......................................... 73
invPerm ..................................................... 75
is.na-methods ............................................. 76
is.null.DN ................................................ 78
isSymmetric-methods .................................... 79
isTriangular .............................................. 79
KhatriRao .................................................. 80
KNex ....................................................... 82
kronecker-methods ........................................ 83
ldenseMatrix-class ...................................... 84
ldiMatrix-class .......................................... 85
lgeMatrix-class .......................................... 85
lsparseMatrix-classes ................................... 86
lsyMatrix-class .......................................... 88
ltrMatrix-class .......................................... 89
lu .......................................................... 90
LU-class ..................................................... 93
mat2triplet ................................................ 94
Matrix ..................................................... 95
Matrix-class .............................................. 97
matrix-products ......................................... 99
MatrixClass ............................................... 101
MatrixFactorization-class ............................ 102
ndenseMatrix-class ..................................... 103
nearPD .................................................... 104
ngeMatrix-class ......................................... 107
nMatrix-class ........................................... 108
nnzero ..................................................... 109
norm ........................................................ 111
nsparseMatrix-classes ................................. 112
nsyMatrix-class ......................................... 114
ntrMatrix-class .......................................... 115
number-class ............................................. 116
abIndex-class

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 2:100000 or c(0:1e5,1000:1e6).

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

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

Slots

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

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

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

Methods

`as.numeric`, `as.integer`, `as.vector` signature(x = "abIndex"): ...

[ signature(x = "abIndex", i = "index", j = "ANY", drop = "ANY"): ...

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

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

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

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

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

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

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

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

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

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

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

Note

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

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

See Also

`rle` (base) which is used here; `numeric`
Examples

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

stopifnot(identical(-3:20,
    as(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**

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

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

**Arguments**

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

**Value**

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

**See Also**

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

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

try( ## (arithmetic) not yet implemented
    abIseq(1, 50, by = 3)
)
```

---

all-methods  "Matrix" Methods for Functions `all()` and `any()`

Description

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

Methods

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

Examples

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

---

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

Description

Methods for function `all.equal()` (from \texttt{R} package \texttt{base}) are defined for all \texttt{Matrix} classes.
atomicVector-class

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

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))))

---

atomicVector-class  Virtual Class "atomicVector" of Atomic Vectors

Description

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

Objects from the Class

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

Methods

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

Extends

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

Author(s)

Martin Maechler
See Also

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

Examples

```r
showClass("atomicVector")
```

---

**Description**

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

**Usage**

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

**Arguments**

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

**Value**

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

**Methods**

- `x = "CsparseMatrix"` method for compressed, sparse, column-oriented matrices.
- `x = "TsparseMatrix"` method for sparse matrices in triplet format.
- `x = "RsparseMatrix"` method for compressed, sparse, row-oriented matrices.
- `x = "ddenseMatrix"` method for dense numeric matrices, including packed numeric matrices.

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), nc = 5))
tril(m5)  # lower triangle
tril(m5, -1)  # strict lower triangle
triu(m5, 1)  # strict upper triangle
band(m5, -1, 2)  # general band
(m65 <- Matrix(rnorm(30), nc = 5))  # not square
triu(m65)  # result in not dtrMatrix unless square
(sm5 <- crossprod(m65))  # symmetric
band(sm5, -1, 1)  # symmetric band preserves symmetry property
as(band(sm5, -1, 1), "sparseMatrix")  # often preferable
```

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

Description

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

Usage

```r
bandSparse(n, m = n, k, diagonals, symmetric = FALSE,
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 \(d = length(k)\) 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`).
bandSparse 11

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

ea 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 *or* a matrix for specifying the diagonals:
stopifnot(identical(B, bandSparse(n, k = bk, diag = bMat)),
identical(Bs, bandSparse(n, k = bk, diag = bMat, symmetric=TRUE)),
inherits(B, "dtCMatrix") # triangular!
Description
Build a block diagonal matrix given several building block matrices.

Usage
\[
\text{bdiag}(...) \\
\text{.bdiag(lst)}
\]

Arguments
\[
... \quad \text{individual matrices or a list of matrices.} \\
lst \quad \text{non-empty list of matrices.}
\]

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

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

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))
```

```r
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))))
```

```r
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") )
```

```r
## 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 \code{"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))
  )
}
```

```r
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]
```
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 a generalization of a pivoting $LDL'$ Cholesky decomposition.

Usage

```r
## 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:

- `x = "dspMatrix"` routine dsptrf(); whereas
- `x = "dsyMatrix"`, and
- `x = "matrix"` use dsytrf().
CAex

References

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

See Also

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

Examples

data(CAex)
dim(CAex)
isSymmetric(CAex)# TRUE
CAs <- as(CAex, "symmetricMatrix")
if(FALSE) # no method defined yet for *sparse* :
  bk. <- BunchKaufman(CAs)
  ## does apply to *dense* symmetric matrices:
  bkCA <- BunchKaufman(as(CAs, "denseMatrix"))
  bkCA

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

---

CAex              Albers' example Matrix with "Difficult" Eigen Factorization

Description

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

Usage

data(CAex)

Format

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

Details

Historical note (2006-03-30): In earlier versions of R, eigen(CAex) fell into an infinite loop
whereas 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"))


Description

The base functions cbind and rbind are defined for an arbitrary number of arguments and hence have the first formal argument ... For that reason, in the past S4 methods could easily be defined for binding together matrices inheriting from Matrix.

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

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.

Usage

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

Arguments

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

Details

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

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

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

Author(s)

Martin Maechler

See Also

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

Examples

(a <- matrix(c(2:1,1:2), 2,2))
D <- Diagonal(2)
if(getRversion() < "3.2.0") {
  M1 <- cbind(0, rBind(a, 7))
  print(M1) # remains traditional matrix

  M2 <- cBind(4, a, D, -1, D, 0) # a sparse Matrix
  print(M2)
} else { ## newer versions of R do not need cBind / rBind:

  M1 <- cbind(0, suppressWarnings(rBind(a, 7)))
  print(M1) # remains traditional matrix

  M2 <- suppressWarnings(cBind(4, a, D, -1, D, 0)) # a sparse Matrix
  print(M2)

  stopifnot(identical(M1, cbind(0, rbind(a, 7))),
            identical(M2, cbind(4, a, D, -1, D, 0)))
}# R >= 3.2.0

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 \texttt{Cholesky}()).
parent  a "\texttt{dsCMatrix}" or "\texttt{dgCMatrix}" matrix object with the same nonzero pattern as
the matrix that generated object. If parent is symmetric, of class "\texttt{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 decom-
position of a matrix with the same nonzero pattern as tcrossprod(parent).
Since Matrix version 1.0-8, other "\texttt{sparseMatrix}" matrices are coerced to
\texttt{dsparseMatrix} and \texttt{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 \texttt{new("dCHMsuper",...)} but are more commonly created via \texttt{Cholesky}(), applied to \texttt{dsCMatrix} or \texttt{lsCMatrix} objects.
For an introduction, it may be helpful to look at the \texttt{expand()} method and examples below.

Slots

of "\texttt{CHMfactor}" and all classes inheriting from it:

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

Slots of the non virtual classes "\texttt{dl|CHM(supersimpl)}":

\texttt{p}: Object of class "\texttt{integer}" of pointers, one for each column, to the initial (zero-based) index of
elements in the column. Only present in classes that contain "\texttt{CHMsimpl}".
\texttt{i}: Object of class "\texttt{integer}" of length \texttt{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
"\texttt{CHMsimpl}".
\texttt{x}: For the "\texttt{d*}" classes: "\texttt{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 for x 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 logarithm = 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′ (= tcrossprod(parent)) is used for A. Further it provides an optional argument mult, a numeric scalar. This method updates the numeric values in object to the decomposition of A + mI where A is the matrix above (either the parent or MM′) and m is the scalar mult. Because only the numeric values are updated this method should be faster than creating and decomposing A + mI. It is not uncommon to want, say, the determinant of A + mI for many different values of m. This method would be the preferred approach in such cases.
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'M1 = M M’ where M <- t(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 ]

---

chol

**Choleski Decomposition - 'Matrix' S4 Generic and Methods**

### Description

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

### Usage

```r
chol(x, ...)
```

# S4 method for signature 'dsCMatrix'

```r
chol(x, pivot = FALSE, ...)
```

# S4 method for signature 'dsparseMatrix'

```r
chol(x, pivot = FALSE, cache = TRUE, ...)
```
chol

Arguments

x a (sparse or dense) square matrix, here inheriting from class Matrix; if x is not positive definite, an error is signalled.

pivot logical indicating if pivoting is to be used. Currently, this is not made use of for dense matrices.

cache logical indicating if the result should be cached in x@factors; note that this argument is experimental and only available for some sparse matrices.

... potentially further arguments passed to methods.

Details

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

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

Value

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

Methods

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

chol signature(x = "dgeMatrix"): works via "dpoMatrix", see class dpoMatrix.

chol signature(x = "dpoMatrix"): Returns (and stores) the Cholesky decomposition of x, via LAPACK routines dlacpy and dpotrf.

chol signature(x = "dppMatrix"): Returns (and stores) the Cholesky decomposition via LAPACK routine dpotrf.

chol signature(x = "dsCMatrix",pivot = "logical"): Returns (and stores) the Cholesky decomposition of x. If pivot is true, the Approximate Minimal Degree (AMD) algorithm is used to create a reordering of the rows and columns of x so as to reduce fill-in.

References


See Also

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

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

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

str(c2)

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

## A sparse example --- exemplifying 'pivot'

(mm <- toeplitz(as(c(10, 0, 1, 0, 3), "sparseVector"))) # 5 x 5
(R <- chol(mm)) ## default: pivot = FALSE
R2 <- chol(mm, pivot=FALSE)
stopifnot( identical(R, R2), all.equal(crossprod(R), mm) )

(R. <- chol(mm, pivot=TRUE))# nice band structure,

## but of course crossprod(R.) is *NOT* equal to mm
## --> see Cholesky() and its examples, for the pivot structure & factorization
stopifnot(all.equal(sqrt(det(mm)), det(R)),
          all.equal(prod(diag(R)), det(R)),
          all.equal(prod(diag(R.)), det(R)))

## a second, even sparser example:

(M2 <- toeplitz(as(c(1,.5, rep(0,12), -.1), "sparseVector")))
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
)

chol2inv-methods

Inverse from Choleski or QR Decomposition – Matrix Methods

Description

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

Even more generally, given an upper triangular matrix \(R\), compute \((R' R)^{-1}\).
Methods

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

See Also

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

Examples

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

---

**Description**

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

**Usage**

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

**Arguments**

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

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

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

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

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

**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 (inheriting from "Matrix") whereas `Cholesky()` returns a "CHMfactor" object, and hence a typical user will rather use `chol(A)`.

**Examples**

data(KNex)
```r
tmt <- with(KNex, crossprod(mm))
str(tmt@factors) # empty list()
(C1 <- Cholesky(tmt)) # uses show(<MatrixFactorization>)
str(tmt@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

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

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

## Visualize the sparseness:
dq <- function(ch) paste("",ch,"", sep="") ## dQuote(<UTF-8>) gives bad plots
image(mtm, main=paste("crossprod(mm) : 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.attr")$dimnames,
                 stringsAsFactors=FALSE,KEEP.OUT.ATTRS=FALSE))
names(rr) <- apply(nn, 1, function(r)
                paste(sub("(.=\")","","\"\", r), collapse="",))
str(rr, max=1)
str(re <- lapply(rr, expand), max=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

**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`).

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

See Also

Classes \( \text{dtrMatrix}, \text{dpoMatrix} \); function \( \text{chol} \).

Function \( \text{Cholesky} \) resulting in class \( \text{CHMfactor} \) objects, not class "Cholesky" ones, see the section 'Note'.

Examples

\[
\begin{align*}
\text{(sm} & \leftarrow \text{as(as(Matrix(diag(5) + 1), "dsyMatrix"), "dspMatrix"))) \\
\text{signif(csm} & \leftarrow \text{chol(sm), 4)}
\end{align*}
\]

\[
\begin{align*}
\text{(pm} & \leftarrow \text{crossprod(Matrix(rnorm(18), nrow} = 6, \text{ncol} = 3)))) \\
\text{(ch} & \leftarrow \text{chol(pm))}
\end{align*}
\]

\[
\begin{align*}
\text{if (toupper(ch@uplo) == "U")} & \# \text{which is TRUE} \\
\text{crossprod(ch)}
\end{align*}
\]

\[
\begin{align*}
\text{stopifnot(all.equal(as(crossprod(ch), "matrix"),} \\
\text{as(pm, "matrix"), tolerance=1e-14))}
\end{align*}
\]

\[
\begin{align*}
\text{colSums} & \quad \text{Form Row and Column Sums and Means}
\end{align*}
\]

Description

Form row and column sums and means for objects, for \( \text{sparseMatrix} \) the result may optionally be sparse (\( \text{sparseVector} \)), too. Row or column names are kept respectively as for \( \text{base} \) matrices and \( \text{colSums} \) methods, when the result is numeric vector.

Usage

\[
\begin{align*}
\text{colSums} \ (x, \text{na.rm} = \text{FALSE}, \text{dims} = 1, \ldots) \\
\text{rowSums} \ (x, \text{na.rm} = \text{FALSE}, \text{dims} = 1, \ldots) \\
\text{colMeans}(x, \text{na.rm} = \text{FALSE}, \text{dims} = 1, \ldots) \\
\text{rowMeans}(x, \text{na.rm} = \text{FALSE}, \text{dims} = 1, \ldots)
\end{align*}
\]

\[
\begin{align*}
\# \text{S4 method for signature 'CsparseMatrix'} \\
\text{colSums}(x, \text{na.rm} = \text{FALSE},} \\
\text{\text{dims} = 1, \text{sparseResult} = \text{FALSE})
\end{align*}
\]

\[
\begin{align*}
\# \text{S4 method for signature 'CsparseMatrix'} \\
\text{rowSums}(x, \text{na.rm} = \text{FALSE},} \\
\text{\text{dims} = 1, \text{sparseResult} = \text{FALSE})
\end{align*}
\]
## S4 method for signature 'CsparseMatrix'
colMeans(x, na.rm = FALSE, 
dims = 1, sparseResult = FALSE)
## S4 method for signature 'CsparseMatrix'
rowMeans(x, na.rm = FALSE, 
dims = 1, sparseResult = FALSE)

### Arguments

- **x** a Matrix, i.e., inheriting from `Matrix`.
- **na.rm** logical. Should missing values (including NaN) be omitted from the calculations?
- **dims** completely ignored by the `Matrix` methods.
- **...** potentially further arguments, for method `<->` generic compatibility.
- **sparseResult** logical indicating if the result should be sparse, i.e., inheriting from class `sparseVector`. 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) --&gt; 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 *automagically* 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, \( \|A\|_1 \), through \texttt{onenormest(.)}. 

Usage

\begin{verbatim}
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)
\end{verbatim}

Arguments

- \( A \) : a square matrix, optional for \texttt{onenormest()}, where instead of \( A \), \( A.x \) and \( At.x \) can be specified, see there.
- \( t \) : number of columns to use in the iterations.
- \( \text{normA} \) : number; (an estimate of) the 1-norm of \( A \), by default \( \text{norm}(A, "1") \); may be replaced by an estimate.
- \( \text{silent} \) : logical indicating if warning and (by default) convergence messages should be displayed.
- \( \text{quiet} \) : logical indicating if convergence messages should be displayed.
- \( A.x, At.x \) : when \( A \) is missing, these two must be given as functions which compute \( A \times x \), or \( t(A) \times x \), respectively.
- \( n \) : \( = \text{nrow}(A) \), only needed when \( A \) is not specified.
- \( \text{iter.max} \) : maximal number of iterations for the 1-norm estimator.
- \( \text{eps} \) : the relative change that is deemed irrelevant.

Details

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

Note that \texttt{onenormest()} uses random vectors and hence both functions’ results are random, i.e., depend on the random seed, see, e.g., \texttt{set.seed().}
Value

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

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

The function onenormest() returns a list with components,

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

Author(s)

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

References


See Also

norm, rcond.

Examples

data(KNex)
mtm <- with(KNex, crossprod(mm))
system.time(ce <- condest(mtm))
sum(abs(ce$v))  ## || v ||_1 == 1
## Prove that || A v || = || A || / est (as ||v|| = 1):
stopifnot(all.equal(norm(mtm %*% ce$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 approxmation 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  Class "CsparseMatrix" of Sparse Matrices in Column-compressed Form

Description

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

Slots

i: Object of class "integer" of length nnzero (number of non-zero elements). These are the 0-based row numbers for each non-zero element in the matrix, i.e., i must be in 0:(nrow(.)-1).

p: integer vector for providing pointers, one for each column, to the initial (zero-based) index of elements in the column. .@p is of length ncol(.) + 1, with p[1] == 0 and p[length(p)] == nnzero, such that in fact, diff(.@p) are the number of non-zero elements for each column.

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

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

Extends

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

Methods

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

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"): ...
coerce signature(from = "TsparseMatrix", to = "CsparseMatrix"): ...
coerce signature(from = "denseMatrix", to = "CsparseMatrix"): ...
diag signature(x = "CsparseMatrix"): ...
gamma signature(x = "CsparseMatrix"): ...
lgamma signature(x = "CsparseMatrix"): ...
log signature(x = "CsparseMatrix"): ...
t signature(x = "CsparseMatrix"): ...
tril signature(x = "CsparseMatrix"): ...
triu signature(x = "CsparseMatrix"): ...

Note

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

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

See Also

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

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

Examples

gtclass("CsparseMatrix")

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

---

**ddenseMatrix-class**

Virtual Class "ddenseMatrix" of Numeric Dense Matrices

**Description**

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

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

Description

The class "ddiMatrix" of numerical diagonal matrices.

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

Objects from the Class

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

Slots

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

Extends

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

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

See Also

Class \texttt{diagonalMatrix} and function \texttt{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)))


Description

This is the virtual class of all dense (S4) matrices. It is the direct superclass of \texttt{ddenseMatrix}, \texttt{ldenseMatrix}.

Extends

class "Matrix" directly.

Slots

exactly those of its superclass "Matrix".

Methods

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

See Also

\texttt{colSums}, \texttt{kronecker}, and other such methods with own help pages.
Its superclass \texttt{Matrix}, and main subclasses, \texttt{ddenseMatrix} and \texttt{sparseMatrix}.
Examples

    showClass("denseMatrix")

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

Description

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

Objects from the Class

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

Slots

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

  ... all other slots are inherited from the superclass "CsparseMatrix".

Methods

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

  coerce  signature(from = "matrix",to = "dgCMatrix")
  coerce  signature(from = "dgCMatrix",to = "matrix")
  coerce  signature(from = "dgCMatrix",to = "dgTMatrix")

  diag  signature(x = "dgCMatrix"): returns the diagonal of x

  dim  signature(x = "dgCMatrix"): returns the dimensions of x

  image  signature(x = "dgCMatrix"): plots an image of x using the levelplot function

  solve  signature(a = "dgCMatrix",b = "..."): see solve-methods, notably the extra argument sparse.

  lu  signature(x = "dgCMatrix"): computes the LU decomposition of a square dgCMatrix object

See Also

  Classes dsCMatrix, dtCMatrix, lu

Examples

  (m <- Matrix(c(0,0,2:0), 3,5))
  str(m)
  m[,1]
Class "dgeMatrix" of Dense Numeric (S4 Class) Matrices

Description

A general numeric dense matrix in the S4 Matrix representation. dgeMatrix is the “standard” class for dense numeric matrices in the Matrix package.

Objects from the Class

Objects can be created by calls of the form new("dgeMatrix",...) or, more commonly, by coercion from the Matrix class (see Matrix) or by Matrix(.).

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"): ...
**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.
dgTMatrix-class

Methods

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

See Also

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

dgTMatrix-class

Sparse matrices in triplet form

Description

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

Objects from the Class

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

Slots

- **i**: integer row indices of non-zero entries in 0-based, 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 \(0:(nrow(.)-1)\).
- **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 \( x_{ik} \)'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.

**See Also**

Class *dgCMatrix* or the superclasses *dsparseMatrix* and *TsparseMatrix*; *uniqTsparse*.

**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, "dgTMatrix"))
str(mT)
mT[1,]
mT[4, drop = FALSE]
stopifnot(identical(mT[lower.tri(mT)], m[lower.tri(m) ]))
mT[lower.tri(mT,diag=TRUE)] <- 0
mT

## Triplet representation with repeated (i,j) entries
## *adds* the corresponding x's:
T2 <- new("dgTMatrix",
    i = as.integer(c(1,1,0,3,3)),
    j = as.integer(c(2,2,4,0,0)), x=10*1:5, Dim=4:5)
str(T2) # contains (i,j,x) slots exactly as above, but
T2 ## has only three non-zero entries, as for repeated (i,j)'s,
    ## the corresponding x's are "implicitly" added
stopifnot(nnzero(T2) == 3)
```
Diagonal

Create Diagonal Matrix Object

Description

Create a diagonal matrix object, i.e., an object inheriting from diagonalMatrix (or a “standard" CsparseMatrix diagonal matrix in cases that is prefered).

Usage

Diagonal(n, x = NULL)

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

Arguments

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

x
  numeric or logical; if missing, a unit diagonal n × n matrix is created.

uplo
  for . symDiagonal (. trDiagonal), the resulting sparse symmetricMatrix (or triangularMatrix) will have slot uplo set from this argument, either "U" or "L". Only rarely will it make sense to change this from the default.

shape
  string of 1 character, one of c("t", "s", "g"), to choose a triangular, symmetric or general result matrix.

unitri
  optional logical indicating if a triangular result should be “unit-triangular", i.e., with diag = "U" slot, if possible. The default, missing, is the same as TRUE.

kind
  string of 1 character, one of c("d", "l", "n"), to choose the storage mode of the result, from classes dsparseMatrix, lsparseMatrix, or nsparseMatrix, respectively.

cols
  integer vector with values from 0:(n-1), denoting the columns to subselect conceptually, i.e., get the equivalent of Diagonal(n,*)[,cols + 1].

Value

Diagonal() returns an object of class ddiMatrix or ldiMatrix (with “ superclass” diagonalMatrix).

.symDiagonal() returns an object of class dsCMatrix or lsCMatrix, i.e., a sparse symmetric matrix. Analogously, . trDiagonal gives a sparse triangularMatrix. This can be more efficient than Diagonal(n) when the result is combined with further symmetric (sparse) matrices, e.g., in kronecker, however not for matrix multiplications where Diagonal() is clearly preferred.

.sparseDiagonal(), the workhorse of . symDiagonal and . trDiagonal returns a CsparseMatrix (the resulting class depending on shape and kind) representation of Diagonal(n), or, when cols are specified, of Diagonal(n)[,cols+1].
diagonalMatrix-class

Author(s)

Martin Maechler

See Also

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

Examples

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

## Use Diagonal() + kronecker() for "repeated-block" matrices:
M1 <- Matrix(0+0:5, 2,3)
(M <- kronecker(Diagonal(3), M1))

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

diagonalMatrix-class   Class "diagonalMatrix" of Diagonal Matrices

Description

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

Objects from the Class

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

Slots

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

Extends

Class "sparseMatrix", directly.
Methods

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

coerce signature(from = "matrix", to = "diagonalMatrix"): ...
coerce signature(from = "Matrix", to = "diagonalMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "generalMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "triangularMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "nMatrix"): ...
coerce signature(from = "diagonalMatrix", to = "matrix"): ...
coerce signature(from = "diagonalMatrix", to = "sparseVector"): ...
t signature(x = "diagonalMatrix"): ...
  and many more methods
solve signature(a = "diagonalMatrix", b, ...): is trivially implemented, of course; see also solve-methods.
which signature(x = "nMatrix"): semantically equivalent to base function which(x, arr.ind).
"Math" signature(x = "diagonalMatrix"): all these group methods return a "diagonalMatrix", apart from cumsum() etc which return a vector also for base matrix.
* signature(e1 = "ddiMatrix", e2="denseMatrix"): arithmetic and other operators from the Ops group have a few dozen explicit method definitions, in order to keep the results diagonal in many cases, including the following:
  / signature(e1 = "ddiMatrix", e2="denseMatrix"): the result is from class ddiMatrix which is typically very desirable. Note that when e2 contains off-diagonal zeros or NAs, we implicitly use 0/x = 0, hence differing from traditional R arithmetic (where 0/0 \rightarrow \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

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

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

Usage

\[
\text{diagN2U}(x, \text{cl} = \text{getClassDef}(\text{class}(x)), \text{checkDense} = \text{FALSE})
\]

\[
\text{diagU2N}(x, \text{cl} = \text{getClassDef}(\text{class}(x)), \text{checkDense} = \text{FALSE})
\]

.diagU2N(x, cl, checkDense = FALSE)

Arguments

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

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

Value

a triangular matrix of the same class but with a different diag slot. For diagU2N (semantically) with identical entries as x, whereas in 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 internal utilities.
dMatrix-class

See Also

"triangularMatrix", "dtCMatrix".

Examples

(T <- Diagonal(7) + triu(Matrix(rpois(49, 1/4), 7,7), k = 1))
(uT <- diag(N2U(T))) # "unitriangular"
(t.u <- diag(N2U(10*T)) # changes the diagonal!
stopifnot(all(T == uT), diag(t.u) == 1,
identical(T, diagU2N(uT)))
T[upper.tri(T)] <- 5
T <- diag(N2U(as(T, "triangularMatrix"))
stopifnot(T@diag == "U")
dT <- as(T, "denseMatrix")
dt <- diag(N2U(dT))
dtU <- diag(N2U(dT, checkDense=TRUE)
stopifnot(is(dtU, "denseMatrix"), is(dt., "sparseMatrix"),
all(dT == dt.), all(dT == dtU),
dt.@diag == "U", dtU@diag == "U")

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 \texttt{max()}, \texttt{min()}, \texttt{range()}, \texttt{prod()}, \texttt{sum()}, \texttt{any()} and \texttt{all()}.

The following methods are also defined for all double matrices:

\texttt{coerce signature(from = "dMatrix", to = "matrix"): ...}
\texttt{expm signature(x = "dMatrix"): computes the "Matrix Exponential", see \texttt{expm}.}
\texttt{zapsmall signature(x = "dMatrix"): ...}

The following methods are defined for all logical matrices:

\texttt{which signature(x = "lsparseMatrix") and many other subclasses of "lMatrix": as the base function \texttt{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 \texttt{useNames} is true, as by default, with \texttt{dimnames}, the same as base::which.}

\textbf{See Also}

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

The numeric matrix classes \texttt{dgeMatrix}, \texttt{dgCMatrix}, and \texttt{Matrix}.
\texttt{drop0(x, tol=1e-10)} is sometimes preferable to (and more efficient than) \texttt{zapsmall(x, digits=10)}.

\textbf{Examples}

\begin{verbatim}
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))
\end{verbatim}

\section*{dpoMatrix-class}

\textit{Positive Semi-definite Dense (Packed \& Non-packed) Numeric Matrices}

\section*{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"
- `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) $\sqrt{\text{var}(\cdot)}$ 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 anymore. The only exceptions, currently, are multiplications, divisions or additions with positive length(.) == 1 numbers (or `logicals`).

See Also

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

```r
h6 <- Hilbert(6)
rcond(h6)
str(h6)
```

```r
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 `spMatrix`.
- `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 `CspMatrix`.

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 `spMatrix`; `nnzero`
Examples

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

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

### dsCMatrix-class

**Numeric Symmetric Sparse (column compressed) Matrices**

#### Description

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

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

#### Objects from the Class

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

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

#### Slots

- **uplo**: A character object indicating if the upper triangle ("U") or the lower triangle ("L") is stored.
- **i**: Object of class "integer" of length nnZ (half number of non-zero elements). These are the row numbers for each non-zero element in the lower triangle of the matrix.
- **p**: (only in class "dsCMatrix"): an integer vector for providing pointers, one for each column, see the detailed description in `CsparseMatrix`.
- **j**: (only in class "dsTMatrix"): Object of class "integer" of length nnZ (as i). These are the column numbers for each non-zero element in the lower triangle of the matrix.
- **x**: Object of class "numeric" of length nnZ – the non-zero elements of the matrix (to be duplicated for full matrix).
factors: Object of class "list" - a list of factorizations of the matrix.

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

Extends

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

Methods

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

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

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

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

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

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

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

coerce signature(from = "dsCMatx", to = "dgTMatrix")

coerce signature(from = "dsCMatx", to = "dgeMatrix")

coerce signature(from = "dsCMatx", to = "matrix")

coerce signature(from = "dsTMatrix", to = "dgeMatrix")

coerce signature(from = "dsTMatrix", to = "dsCMatx")

coerce signature(from = "dsTMatrix", to = "dsyMatrix")

coerce signature(from = "dsTMatrix", to = "matrix")

See Also

Classes dgCMatx, dgTMatrix, dgeMatrix and those mentioned above.

Examples

mm <- Matrix(toeplitz(c(10, 0, 1, 0, 3)), sparse = TRUE)

mm # automatically dsCMatx

str(mm)
## how would we go from a manually constructed Tsparse* :

```r
mT <- as(mm, "dgTMatrix")
```

## Either

```r
(symM <- as(mT, "symmetricMatrix"))# dsT
(symC <- as(symM, "CsparseMatrix"))# dsC
```

## or

```r
sC <- Matrix(mT, sparse=TRUE, forceCheck=TRUE)
```

```r
sym2 <- as(symC, "TsparseMatrix")
```

## --> the same as 'symM', a "dsTMatrix"

---

### 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

```r
showClass("dsparseMatrix")
```
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

(m0 <- new("dsRMMatrix"))
m2 <- new("dsRMMatrix", 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 \ Non-packed) Numeric Matrices

Description

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

Objects from the Class

Objects can be created by calls of the form new("dsyMatrix",...) 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 <- 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 dpoMatrix, dppMatrix and corMatrix.
Classes dgeMatrix and Matrix; solve, norm, rcond, t

Examples

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

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

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

---

**dtCMatrix-class**

Triangular, (compressed) sparse column matrices

**Description**

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

The "dtTMatrix" class is a class of triangular, sparse matrices in triplet format.
Objects from the Class

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

Slots

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

Extends

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

Methods

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

See Also

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

```r
showClass("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, "dtCMatrix"))
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,0L:2, 5L), Dim = c(5L, 5L),
       x = rep(1, 5), diag = "U")
U5
(iu <- solve(U5)) # contains one '/quotesingle.Var 0/quotesingle.Var'
validObject(iu2 <- solve(U5, Diagonal(5)))# failed in earlier versions
I5 <- iu %*% U5 # should equal the identity matrix
i5 <- iu2 %*% U5
m53 <- matrix(1:15, 5,3, dimnames=list(NULL,letters[1:3]))
asDiag <- function(M) as(drop0(M), "diagonalMatrix")
stopifnot(
         all.equal(Diagonal(5), asDiag(I5), tolerance=1e-14) ,
         all.equal(Diagonal(5), asDiag(i5), tolerance=1e-14) ,
         identical(list(NULL, dimnames(m53)[[2]]), dimnames(solve(U5, m53)))
)
```

dtpMatrix-class

```
Packed Triangular Dense Matrices - "dtpMatrix"
```

Description

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

Objects from the Class

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

- **uplo**: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular.
- **diag**: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N"; see `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 $d \times d$, $\text{length}(x)$ 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 \( \prod(\text{diag}(x)) \), 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 forward-solve, see `solve-methods`.
- `t` signature(x = "dtpMatrix"): t(x) remains a "dtpMatrix", lower triangular if x is upper triangular, and vice versa.

**See Also**

Class `dtrMatrix`

**Examples**

```r
showClass("dtrMatrix")
exa(mple("dtrMatrix-class", echo=FALSE)
(p1 <- as(T2, "dtpMatrix"))
str(p1)
(pp <- as(T, "dtpMatrix"))
ip1 <- solve(p1)
stopifnot(length(p1@x) == 3, length(pp@x) == 3,
```
p1 @ uplo == T2 @ uplo, pp @ uplo == T @ uplo,
identical(t(pp), p1), identical(t(p1), pp),
all((l.d <- p1 - T2) == 0), is(l.d, "dtpMatrix"),
all((u.d <- pp - T ) == 0), is(u.d, "dtpMatrix"),
l.d@uplo == T2@uplo, u.d@uplo == T@uplo,
identical(t(ip1), solve(pp)), is(ip1, "dtpMatrix"),
all.equal(as(solve(p1,p1), "diagonalMatrix"), Diagonal(2)))

---

dtRMMatrix-class

**Description**

The dtRMMatrix 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("dtRMMatrix",...)`.

**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 dsRMMatrix class.)
- **x**: Object of class "numeric" - the non-zero elements of the matrix.
- **Dim**: The dimension (a length-2 "integer")
- **Dimnames**: corresponding names (or NULL), inherited from the Matrix, see there.

**Extends**

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

**Methods**

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

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

Examples

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

---

dtrMatrix-class

Triangular, dense, numeric matrices

Description

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

Objects from the Class

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

Slots

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

Extends

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

Methods

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

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


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

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

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

rccond signature(x = "dtrMatrix", norm = "character")

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

+,-,\*,\ldots,==,\geq,\ldots 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, "dgeMatrix"))

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

---

expand

Expand a (Matrix) Decomposition into Factors

Description

Expands decompositions stored in compact form into factors.

Usage

expand(x, ...)

Arguments

x a matrix decomposition.

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

Details

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

Value

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

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

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

Examples

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

expm

Matrix Exponential

Description
Compute the exponential of a matrix.

Usage

```r
expm(x)
```

Arguments

- `x` a matrix, typically inheriting from the dMatrix class.

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

Value
The matrix exponential of x.

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

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

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


See Also

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

Examples

```r
(m1 <- Matrix(c(1,0,1,1), nc = 2))
(e1 <- expm(m1)) ; e <- exp(1)
stopifnot(all.equal(e1@x, c(e,0,e,e), tolerance = 1e-15))
(m2 <- Matrix(c(-49, -64, 24, 31), nc = 2))
(e2 <- expm(m2))
(m3 <- Matrix(cbind(0,rbind(6*diag(3),0))))# sparse!
(e3 <- expm(m3)) # upper triangular
```

---

**externalFormats**

*Read and write external matrix formats*

**Description**

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

**Usage**

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

**Arguments**

<table>
<thead>
<tr>
<th>obj</th>
<th>a real sparse matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>for <code>writeMM</code> - the name of the file to be written. For <code>readHB</code> and <code>readMM</code> 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 &quot;rua&quot; or &quot;rsa&quot;. Those storing matrices in the MatrixMarket format usually end in &quot;mtx&quot;. Alternatively, <code>readHB</code> and <code>readMM</code> accept connection objects.</td>
</tr>
<tr>
<td>...</td>
<td>optional additional arguments. Currently none are used in any methods.</td>
</tr>
</tbody>
</table>
The `readHB` and `readMM` functions return an object that inherits from the "Matrix" class. Methods for the `writeMM` generic functions usually return `NULL` and, as a side effect, the matrix obj is written to file in the MatrixMarket format (writeMM).

**Note**

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

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

**References**

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

**Examples**

```r
str(pores <- readMM(system.file("external/pores_1.mtx", package = "Matrix")))
str(utm <- readHB(system.file("external/utm300.rua", package = "Matrix")))
str(lundA <- readMM(system.file("external/lund_a.mtx", package = "Matrix")))
str(lundA <- readHB(system.file("external/lund_a.rsa", package = "Matrix")))
```

## Not run:
```
## NOTE: The following examples take quite some time
## ---- even on a fast internet connection:
## if(FALSE) # the URL has been corrected, but we need an un-tar step!
str(sm <-
  readHB(gzcon(url("https://www.cise.ufl.edu/research/sparse/RB/Boeing/msc00726.tar.gz"))))
```

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

## End(Not run)
```
data(KNex)
## 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")]
```

# very simple export - in triplet format - to text file:
```
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()`:
facmul

**Multiplication by Decomposition Factors**

**Description**

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

**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**

Examples

```r
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

```
Force a Matrix to 'symmetricMatrix' Without Symmetry Checks
```

Description

Force a square matrix \texttt{x} to a \texttt{symmetricMatrix}, \textbf{without} a symmetry check as it would be applied for \texttt{as(x,"symmetricMatrix")}.

Usage

```
forceSymmetric(x, uplo)
```

Arguments

\texttt{x} any square matrix (of numbers), either ""traditional"" (\texttt{matrix}) or inheriting from \texttt{Matrix}.

\texttt{uplo} optional string, "U" or "L" indicating which "triangle" half of \texttt{x} should determine the result. The default is "U" unless \texttt{x} already has a \texttt{uplo} slot (i.e., when it is \texttt{symmetricMatrix}, or \texttt{triangularMatrix}), where the default will be \texttt{x@uplo}.

Value

a square matrix inheriting from class \texttt{symmetricMatrix}.

See Also

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

Examples

```
## 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)

---

### Description

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

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

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

### Usage

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

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

### Arguments

- **x**: an R object inheriting from class `sparseMatrix`.
- **zero.print**: character which should be used for structural zeroes. The default "." may occasionally be replaced by " " (blank); using "0" would look almost like `print()`ing of non-sparse matrices.
- **align**: a string specifying how the zero.print codes should be aligned, see `formatSpMatrix`.
- **m**: (optional) a (standard R) matrix version of x.
- **asLogical**: should the matrix be formatted as a logical matrix (or rather as a numeric one); mostly for `formatSparseM()`.
- **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").
- **digits**: significant digits to use for printing, see `print.default`.
generalMatrix-class

- **cx** (optional) character matrix; a formatted version of x, still with strings such as "0.00" for the zeros.
- **IN0** (optional) integer vector, specifying the location of the non-zeros of x.
- **col.names, note.dropping.colnames** see formatSpMatrix.
- **dn** dimnames to be used; a list (of length two) with row and column names (or NULL).

**Value**

A character matrix like cx, where the zeros have been replaced with (padded versions of) 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, 11,0,0,0,0,-7,0), 4,9))
fm <- formatSparseM(m)
noquote(fm)
## nice, but this is nicer (with "units" vertically aligned):
print(fm, quote=FALSE, right=TRUE)
## and "the same" as :
Matrix(m)

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

---

**generalMatrix-class**

**Class "generalMatrix" of General Matrices**

**Description**

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

**Objects from the Class**

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

 factors ,
 Dim ,
 Dimnames: all slots inherited from compMatrix; see its description.

 Extends

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

 See Also

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

 graph-sparseMatrix  Conversions "graph" <- (sparse) Matrix

 Description

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

 as(from, Class)

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

 Usage

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

 Arguments

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

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

See Also

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

Examples

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

  show(a2 <- sparseMatrix(i= c(1:4,4), j=c(2:4,1:2), x = TRUE, dimnames=dns))
  show(g2 <- as(a2, "graph")) # directed
  # now if you want it undirected:
  show(g3 <- T2graph(as(a2,"TsparseMatrix"), edgemode="undirected"))
  show(m3 <- as(g3,"Matrix"))
  show( graph2T(g3) ) # a "pattern Matrix" (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.
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”: \( y \text{lim} \) is for the rows (= 1st index) and \( x \text{lim} \) for the columns (= 2nd index). For convenience, when the limits are integer valued, they are both extended by 0.5; also, \( y \text{lim} \) 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 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).
**image-methods**

useAbs  logical indicating if abs(x) should be shown; if TRUE, the former (implicit) default, the default col.regions will be grey colors (and no colorkey drawn). The default is FALSE unless the matrix has no negative entries.

colorkey  logical indicating if a color key aka ‘legend’ should be produced. Default is to draw one, unless useAbs is true. You can also specify a list, see levelplot, such as list(raster=TRUE) in the case of rastering.

col.regions  vector of gradually varying colors; see levelplot.

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

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")
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(doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA")) ||
   identical("true", unname(Sys.getenv("R_PKG_CHECKING_doExtras")))) {
   ## Using raster graphics: For PDF this would give a 77 MB file,
   ## however, for such a large matrix, this is typically considerably
   ## *slower* (than vector graphics rectangles) in most cases :
   if(doPNG <- !dev.interactive())
     png("image-USCounties-raster.png", width=3200, height=3200)
   image(USCounties, useRaster = TRUE) # should not suffer from anti-aliasing
   if(doPNG)
     dev.off()
   ## and now look at the *.png image in a viewer you can easily zoom in and out
}#only if(doExtras)

index-class

Virtual Class "index" - Simple Class for Matrix Indices

Description

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

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

Objects from the Class

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

See Also
[-methods, and
Subassign-methods, also for examples.

Examples

showClass("index")

Description

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

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

Details

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

Permutation matrices (of class 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 drop=FALSE) stays within the "indMatrix" class,
all other subsetting/indexing operations ("column-indexing", including diag) on "indMatrix" objects treats them as nonzero-pattern matrices (ngTMatrix specifically), such that non-matrix
subsetting results in 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 new("indMatrix",...) or by coercion from an integer
index vector, see below.

Slots

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

Dim: 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.

class 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, ...

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

class 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")
(sm1 <- as(rep(c(2,3,1), e=3), "indMatrix"))
stopifnot(all(sm1 == 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")

as(sm1, "ngTMatrix")
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 <- suppressWarnings(rBind(I1, I2)))
stopifnot(is(I12, "indMatrix"),
          if(getRversion() >= "3.2.0") identical(I12, rbind(I1, I2)) else TRUE,
          colSums(I12) == c(2L,2:4,4:2))
```

---

**invPerm**  

*Inverse Permutation Vector*

Description

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

Usage

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

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

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 \texttt{order(p)} or \texttt{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, \texttt{pMatrix}.

Examples

\begin{verbatim}
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)
)
\end{verbatim}

\---

\textbf{is.na-methods} \hspace{1cm} \textit{is.na()}, \textit{is.infinite()} Methods for 'Matrix' Objects

Description

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

\begin{itemize}
\item \texttt{x = "denseMatrix"} returns a "nMatrix" object of same dimension as x, with TRUE's whenever x is \texttt{NA}, finite, or infinite, respectively.
\item \texttt{x = "sparseMatrix"} ditto.
\end{itemize}
**Usage**

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

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

**Arguments**

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

**See Also**

`NA`, `is.na`, `is.finite`, `is.infinite`; `nMatrix`, `denseMatrix`, `sparseMatrix`.

The `sparseVector` class.

**Examples**

```r
M <- Matrix(1:6, nrow=4, ncol=3,
dimnames = list(c("a", "b", "c", "d"), c("A", "B", "C")))
stopifnot(all(!is.na(M)))
M[2:3,2] <- NA
is.na(M)
if(exists("anyNA", mode="function"))
anyNA(M)

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

inA <- is.na(A)
stopifnot(sum(inA) == 1+1+5)
```
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 dimnames dn are “like” NULL, or list(NULL,NULL), as they can easily be for the traditional R matrices (matrix) which have no formal class definition, and hence much freedom in how their dimnames look like.

Usage

is.null.DN(dn)

Arguments

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

Value

logical TRUE or FALSE.

Note

This function is really to be used on “traditional” matrices rather than those inheriting from Matrix, as the latter will always have dimnames list(NULL,NULL) exactly, in such a case.

Author(s)

Martin Maechler

See Also

is.null, dimnames, matrix.

Examples

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(capture.output(m) -> cm,
    capture.output(m1)),
  identical(capture.output(m) -> cm,
    capture.output(m1)),
isSymmetric-methods

Methods for Function isSymmetric in Package 'Matrix'

Description

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

See Also

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

Examples

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

## Look at implementations:
showMethods("isSymmetric", includeDefs=TRUE)# "ANY": base's S3 generic; 6 more

isTriangular

isTriangular() and isDiagonal() Methods

Description

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

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

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

Usage

isDiagonal(object)

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

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

Value

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

See Also

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

Examples

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

isDiagonal(matrix(c(2,0,0,1), 2,2)) # TRUE

KhatriRao

Khatri-Rao Matrix Product

Description

Computes Khatri-Rao products for any kind of matrices.

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

Usage

KhatriRao(X, Y = X, FUN = "+", 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.
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 \( X (n \times k) \) and \( Y (m \times k) \), is of dimension \((n \cdot m) \times k\), where the \( j \)-th column, \( R[,j] \) is the kronecker product \( \text{kronecker}(X[,j],Y[,j]) \).

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))

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

stopifnot(dim(KhatriRao(m,d)) == c(nrow(m)*nrow(d), ncol(d)))
## border cases / checks:
```

---

### Value

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

### 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))

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

stopifnot(dim(KhatriRao(m,d)) == c(nrow(m)*nrow(d), ncol(d)))
## border cases / checks:
```
zm <- nm; zm[] <- 0 # all 0 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, m) == 0), identical(dim(K3), dim(Kdm)),
  all(K4 <- KhatriRao(m, d0) == 0), all(KhatriRao(m0, d0) == 0),
  all(KhatriRao(d0, m0) == 0), all(KhatriRao(m0, m0) == 0),
  identical(dimnames(KhatriRao(m, d0, make.dimnames=TRUE)), dimnames(Kmd)))

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

data(KNex)

References


Examples

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

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

head(round(sparse.sol,3))

## Compare with QR-based solution ("more accurate, but slightly slower"):

system.time(
  sp.sol2 <- with(KNex, qr.coef(qr(mm), y))
)

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

Computes Kronecker products for objects inheriting from "Matrix".

In order to preserve sparsity, we treat $0 \times \text{NA}$ as $0$, not as $\text{NA}$ as usually in R (and as used for the base function `kronecker`).

Methods

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

Examples

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

## should also work with special-cased logical matrices
l3 <- upper.tri(matrix(,3,3))
M <- Matrix(l3)
(N <- as(M, "nsparseMatrix")(upper triangular)
N2 <- as(N, "generalMatrix") # (lost "t"riangularity)
MM <- kronecker(M,M)
NN <- kronecker(N,N) # "dTMatrix" i.e. did keep
NN2 <- kronecker(N2,N2)
stopifnot(identical(NN, MM),
          is(NN2, "sparseMatrix"), all(NN2 == NN),
          is(NN, "triangularMatrix"))
```
IdenseMatrix-class

Virtual Class "ldenseMatrix" of Dense Logical Matrices

Description

IdenseMatrix 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"): ...

class 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.
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)
(IM <- Matrix(matrix(rnorm(28), 4,7) > 0))# a simple random lgeMatrix
set.seed(11)
(IC <- Matrix(matrix(rnorm(28), 4,7) > 0))# a simple random lgCMatrix
as(IM, "lgCMatrix")

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 (TsparseMatrix) matrices such as lgTMatrix may contain duplicated pairs of indices \((i, j)\) as for the corresponding numeric class dgTMatrix where for such pairs, the corresponding \(x\) slot entries are added. For logical matrices, the \(x\) entries corresponding to duplicated index pairs \((i, j)\) are "added" as well if the addition is defined as logical or, i.e., "TRUE + TRUE |\to| TRUE" and "TRUE + FALSE |\to| TRUE". Note the use of 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 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 \(x \neq 0\).

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

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

Methods

- **coerce** signature(from = "dgCMatrix",to = "lgCMatrix")
- **t** signature(x = "lgCMatrix"): returns the transpose of x
- **which** signature(x = "lsparseMatrix"), semantically equivalent to base function which(x, arr.ind); for details, see the IMatrix class documentation.
lsyMatrix-class

See Also

the class dgCMatrix and dgTMatrix

Examples

```
(m <- Matrix(c(0,0,2:0), 3,5, dimnames=list(LETTERS[1:3],NULL)))
(lm <- (m > 1))  # lgC
!lm  # no longer sparse
stopifnot(is(lm,"lsparseMatrix"),
          identical(!lm, m <= 1))
```

data(KNex)
```
str(mmG.1 <- (KNex $ mm) > 0.1)# "lgC..."
table(mmG.1@x)# however with many `<non-structural zeros'>
```

## from logical to nz_pattern -- okay when there are no NA's :

```
mmG.1 <- as(mmG.1, "nMatrix") # <<< has "TRUE" also where mmG.1 had FALSE
```

## from logical to "double"
```
dmG.1 <- as(mmG.1, "dMatrix")# has '0' and back:
lmG.1 <- as(dmG.1, "lMatrix")# has no extra FALSE, i.e. drop0() included
stopifnot(identical(mmG.1, as((KNex $ mm) != 0,"nMatrix")),
          validObject(lmG.1), all(lmG.1@x),
          # same "logical" but lmG.1 has no 'FALSE' in x slot:
          all(lmG.1 == mmG.1))
```

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

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

Extends

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

Methods

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

See Also

lgeMatrix, Matrix, t

Examples

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

ltrMatrix-class  Triangular Dense Logical Matrices

Description

The "ltrMatrix" class is the class of triangular, dense, logical matrices in nonpacked storage. The "ltmMatrix" 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="ltrMatrix") for details.

See Also

Classes lgeMatrix, Matrix; function t

Examples

showClass("ltrMatrix")

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

---

lu 

(Generalized) Triangular Decomposition of a Matrix

Description

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

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

Arguments

- **x**: a dense or sparse matrix, in the latter case of square dimension. No missing values or IEEE special values are allowed.
- **warnSing**: (when `x` is a "denseMatrix") logical specifying if a warning should be signalled when `x` is singular.
- **errSing**: (when `x` is a "sparseMatrix") logical specifying if an error (see `stop`) should be signalled when `x` is singular. When `x` is singular, `lu(x, errSing=FALSE)` returns `NA` instead of an LU decomposition. No warning is signalled and the useR should be careful in that case.
- **order**: logical or integer, used to choose which fill-reducing permutation technique will be used internally. Do not change unless you know what you are doing.
- **tol**: positive number indicating the pivoting tolerance used in `cs_lu`. Do only change with much care.
- **keep.dimnames**: logical indicating that `dimnames` should be propagated to the result, i.e., “kept”. This was hardcoded to `FALSE` in upto `Matrix` version 1.2-0. Setting to `FALSE` may gain some performance.
- **...**: further arguments passed to or from other methods.

Details

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

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

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

Value

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

## permute rows and columns of original matrix
pm <- pm[pmLU@p + 1L, pmLU@q + 1L]
plU <- drop0(pmLU@U); pmLU@U # L %*% U -- dropping extra zeros

## equal up to "rounding"
plU[1:4, 1:5]
plLU[1:4, 1:5]
```
**LU-class**

**LU (dense) Matrix Decompositions**

**Description**

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

**Details**

The decomposition is of the form

\[ A = PLU \]

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

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

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

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

**Objects from the Class**

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

**Extends**

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

"denseLU" directly extends "LU".

**Slots**

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

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

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

**Methods**

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

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

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

Examples

```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")))
```

mat2triplet

Map Matrix to its Triplet Representation

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

```r
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))
stopifnot(i == sort(trA$i), sort(j) == trA$j, x == sort(trA$x))

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

Description

Construct a Matrix of a class that inherits from Matrix.

Usage

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

Arguments

data an optional numeric data vector or matrix.
nrow when data is not a matrix, the desired number of rows
ncol when data is not a matrix, the desired number of columns
byrow  logical. If FALSE (the default) the matrix is filled by columns, otherwise the
matrix is filled by rows.
dimnames a dimnames attribute for the matrix: a list of two character components. They
are set if not NULL (as per default).
sparse  logical or NULL, specifying if the result should be sparse or not. By default, it is
made sparse when more than half of the entries are 0.
doDiag  logical indicating if a diagonalMatrix object should be returned when the re-
sulting 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) determinant of x. The method chosen depends on the actual Matrix class of x. Note that det also works for all our matrices, calling the appropriate determinant() method. The Matrix::.det

# a 3 by 2 matrix (+ integer warning)
Matrix(1:6, 3, 2)

## 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"))
Matrix-class

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 automagically (if they first call as.matrix() on their principal argument), e.g., apply, eigen, svd or kappa all do work via coercion to a “traditional” (dense) matrix.

Note

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

Author(s)

Douglas Bates <bates@stat.wisc.edu> and Martin Maechler

See Also

the classes dgeMatrix, dgCMatrix, and function Matrix for construction (and examples). Methods, e.g., for kronecker.
Examples

```
slotNames("Matrix")

c1 <- getClass("Matrix")
names(c1@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 \texttt{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. \texttt{boolArith} = TRUE for regular ("non cross") matrix products, \texttt{%%} cannot be specified. Instead, we provide the \texttt{%%} operator for \texttt{boolean} matrix products.

### Value

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

### Methods

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

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

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

crossprod signature(x = "CsparseMatrix",y = "missing") returns \(t(x) \%\% x\) as an \texttt{dsCMatrix} object.

crossprod signature(x = "TsparseMatrix",y = "missing") returns \(t(x) \%\% x\) as an \texttt{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 may be incomplete and partly missing. Please report such omissions if detected!
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, crossprod and `%*%`.

**Examples**

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

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

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

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

---

**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, ...)`

**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.
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

```r
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

```r
## Signature
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**

**Nearest Positive Definite Matrix**

**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 \( X \leftarrow (X + t(X))/2 \) should be done, after \( X \leftarrow tcrossprod(Qd,Q) \); some doubt if this is necessary.
doDykstra logical indicating if Dykstra's correction should be used; true by default. If false, the algorithm is basically the direct fixpoint iteration $Y_k = P_U(P_S(Y_{k-1}))$.

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

ensureSymmetry logical; by default, \texttt{symmpart(x)} is used whenever \texttt{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 (\texttt{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 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 diag(.) <- 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 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 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 (\texttt{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.
References


See Also

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

Examples

```r
## Higham(2002), p.334f - simple example
A <- matrix(1, 3,3); A[1,3] <- A[3,1] <- 0
n.A <- nearPD(A, corr=TRUE, do2eigen=FALSE)
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$mat, 2)
norm(m - near.m$mat[1:2,]) # 1.102 / 1.08
if(require("sfsmisc")) {
  m2 <- posdefify(m) # a simpler approach
  norm(m - m2) # 1.185, i.e., slightly "less near"
}
round(nearPD(m, only.values=TRUE), 9)

## A longer example, extended from Jens' original,
## showing the effects of some of the options:
pr <- Matrix(c(1, 0.477, 0.644, 0.478, 0.651, 0.826,
        0.477, 1, 0.516, 0.233, 0.682, 0.75,
        0.644, 0.516, 1, 0.599, 0.581, 0.742,
        0.478, 0.233, 0.599, 1, 0.741, 0.8,
        0.651, 0.682, 0.581, 0.741, 1, 0.798,
        0.826, 0.75, 0.742, 0.8, 0.798, 1),
```
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

Class "nMatrix" of Non-zero Pattern Matrices

Description

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

Slots

Common to all matrix object in the package:

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

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

Methods

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

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

coerce signature(from = "dMatrix",to = "nMatrix"), and

coerce signature(from = "lMatrix",to = "nMatrix"): For dense matrices with NAs, these coercions are valid since Matrix version 1.2.0 (still with a warning or a 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, dsparseMatrix, 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 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.
nan.counted  a logical describing how NAs should be counted. There are three possible settings for nan.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 nan.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 0s in x, counting NA’s depending on the nan.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

m <- Matrix(0+1:28, nrow = 4)
m[3, c(2,4:5,7)] <- m[3, 1:4] <- m[1:3, 6] <- 0
(mT <- as(m, "dgTMatrix"))
nnzero(mT)
(S <- crossprod(mT))
nnzero(S)
str(S)  # slots are smaller than nnzero()
stopifnot(nnnzero(S) == sum(as.matrix(S) != 0))# failed earlier

data(KNex)
M <- KNex$mm
class(M)
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.
- **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 (svd) of \( x \).

The default is "O". Only the first character of type[1] is used.

...  
进一步的参数传递给或从其他方法。

**Details**

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

**Value**

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

**References**

See Also

onenormest(), an approximate randomized estimate of the 1-norm condition number, efficient for large sparse matrices.
The norm() function from R's base package.

Examples

```r
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"),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 ntrRMatrix).

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

Slots

uplo: Object of class "character". Must be either "U", for upper triangular, and "L", for lower triangular. Present in the triangular and symmetric classes but not in the general class.

diag: Object of class "character". Must be either "U", for unit triangular (diagonal is all ones), or "N" for non-unit. The implicit diagonal elements are not explicitly stored when diag is "U". Present in the triangular classes only.

p: Object of class "integer" of pointers, one for each column (row), to the initial (zero-based) index of elements in the column. Present in compressed column-oriented and compressed row-oriented forms only.

i: Object of class "integer" of length nnzero (number of non-zero elements). These are the row numbers for each TRUE element in the matrix. All other elements are FALSE. Present in triplet and compressed column-oriented forms only.

j: Object of class "integer" of length nnzero (number of non-zero elements). These are the column numbers for each TRUE element in the matrix. All other elements are FALSE. Present in triplet and compressed column-oriented forms only.

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

Methods

coerce signature(from = "dgCMatrix", to = "ngCMatrix"), and many similar ones; typically you should coerce to "nsparseMatrix" (or "nMatrix"). Note that coercion to a sparse pattern matrix records all the potential non-zero entries, i.e., explicit ("non-structural") zeroes are coerced to TRUE, not FALSE, see the example.

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

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

See Also

the class dgCMatrix

Examples

(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
nnm <- !mm # no longer sparse
(nnm <- as(nnm, "spMatrix"))# "lgCMatrix"
## consistency check:
stopifnot(xor(as( mm, "matrix"),

as(nnm, "matrix")))
### nsyMatrix-class

#### 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.
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, "nsyMatrix")) # -> "nsy"
str (sM <- as(sM, "nspMatrix")) # -> "nsp": packed symmetric
```

---

**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

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

Examples

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

number-class

Class "number" of Possibly Complex Numbers

Description

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

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

Objects from the Class

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

Examples

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

pMatrix-class

Permutation matrices

Description

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

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

Details

Matrix multiplication with permutation matrices is equivalent to row or column permutation. Here are the four different cases for an arbitrary matrix $M$ and a permutation matrix $P$ (where we assume matching dimensions):
\[
P \cdot M = M \cdot P = M[i(p)] \\
PM = P \cdot M = M[p] \\
PM' = \text{tcrossprod}(M, P) \approx M[t(P)] = M[p] \\
\]

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 (\( \text{solve} \)) and transposition (\( t \)) are the same for permutation matrices \( P \).

**Objects from the Class**

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

**Slots**

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

**Extends**

Class "indMatrix", directly.

**Methods**

\%\%\% signature(x = "matrix", y = "pMatrix") and other signatures (use \text{showMethods("\%\%",class="pMatrix")}: ...

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

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

- \textbf{coerce} signature(from = "pMatrix", to = "matrix"): coercion to a traditional FALSE/TRUE matrix of mode logical. (in earlier version of \text{Matrix}, it resulted in a 0/1-integer matrix; logical makes slightly more sense, corresponding better to the “natural” \text{spMatrix} counterpart, "ngTMatrix").

- \textbf{coerce} signature(from = "pMatrix", to = "ngTMatrix"): coercion to sparse logical matrix of class \text{ngTMatrix}.

- \textbf{determinant} signature(x = "pMatrix", logarithm="logical"): Since permutation matrices are orthogonal, the determinant must be +1 or -1. In fact, it is exactly the \textit{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.

\textbf{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

\[ P <- as(p, \text{"pMatrix"}) \]
\[ p <- P@perm \]

see also the ‘Examples’.

“Row-indexing” a permutation matrix typically returns an "indMatrix". See "indMatrix" for all other subsetting/indexing and subassignment (\( A[\ldots] <-v \)) operations.

See Also

\texttt{invPerm(p)} computes the inverse permutation of an integer (index) vector \( p \).

Examples

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

set.seed(11)
## random permutation matrix :
(p10 <- as(sample(10),"pMatrix"))

## Permute rows / columns of a numeric matrix :
(mm <- round(array(rnorm(3 * 3), c(3, 3)), 2))
mm %*% pm1
pm1 %*% mm
try(as(as.integer(c(3,3,1)), "pMatrix"))# Error: not a permutation

as(pm1, "ngTMatrix")
p10[1:7, 1:4] # gives an "ngTMatrix" (most economic!)

## row-indexing of a <pMatrix> keeps it as an <indMatrix>:
p10[1:3,]
\end{verbatim}
Format and Print Sparse Matrices Flexibly

Description

Format and print sparse matrices flexibly. These are the “workhorses” used by the format, show and print methods for sparse matrices. If \( x \) is large, printSpMatrix2(\( x \)) calls printSpMatrix() twice, namely, for the first and the last few rows, suppressing those in between, and also suppresses columns when \( x \) is too wide.

printSpMatrix() basically prints the result of formatSpMatrix().

Usage

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

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

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

Arguments

\( x \) an \( \mathbf{R} \) object inheriting from class \texttt{sparseMatrix}.

\( \text{digits} \) significant digits to use for printing, see \texttt{print.default}, the default, \texttt{NULL}, corresponds to using \texttt{getOption("digits")}.

\( \text{maxp} \) integer, default from \texttt{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.

\( \text{cld} \) the class definition of \( x \); must be equivalent to \texttt{getClassDef(class(x))} and exists mainly for possible speedup.

\( \text{zero.print} \) character which should be printed for \textit{structural} zeroes. The default "." may occasionally be replaced by " " (blank); using "\( \theta \)" would look almost like \texttt{print()}ing of non-sparse matrices.
printSpMatrix

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 "I" 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 \texttt{sparseMatrix} and the classes extending it; maybe \texttt{sparseMatrix} or \texttt{spMatrix} as simple constructors of such matrices.

The underlying utilities \texttt{formatSparseM} and \texttt{.formatSparseSimple()} (on the same page).

Examples

\begin{verbatim}
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")
\end{verbatim}
backPermute     logical indicating if the rows of the $R$ matrix should be back permuted such that 
               `qrR()`'s result can be used directly to reconstruct the original matrix $X$.
row.names       logical indicating if `rownames` should propagated to the result.
...             further arguments passed to or from other methods

Methods

```r
x = "dgCMatrix"  QR decomposition of a general sparse double-precision matrix with 
                 nrow(x) >= ncol(x). Returns an object of class "sparseQR".

x = "sparseMatrix" works via "dgCMatrix".
```

See Also

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

Examples

```r
##------------- example of pivoting -- from base's 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)))

# Convert to sparseMatrix
X <- as(X, "sparseMatrix")
X # is singular, columns "b2" and "c3" are "extra"

##----- regular case ------------------------------------------
Xr <- X[, -c(3,6)] # the "regular" (non-singular) version of X

## regular case:
qXr <- qr(Xr)
qxr <- qr(as.matrix(Xr))
qcfXY <- qr.coef(qXr, Y) # 4x1 dgeMatrix

##----- interactive checks -----------------------------------

## regular case:
qXr <- qr(Xr)
qxr <- qr(as.matrix(Xr))
qcfXY <- qr.coef(qXr, Y) # 4x1 dgeMatrix

doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA")) ||
identical("true", unname(Sys.getenv("R_PKG_CHECKING_doExtras")))

doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA")) ||
identical("true", unname(Sys.getenv("R_PKG_CHECKING_doExtras")))

tolE <- if(doExtras) 1e-15 else 1e-13

stopifnot( 
  all.equal(qr.coef(qXr, Y), cf, tol=tolE), 
  getRversion() <= "3.4.1" ||
  all.equal(qr.coef(qXr, Y), m(cf), tol=tolE),
  all.equal(qcfXY, cf, tol=tolE),
)
```
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", "qrLINPACK", "qr", 
                      "useGrad", "maybeGrad"), 
           sval = svd(x, 0, 0)$d, warn.t = TRUE, warn.qr = TRUE) 

qr2rankMatrix(qr, tol = NULL, isBqr = is.qr(qr), do.warn = TRUE)
```

---

**rankMatrix**

**Rank of a Matrix**

**Description**

We provide several methods, the default corresponding to Matlab’s definition.

**Usage**

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

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, \( \max(\operatorname{dim}(x)) \times \operatorname{.Machine\$double\_eps} \) is according to Matlab's default (for its only method which is our method="tolNorm2").
- **method**: a character string specifying the computational method for the rank, can be abbreviated:
  - "tolNorm2": the number of singular values \( \geq \text{tol} \times \max(sval) \);
  - "qrLINPACK": for a dense matrix, this is the rank of \( \text{qr}(x, \text{tol}, \text{LAPACK}=\text{FALSE}) \) (which is \( \text{qr}(\ldots) \cdot \text{rank} \));
    - This ("qr*", dense) version used to be the recommended way to compute a matrix rank for a while in the past.
    - For sparse \( x \), this is equivalent to "qr.R".
  - "qr.R": this is the rank of triangular matrix \( R \), where \( \text{qr}() \) uses LAPACK or a "sparseQR" method (see qr-methods) to compute the decomposition QR. The rank of \( R \) is then defined as the number of “non-zero” diagonal entries \( d_i \) of \( R \), and “non-zero”’s fulfill \( |d_i| \geq \text{tol} \cdot \max(|d_i|) \).
  - "qr": is for back compatibility: for dense \( x \), it corresponds to "qrLINPACK", whereas for sparse \( x \), it uses "qr.R".
    - For all the "qr*" methods, singular values sval are not used, which may be crucially important for a large sparse matrix \( x \), as in that case, when sval is not specified, the default, computing \( \text{svd}() \) currently coerces \( x \) to a dense matrix.
  - "useGrad": considering the “gradient” of the (decreasing) singular values, the index of the smallest gap.
  - "maybeGrad": choosing method "useGrad" only when that seems reasonable; otherwise using "tolNorm2".
- **sval**: numeric vector of non-increasing singular values of \( x \); typically unspecified and computed from \( x \) when needed, i.e., unless method = "qr".
- **warn.t**: logical indicating if rankMatrix() should warn when it needs \( t(x) \) instead of \( x \). Currently, for method = "qr" only, gives a warning by default because the caller often could have passed \( t(x) \) directly, more efficiently.
- **warn.qr**: in the QR cases (i.e., if method starts with "qr"), rankMatrix() calls qr2rankMatrix(\ldots, do.warn = warn.qr), see below.
- **qr**: an R object resulting from \( \text{qr}(x, \ldots) \), i.e., typically inheriting from class "qr" or "sparseQR".
- **isBqr**: logical indicating if \( qr \) is resulting from base \( qr() \). (Otherwise, it is typically from Matrix package sparse \( qr() \).
- **do.warn**: 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!

Details

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

If \( x \) is a matrix of all 0 (or of zero dimension), the rank is zero; otherwise, typically a positive integer in \( 1:\min(\dim(x)) \) 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 NaN entries. Then, a warning is signalled (unless \( \text{warn.qr / do.warn} \) is not true) and NA (specifically, \( \text{NA_integer_} \) is returned.

Note

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

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

\begin{enumerate}
\item \text{rankMatrix(cbind(1, 0, 1:3)) \# 2}
\item \text{(mths <- eval(formals(rankMatrix)$method))}
\item \text{H12 <- Hilbert(12)}
\item \text{rankMatrix(H12, tol = 1e-20) \# 12; but 11 with default method & tol.}
\item \text{sapply(mths, function(.m.) rankMatrix(H12, method = .m.))}
\item \text{\# tolNorm2 qr.R qrLINPACK qr useGrad maybeGrad}
\item \text{\# 11 12 11 12 11 11}
\item \text{\# The meaning of 'tol' for method="qrLINPACK" and *dense* \( x \) is not entirely "scale free"}
\item \text{rMQL <- function(ex, M) rankMatrix(M, method="qrLINPACK", tol = 10^{-ex})}
\item \text{rMQR <- function(ex, M) rankMatrix(M, method="qr.R", tol = 10^{-ex})}
\item \text{sapply(5:15, rMQL, M = H12) \# result is platform dependent}
\item \text{\# 7 7 8 10 11 11 11 12 12 12 \ (x86_64)}
\item \text{sapply(5:15, rMQL, M = 1000 \times H12) \# not identical unfortunately}
\item \text{\# 7 7 8 10 11 11 12 12 12 12}
\item \text{sapply(5:15, rMQR, M = H12)}
\item \text{\# 5 6 7 8 8 9 10 10 11 11}
\item \text{sapply(5:15, rMQR, M = 1000 \times H12) \# the same*}
\item \text{\# "sparse" case:}
\item \text{M15 <- kronecker(diag(x=c(100,1,10)), Hilbert(5))}
\end{enumerate}
rcond

Estimate the Reciprocal Condition Number

Description

Estimate the reciprocal of the condition number of a matrix.

This is a generic function with several methods, as seen by showMethods(rcond).

Usage

rcond(x, norm, ...)

## S4 method for signature 'sparseMatrix,character'

rcond(x, norm, useInv=FALSE, ...)

Arguments

x

an R object that inherits from the Matrix class.

norm

character string indicating the type of norm to be used in the estimate. The default is "0" for the 1-norm ("0" is equivalent to "1"). For sparse matrices, when useInv=TRUE, norm can be any of the kinds allowed for norm; otherwise, the other possible value is "I" for the infinity norm, see also norm.
rcond

useInv
logical (or "Matrix" containing solve(x)). If not false, compute the reciprocal condition number as \(1/(\|x\| \cdot \|x^{-1}\|)\), where \(x^{-1}\) is the inverse of \(x\), solve(x). This may be an efficient alternative (only) in situations where solve(x) is fast (or known), e.g., for (very) sparse or triangular matrices.

Note that the result may differ depending on useInv, as per default, when it is false, an approximation is computed.

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

Value
An estimate of the reciprocal condition number of \(x\).

BACKGROUND
The condition number of a regular (square) matrix is the product of the norm of the matrix and the norm of its inverse (or pseudo-inverse).

More generally, the condition number is defined (also for non-square matrices \(A\)) as

\[
\kappa(A) = \frac{\max_{\|v\|=1} \|Av\|}{\min_{\|v\|=1} \|Av\|}.
\]

Whenever \(x\) is not a square matrix, in our method definitions, this is typically computed via rcond(qr.R(qr(X)),...) where \(X\) is \(x\) or \(t(x)\).

The condition number takes on values between 1 and infinity, inclusive, and can be viewed as a factor by which errors in solving linear systems with this matrix as coefficient matrix could be magnified.

rcond() computes the reciprocal condition number \(1/\kappa\) with values in \([0, 1]\) and can be viewed as a scaled measure of how close a matrix is to being rank deficient (aka “singular”).

Condition numbers are usually estimated, since exact computation is costly in terms of floating-point operations. An (over) estimate of reciprocal condition number is given, since by doing so overflow is avoided. Matrices are well-conditioned if the reciprocal condition number is near 1 and ill-conditioned if it is near zero.

References

See Also
norm, kappa() from package base computes an approximate condition number of a “traditional” matrix, even non-square ones, with respect to the \(p = 2\) (Euclidean) norm. solve.
condest, a newer approximate estimate of the (1-norm) condition number, particularly efficient for large sparse matrices.
Examples

```r
x <- Matrix(rnorm(9), 3, 3)
rcond(x)
## typically "the same" (with more computational effort):
1 / (norm(x) * norm(solve(x)))
rcond(Hilbert(9)) # should be about 9.1e-13

## For non-square matrices:
rcond(x1 <- cbind(1,1:10)) # 0.05278
rcond(x2 <- cbind(x1, 2:11)) # practically 0, since x2 does not have full rank

## sparse
(S1 <- Matrix(rbind(0:1,0, diag(3:-2))))
rcond(S1)
m1 <- as(S1, "denseMatrix")
all.equal(rcond(S1), rcond(m1))

## wide and sparse
rcond(Matrix(cbind(0, diag(2:-1))))
```

```r
## 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)) # 12800^2
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)`
replValue-class

Arguments

- **x**: numeric vector
- **times**: integer (valued) scalar: the number of repetitions

Value

A vector of class `abIndex`

See Also

- `rep.int()`, the base function; `abIseq`, `abIndex`.

Examples

```r
(ab <- rep2abI(2:7, 4))
stopifnot(identical(as(ab, "numeric"),
                      rep(2:7, 4)))
```

replValue-class  Virtual Class "replValue" - Simple Class for subassignment Values

Description

The class "replValue" is a virtual class used for values in signatures for sub-assignment of `Matrix` matrices.

In fact, it is a simple class union (`setClassUnion`) of "numeric" and "logical" (and maybe "complex" in the future).

Objects from the Class

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

See Also

- `Subassign-methods`, also for examples.

Examples

```r
showClass("replValue")
```
Class "rleDiff" is for compactly storing long vectors which mainly consist of linear stretches. For such a vector \( x \), \( \text{diff}(x) \) consists of constant stretches and is hence well compressable via \( \text{rle()} \).

**Objects from the Class**

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

Currently experimental, see below.

**Slots**

- `first`: A single number (of class "numLike", a class union of "numeric" and "logical").
- `rle`: Object of class "rle", basically a `list` with components "lengths" and "values", see `rle()`. As this is used to encode potentially huge index vectors, lengths may be of type `double` here.

**Methods**

There is a simple `show` method only.

**Note**

This is currently an experimental auxiliary class for the class `abIndex`, see there.

**See Also**

- `rle`, `abIndex`.

**Examples**

```r
showClass("rleDiff")
ab <- c(abIseq(2, 100), abIseq(20, -2))
ab@rleD # is "rleDiff"
```
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 nspamseMatrix.

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 sample.int(nrow*ncol,nnz), then—if rand.x is not NULL—gets x <- rand.x(nnz) and calls sparseMatrix(i=i,j=j,x=x,...). When rand.x=NULL, sparseMatrix(i=i,j=j,...) will return a pattern matrix (i.e., inheriting from nspamseMatrix).

Value

a sparseMatrix, say M of dimension (nrow, ncol), i.e., with dim(M) == c(nrow,ncol), if symmetric is not true, with nzM <- nnzero(M) fulfilling nzM <= nnz and typically, nzM == nnz.

Author(s)

Martin Maechler
Examples

```r
set.seed(17)# to be reproducible
M <- rsparsematrix(8, 12, nnz = 30) # small example, not very sparse
M
M1 <- rsparsematrix(1000, 20, nnz = 123, rand.x = runif)
summary(M1)

## a random *symmetric* Matrix
(S9 <- rsparsematrix(9, 9, nnz = 10, symmetric=TRUE)) # dsCMatrix
nnzero(S9)# ~ 20: as 'nnz' only counts one "triangle"

## a random pattern* aka boolean Matrix (no 'x' slot):
(n7 <- rsparsematrix(5, 12, nnz = 10, rand.x = NULL))

## a [T]riplet representation sparseMatrix:
T2 <- rsparsematrix(40, 12, nnz = 99, repr = "T")
head(T2)
```

RsparseMatrix-class  

Class "RsparseMatrix" of Sparse Matrices in Column-compressed Form

Description

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

Slots

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

Extends

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

Methods

Only few methods are defined currently on purpose, since we rather use the CsparseMatrix in Matrix. Recently, more methods were added but beware that these typically do not return "RsparseMatrix" results, but rather Csparse* or Tsparse* ones.

- `t` signature(x = "RsparseMatrix"): ...
- `coerce` signature(from = "RsparseMatrix", to = "CsparseMatrix"): ...
- `coerce` signature(from = "RsparseMatrix", to = "TsparseMatrix"): ...
See Also

its superclass, \texttt{sparseMatrix}, and, e.g., class \texttt{dgRMatrix} for the links to other classes.

Examples

\begin{verbatim}
showClass("RsparseMatrix")
\end{verbatim}

\section*{Schur Decomposition of a Matrix}

Description

Computes the Schur decomposition and eigenvalues of a square matrix; see the BACKGROUND information below.

Usage

\begin{verbatim}
Schur(x, vectors, ...)
\end{verbatim}

Arguments

\begin{itemize}
  \item \textbf{x} \hspace{1cm} numeric square Matrix (inherit from class \texttt{"Matrix"}) or traditional \texttt{matrix}. Missing values (NAs) are not allowed.
  \item \textbf{vectors} \hspace{1cm} logical. When \texttt{TRUE} (the default), the Schur vectors are computed, and the result is a proper \texttt{MatrixFactorization} of class \texttt{Schur}.
  \item \texttt{...} \hspace{1cm} further arguments passed to or from other methods.
\end{itemize}

Details

Based on the Lapack subroutine \texttt{dgees}.

Value

If \texttt{vectors} are \texttt{TRUE}, as per default: If \texttt{x} is a \texttt{Matrix} an object of class \texttt{Schur}, otherwise, for a traditional \texttt{matrix \texttt{x}}, a \texttt{list} with components \texttt{T}, \texttt{Q}, and \texttt{EValues}.

If \texttt{vectors} are \texttt{FALSE}, a list with components

\begin{itemize}
  \item \texttt{T} \hspace{1cm} the upper quasi-triangular (square) matrix of the Schur decomposition.
  \item \texttt{EValues} \hspace{1cm} the vector of \texttt{numeric} or \texttt{complex} eigen values of \texttt{T} or \texttt{A}.
\end{itemize}

BACKGROUND

If \texttt{A} is a square matrix, then \texttt{A = Q T t(Q)}, where \texttt{Q} is orthogonal, and \texttt{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 \texttt{A} are the same as those of \texttt{T}, which are easy to compute. The Schur form is used most often for computing non-symmetric eigenvalue decompositions, and for computing functions of matrices such as matrix exponentials.
References


Examples

```r
Schur(Hilbert(9)) # Schur factorization (real eigenvalues)

(A <- Matrix(round(rnorm(5*5, sd = 100)), nrow = 5))
(Sch.A <- Schur(A))

eTA <- eigen(Sch.A@T)
str(SchA <- Schur(A, vectors=FALSE))# no 'T' ==> simple list
stopifnot(all.equal(eTA$values, eigen(A)$values, tolerance = 1e-13),
  all.equal(eTA$values,
    local({z <- Sch.A@EValues
      z[order(Mod(z), decreasing=TRUE)]}), tolerance = 1e-13),
  identical(SchA$T, Sch.A@T),
  identical(SchA$EValues, Sch.A@EValues))

## For the faint of heart, we provide Schur() also for traditional matrices:

a.m <- function(M) unname(as(M, "matrix"))
a <- a.m(A)
Sch.a <- Schur(a)
stopifnot(identical(Sch.a, list(Q = a.m(Sch.A @ Q),
  T = a.m(Sch.A @ T),
  EValues = Sch.A@EValues)),
  all.equal(a, with(Sch.a, Q %*% T %*% t(Q))))
```

---

**Schur-class**

*Class “Schur” of Schur Matrix Factorizations*

**Description**

Class “Schur” is the class of Schur matrix factorizations. These are a generalization of eigen value (or “spectral”) decompositions for general (possibly asymmetric) square matrices, see the `Schur()` function.

**Objects from the Class**

Objects of class "Schur" are typically created by `Schur()`.

**Slots**

"Schur" has slots

- `T`: Upper Block-triangular *Matrix* object.
- `Q`: Square *orthogonal* "Matrix".
EValues: numeric or complex vector of eigenvalues of $T$.
Dim: the matrix dimension: equal to $c(n,n)$ of class "integer".

Extends
Class "MatrixFactorization", directly.

See Also
Schur() for object creation: MatrixFactorization.

Examples
showClass("Schur")
Schur(M <- Matrix(c(1:7, 10:2), 4,4))
## Trivial, of course:
str(Schur(Diagonal(5)))

## for more examples, see Schur()

solve-methods

Methods in Package Matrix for Function solve()

Description
Methods for function solve to solve a linear system of equations, or equivalently, solve for $X$ in

$$AX = B$$

where $A$ is a square matrix, and $X$, $B$ are matrices or vectors (which are treated as 1-column matrices), and the R syntax is

$$X <- \text{solve}(A,B)$$

In solve(a,b) in the Matrix package, a may also be a MatrixFactorization instead of directly a matrix.

Usage

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

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

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

Arguments

a a square numeric matrix, A, typically of one of the classes in Matrix. Logical matrices are coerced to corresponding numeric ones.

b numeric vector or matrix (dense or sparse) as RHS of the linear system Ax = b.

system only if a is a CHMfactor: character string indicating the kind of linear system to be solved, see below. Note that the default, "A", does not solve the triangular system (but "L" does).

sparse only when a is a sparseMatrix, i.e., typically a dgCMatrix: logical specifying if the result should be a (formally) sparse matrix.

tol only used when a is sparse, in the isSymmetric(a,tol=*) test, where that applies.

... 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 <-solve(a,b,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,"dgeMatrix"), using the 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 = "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:

```r
n1 <- 7; n2 <- 3
dd <- data.frame(a = gl(n1,n2), b = gl(n2,1,n1*n2))# balanced 2-way
X <- sparse.model.matrix(~ -1+ a + b, dd)# no intercept --> even sparser
XXt <- tcrossprod(X)
diag(XXt) <- rep(c(0,0,1,0), length.out = nrow(XXt))
n <- nrow(ZZ <- kronecker(XXt, Diagonal(x=c(4,1))))
image(a <- 2*Diagonal(n) + ZZ %*% Diagonal(x=c(10, rep(1, n-1))))
isSymmetric(a) # FALSE
image(drop0(skewpart(a)))
image(ia0 <- solve(a)) # checker board, dense [but really, a is singular!]
try(solve(a, sparse=TRUE))##-> error [ TODO: assertError ]
ia. <- solve(a, sparse=TRUE, tol = 1e-19)##-> *no* error
```
if(R.version$arch == "x86_64")
## 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 %*% a ; image(I.)
I0 <- drop0(zapsmall(I.)); image(I0)
.I <- a %*% iad
.I0 <- drop0(zapsmall(.I))
stopifnot( all.equal(as(I0, "diagonalMatrix"), Diagonal(n)),
all.equal(as(.I0,"diagonalMatrix"), Diagonal(n)) )

### sparse.model.matrix

#### Construct Sparse Design / Model Matrices

**Description**

Construct a sparse model or “design” matrix, 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**

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

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

```r
fac2Sparse(from, to = c("d", "i", "l", "n", "z"),
drop.unused.levels = TRUE, repr = c("C","T","R"), 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.
sparse.model.matrix

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) ==
           Matrix(model.matrix(~ a + b, dd), sparse=TRUE)),
          all(sparse.model.matrix(~ 0+ a + b, dd) ==
              Matrix(model.matrix(~ 0+ a + b, dd), sparse=TRUE)))

(ff <- gl(3,3,, c("X","Y", "Z")))
fac2sparse(ff) # 3 x 12 sparse Matrix of class "dgCMatrix"
##
## X 1 1 1 . . . . . . . .
## Y . . . . 1 1 1 . . . .
## Z . . . . . . . . 1 1 1 1
## can also be computed via sparse.model.matrix():
f30 <- gl(3,0 )
f12 <- gl(3,0, 12)
stopifnot(
  all.equal(t( fac2sparse(ff) ),
              sparse.model.matrix(~ 0+ff),
              tolerance = 0, check.attributes=FALSE),
  is(M <- fac2sparse(f30, drop= TRUE),"CsparseMatrix"), dim(M) == c(0, 0),
  is(M <- fac2sparse(f30, drop=FALSE),"CsparseMatrix"), dim(M) == c(3, 0),
  is(M <- fac2sparse(f12, drop= TRUE),"CsparseMatrix"), dim(M) == c(0,12),
  is(M <- fac2sparse(f12, drop=FALSE),"CsparseMatrix"), dim(M) == c(3,12)
)

### Sparse LU decomposition of a square sparse matrix

Description

Objects of this class contain the components of the LU decomposition of a sparse square matrix.
Objects from the Class

Objects can be created by calls of the form `new("sparseLU",...)` but are more commonly created by function `lu()` applied to a sparse matrix, such as a matrix of class `dgCMatrix`.

Slots

- **L**: Object of class "`dtCMatrix`", the lower triangular factor from the left.
- **U**: Object of class "`dtCMatrix`", the upper triangular factor from the right.
- **p**: Object of class "`integer`", permutation applied from the left.
- **q**: Object of class "`integer`", permutation applied from the right.
- **Dim**: the dimension of the original matrix; inherited from class `MatrixFactorization`.

Extends

Class "`LU`", directly. Class "`MatrixFactorization`", by class "LU".

Methods

- **expand** signature(x = "sparseLU") Returns a list with components \( P \), \( L \), \( U \), and \( Q \), where \( P \) and \( Q \) represent fill-reducing permutations, and \( L \) and \( U \) the lower and upper triangular matrices of the decomposition. The original matrix corresponds to the product \( P'LUQ \).

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 :
dimnames(A) <- dnA <- 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
```
SparseM-conversions

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.

```
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 = "sparseMatrix", to = "matrix.csr" ...
from = "matrix.csr", to = "dgCMatrix" ...
from = "matrix.coo", to = "dgCMatrix" ...
from = "matrix.csr", to = "Matrix" ...
from = "matrix.csc", to = "Matrix" ...
from = "matrix.coo", to = "Matrix" ...
```

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 a compressed, column-oriented, sparse matrix, inheriting from class CsparseMatrix (or TsparseMatrix if giveCsparse is false), from locations (and values) of its non-zero entries.

This is the recommended user interface rather than direct `new("***Matrix",....)` calls.

**Usage**

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

**Arguments**

- `i,j` integer vectors of the same length specifying the locations (row and column indices) of the non-zero (or non-TRUE) entries of the matrix. Note that for repeated pairs `(i_k,j_k)`, when `x` is not missing, the corresponding `x_k` are *added*, in consistency with the definition of the "TsparseMatrix" class, unless `use.last.ij` is true, in which case only the *last* of the corresponding `(i_k,j_k,x_k)` triplet is used.
- `p` numeric (integer valued) vector of pointers, one for each column (or row), to the initial (zero-based) index of elements in the column (or row). Exactly one of `i`, `j` or `p` must be missing.
- `x` optional values of the matrix entries. If specified, must be of the same length as `i` / `j`, or of length one where it will be recycled to full length. If missing, the resulting matrix will be a 0/1 pattern matrix, i.e., extending class nsparseMatrix.
- `dims` optional, non-negative, integer, dimensions vector of length 2. Defaults to `c(max(i),max(j))`.
- `dimnames` optional list of `dimnames`; if not specified, none, i.e., `NULL` ones, are used.
- `symmetric` logical indicating if the resulting matrix should be symmetric. In that case, only the lower or upper triangle needs to be specified via `(i/j/p)`.
- `triangular` logical indicating if the resulting matrix should be triangular. In that case, the lower or upper triangle needs to be specified via `(i/j/p)`.
- `index1` logical scalar. If `TRUE`, the default, the index vectors `i` and/or `j` are 1-based, as is the convention in R. That is, counting of rows and columns starts at 1. If `FALSE` the index vectors are 0-based so counting of rows and columns starts at 0; this corresponds to the internal representation.
- `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.
sparseMatrix

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.

check  logical indicating if a validity check is performed; do not set to FALSE unless you know what you’re doing!

use.last.ij  logical indicating if in the case of repeated, i.e., duplicated pairs \((i_k, j_k)\) only the last one should be used. The default, FALSE, corresponds to the "TsparseMatrix" definition.

Details

Exactly one of the arguments \(i, j\) and \(p\) must be missing.

In typical usage, \(p\) is missing, \(i\) and \(j\) are vectors of positive integers and \(x\) is a numeric vector. These three vectors, which must have the same length, form the triplet representation of the sparse matrix.

If \(i\) or \(j\) is missing then \(p\) must be a non-decreasing integer vector whose first element is zero. It provides the compressed, or “pointer” representation of the row or column indices, whichever is missing. The expanded form of \(p\), \(\text{rep(seq_along(dp), dp)}\) 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 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 TsparseMatrix is constructed, with numerical values given by \(x\), if non-missing. Note that in that case, when some pairs \((i_k, j_k)\) are repeated (aka “duplicated”), the corresponding \(x_k\) are added, in consistency with the definition of the "TsparseMatrix" class, unless use.last.ij is set to true.

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

The reason for returning a CsparseMatrix object instead of the triplet format by default is that the compressed column form is easier to work with when performing matrix operations. In particular, if there are no zeros in \(x\) then a CsparseMatrix is a unique representation of the sparse matrix.

Value

A sparse matrix, by default (from repr = "C") in compressed, column-oriented form, as an R object inheriting from both CsparseMatrix and generalMatrix.

Note

You do need to use index1 = FALSE (or add + 1 to \(i\) and \(j\)) if you want use the 0-based \(i\) (and \(j\)) slots from existing sparse matrices.

See Also

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

Random sparse matrices via rsparsematrix().

The standard R xtabs(*, sparse=TRUE), for sparse tables and sparse.model.matrix() for building sparse model matrices.

Consider CsparseMatrix and similar class definition help files.

Examples

```r
## simple example
i <- c(1,3:8); j <- c(2,9,6:10); x <- 7 * (1:7)
(A <- sparseMatrix(i, j, x = x)) ## 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)
str(m)
```
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.
sparseMatrix-class

Extends
Class "Matrix", directly.

Methods

```r
show (object = "sparseMatrix"): The show method for sparse matrices prints "structural" zeroes as "." using printSpMatrix() which allows further customization.
```

```r
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().
```

```r
format signature(x = "sparseMatrix"): ....
The format method for sparse matrices, see formatSpMatrix() for details such as the extra optional arguments.
```

```r
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.
```

```r
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.
```

```r
rbind2 (x = *, y = *): binding matrices together row-wise, see cbind2 above.
```

```r
determinant (x = "sparseMatrix", logarithm=TRUE): determinant() methods for sparse matrices typically work via Cholesky or lu decompositions.
```

```r
diag (x = "sparseMatrix"): extracts the diagonal of a sparse matrix.
```

```r
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)).
```

```r
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]))

---

sparseQR-class

Sparse QR decomposition of a sparse matrix

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 \(A[p+1,]\) 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 \(A[,q+1]\).

\(R\) 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 \(Q^* := P^{-1}Q = P'Q\) 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

\[ PA = Q^* R, \]
sparseQR-class

in R as

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

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.coef` signature(`qr = "sparseQR",y = "numeric"`): ...
- `qr.fitted` signature(`qr = "sparseQR",y = "ddenseMatrix"`): ...
- `qr.fitted` signature(`qr = "sparseQR",y = "matrix"`): ...
- `qr.fitted` signature(`qr = "sparseQR",y = "numeric"`): ...
- `qr.qty` signature(`qr = "sparseQR",y = "ddenseMatrix"`): ...
- `qr.qty` signature(`qr = "sparseQR",y = "matrix"`): ...
- `qr.qty` signature(`qr = "sparseQR",y = "numeric"`): ...
- `qr.qy` signature(`qr = "sparseQR",y = "ddenseMatrix"`): ...
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

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
str(sv <- sparseVector(x = 1:10, i = sample(999, 10), length=1000))
sx <- c(0,0,3, 3.2, 0,0,0,-3:1,0,0,2,0,0,5,0,0)
ss <- as(sx, "sparseVector")
stopifnot(identical(ss,
sparseVector(x = c(2, -1, -2, 3, 1, -3, 5, 3.2),
i = c(15L, 10:9, 3L,12L,8L,18L, 4L), length = 20L)))
(ns <- sparseVector(i= c(7, 3, 2), length = 10))
stopifnot(identical(ns,
new("nsparseVector", length = 10, i = c(2, 3, 7))))

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.
sparseVector-class

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.

toepz signature(x = "sparseVector"): as toepz(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 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") and

is.na, is.finite, is.infinite (x = "nsparseVector") 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

```r
ggetClass("sparseVector")
ggetClass("dsparseVector")
ggetClass("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)))), tol=0),
  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 **sparseMatrix()** and hence somewhat deprecated.

**Usage**

```r
spMatrix(nrow, ncol, i = integer(), j = integer(), x = numeric())
```

**Arguments**

- `nrow, ncol` integers specifying the desired number of rows and columns.
- `i,j` integer vectors of the same length specifying the locations of the non-zero (or non-TRUE) entries of the matrix.
spMatrix

\[ x \]

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 \texttt{R} object inheriting from both \texttt{TsparseMatrix} and \texttt{generalMatrix}. The matrix \( M \) will have \( M[i[k], j[k]] == x[k] \), for \( k = 1, 2, \ldots, n \), where \( n = \text{length}(i) \) and \( M[i', j'] == 0 \) for all other pairs \((i', j')\).

**See Also**

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

Consider \texttt{TsparseMatrix} and similar class definition help files.

**Examples**

```r
## simple example
A <- spMatrix(10, 20, i = c(1, 3:8), j = c(2, 9, 6:10), x = 7 * (1:7))
A # a "dgTMatrix"
summary(A)
str(A) # note that *internally* 0-based indices \((i, j)\) are used

L <- spMatrix(9, 30, i = rep(1:9, 3), 1:27, (1:27) %% 4 != 1)
L # an "lgTMatrix"
```

## A simplified predecessor of \texttt{Matrix}' \texttt{rsparsematrix()} function:

```r
rSpMatrix <- function(nrow, ncol, nnz, 
  rand.x = function(n) round(rnorm(nnz), 2)) {
  ## Purpose: random sparse matrix
  ## Arguments: \( (nrow, ncol) \): dimension 
  ## \texttt{nnz} : number of non-zero entries 
  ## \texttt{rand.x} : random number generator for 'x' slot
  ## Author: Martin Maechler, Date: 14.-16. May 2007
  stopifnot((nnz <- as.integer(nnz)) >= 0, 
    nrow >= 0, ncol >= 0, nnz <= nrow * ncol)
  spMatrix(nrow, ncol, 
    i = sample(nrow, nnz, replace = TRUE),
    j = sample(ncol, nnz, replace = TRUE),
    x = rand.x(nnz))
}
```

M1 <- rSpMatrix(100000, 20, nnz = 200)
summary(M1)

symmetricMatrix-class  Virtual Class of Symmetric Matrices in Package Matrix

Description

The virtual class of symmetric matrices, "symmetricMatrix", from the package Matrix contains
numeric and logical, dense and sparse matrices, e.g., see the examples 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), inher-
ited 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 (typi-
cally sparse) symmetric matrices.
isSymmetric signature(object = "symmetricMatrix"): returns TRUE trivially.
There’s a C function symmetricMatrix_validate() called by the internal validity checking func-
tions, 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.
symmpart

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

## 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
dimnames(sy) # both parts - as sy *is* symmetrical

showClass("symmetricMatrix")

## The names of direct subclasses:
scl <- getClass("symmetricMatrix")@subclasses
directly <- sapply(lapply(scl, slot, "by"), length) == 0
names(scl)[directly]

## Methods -- applicable to all subclasses above:
showMethods(classes = "symmetricMatrix")

---

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 \(x == symmpart(x) + skewpart(x)\) 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`).

See Also

`isSymmetric`.

Examples

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

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

dn <- dimnames(m) <- list(row = c("r1", "r2"), col = c("var.1", "var.2"))
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, i.e., the length of the x slot does not depend on diag.

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

Extends

Class "Matrix", directly.

Methods

There's a C function triangularMatrix_validity() called by the internal validity checking functions.

Currently, Schur, isSymmetric and as() (i.e. coerce) have methods with triangularMatrix in their signature.

See Also

isTriangular() for testing any matrix for triangularity; classes symmetricMatrix, and, e.g., dtrMatrix for numeric dense matrices, or ltCMatrix for a logical sparse matrix subclass of "triangularMatrix".

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".

Methods

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

Note

Most operations with sparse matrices are performed using the compressed, column-oriented or CsparseMatrix representation. The triplet representation is convenient for creating a sparse matrix or for reading and writing such matrices. Once it is created, however, the matrix is generally coerced to a `CspMatrix` for further operations.

Note that all new(.), `spMatrix` and `sparseMatrix(*, 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 "dgTMatrix".

For convenience, methods for some operations such as `%*%` and crossprod are defined for `TsparseMatrix` objects. These methods simply coerce the `TsparseMatrix` object to a `CspMatrix` object then perform the operation.

See Also

- its superclass, `sparseMatrix`, and the `dgTMatrix` class, for the links to other classes.

Examples

```
showClass("TsparseMatrix")
## or just the subclasses' names
names(getClass("TsparseMatrix")@subclasses)

T3 <- spMatrix(3,4, i=c(1,3:1), j=c(2,4:2), x=1:4)
T3 # only 3 non-zero entries, 5 = 1+4 !
```
uniqTsparse

Unique (Sorted) TsparseMatrix Representations

Description

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

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

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

\texttt{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 \texttt{or}.

Usage

\begin{verbatim}
uniqTsparse(x, class.x = c(class(x)))
anyDuplicatedT(x, di = dim(x))
\end{verbatim}

Arguments

- \(x\): a sparse matrix stored in triplet form, i.e., inheriting from class \texttt{TsparseMatrix}.
- \(\text{class.x}\): optional character string specifying \texttt{class(x)}.
- \(\text{di}\): the matrix dimension of \(x\), \texttt{dim}(x).

Value

\texttt{uniqTsparse(x)} returns a \texttt{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.

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

See Also

\texttt{TsparseMatrix}, for uniqueness, notably \texttt{dgTMatrix}.

Examples

\begin{verbatim}
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,
\end{verbatim}
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, L2u@i == T2u@i, N2@i == L2@i, L2@i == T2@i,
N2u@j == L2u@j, L2u@j == T2u@j, N2@j == L2@j, L2@j == T2@j)
# now with a nasty NA [partly failed in Matrix 1.1-5]:
validObject(L2.N)
(m2N <- as.matrix(L2.N)) # looks "ok"
il <- as.integer(m2N)
stopifnot(identical(10L, which(is.na(match(il, 0:1)))))
symnum(m2N)

---

unpack

Representation of Packed and Unpacked (Dense) Matrices

Description

“Packed” matrix storage here applies to dense matrices (`denseMatrix`) only, and there is available only for symmetric (`symmetricMatrix`) or triangular (`triangularMatrix`) matrices, where only one triangle of the matrix needs to be stored.

`unpack()` unpacks “packed” matrices, where `pack()` produces “packed” matrices.

Usage

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

unpack(x, ...)
Arguments

x for unpack(): a matrix stored in packed form, e.g., of class "d?pMatrix" where "?" is "t" for triangular or "s" for symmetric.

for pack(): a (symmetric or triangular) matrix stored in full storage.

symmetric logical (including NA) for optionally specifying if x is symmetric (or rather triangular).

upperTri (for the triangular case only) logical (incl. NA) indicating if x is upper (or lower) triangular.

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

Details

These are generic functions with special methods for different types of packed (or non-packed) symmetric or triangular dense matrices. Use showMethods("unpack") to list the methods for unpack(), and similarly for pack().

Value

for unpack(): A Matrix object containing the full-storage representation of x.

for pack(): A packed Matrix (i.e. of class "..pMatrix") representation of x.

Examples

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

(s <- crossprod(matrix(sample(15), 5,3))) # traditional symmetric matrix
(sp <- pack(s))
mt <- as.matrix(tt <- tril(s))
(pt <- pack(mt))
stopifnot(identical(pt, pack(tt)),
  dim(s) == dim(sp), all(s == sp),
  dim(mt) == dim(pt), all(mt == pt), all(mt == tt))
showMethods("pack")

Unused-classes

Virtual Classes Not Yet Really Implemented and Used

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")

updown  Up- and Down-Dating a Cholesky Decomposition

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 sub class of "CHMfactor").

Methods
signature(update = "character", C = "mMatrix", L = "CHMfactor") ..
signature(update = "logical", C = "mMatrix", L = "CHMfactor") ..

Author(s)
Contributed by Nicholas Nagle, University of Tennessee, Knoxville, USA

References
CHOLMOD manual, currently beginning of chapter 18. ...

See Also
Cholesky,
Examples

dn <- list(LETTERS[1:3], letters[1:5])
## pointer vectors can be used, and the (i,x) slots are sorted if necessary:
m <- sparseMatrix(i = c(3,1, 3:2, 2:1), p= c(0:2, 4,4,6), x = 1:6, dimnames = dn)
cA <- Cholesky(A <- crossprod(m) + Diagonal(5))
166 * as(cA, "Matrix") ^ 2
uc1 <- updown("+", Diagonal(5), cA)
## Hmm: this loses positive definiteness:
uc2 <- updown("-", 2*Diagonal(5), cA)
image(show(as(cA, "Matrix")))
image(show(c2 <- as(uc2,"Matrix")))# severely negative entries
##---> Warning

USCounties
USCounties Contiguity Matrix

Description

This matrix represents the contiguities of 3111 US counties using the Queen criterion of at least a single shared boundary point. The representation is as a row standardised spatial weights matrix transformed to a symmetric matrix (see Ord (1975), p. 125).

Usage

data(USCounties)

Format

A 3111² 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.
Details

The data were created into \( \mathcal{R} \) 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 \( \mathcal{R} \) 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

data(wrld_1deg)
(n <- ncol(wrld_1deg))
IM <- .symDiagonal(n)
doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA")) ||
identical("true", unname(Sys.getenv("R_PKG_CHECKING_doExtras")))
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))
)

system.time(MJ3 <- n * log(rho) + Matrix:::ldetL2up(C1, nWC, 1/rho))

system.time(MJ4 <- n * log(rho) + Matrix:::ldetL2up(C2, nWC, 1/rho))
stopifnot(all.equal(MJ, MJ2),
  all.equal(MJ, MJ3),
  all.equal(MJ, MJ4))
Description

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

Methods

There are more than these:

- \texttt{x = "Matrix", i = "missing", j = "missing", drop= "ANY"} ...
- \texttt{x = "Matrix", i = "numeric", j = "missing", drop= "missing"} ...
- \texttt{x = "Matrix", i = "missing", j = "numeric", drop= "missing"} ...
- \texttt{x = "dsparseMatrix", i = "missing", j = "numeric", drop= "logical"} ...
- \texttt{x = "dsparseMatrix", i = "numeric", j = "missing", drop= "logical"} ...
- \texttt{x = "dsparseMatrix", i = "numeric", j = "numeric", drop= "logical"} ...

See Also

\cite{methods} for subassignment to "Matrix" objects. \texttt{Extract} about the standard extraction.

Examples

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

Description

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

\textbf{Note:} Contrary to standard \texttt{matrix} assignment in base R, in \texttt{x[...] <- val} it is typically an error (see \texttt{stop}) when the type or class of \texttt{val} would require the class of \texttt{x} to be changed, e.g., when \texttt{x} is logical, say "lsparseMatrix", and \texttt{val} is numeric. In other cases, e.g., when \texttt{x} is a "nsparseMatrix" and \texttt{val} is not TRUE or FALSE, a warning is signalled, and \texttt{val} is "interpreted" as \texttt{logical}, and (logical) \texttt{NA} is interpreted as TRUE.
Methods

There are many many more than these:

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

See Also

[-methods for subsetting "Matrix" objects; the index class; Extract about the standard subset assignment (and extraction).

Examples

```r
set.seed(101)
(a <- m <- Matrix(round(rnorm(7*4),2), nrow = 7))

a[] <- 2.2 # <<- replaces **every** entry
a
## as do these:
a[,] <- 3 ; a[TRUE,] <- 4

m[2, 3] <- 3.14 # simple number
m[3, 3:4]<- 3:4 # simple numeric of length 2

## sub matrix assignment:
m[-(4:7), 3:4] <- cbind(1,2:4) #-> upper right corner of 'm'
m[3:5, 2:3] <- 0
m[6:7, 1:2] <- Diagonal(2)

## 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., R objects of class nMatrix, it is natural to allow matrix products using boolean instead of numerical arithmetic.

In package Matrix, we use the binary operator %&% (aka “infix”) function for this and provide methods for all our matrices and the traditional R matrices (see 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 = "sparseVector", y = "mMatrix")
signature(x = "mMatrix", y = "sparseVector")
signature(x = "sparseVector", y = "sparseVector")

Note

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

Examples

set.seed(7)
L <- Matrix(rnorm(20) > 1, 4,5)
(N <- as(L, "nMatrix"))
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 pattern' (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), 97
!,ldenseMatrix-method
  (ldenseMatrix-class), 84
!,ldiMatrix-method (ldiMatrix-class), 85
!,lgeMatrix-method (lgeMatrix-class), 85
!,lsparseMatrix-method
  (lsparseMatrix-classes), 86
!,ltrMatrix-method (ltrMatrix-class), 89
!,ldiMatrix-method (ltrMatrix-class), 89
!,ndenseMatrix-method
  (ndenseMatrix-class), 103
!,ngeMatrix-method (ngeMatrix-class), 107
!,nsparseMatrix-method
  (nsparseMatrix-classes), 112
!,ntrMatrix-method (ntrMatrix-class), 115
!,sparseVector-method
  (sparseVector-class), 151

* Choleski
  Cholesky, 23
* IO
  externalFormats, 62
* algebra
  band, 9
  bandSparse, 10
  CHMfactor-class, 17
  chol, 20
  chol2inv-methods, 22
  Cholesky, 23
  Cholesky-class, 26
  colSums, 27
  dgCMatrix-class, 36
  dgeMatrix-class, 37
  dgRMatrix-class, 38
  dgTMatrix-class, 39
  Diagonal, 41
  dMatrix-class, 45
dpoMatrix-class, 46
dsCMatrix-class, 49
dsRMatrix-class, 52
dtCMatrix-class, 54
dtRMatrix-class, 58
  expand, 60
  expm, 61
  externalFormats, 62
  fciacul, 64
  Hilbert, 69
  lsparseMatrix-classes, 86
  lu, 90
  LU-class, 93
  Matrix, 95
  Matrix-class, 97
  matrix-products, 99
  nearPD, 104
  nMatrix-class, 108
  norm, 111
  nsparseMatrix-classes, 112
  qr-methods, 121
  rankMatrix, 123
  rcond, 126
  Schur, 133
  sparseQR-class, 148
  unpack, 162
* arithmetic
  invPerm, 75
* arith
  all.equal-methods, 7
  colSums, 27
  symmpart, 157
* array
  [ methods, 168
  [ methods, 168
  bandSparse, 10
  bdiag, 12
cBind, 16

171
chol, 20
Cholesky, 23
colSums, 27
Diagonal, 41
drop0, 48
externalFormats, 62
facmul, 64
forceSymmetric, 65
Hilbert, 69
KhatriRao, 80
kronecker-methods, 83
lu, 90
Matrix, 95
nearPD, 104
qr-methods, 121
rankMatrix, 123
rcond, 126
rsparsematrix, 131
sparseMatrix, 143
sparseQR-class, 148
sparseVector, 150
spMatrix, 154
symmpart, 157
unpack, 162
* attribute
  nnzero, 109
* classes
  abIndex-class, 4
  abIseq, 6
  atomicVector-class, 8
  CHMfactor-class, 17
  Cholesky-class, 26
  compMatrix-class, 29
  CsparseMatrix-class, 32
  ddenseMatrix-class, 33
  ddiMatrix-class, 34
  denseMatrix-class, 35
  dgCMatrix-class, 36
  dgeMatrix-class, 37
  dRMatrix-class, 38
  dtMatrix-class, 39
diagonalMatrix-class, 42
diagU2N, 44
dMatrix-class, 45
dpoMatrix-class, 46
dsCMatrix-class, 49
dspMatrix-class, 51
dsRMatrix-class, 52
dsyMatrix-class, 53
dtCMatrix-class, 54
dtpMatrix-class, 56
dtrRMatrix-class, 58
dtrMatrix-class, 59
generalMatrix-class, 67
index-class, 72
indMatrix-class, 73
ldenseMatrix-class, 84
ldiMatrix-class, 85
lgeMatrix-class, 85
lsparseMatrix-classes, 86
lsyMatrix-class, 88
ltrMatrix-class, 89
LU-class, 93
mat2triplet, 94
Matrix-class, 97
MatrixClass, 101
MatrixFactorization-class, 102
ndenseMatrix-class, 103
ngeMatrix-class, 107
mMatrix-class, 108
nsparseMatrix-classes, 112
nsyMatrix-class, 114
ntrMatrix-class, 115
number-class, 116
pMatrix-class, 116
replValue-class, 129
rleDiff-class, 130
RsparseMatrix-class, 132
Schur-class, 134
sparseLU-class, 140
sparseMatrix-class, 146
sparseQR-class, 148
sparseVector-class, 151
symmetricMatrix-class, 156
triangularMatrix-class, 158
tspMatrix-class, 159
uniqTsparse, 161
Unused-classes, 163
* datasets
  CAex, 15
  KNex, 82
  USCounties, 165
  wrld_1deg, 166
* distribution
  rsparsematrix, 131
* graph
INDEX

graph-sparseMatrix, 68
* hplot
  image-methods, 70
* manip
  abIseq, 6
cBind, 16
mat2triplet, 94
rep2abI, 128
* math
  expm, 61
* methods
[-methods, 168
[<--methods, 168
%%%-methods, 169
all-methods, 7
all.equal-methods, 7
band, 9
BunchKaufman-methods, 14
chol2inv-methods, 22
image-methods, 70
is.na-methods, 76
isSymmetric-methods, 79
isTriangular, 79
KhatriRao, 80
kronecker-methods, 83
matrix-products, 99
qr-methods, 121
solve-methods, 135
SparseM-conversions, 142
updown, 164
* models
  sparse.model.matrix, 138
* print
  formatSparseM, 66
  printSpMatrix, 119
* utilities
diagU2N, 44
drop0, 48
formatSparseM, 66
graph-sparseMatrix, 68
is.null.DN, 78
mat2triplet, 94
uniqTsparse, 161
*,Matrix,ddiMatrix-method
  (diagonalMatrix-class), 42
*,ddenseMatrix,ldiMatrix-method
  (diagonalMatrix-class), 42
*,ddiMatrix,Matrix-method
  (diagonalMatrix-class), 42
*,ddiMatrix,ddenseMatrix-method
  (diagonalMatrix-class), 42
*,ddiMatrix,ldenseMatrix-method
  (diagonalMatrix-class), 42
*,ddiMatrix,ldiMatrix-method
  (diagonalMatrix-class), 42
*,ldenseMatrix,ddiMatrix-method
  (diagonalMatrix-class), 42
*,ldenseMatrix,ldiMatrix-method
  (diagonalMatrix-class), 42
*,ldiMatrix,Matrix-method
  (diagonalMatrix-class), 42
*,ldiMatrix,ddenseMatrix-method
  (diagonalMatrix-class), 42
*,ldiMatrix,ldenseMatrix-method
  (diagonalMatrix-class), 42
*,ldiMatrix,ndenseMatrix-method
  (diagonalMatrix-class), 42
*,ndenseMatrix,ddiMatrix-method
  (diagonalMatrix-class), 42
*,ndenseMatrix,ldiMatrix-method
  (diagonalMatrix-class), 42
+,Matrix,missing-method
  (Matrix-class), 97
+,dgTMatrix,dgTMatrix-method
  (dgTMatrix-class), 39
-,Matrix,missing-method
  (Matrix-class), 97
-,ddiMatrix,missing-method
  (diagonalMatrix-class), 42
-,denseMatrix,missing-method
  (denseMatrix-class), 35
-,dsparseVector,missing-method
  (sparseVector-class), 151
-,indMatrix,missing-method
  (indMatrix-class), 73
-,ldiMatrix,missing-method
  (diagonalMatrix-class), 42
-,lsparseMatrix,missing-method
  (lsparseMatrix-classes), 86
-,nsparseMatrix,missing-method
  (nsparseMatrix-classes), 112
-,pMatrix,missing-method
Index
(matrix-products), 99
%%, ddenseMatrix, matrix-method
(matrix-products), 99
%%, ddenseMatrix, ndenseMatrix-method
(matrix-products), 99
%%, denseMatrix, diagonalMatrix-method
(matrix-products), 99
%%, dgeMatrix, dgeMatrix-method
(matrix-products), 99
%%, dgeMatrix, diagonalMatrix-method
(matrix-products), 99
%%, dgeMatrix, dsparseMatrix-method
(matrix-products), 99
%%, dgeMatrix, dpMatrix-method
(matrix-products), 99
%%, dgeMatrix, matrix-method
(matrix-products), 99
%%, dgeMatrix, numLike-method
(matrix-products), 99
%%, diagonalMatrix, CsparseMatrix-method
(matrix-products), 99
%%, diagonalMatrix, denseMatrix-method
(matrix-products), 99
%%, diagonalMatrix, dgeMatrix-method
(matrix-products), 99
%%, diagonalMatrix, diagonalMatrix-method
(matrix-products), 99
%%, diagonalMatrix, lgeMatrix-method
(matrix-products), 99
%%, diagonalMatrix, matrix-method
(matrix-products), 99
%%, diagonalMatrix, sparseMatrix-method
(matrix-products), 99
%%, dspMatrix, ddenseMatrix-method
(matrix-products), 99
%%, dspMatrix, matrix-method
(matrix-products), 99
%%, dsparseMatrix, ddenseMatrix-method
(matrix-products), 99
%%, dsparseMatrix, dgeMatrix-method
(matrix-products), 99
%%, dMatrix, sparseVector-method
(matrix-products), 99
%%, matrix, CsparseMatrix-method
(matrix-products), 99
%%, matrix, matrix-method
(matrix-products), 99
%%, dtMatrix, matrix-method
(matrix-products), 99
%%, dtMatrix, mmatrix-method
(matrix-products), 99
%%, dtrMatrix, ddenseMatrix-method
(matrix-products), 99
%%, dtrMatrix, dtrMatrix-method
(matrix-products), 99
%%, dtrMatrix, matrix-method
(matrix-products), 99
%%, indMatrix, Matrix-method
(matrix-products), 99
%%, indMatrix, indMatrix-method
(matrix-products), 99
%%, indMatrix, matrix-method
(matrix-products), 99
%%, integer, dMatrix-method
(matrix-products), 99
%%, lMatrix, dMatrix-method
(matrix-products), 99
%%, lMatrix, lMatrix-method
(matrix-products), 99
%%, lMatrix, nMatrix-method
(matrix-products), 99
%%, ldenseMatrix, ddenseMatrix-method
(matrix-products), 99
%%, ldenseMatrix, ldenseMatrix-method
(matrix-products), 99
%%, ldenseMatrix, lsparseMatrix-method
(matrix-products), 99
%%, ldenseMatrix, matrix-method
(matrix-products), 99
%%, ldenseMatrix, ndenseMatrix-method
(matrix-products), 99
%%, lgCMatrix, lgCMatrix-method
(matrix-products), 99
%%, lgeMatrix, diagonalMatrix-method
(matrix-products), 99
%%, lgeMatrix, diagonalMatrix-method
(matrix-products), 99
%%, lgeMatrix, lgeMatrix-method
(matrix-products), 99
%%, lsparseMatrix, ldenseMatrix-method
(matrix-products), 99
%%, lsparseMatrix, lsparseMatrix-method
(matrix-products), 99
%%, mMatrix, sparseVector-method
(matrix-products), 99
%%, matrix, CsparseMatrix-method
(matrix-products), 99
%%, matrix, matrix-method
(matrix-products), 99
%%, matrix, ddenseMatrix-method
(matrix-products), 99
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
\%
(diagonalMatrix-class), 42
^,ddenseMatrix,ddiMatrix-method
(diagonalMatrix-class), 42
^,ddenseMatrix,ldiMatrix-method
(diagonalMatrix-class), 42
^,ddiMatrix,Matrix-method
(diagonalMatrix-class), 42
^,ddiMatrix,ddenseMatrix-method
(diagonalMatrix-class), 42
^,ddiMatrix,ldenseMatrix-method
(diagonalMatrix-class), 42
^,ldenseMatrix,ddiMatrix-method
(diagonalMatrix-class), 42
^,ldenseMatrix,ldiMatrix-method
(diagonalMatrix-class), 42
^,ldiMatrix,Matrix-method
(diagonalMatrix-class), 42
^,ldiMatrix,ddenseMatrix-method
(diagonalMatrix-class), 42
^,ldiMatrix,ldenseMatrix-method
(diagonalMatrix-class), 42
^,ldiMatrix,ndenseMatrix-method
(diagonalMatrix-class), 42
^,ndenseMatrix,ddiMatrix-method
(diagonalMatrix-class), 42
^,ndenseMatrix,ldiMatrix-method
(diagonalMatrix-class), 42
^,ndenseMatrix,ndenseMatrix-method
(diagonalMatrix-class), 42
^,nsparseMatrix,ldiMatrix-method
(diagonalMatrix-class), 42
all.equal, 7
all.equal,abIndex,abIndex-method
(all.equal-methods), 7
all.equal,abIndex,numLike-method
(all.equal-methods), 7
all.equal,ANY,Matrix-method
(all.equal-methods), 7
all.equal,ANY,spMatrix-method
(all.equal-methods), 7
all.equal,ANY,sparseMatrix-method
(all.equal-methods), 7
all.equal,ANY,sparseVector-method
(all.equal-methods), 7
all.equal,Matrix,ANY-method
(all.equal-methods), 7
all.equal,Matrix,Matrix-method
(all.equal-methods), 7
all.equal,numLike,abIndex-method
(all.equal-methods), 7
all.equal,spMatrix,ANY-method
(all.equal-methods), 7
all.equal,spMatrix,spMatrix-method
(all.equal-methods), 7
all.equal,spMatrix,spVector-method
(all.equal-methods), 7
all.equal,spVector,ANY-method
(all.equal-methods), 7
all.equal,spVector,spMatrix-method
(all.equal-methods), 7
all.equal,spVector,spVector-method
(all.equal-methods), 7
all.equal-methods, 7
all.equal.numeric, 8
any, 7, 46
any,ddiMatrix-method
(diagonalMatrix-class), 42
any,ldiMatrix-method
(diagonalMatrix-class), 42
any,lMatrix-method (all-methods), 7
any,nsparseMatrix-method
(nspMatrix-classes), 112
anyDuplicated, 161
anyDuplicatedT (uniqTsparse), 161
anyNA,dMatrix-method (is.na-methods), 76
anyNA,iMatrix-method (is.na-methods), 76
anyNA,lMatrix-method (is.na-methods), 76
anyNA,nsparseMatrix-method
(is.na-methods), 76
anyNA,nsparseMatrix-method

abbreviate, 120
abIndex, 6, 128–130
abIndex-class, 4
abIseq, 5, 6, 129
abIseq1 (abIseq), 6
abs, 71
all, 7, 46
all,ddiMatrix-method
(diagonalMatrix-class), 42
all,ldenseMatrix-method (all-methods), 7
all,ldiMatrix-method
(diagonalMatrix-class), 42
all,lsparseMatrix-method (all-methods), 7
all,lsyMatrix-method (all-methods), 7
all,Matrix-method (all-methods), 7
all,nsparseMatrix-method
(nspMatrix-classes), 112
all-methods, 7
(is.na-methods), 76
anyNA, nsparseVector-method
(is.na-methods), 76
anyNA, sparseVector-method
(is.na-methods), 76
anyNA, xMatrix-method (is.na-methods), 76
anyNA, zMatrix-method (is.na-methods), 76
apply, 98
Arith, 37, 45
Arith, abIndex, abIndex-method
(abIndex-class), 4
Arith, abIndex, numLike-method
(abIndex-class), 4
Arith, CsparseMatrix, CsparseMatrix-method
(CsparseMatrix-class), 32
Arith, CsparseMatrix, numeric-method
(CsparseMatrix-class), 32
Arith, ddenseMatrix, ddenseMatrix-method
(ddenseMatrix-class), 33
Arith, ddenseMatrix, logical-method
(ddenseMatrix-class), 33
Arith, ddenseMatrix, numeric-method
(ddenseMatrix-class), 33
Arith, ddenseMatrix, sparseVector-method
(sparseVector-class), 151
Arith, ddiMatrix, logical-method
(diagonalMatrix-class), 42
Arith, ddiMatrix, numeric-method
(diagonalMatrix-class), 42
Arith, dgCMatrix, dgCMatrix-method
(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, dgeMatrix, sparseVector-method
(sparseVector-class), 151
Arith, dMatrix, dMatrix-method
(dMatrix-class), 45
Arith, dpoMatrix, logical-method
(dpoMatrix-class), 46
Arith, dpoMatrix, numeric-method
(dpoMatrix-class), 46
Arith, dppMatrix, logical-method
(dpoMatrix-class), 46
Arith, dppMatrix, numeric-method
(dpoMatrix-class), 46
Arith, dsCMATRIX, dsCMATRIX-method
(dsCMATRIX-class), 49
Arith, dsparseMatrix, logical-method
(dsparseMatrix-class), 51
Arith, dsparseMatrix, numeric-method
(dsparseMatrix-class), 51
Arith, dsparseVector, dsparseVector-method
(sparseVector-class), 151
Arith, dtCMATRIX, dtCMATRIX-method
(dtCMATRIX-class), 54
Arith, ldiMatrix, logical-method
(diagonalMatrix-class), 42
Arith, ldiMatrix, numeric-method
(diagonalMatrix-class), 42
Arith, lgCMatrix, lgCMATRIX-method
(lsparseMatrix-classes), 86
Arith, lgeMatrix, lgeMatrix-method
(lgeMatrix-class), 85
Arith, lgTMatrix, lgTMatrix-method
(lsparseMatrix-classes), 86
Arith, lMatrix, logical-method
(dMatrix-class), 45
Arith, lMatrix, numeric-method
(dMatrix-class), 45
Arith, logical, ddenseMatrix-method
(ddenseMatrix-class), 33
Arith, logical, ddiMatrix-method
(diagonalMatrix-class), 42
Arith, logical, dgCMatrix-method
(dgCMatrix-class), 36
Arith, logical, dgeMatrix-method
(dgeMatrix-class), 37
Arith, logical, dpoMatrix-method
(dpoMatrix-class), 46
Arith, logical, dppMatrix-method
(dpoMatrix-class), 46
Arith, logical, dsparseMatrix-method
(dsparseMatrix-class), 51
Arith, logical, ldiMatrix-method
(diagonalMatrix-class), 42
Arith, logical, lMatrix-method
(dMatrix-class), 45
Arith, logical, nMatrix-method
INDEX

(sparseVector-class), 151
as_dsTMatrix_listw, 165, 167
atomicVector-class, 8
attributes, 80
backsolve, 137
band, 9, 10, 11, 42
band,CsparseMatrix-method (band), 9
band,ddenseMatrix-method (band), 9
band,denseMatrix-method (band), 9
band,ldenseMatrix-method (band), 9
band,RsparseMatrix-method (band), 9
band,TsparseMatrix-method (band), 9
band-methods (band), 9
bandSparse, 9, 10, 12, 42, 96, 145
bdiag, 11, 12, 96, 145, 155
BunchKaufman, 14, 15, 26, 29
BunchKaufman (BunchKaufman-methods), 14
BunchKaufman,dspMatrix-method (BunchKaufman-methods), 14
BunchKaufman,dsyMatrix-method (BunchKaufman-methods), 14
BunchKaufman,matrix-method (BunchKaufman-methods), 14
BunchKaufman-class (Cholesky-class), 26
BunchKaufman-methods, 14
c, 5
c.abIndex (abIseq), 6
c.sparseVector (sparseVector-class), 151
CAex, 15
cBind, 16
cbind, 16, 17, 147
cbind2, 16, 17
cbind2, ANY, Matrix-method (Matrix-class), 97
cbind2, atomicVector, ddiMatrix-method (diagonalMatrix-class), 42
cbind2, atomicVector, ldiMatrix-method (diagonalMatrix-class), 42
cbind2, atomicVector, Matrix-method (Matrix-class), 97
cbind2, ddiMatrix, atomicVector-method (diagonalMatrix-class), 42
cbind2, ddiMatrix, matrix-method (diagonalMatrix-class), 42
cbind2, denseMatrix, denseMatrix-method (denseMatrix-class), 35
cbind2, denseMatrix, matrix-method (denseMatrix-class), 35
cbind2, denseMatrix, numeric-method (denseMatrix-class), 35
cbind2, denseMatrix, sparseMatrix-method (sparseMatrix-class), 146
cbind2, diagonalMatrix, sparseMatrix-method (diagonalMatrix-class), 42
cbind2, ldiMatrix, atomicVector-method (diagonalMatrix-class), 42
cbind2, ldiMatrix, matrix-method (diagonalMatrix-class), 42
cbind2, Matrix, ANY-method (Matrix-class), 97
cbind2, Matrix, atomicVector-method (Matrix-class), 97
cbind2, Matrix, denseMatrix-method (Matrix-class), 146
cbind2, NULL, Matrix-method (Matrix-class), 97
cbind2, numeric, denseMatrix-method (denseMatrix-class), 35
cbind2, numeric, sparseMatrix-method (sparseMatrix-class), 146
cbind2, sparseMatrix, denseMatrix-method (sparseMatrix-class), 146
cbind2, sparseMatrix, diagonalMatrix-method (diagonalMatrix-class), 42
cbind2, sparseMatrix, matrix-method (sparseMatrix-class), 146
cbind2, sparseMatrix, numeric-method (sparseMatrix-class), 146
cbind2, sparseMatrix, sparseMatrix-method (sparseMatrix-class), 146
character, 5, 11, 45, 74, 97, 101, 102, 108, 117, 139, 143
CHMfactor, 24, 26, 27, 103, 136, 137, 164
CHMfactor-class, 17
CHMsimpl, 24
CHMsimpl-class (CHMfactor-class), 17
CHMsuper, 24
CHMsuper-class (CHMfactor-class), 17

chol, 15, 20, 21, 23, 24, 26, 27, 29, 37, 47, 50, 92

chol,ddenseMatrix-method (chol), 20
chol,ldiMatrix-method (chol), 20
chol,dgeMatrix-method (chol), 20
chol,dpoMatrix-method (chol), 20
chol,dppMatrix-method (chol), 20
chol,dsCMatrix-method (chol), 20
chol,dsparseMatrix-method (chol), 20
chol,dtCMatrix-method (chol), 20
chol,dtRMatrix-method (chol), 20
chol,dtTMatrix-method (chol), 20
chol,ldiMatrix-method (chol), 20
chol,lsCMatrix-method (chol), 20
chol,nsCMatrix-method (chol), 20
chol,Matrix-method (chol), 20

chol-methods (chol), 20

chol2inv, ANY-method (chol2inv-methods), 22

chol2inv,CHMfactor-method (chol2inv-methods), 22
chol2inv,denseMatrix-method (chol2inv-methods), 22
chol2inv,diagonalMatrix-method (chol2inv-methods), 22
chol2inv,dtrMatrix-method (chol2inv-methods), 22
chol2inv,sparseMatrix-method (chol2inv-methods), 22
chol2inv-methods, 22

Cholesky, 15, 18, 20, 21, 23, 24, 26, 27, 29, 50, 61, 103, 147, 164

Cholesky,CsparseMatrix-method (Cholesky), 23
Cholesky,dsCMatrix-method (Cholesky), 23
Cholesky,Matrix-method (Cholesky), 23
Cholesky,nsparseMatrix-method (Cholesky), 23
Cholesky,sparseMatrix-method (Cholesky), 23
Cholesky-class, 26

CholeskyFactorization-class (MatrixFactorization-class), 102
class, 4, 8, 11, 17, 44, 78, 79, 101, 105, 124, 129, 143, 150, 151, 168
coerce, 159
coerce,abIndex,integer-method (abIndex-class), 4
coerce,abIndex,numeric-method (abIndex-class), 4
coerce,abIndex,seqMat-method (abIndex-class), 4
c coerce,abIndex,-vector-method (abIndex-class), 4
c co erce,ANY,denseMatrix-method (denseMatrix-class), 35
c coerce,ANY,Matrix-method (Matrix-class), 97
c coerce,ANY,nsparseVector-method (sparseVector-class), 151
c coerc e,ANY, sparseMatrix-method (sparseMatrix-class), 146
c coerc e, ANY, sparseVector-method (sparseVector-class), 151
coerce,atomicVector,dsparseVector-method (sparseVector-class), 151
coerce,atomicVector,sparseVector-method (sparseVector-class), 151
coerce,BunchKaufman,1Matrix-method (Cholesky-class), 26
coerce,CHMfactor,Matrix-method (CHMfactor-class), 17
coerce,CHMfactor,pMatrix-method (CHMfactor-class), 17
coerce,CHMfactor,sparseMatrix-method (CHMfactor-class), 17
coerce,CHMfactor,triangularMatrix-method (CHMfactor-class), 17
coerce,Cholesky,1Matrix-method (Cholesky-class), 26
coerce,Cholesky,dsCMatrix-method (Cholesky-class), 32
coerce,Cholesky,Matrix-method (Cholesky-class), 32
coerce,Cholesky,nsparseMatrix-method (Cholesky-class), 32
coerce,Cholesky,sparseMatrix-method (Cholesky-class), 32
coerce,CsparseMatrix,matrix.coo-method (SparseM-conversions), 142
coerce, CsparseMatrix, matrix.csc-method (SparseM-conversions), 142
coerce, CsparseMatrix, matrix.csr-method (SparseM-conversions), 142
coerce, CsparseMatrix, nMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, nsparseMatrix-method (CsparseMatrix-class), 32
coerce, CsparseMatrix, sparseVector-method (sparseVector-class), 151
coerce, CsparseMatrix, symmetricMatrix-method (symmetricMatrix-class), 156
coerce, ddiMatrix, CsparseMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, dddenseMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, dgCMatrix-method (dgMatrix-class), 36
coerce, ddiMatrix, dgeMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, ddCMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, symmetricMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, triangularMatrix-method (diagonalMatrix-class), 42
coerce, ddiMatrix, tspMatrix-method (diagonalMatrix-class), 42
coerce, denseMatrix, CsparseMatrix-method (denseMatrix-class), 35
coerce, denseMatrix, generalMatrix-method (denseMatrix-class), 35
coerce, denseMatrix, RsparseMatrix-method (RsparseMatrix-class), 132
coerce, denseMatrix, sparseMatrix-method (denseMatrix-class), 35
coerce, denseMatrix, symmetricMatrix-method (symmetricMatrix-class), 156
coerce, denseMatrix, tspMatrix-method (denseMatrix-class), 35
coerce, dgCMatrix, dgeMatrix-method (dgMatrix-class), 36
coerce, dgCMatrix, dgTMatrix-method (dgMatrix-class), 36
coerce, dgCMatrix, dsCMatrix-method (dgMatrix-class), 36
coerce, dgCMatrix, dtCMatrix-method (dgMatrix-class), 36
coerce, dgCMatrix, lgeMatrix-method (lgeMatrix-class), 37
coerce, dgCMatrix, lgeMatrix-method (dgMatrix-class), 37
coerce, dgCMatrix, matrix-method (dgMatrix-class), 36
coerce, dgCMatrix, ngCMatrix-method (dgMatrix-class), 36
coerce, dgCMatrix, vector-method (dgMatrix-class), 36
coerce, dgCMatrix, dspMatrix-method (dsyMatrix-class), 49
coerce, dgCMatrix, dsTMatrix-method (dsyMatrix-class), 49
coerce, dgCMatrix, dsyMatrix-method (dsyMatrix-class), 36
coerce, dgCMatrix, dtTMatrix-method (dtTMatrix-class), 49
coerce, dgTMatrix, dgCMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dgTMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtCMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtCMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtTMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtTMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dgTMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dgTMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtCMatrix-method (dgTMatrix-class), 36
coerce, dgTMatrix, dtTMatrix-method (dgTMatrix-class), 36
coerce,dgTMatrix, lgeMatrix-method (lgeMatrix-class), 37
coerce, dgTMatrix, matrix-method (dgTMatrix-class), 37
coerce, dgTMatrix, triangularMatrix-method (dgTMatrix-class), 37
coerce, dgTMatrix, triangularMatrix-method (dgTMatrix-class), 37
coerce, dgTMatrix, RsparseMatrix-method (RsparseMatrix-class), 132
coerce, dgTMatrix, dgCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dgCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dgCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dgCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dsTMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dtCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dtCMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dtTMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dtTMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dgTMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dgTMatrix-method (dgTMatrix-class), 39
coerce, dgTMatrix, dtTMatrix-method
  (dgTMatrix-class), 39
coerce, dgTMatrix, graphNEL-method
  (graph-sparseMatrix), 68
coerce, dgTMatrix, matrix-method
  (dgTMatrix-class), 39
coerce, dgTMatrix, matrix.coo-method
  (SparseM-conversions), 142
coerce, dgTMatrix, symmetricMatrix-method
  (dgTMatrix-class), 39
coerce, dgTMatrix, triangularMatrix-method
  (dgTMatrix-class), 39
coerce, diagonalMatrix, denseMatrix-method
  (diagonalMatrix-class), 42
coerce, diagonalMatrix, generalMatrix-method
  (diagonalMatrix-class), 42
coerce, diagonalMatrix, matrix-method
  (diagonalMatrix-class), 42
coerce, diagonalMatrix, nMatrix-method
  (diagonalMatrix-class), 42
coerce, diagonalMatrix, nsparseMatrix-method
  (diagonalMatrix-class), 42
coerce, diagonalMatrix, sparseVector-method
  (sparseVector-class), 151
coerce, dMatrix, lMatrix-method
  (dMatrix-class), 45
coerce, dMatrix, nMatrix-method
  (nMatrix-class), 108
coerce, dpoMatrix, corMatrix-method
  (dpoMatrix-class), 46
coerce, dpoMatrix, dppMatrix-method
  (dpoMatrix-class), 46
coerce, dpoMatrix, lMatrix-method
  (dpoMatrix-class), 46
coerce, dpoMatrix, nMatrix-method
  (dpoMatrix-class), 46
coerce, dpoMatrix, sparseMatrix-method
  (dpoMatrix-class), 46
coerce, dppMatrix, dpoMatrix-method
  (dpoMatrix-class), 46
coerce, dppMatrix, lMatrix-method
  (dpoMatrix-class), 46
coerce, dppMatrix, nMatrix-method
  (dpoMatrix-class), 46
coerce, dppMatrix, sparseMatrix-method
  (dpoMatrix-class), 46
coerce, dsCMatrix, dgeMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dgTMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dpoMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dsRMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dsTMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dsysyMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, generalMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, lsCMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, matrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, nsCMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, vector-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dgeMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dpoMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, dppMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, lMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, nMatrix-method
  (dsCMatrix-class), 49
coerce, dsCMatrix, sparseMatrix-method
  (dsCMatrix-class), 49
coerce, dsTMatrix, matrix-method
  (dsCMatrix-class), 49
coerce, dsyMatrix, corMatrix-method
  (dsyMatrix-class), 53
coerce, dsyMatrix, dpoMatrix-method
  (dsyMatrix-class), 53
coerce, dsyMatrix, dsCMatrix-method
  (dsCMatrix-class), 49
coerce, dsyMatrix, dspMatrix-method
  (dsyMatrix-class), 53
coerce, dsyMatrix, dtrMatrix-method
  (dsyMatrix-class), 59
coerce, dsyMatrix, dtpMatrix-method
  (dsyMatrix-class), 59
coerce, dsyMatrix, dtMMatrix-method
  (dsyMatrix-class), 54
coerce, dsyMatrix, dgeMatrix-method
  (dsyMatrix-class), 54
coerce, dsyMatrix, dtTMatrix-method
  (dsyMatrix-class), 54
coerce, dsyMatrix, generalMatrix-method
  (dsyMatrix-class), 54
coerce, dsyMatrix, matrix-method
  (dsyMatrix-class), 53
coerce, dtCMatrix, dgeMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, dgeMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, dgeMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, dtMMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, dtTMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, generalMatrix-method
  (dtCMatrix-class), 54
coerce, dtCMatrix, matrix-method
  (dtCMatrix-class), 53
coerce, dtCMatrix, sparseMatrix-method
  (dtCMatrix-class), 54
coerce, factor, dgCMatrix-method
  (dgCMatrix-class), 36
coerce, factor, sparseMatrix-method
  (sparseMatrix-class), 146
coerce, graph, CsparseMatrix-method
  (sparseMatrix-class), 68
coerce, graph, Matrix-method
  (sparseMatrix-class), 68
coerce, graph, graphAM-method
  (graph-sparseMatrix), 68
coerce, graph, graphAM-method
  (graph-sparseMatrix), 68
coerce, graph, CsparseMatrix-method
  (graph-sparseMatrix), 68
coerce, graph, CsparseMatrix-method
  (graph-sparseMatrix), 68
coerce, graphAM, sparseMatrix-method
  (graph-sparseMatrix), 68
coerce, graphAM, sparseMatrix-method
  (graph-sparseMatrix), 68
coerce, graphNEL, CsparseMatrix-method
  (graph-sparseMatrix), 68
coerce, graphNEL, dsCMatrix-method
  (dsCMatrix-class), 49
coerce, graphNEL, TsparseMatrix-method
  (graph-sparseMatrix), 68
coerce, indMatrix, CsparseMatrix-method
  (indMatrix-class), 73
coerce, indMatrix, dMatrix-method
  (indMatrix-class), 73
coerce, indMatrix, dsparseMatrix-method
  (indMatrix-class), 73
coerce, indMatrix, lMatrix-method
  (indMatrix-class), 73
coerce, dtrMatrix, CsparseMatrix-method
  (dtrMatrix-class), 59
coerce, dtrMatrix, dtpMatrix-method
  (dtrMatrix-class), 59
coerce, dtrMatrix, dgeMatrix-method
  (dtrMatrix-class), 54
coerce, dtrMatrix, dgTMatrix-method
  (dtrMatrix-class), 54
coerce, indMatrix, lsparseMatrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, matrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, ngeMatrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, ngTMatrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, nMatrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, nsparseMatrix-method

(coerce, indMatrix-class), 73

coerce, indMatrix, TsparseMatrix-method

(coerce, indMatrix-class), 73

coerce, integer, indMatrix-method

(coerce, integer-class), 73

coerce, integer, pMatrix-method

(coerce, integer-class), 116

coerce, ldenseMatrix, matrix-method

(ldenseMatrix-class), 84

coerce, ldiMatrix, CsparseMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, ldenseMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, lgCMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, ldenseMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, lgCMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, lgTMatrix-method

(diagonalMatrix-class), 42

coerce, ldiMatrix, l(Matrix)-method

(dMatrix-class), 45

coerce, lgCMatrix, dgCMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, lgeMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, lTMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, lsCMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, ltrMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, mMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, symmetricMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, triangularMatrix-method

(lsparseMatrix-classes), 86

coerce, lgCMatrix, triangularMatrix-method

(lsparseMatrix-classes), 86

coerce, list, indMatrix-method

(indMatrix-class), 73

coerce, lMatrix, dgCMatrix-method

(dMatrix-class), 45

coerce, lMatrix, dMatrix-method

(dMatrix-class), 45

coerce, lMatrix, nMatrix-method

(nMatrix-class), 108

coerce, logical, abIndex-method

(abIndex-class), 4

coerce, lsCMatrix, dgTMatrix-method

(lsparseMatrix-classes), 86

coerce, lsCMatrix, dsCMatrix-method

(lsparseMatrix-classes), 86
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>coerce,lsCMatrix,generalMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsCMatrix,lgCMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsCMatrix,lgTMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsCMatrix,lstMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsCMatrix,matrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsCMatrix,lsyMatrix-method</td>
<td>(lsyMatrix-class)</td>
<td>88</td>
</tr>
<tr>
<td>coerce,lsTMatrix,dstMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsTMatrix,lgCMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsTMatrix,lsyMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,lsyMatrix,dsyMatrix-method</td>
<td>(lsyMatrix-class)</td>
<td>88</td>
</tr>
<tr>
<td>coerce,lsyMatrix/lgMatrix-method</td>
<td>(lsyMatrix-class)</td>
<td>88</td>
</tr>
<tr>
<td>coerce,lsyMatrix/lspMatrix-method</td>
<td>(lsyMatrix-class)</td>
<td>88</td>
</tr>
<tr>
<td>coerce,ltCMatrix,dMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltCMatrix,dtCMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltCMatrix,lgCMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltCMatrix,ltMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltCMatrix,matrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltMatrix,dpMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltMatrix/lgMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltMatrix,ltMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltMatrix/dtMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltMatrix,generalMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltMatrix,ltTMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltTMatrix,dtTMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltTMatrix,generalMatrix-method</td>
<td>(lsparseMatrix-classes)</td>
<td>86</td>
</tr>
<tr>
<td>coerce,ltTMatrix,lgMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltTMatrix,lsyMatrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,ltTMatrix,matrix-method</td>
<td>(ltrMatrix-class)</td>
<td>89</td>
</tr>
<tr>
<td>coerce,Matrix,complex-method</td>
<td>(Matrix-class)</td>
<td>97</td>
</tr>
<tr>
<td>coerce,Matrix,corMatrix-method</td>
<td>(dpoMatrix-class)</td>
<td>46</td>
</tr>
<tr>
<td>coerce,Matrix,corMatrix-method</td>
<td>(dpoMatrix-class)</td>
<td>46</td>
</tr>
<tr>
<td>coerce,Matrix,CsparseMatrix-method</td>
<td>(Matrix-class)</td>
<td>97</td>
</tr>
<tr>
<td>coerce,Matrix,CsparseMatrix-method</td>
<td>(CsparseMatrix-class)</td>
<td>32</td>
</tr>
<tr>
<td>coerce,Matrix,corMatrix-method</td>
<td>(Matrix-class)</td>
<td>97</td>
</tr>
<tr>
<td>coerce,matrix,corMatrix-method</td>
<td>(dgCMatrix-class)</td>
<td>36</td>
</tr>
<tr>
<td>coerce,matrix,corMatrix-method</td>
<td>(dgeMatrix-class)</td>
<td>37</td>
</tr>
<tr>
<td>coerce,matrix,corMatrix-method</td>
<td>(RsparseMatrix-class)</td>
<td>132</td>
</tr>
<tr>
<td>coerce,matrix,corMatrix-method</td>
<td>(dgTMatrix-class)</td>
<td>39</td>
</tr>
<tr>
<td>coerce,MATRIX,dgCMatrix-method</td>
<td>(diagonalMatrix-class)</td>
<td>42</td>
</tr>
</tbody>
</table>
coerce,matrix,diagonalMatrix-method (diagonalMatrix-class), 42
coerce,Matrix,dpoMatrix-method (dpoMatrix-class), 46
coerce,matrix,dpoMatrix-method (dpoMatrix-class), 46
coerce,Matrix,dppMatrix-method (dpoMatrix-class), 46
coerce,matrix,dppMatrix-method (dpoMatrix-class), 46
coerce,matrix,dsCMatrix-method (dsCMatrix-class), 49
coerce,matrix,dsyMatrix-method (dsyMatrix-class), 53
cache,matrix,dsyMatrix-method (dsyMatrix-class), 53
cache,matrix,dtCMatrix-method (dtCMatrix-class), 54
cache,matrix,dtMatrix-method (dtCMatrix-class), 56
cache,matrix,dtMatrix-method (dtCMatrix-class), 59
cache,matrix,dtTMatrix-method (dtCMatrix-class), 54
cache,matrix,dtTMatrix-method (dtCMatrix-class), 59
cache,matrix,dsyMatrix-method (dsyMatrix-class), 53
cache,matrix,dtCMatrix-method (dsyMatrix-class), 53
cache,matrix,dtMatrix-method (dsyMatrix-class), 54
cache,matrix,dtTMatrix-method (dsyMatrix-class), 54
cache,matrix,ldenseMatrix-method (ldenseMatrix-class), 84
cache,matrix,lgCMatrix-method (ldenseMatrix-class), 84
cache,matrix,lgCMatrix-method (lgCMatrix-class), 86
cache,matrix,lgCMatrix-method (lgeMatrix-class), 85
cache,matrix,lgTMatrix-method (lgeMatrix-class), 85
cache,matrix,lgTMatrix-method (lgTMatrix-class), 86
cache,matrix,lgTMatrix-method (lspMatrix-class), 86
cache,matrix,lgTMatrix-method (lsyMatrix-class), 88
cache,matrix,lsyMatrix-method (lsyMatrix-class), 88
cache,matrix,lsyMatrix-method (lspMatrix-class), 86
cache,matrix,lsyMatrix-method (lspMatrix-class), 86
coerce,matrix,ltCMatrix-method (ltrMatrix-class), 89
cache,matrix,ltMatrix-method (ltrMatrix-class), 89
cache,matrix,ltTMatrix-method (ltrMatrix-class), 89
cache,matrix,ltTMatrix-method (ltrMatrix-class), 89
cache,matrix,ltCMatrix-method (ltrMatrix-class), 89
cache,matrix,ltTMatrix-method (ltrMatrix-class), 89
coerce,matrix,Matrix-method (Matrix-class), 97
coerce,matrix,Matrix-method (Matrix-class), 97
coerce,matrix,Matrix-method (Matrix-class), 97
coerce,matrix,Matrix-method (Matrix-class), 97
coerce,matrix,nMatrix-method (nMatrix-class), 108
coerce,matrix,nsyMatrix-method (nsyMatrix-class), 114
coerce,matrix,nsyMatrix-method (nsyMatrix-class), 114
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,ntMatrix-method (ntMatrix-class), 115
coerce,matrix,lsyMatrix-method (lspMatrix-class), 86
coerce,matrix,lsyMatrix-method (lspMatrix-class), 86
coerce,matrix,lsyMatrix-method (lsyMatrix-class), 88
coerce, matrix, TsparseMatrix-method (TsparseMatrix-class), 159
coerce, Matrix, vector-method (Matrix-class), 97
coerce, matrix.coo, CsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.coo, dgCMatrix-method (SparseM-conversions), 142
coerce, matrix.coo, dgTMatrix-method (SparseM-conversions), 142
coerce, matrix.coo, Matrix-method (SparseM-conversions), 142
coerce, matrix.coo, TsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csc, CsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csc, dgCMatrix-method (SparseM-conversions), 142
coerce, matrix.csc, Matrix-method (SparseM-conversions), 142
coerce, matrix.csc, TsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csr, CsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csr, dgCMatrix-method (SparseM-conversions), 142
coerce, matrix.csr, Matrix-method (SparseM-conversions), 142
coerce, matrix.csr, TsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csr, RsparseMatrix-method (SparseM-conversions), 142
coerce, matrix.csr, TsparseMatrix-method (SparseM-conversions), 142
coerce, ndenseMatrix, CsparseMatrix-method (ndenseMatrix-class), 103
coerce, ndenseMatrix, ldenseMatrix-method (ndenseMatrix-class), 103
coerce, ndenseMatrix, matrix-method (ndenseMatrix-class), 103
coerce, ndenseMatrix, nspMatrix-method (ndenseMatrix-class), 103
coerce, ndenseMatrix, sparseMatrix-method (ndenseMatrix-class), 103
coerce, ndenseMatrix, TsparseMatrix-method (ndenseMatrix-class), 103
coerce, ngCMatrix, dgCMatrix-method (nsparseMatrix-classes), 112
coerce, ngCMatrix, dMatrix-method (nsparseMatrix-classes), 112
coerce, ngCMatrix, dsparseMatrix-method (nsparseMatrix-classes), 112
coerce, ngCMatrix, lgCMatrix-method (nsparseMatrix-classes), 112
coerce, ngCMatrix, lMatrix-method (nsparseMatrix-classes), 112
coerce, ngCMatrix, lsparseMatrix-method (nsparseMatrix-classes), 112
coerce,ngTMatrix,lgTMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,lMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,matrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,ngCMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,ngeMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,ntTMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,symmetricMatrix-method
(nsparseMatrix-classes), 112
coerce,ngTMatrix,triangularMatrix-method
(nsparseMatrix-classes), 112
coerce,nMatrix,dMatrix-method
(nMatrix-class), 108
coerce,nMatrix,indMatrix-method
(indMatrix-class), 73
coerce,nMatrix,lMatrix-method
(nMatrix-class), 108
coerce,nMatrix,matrix-method
(nMatrix-class), 108
coerce,nMatrix,pMatrix-method
(pMatrix-class), 108
coerce,nsCMatrix,dMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,dsCMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,dsparseMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,generalMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,lMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,lsCMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,lsCMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,lsparseMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,matCMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,matrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,ngCMatrix-method
(nsparseMatrix-classes), 112
coerce,nsCMatrix,nsTMatrix-method
(nsparseMatrix-classes), 112
coerce,nsparseMatrix,dsparseMatrix-method
(nsparseMatrix-classes), 112
coerce,nsyMatrix,dsyMatrix-method
(nsyMatrix-class), 114
coerce,nsyMatrix,lsyMatrix-method
(ndenseMatrix-class), 103
coerce,nsyMatrix,ngeMatrix-method
(nsyMatrix-class), 114
coerce,nsyMatrix,nsyMatrix-method
(nsyMatrix-class), 114
coerce,ntCMatrix,dMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,dsparseMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,dtCMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,lMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,lsCMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,ltCMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,dsCMatrix-method
(nsparseMatrix-classes), 112
coerce,ntCMatrix,dsyMatrix-method
(nsyMatrix-class), 114
coerce,ntCMatrix,dsyMatrix-method
(ndenseMatrix-class), 103
coerce,ntCMatrix,ngCMatrix-method
(nsyMatrix-class), 114
coerce,ntCMatrix,nsyMatrix-method
(nsyMatrix-class), 114
coerce,ntCMatrix,dsyMatrix-method
(nsyMatrix-class), 114
coerce,ntCMatrix,dsyMatrix-method
(nsyMatrix-class), 114
INDEX

coerce,ntCMatrix, matrix-method
  (nsparseMatrix-classes), 112
coerce,ntCMatrix, ngCMatrix-method
  (nsparseMatrix-classes), 112
coerce,ntCMatrix,TsparseMatrix-method
  (nsparseMatrix-classes), 112
coerce,ntpMatrix,dtpMatrix-method
  (ntrMatrix-class), 115
coerce,ntpMatrix,ltpMatrix-method
  (ndenseMatrix-class), 103
coerce,ntpMatrix,ngeMatrix-method
  (ntrMatrix-class), 115
coerce,ntpMatrix,ntrMatrix-method
  (ntrMatrix-class), 115
coerce,ntrMatrix,dtrMatrix-method
  (ntrMatrix-class), 115
coerce,ntrMatrix,ldtrMatrix-method
  (ntrMatrix-class), 103
coerce,ntrMatrix,ngtMatrix-method
  (ntrMatrix-class), 115
coerce,ntrMatrix,ntpMatrix-method
  (ntrMatrix-class), 115
coerce,nTMatrix, dTMatrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, generalMatrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, matrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, ngCMatrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, ngtMatrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, nCMatrix-method
  (nsparseMatrix-classes), 112
coerce,nTMatrix, ntrMatrix-method
  (nsparseMatrix-classes), 112
coerce, numeric, abIndex-method
  (abIndex-class), 4
coerce, numeric, CsparseMatrix-method
  (CsparseMatrix-class), 32
coerce, numeric, indMatrix-method
  (indMatrix-class), 73
coerce, numeric, pMatrix-method
  (pMatrix-class), 116
coerce, numeric, seqMat-method
  (abIndex-class), 4
coerce, numeric, TsparseMatrix-method
  (TsparseMatrix-class), 159
coerce, numLike, dgeMatrix-method
  (dgeMatrix-class), 37
coerce, pBunchKaufman, lMatrix-method
  (Cholesky-class), 26
coerce, pCholesky, lMatrix-method
  (Cholesky-class), 26
coerce, pMatrix, CsparseMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, dMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, dsparseMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, lMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, nMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, nspMatrix-method
  (pMatrix-class), 116
coerce, pMatrix, TMatrix-method
  (pMatrix-class), 116
coerce, RsparseMatrix, CsparseMatrix-method
  (RsparseMatrix-class), 132
coerce, RsparseMatrix, denseMatrix-method
  (RsparseMatrix-class), 132
coerce, RsparseMatrix, dgeMatrix-method
  (RsparseMatrix-class), 132
coerce, RsparseMatrix, dMatrix-method
  (RsparseMatrix-class), 132
coerce, RsparseMatrix, dgeMatrix-method
  (RsparseMatrix-class), 132
INDEX

coerce,RsparseMatrix,TsparseMatrix-method
(RsparseMatrix-class), 132

coerce,seqMat,abIndex-method
(abIndex-class), 4

coerce,seqMat,numeric-method
(abIndex-class), 4

coerce,sparseMatrix,generalMatrix-method
(sparseMatrix-class), 146

coerce,sparseMatrix,graph-method
(graph-sparseMatrix), 68

coerce,sparseMatrix,graphNEL-method
(graph-sparseMatrix), 68

coerce,sparseMatrix,indMatrix-method
(indMatrix-class), 73

coerce,sparseMatrix,pMatrix-method
(pMatrix-class), 116

coerce,sparseMatrix,RsparseMatrix-method
(RsparseMatrix-class), 132

coerce,sparseMatrix,sparseVector-method
(sparseVector-class), 151

coerce,sparseMatrix,symmetricMatrix-method
(sparseMatrix-class), 146

coerce,sparseMatrix,triangularMatrix-method
(triangularMatrix-class), 158

coerce,TsparseMatrix,1Matrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,IsparseMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,generalMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,graph-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,graphNEL-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,indMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,IsparseMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,1Matrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,IsparseMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,1Matrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,IsparseMatrix-method
(TsparseMatrix-class), 159

coerce,TsparseMatrix,1Matrix-method
(TsparseMatrix-class), 159

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27

colMeans (colSums), 27
INDEX

colSums, denseMatrix-method (colSums), 27
colSums, dgCMatrix-method (colSums), 27
colSums, dgeMatrix-method (dgeMatrix-class), 37
colSums, diagonalMatrix-method (colSums), 27
colSums, igCMatrix-method (colSums), 27
colSums, indMatrix-method (indMatrix-class), 73
colSums, lgCMatrix-method (colSums), 27
colSums, ngCMatrix-method (colSums), 27
colSums, RsparseMatrix-method (colSums), 27
colSums, TsparseMatrix-method (colSums), 27
Compare, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class), 32
Compare, dMatrix, logical-method (dMatrix-class), 45
Compare, dMatrix, numeric-method (dMatrix-class), 45
Compare, lgeMatrix, lgeMatrix-method (lgeMatrix-class), 85
Compare, lMatrix, logical-method (lMatrix-class), 45
Compare, logical, dMatrix-method (dMatrix-class), 45
Compare, logical, nMatrix-method (nMatrix-class), 108
Compare, lsparseMatrix, lsparseMatrix-method (lsparseMatrix-classes), 86
Compare, ngeMatrix, ngeMatrix-method (ngeMatrix-class), 107
Compare, nMatrix, logical-method (nMatrix-class), 108
Compare, nMatrix, nMatrix-method (nMatrix-class), 108
Compare, nMatrix, numeric-method (nMatrix-class), 108
Compare, numeric, dMatrix-method (dMatrix-class), 45
Compare, numeric, lMatrix-method (lMatrix-class), 45
Compare, numeric, nMatrix-method (nMatrix-class), 108

Complex, 9, 133, 163
compMatrix, 68
compMatrix-class, 29
condest, 30, 30, 127
contrasts, 138
corMatrix, 54
cov2cor, Matrix-method (Matrix-class), 97
cov2cor, sparseMatrix-method (sparseMatrix-class), 146
crossprod, 32, 37, 47, 74, 99, 101, 137, 147
crossprod (matrix-products), 99
crossprod, ANY, Matrix-method (matrix-products), 99
crossprod, ANY, TsparseMatrix-method (matrix-products), 99
crossprod, CsparseMatrix, CsparseMatrix-method (matrix-products), 99
crossprod, CsparseMatrix, ddenseMatrix-method (matrix-products), 99
crossprod, CsparseMatrix, diagonalMatrix-method (matrix-products), 99
crossprod, CsparseMatrix, matrix-method (matrix-products), 99
crossprod, CsparseMatrix, missing-method (matrix-products), 99
crossprod, CsparseMatrix, numLike-method (matrix-products), 99
crossprod, ddenseMatrix, CsparseMatrix-method (matrix-products), 99
crossprod, ddenseMatrix, ddenseMatrix-method (matrix-products), 99
crossprod, ddenseMatrix, dgCMatrix-method (matrix-products), 99
crossprod, ddenseMatrix, dgeMatrix-method (matrix-products), 99
crossprod, ddenseMatrix, ldenseMatrix-method (matrix-products), 99
crossprod, ddenseMatrix, matrix-method (matrix-products), 99
crossprod, ddenseMatrix, missing-method (matrix-products), 99
crossprod, ddenseMatrix, ndenseMatrix-method (matrix-products), 99
crossprod, dgCMatrix, dgeMatrix-method
matrix-products, 99
crossprod,dgeMatrix,dgeMatrix-method
(matrix-products), 99
crossprod,dgeMatrix,dsparseMatrix-method
(matrix-products), 99
crossprod,dgeMatrix,matrix-method
(matrix-products), 99
crossprod,dgeMatrix,missing-method
(matrix-products), 99
crossprod,diagonalMatrix,CsparseMatrix-method
(matrix-products), 99
crossprod,diagonalMatrix,dgeMatrix-method
(matrix-products), 99
crossprod,diagonalMatrix,diagonalMatrix-method
(matrix-products), 99
crossprod,diagonalMatrix,lgeMatrix-method
(matrix-products), 99
crossprod,diagonalMatrix,matrix-method
(matrix-products), 99
crossprod,diagonalMatrix,missing-method
(matrix-products), 99
crossprod,diagonalMatrix,sparseMatrix-method
(matrix-products), 99
crossprod,dsparseMatrix,ddenseMatrix-method
(matrix-products), 99
crossprod,dsparseMatrix,dgeMatrix-method
(matrix-products), 99
crossprod,dtpMatrix,ddenseMatrix-method
(matrix-products), 99
crossprod,dtpMatrix,matrix-method
(matrix-products), 99
crossprod,dtrMatrix,ddenseMatrix-method
(matrix-products), 99
crossprod,dtrMatrix,dtrMatrix-method
(matrix-products), 99
crossprod,dtrMatrix,matrix-method
(matrix-products), 99
crossprod,dtrMatrix,missing-method
(matrix-products), 99
crossprod,indMatrix,indMatrix-method
(matrix-products), 99
crossprod,indMatrix,Matrix-method
(matrix-products), 99
crossprod,indMatrix,matrix-method
(matrix-products), 99
crossprod,indMatrix,missing-method
(matrix-products), 99
crossprod,ldenseMatrix,ddenseMatrix-method
(matrix-products), 99
crossprod,ldenseMatrix,ldenseMatrix-method
(matrix-products), 99
crossprod,ldenseMatrix,lsparseMatrix-method
(matrix-products), 99
crossprod,ldenseMatrix,matrix-method
(matrix-products), 99
crossprod,ldenseMatrix,missing-method
(matrix-products), 99
crossprod,ldenseMatrix,ndenseMatrix-method
(matrix-products), 99
crossprod,lgCMatrix,missing-method
(matrix-products), 99
crossprod,lgTMatrix,missing-method
(matrix-products), 99
crossprod,lsparseMatrix,ldenseMatrix-method
(matrix-products), 99
crossprod,lsparseMatrix,lsparseMatrix-method
(matrix-products), 99
crossprod,lsparseMatrix,missing-method
(matrix-products), 99
crossprod,Matrix,ANY-method
(matrix-products), 99
crossprod,matrix,CsparseMatrix-method
(matrix-products), 99
crossprod,matrix,dgeMatrix-method
(matrix-products), 99
crossprod,matrix,diagonalMatrix-method
(matrix-products), 99
crossprod,matrix,dtrMatrix-method
(matrix-products), 99
crossprod,matrix,Matrix-method
(matrix-products), 99
crossprod,matrix,missing-method
(matrix-products), 99
crossprod,matrix,numLike-method
(matrix-products), 99
crossprod,matrix,TsparseMatrix-method
(matrix-products), 99
crossprod,mMatrix,sparseVector-method
determinant,dgCMatrix,logical-method (sparseMatrix-class), 146

determinant,dgCMatrix,logical-method (dgCMatrix-class), 146

determinant,dgCMatrix,missing-method (dgCMatrix-class), 146

determinant,diagonalMatrix,logical-method (diagonalMatrix-class), 42

determinant,dpoMatrix,logical-method (dpoMatrix-class), 46

determinant,dpoMatrix,missing-method (dpoMatrix-class), 46

determinant,dppMatrix,logical-method (dpoMatrix-class), 46

determinant,dppMatrix,missing-method (dpoMatrix-class), 46

determinant,dtrMatrix,logical-method (dtrMatrix-class), 59

determinant,dtrMatrix,missing-method (dtrMatrix-class), 59

determinant,indMatrix,logical-method (indMatrix-class), 73

determinant,Matrix,logical-method (Matrix-class), 97

determinant,Matrix,missing-method (Matrix-class), 97

determinant,pMatrix,logical-method (pMatrix-class), 116

determinant,sparseMatrix,logical-method (sparseMatrix-class), 146

determinant,sparseMatrix,missing-method (sparseMatrix-class), 146

dgCMatrix, 15, 18–20, 33, 38–40, 46, 50, 51, 53, 55, 59, 88, 91, 98, 100, 113, 122, 136, 141, 149, 150

dgCMatrix-class, 36

dgeMatrix, 33, 34, 46, 47, 50, 53–55, 59, 91, 94, 98, 149, 150

dgeMatrix-class, 37

dgRMATRIX-class, 51, 133

dgTMatrix-class, 39
diag, 11, 42, 73, 104
diag,CsparseMatrix-method (CsparseMatrix-class), 32

diag,ddenseMatrix-method (ddenseMatrix-class), 33

diag,dgCMATRIX-method (dgCMatrix-class), 36

diag,dgEMATRIX-method (dgeMatrix-class), 37

diag,dgRMATRIX-method (dgRMATRIX-class), 38
diag,diagonalMatrix-method (diagonalMatrix-class), 42
diag,dspMatrix-method (dspMatrix-class), 53
diag,dsyMatrix-method (dsyMatrix-class), 53

diag,dtpMatrix-method (dtpMatrix-class), 56

diag,dsyMatrix-method (dsyMatrix-class), 53

diag,dsyMatrix-method (dsyMatrix-class), 88
diag,ldenseMatrix-method (ldenseMatrix-class), 84

diag,ldenseMatrix-method (ldenseMatrix-class), 85

diag,lsyMatrix-method (lsyMatrix-class), 88
diag,lsyMatrix-method (lsyMatrix-class), 88

diag,ltpMatrix-method (ltpMatrix-class), 89

diag,ltMatrix-method (ltMatrix-class), 89

diag,ltMatrix-method (ltMatrix-class), 89

diag,Matrix-method (Matrix-class), 97
diag,ndenseMatrix-method (ndenseMatrix-class), 103
diag,sparseMatrix-method (sparseMatrix-class), 146
diag<-,dgeMatrix-method (dgeMatrix-class), 37
INDEX

diag<-, dspMatrix-method
   (dsyMatrix-class), 53
diag<-, dsyMatrix-method
   (dsyMatrix-class), 53
diag<-, dtpMatrix-method
   (dtpMatrix-class), 56
diag<-, dtrMatrix-method
   (dtrMatrix-class), 59
diag<-, lgeMatrix-method
   (lgeMatrix-class), 85
diag<-, lspMatrix-method
   (lsyMatrix-class), 88
diag<-, lsyMatrix-method
   (lsyMatrix-class), 88
diag<-, ltpMatrix-method
   (ltrMatrix-class), 89
diag<-, ltrMatrix-method
   (ltrMatrix-class), 89
diag<-, ngeMatrix-method
   (ngeMatrix-class), 107
diag<-, nspMatrix-method
   (nsyMatrix-class), 114
diag<-, nsysMatrix-method
   (nsyMatrix-class), 114
diag<-, ntpMatrix-method
   (ntrMatrix-class), 115
diag<-, ntrMatrix-method
   (ntrMatrix-class), 115
diagN2U (diagU2N), 44
Diagonal, 12, 24, 35, 41, 43, 85, 96, 144, 145, 155
diagonalMatrix, 12, 34, 35, 41, 42, 68, 80, 85, 96
diagonalMatrix-class, 42
diagU2N, 44
diff, 98, 130
diff, Matrix-method (Matrix-class), 97
dim, 98, 120, 131, 161
dim, dgCMatrix-method (dgCMatrix-class), 36
dim, dgEMatrix-method (dgeMatrix-class), 37
dim, dgRMatrix-method (dgRMatrix-class), 38
dim, Matrix-method (Matrix-class), 97
dim, MatrixFactorization-method
   (MatrixFactorization-class), 102
dim<-, denseMatrix-method
   (denseMatrix-class), 35
dim<-, Matrix-method (Matrix-class), 97
dim<-, sparseMatrix-method
   (sparseMatrix-class), 146
dim<-, sparseVector-method
   (sparseVector-class), 151
dimnames, 28, 34, 42, 46, 67, 78, 80, 85, 91, 96, 98, 143, 156
dimnames, dgeMatrix-method
   (dgeMatrix-class), 37
dimnames, Matrix-method (Matrix-class), 97
dimnames, symmetricMatrix-method
   (symmetricMatrix-class), 156
dimnames<-, compMatrix, list-method
   (compMatrix-class), 29
dimnames<-, compMatrix, NULL-method
   (compMatrix-class), 29
dimnames<-, Matrix, list-method
   (Matrix-class), 97
dimnames<-, Matrix, NULL-method
   (Matrix-class), 97
dMatrix, 34, 61
dMatrix-class, 45
dnearneigh, 167
double, 130, 139
dpoMatrix, 21, 26, 27, 54, 70, 104, 105
dpoMatrix-class, 46
dppMatrix, 54
dppMatrix-class (dpoMatrix-class), 46
drop, abIndex-method (abIndex-class), 4
drop, Matrix-method (Matrix-class), 97
drop0, 19, 46, 48, 110, 151
dsCMatrix, 18, 19, 24, 36, 41, 165, 166
dsCMatrix-class, 49
dspMatrix, 18, 34, 40, 41, 50, 52
dspMatrix-class, 51
dspMatrix-class (sparseVector-class), 151
dspMatrix-class (dmsyMatrix-class), 53
dsRMatrix-class, 52
dsTMatrix, 167
dsTMatrix-class (dsCMatrix-class), 49
dsyMatrix, 26, 38, 47, 157
dsyMatrix-class, 53
dtCMatrix, 36, 43, 141
dtCMatrix-class, 54
INDEX

dtpMatrix, 60
dtpMatrix-class, 56
dtrMatrix, 26, 27, 38, 55, 57, 93, 159
dtrMatrix-class, 58
dTRMatrix-class, 59
dtTMatrix-class (dtCMatrix-class), 54
eigen, 15, 98
eigen, dgeMatrix, logical-method (dgeMatrix-class), 37
eigen, dgeMatrix, missing-method (dgeMatrix-class), 37
eigen, Matrix, ANY, logical-method (dgeMatrix-class), 37
eigen, Matrix, ANY, missing-method (dgeMatrix-class), 37
expand, 24, 60, 92, 94
expand, CHMfactor-method (CHMfactor-class), 17
expand, denseLU-method (LU-class), 93
expand, MatrixFactorization-method (MatrixFactorization-class), 102
expand, sparseLU-method (sparseLU-class), 140
expm, 46, 61, 62
expm, ddiMatrix-method (expm), 61
expm, dgeMatrix-method (expm), 61
expm, dMatrix-method (expm), 61
expm, Matrix-method (expm), 61
expm, matrix-method (expm), 61
expm, symmetricMatrix-method (expm), 61
expm, triangularMatrix-method (expm), 61
extends, 96
externalFormats, 62
Extract, 168, 169
fac2Sparse (sparse.model.matrix), 138
fac2sparse, 139
facmul, 61, 64
factor, 138, 139
FALSE, 78
forceSymmetric, 65, 79
forceSymmetric, corMatrix, character-method (forceSymmetric), 65
forceSymmetric, corMatrix, missing-method (forceSymmetric), 65
forceSymmetric, CsparseMatrix, ANY-method (forceSymmetric), 65
forceSymmetric, denseMatrix, character-method (forceSymmetric), 65
forceSymmetric, denseMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsCMatrix, character-method (forceSymmetric), 65
forceSymmetric, dsCMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsCMatrix, character-method (forceSymmetric), 65
forceSymmetric, dsCMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsRMatrix, character-method (forceSymmetric), 65
forceSymmetric, dsRMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsTMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsyMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, character-method (forceSymmetric), 65
forceSymmetric, dpoMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, dsyMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, character-method (forceSymmetric), 65
forceSymmetric, dppMatrix, missing-method (forceSymmetric), 65
forceSymmetric, lsCMatrix, character-method (forceSymmetric), 65
forceSymmetric, lsCMatrix, missing-method (forceSymmetric), 65
forceSymmetric, lspMatrix, character-method (forceSymmetric), 65
forceSymmetric, lspMatrix, missing-method (forceSymmetric), 65
forceSymmetric, lsRMatrix, character-method (forceSymmetric), 65
forceSymmetric, lsRMatrix, missing-method (forceSymmetric), 65
forceSymmetric, lsTMatrix, character-method (forceSymmetric), 65

forceSymmetric, lSTMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, lSyMatrix, character-method
(forceSymmetric), 65
forceSymmetric, lSyMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, matrix, ANY-method
(forceSymmetric), 65
forceSymmetric, Matrix, missing-method
(forceSymmetric), 65
forceSymmetric, nCMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nCMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nRMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nRMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nTMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nTMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nsCMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nsCMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nsMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nsMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nsRMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nsRMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nsTMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nsTMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, nsyMatrix, character-method
(forceSymmetric), 65
forceSymmetric, nsyMatrix, missing-method
(forceSymmetric), 65
forceSymmetric, sparseMatrix, ANY-method
(forceSymmetric), 65
format, 66, 67, 119, 147
format, sparseMatrix-method
(sparseMatrix-class), 146
formatSparseM, 66, 120, 121
formatSpMatrix, 66, 67, 147
formatSpMatrix (printSpMatrix), 119
forwardsolve, 137
function, 80, 131
gamma, dgCMatrix-method
(dgCMatrix-class), 36
gamma, dMatrix-method (dMatrix-class), 45
generalMatrix, 74, 144, 155, 158
generalMatrix-class, 67
get.adjacency, 69
get.gpar, 71
getClassDef, 119
getOption, 119
getValidity, 156
graph, 68
graph-sparseMatrix, 68
graph.adjacency, 69
graph2T (graph-sparseMatrix), 68
grey, 71
grid.raster, 70
grid.rect, 71
head, 152
head, Matrix-method (Matrix-class), 97
head, sparseVector-method
(sparseVector-class), 151
Hilbert, 69
image, 70, 98
image, ANY-method (image-methods), 70
image, CHMfactor-method (image-methods), 70
image, dgCMatrix-method (image-methods), 70
image, dgRMatrix-method (image-methods), 70
image, dgTMatrix-method (image-methods), 70
image, dsparseMatrix-method
(image-methods), 70
image, lsparseMatrix-method
(image-methods), 70
image, Matrix-method (image-methods), 70
image, nsparseMatrix-method
(image-methods), 70
image-methods, 70
iMatrix-class (Unused-classes), 163
index, 169
index-class, 72
indMatrix, 117, 118
indMatrix-class, 73
integer, 9, 32, 39, 49, 55, 73, 110, 161
invisible, 120
invPerm, 75, 117, 118
is, 70
is.atomic, 8, 9
is.finite, 76, 77
is.finite, abIndex-method
(abIndex-class), 4
is.finite, ddenseMatrix-method
(is.na-methods), 76
is.finite, dgeMatrix-method (is.na-methods), 76
is.finite, diagonalMatrix-method (is.na-methods), 76
is.finite, dsparseMatrix-method (is.na-methods), 76
is.finite, indMatrix-method (is.na-methods), 76
is.finite, lMatrix-method (is.na-methods), 76
is.finite, nMatrix-method (is.na-methods), 76
is.finite, nsparseVector-method (sparseVector-class), 151
is.finite, sparseVector-method (sparseVector-class), 151
is.finite, 76, 77
is.finite, abIndex-method (abIndex-class), 4
is.finite, ddenseMatrix-method (is.na-methods), 76
is.finite, diagonalMatrix-method (is.na-methods), 76
is.finite, dsparseMatrix-method (is.na-methods), 76
is.finite, indMatrix-method (is.na-methods), 76
is.finite, lMatrix-method (is.na-methods), 76
is.finite, nMatrix-method (is.na-methods), 76
is.finite, nsparseVector-method (sparseVector-class), 151
is.finite, sparseVector-method (sparseVector-class), 151
is.infinite, 76, 77
is.infinite, abIndex-method (abIndex-class), 4
is.infinite, ddenseMatrix-method (is.na-methods), 76
is.infinite, diagonalMatrix-method (is.na-methods), 76
is.infinite, dsparseMatrix-method (is.na-methods), 76
is.infinite, indMatrix-method (is.na-methods), 76
is.infinite, lMatrix-method (is.na-methods), 76
is.infinite, nMatrix-method (is.na-methods), 76
is.infinite, nsparseVector-method (sparseVector-class), 151
is.infinite, sparseVector-method (sparseVector-class), 151
is.na, 76, 77
is.na, abIndex-method (abIndex-class), 4
is.na, denseMatrix-method (is.na-methods), 76
is.na, indMatrix-method (is.na-methods), 76
is.na, nMatrix-method (is.na-methods), 76
is.na, nsparseMatrix-method (nsparseMatrix-classes), 112
is.na, nsparseVector-method (sparseVector-class), 151
is.na, sparseMatrix-method (is.na-methods), 76
is.na, sparseVector-method (is.na-methods), 76
is.na, sparseVector-method (sparseVector-class), 151
is.null, 78
is.null, DN, 78
isDiagonal, 43
isDiagonal (isTriangular), 79
isDiagonal, CsparseMatrix-method (isTriangular), 79
isDiagonal, denseMatrix-method (isTriangular), 79
isDiagonal, diagonalMatrix-method (isTriangular), 79
isDiagonal, matrix-method (isTriangular), 79
isDiagonal, sparseMatrix-method (isTriangular), 79
isDiagonal, symmetricMatrix-method (isTriangular), 79
isDiagonal, triangularMatrix-method (isTriangular), 79
isDiagonal, TsparseMatrix-method (isTriangular), 79
isDiagonal-methods (isTriangular), 79
isLDL (CHMfactor-class), 17
isSparseVector-class (sparseVector-class), 151
isSymmetric, 79, 80, 105, 136, 157–159
isSymmetric, denseMatrix-method (isSymmetric-methods), 79
isSymmetric, dgCMatrix-method (dgCMatrix-class), 36
isSymmetric, diagonalMatrix-method (isSymmetric-methods), 79
isSymmetric, indMatrix-method (indMatrix-class), 73
isSymmetric, sparseMatrix-method (isSymmetric-methods), 79
isSymmetric, symmetricMatrix-method (isSymmetric-methods), 79
isSymmetric, triangularMatrix-method (isSymmetric-methods), 79
isSymmetric-methods, 79, 157
isTriangular, 79, 159
isTriangular, BunchKaufman-method (isTriangular), 79
isTriangular, Cholesky-method (isTriangular), 79
isTriangular, CsparseMatrix-method
lgamma, dgCMatrix-method
    (dgCMatrix-class), 36
lgamma, dMatrix-method (dMatrix-class), 45
lgCMatrix, 86
lgCMatrix-class (lsparseMatrix-classes), 86
lgeMatrix, 84, 89, 90
lgeMatrix-class, 85
lgrMatrix-class (lsparseMatrix-classes), 86
lgTMatrix-class (lsparseMatrix-classes), 86
list, 12, 29, 31, 42, 71, 74, 94, 98, 130, 133, 139
lMatrix, 84, 85, 87, 90, 96, 100, 103, 104, 109, 113
lMatrix-class (dMatrix-class), 45
log, CsparseMatrix-method (CsparseMatrix-class), 32
log, ddenseMatrix-method (ddenseMatrix-class), 33
log, denseMatrix-method (denseMatrix-class), 35
log, dgCMatrix-method (dgCMatrix-class), 36
log, dgeMatrix-method (dgeMatrix-class), 37
log, diagonalMatrix-method (diagonalMatrix-class), 42
log, dMatrix-method (dMatrix-class), 45
log, sparseMatrix-method (sparseMatrix-class), 146
log, sparseVector-method (sparseVector-class), 151
Logic, ANY, Matrix-method (Matrix-class), 97
Logic, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class), 32
Logic, dMatrix, logical-method (dMatrix-class), 45
Logic, dMatrix, numeric-method (dMatrix-class), 45
Logic, dMatrix, sparseVector-method (sparseVector-class), 151
Logic, ldenseMatrix, lsparseMatrix-method (ldenseMatrix-class), 84
Logic, lgCMatrix, lgCMatrix-method (lsparseMatrix-classes), 86
Logic, lgeMatrix, lgeMatrix-method (lgeMatrix-class), 85
Logic, lgTMatrix, lgTMatrix-method (lsparseMatrix-classes), 86
Logic, lMatrix, logical-method (dMatrix-class), 45
Logic, lMatrix, numeric-method (dMatrix-class), 45
Logic, lMatrix, sparseVector-method (sparseVector-class), 151
Logic, logical, dMatrix-method (dMatrix-class), 45
Logic, logical, lMatrix-method (dMatrix-class), 45
Logic, logical, Matrix-method (Matrix-class), 97
Logic, logical, nMatrix-method (nMatrix-class), 108
Logic, lsCMatrix, lsCMatrix-method (lsparseMatrix-classes), 86
Logic, ldenseMatrix, ldenseMatrix-method (ldenseMatrix-class), 84
Logic, lsparseMatrix, lsparseMatrix-method (lsparseMatrix-classes), 86
Logic, lsparseVector, lsparseVector-method (sparseVector-class), 151
Logic, ltCMatrix, ltCMatrix-method (lsparseMatrix-classes), 86
Logic, Matrix, ANY-method (Matrix-class), 97
Logic, Matrix, logical-method (Matrix-class), 97
Logic, Matrix, nMatrix-method (nMatrix-class), 108
Logic, ngeMatrix, ngeMatrix-method (ngeMatrix-class), 107
Logic, nMatrix, logical-method (nMatrix-class), 108
Logic, nMatrix, Matrix-method (nMatrix-class), 108
Logic, nMatrix, nMatrix-method (nMatrix-class), 108
Logic, nMatrix, numeric-method (nMatrix-class), 108
Logic, nMatrix, sparseVector-method (sparseVector-class), 151
Logic, nsparseVector, nsparseVector-method
INDEX

(sparsesVector-class), 151
Logic, numeric, dMatrix-method
(dMatrix-class), 45
Logic, numeric, IMatrix-method
(dMatrix-class), 45
Logic, numeric, nMatrix-method
(nMatrix-class), 108
Logic, sparseVector, dMatrix-method
(sparseVector-class), 151
Logic, sparseVector, IMatrix-method
(sparseVector-class), 151
Logic, sparseVector, nMatrix-method
(sparseVector-class), 151
Logic, triangularMatrix, sparseVector-method
(sparseVector-class), 151
Logic, triangularMatrix, diagonalMatrix-method
(diagonalMatrix-class), 42
logical, 19, 46, 47, 73, 78, 79, 85, 94, 96, 100–102, 110, 124, 152, 168
logm, 61, 62
lsCMatrix, 18, 41, 157
lsCMatrix-class
(lsparseMatrix-classes), 86
lsCMatrix-class
(lsparseMatrix-classes), 86
lsparseMatrix, 41
lsparseMatrix-class
(lsparseMatrix-classes), 86
lsparseMatrix-classes, 86
lsparseVector-class
(sparseVector-class), 151
lsyMatrix-class
(lsparseMatrix-classes), 86
lsyMatrix-class
(lsparseMatrix-classes), 86
ltCMatrix, 159
ltCMatrix-class
(lsparseMatrix-classes), 86
ltPMatrix-class
(ltrMatrix-class), 89
ltrMatrix, 86
ltrMatrix-class
(lsparseMatrix-classes), 86
ltrMatrix-class
(lsparseMatrix-classes), 86
ltTMatrix-class
(lsparseMatrix-classes), 86
LU, 103, 141
lu, 15, 29, 30, 36, 61, 90, 91, 93, 94, 136, 137, 141, 147
lu, ddenseMatrix-method
(ddenseMatrix-class), 33
lu, dgCMatrix-method (lu), 90
lu, dgeMatrix-method (lu), 90
lu, dsparseMatrix-method
(dsparseMatrix-class), 51
lu, dtCMatrix-method (lu), 90
lu, matrix-method (lu), 90
lu, sparseMatrix-method
(sparseMatrix-class), 146
LU-class, 93
mat2triplet, 94
Math, CsparseMatrix-method
(CsparseMatrix-class), 32
Math, ddenseMatrix-method
(ddenseMatrix-class), 33
Math, denseMatrix-method
(denseMatrix-class), 35
Math, dgeMatrix-method
(dgeMatrix-class), 37
Math, diagonalMatrix-method
(diagonalMatrix-class), 42
Math, sparseMatrix-method
(sparseMatrix-class), 146
Math, sparseVector-method
(sparseVector-class), 151
Math2, dMatrix-method (dMatrix-class), 45
Math2, dsparseVector-method
(sparseVector-class), 151
Math2, Matrix-method (Matrix-class), 97
Math2, sparseVector-method
(sparseVector-class), 151
matrix, 7, 27, 43, 65, 66, 74, 78, 94, 96, 98, 104, 110, 117, 133, 168, 169
Matrix-class, 97
matrix-products, 99
matrix.csr, 142
MatrixClass, 101
MatrixFactorization, 24, 26, 29, 93, 133, 135, 137, 141
MatrixFactorization-class, 102
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nsTMatrix, nsTMatrix-class</td>
<td>68</td>
</tr>
<tr>
<td>nsyMatrix, nsyMatrix-class</td>
<td>108</td>
</tr>
<tr>
<td>ntCMatrix, ntCMatrix-class</td>
<td>114</td>
</tr>
<tr>
<td>ntpMatrix-class</td>
<td>115</td>
</tr>
<tr>
<td>NULL</td>
<td>116</td>
</tr>
<tr>
<td>numeric</td>
<td>4, 5, 9, 27, 28, 39, 47, 51, 82, 110, 133</td>
</tr>
<tr>
<td>onenormest</td>
<td>112</td>
</tr>
<tr>
<td>Ops</td>
<td>5, 43, 60, 152</td>
</tr>
<tr>
<td>Ops, abIndex, abIndex-method</td>
<td>4</td>
</tr>
<tr>
<td>Ops, abIndex, numeric-method</td>
<td>4</td>
</tr>
<tr>
<td>Ops, ANY, abIndex-method(abIndex-class)</td>
<td>4</td>
</tr>
<tr>
<td>Ops, ANY, ddiMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ANY, ldMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ANY, Matrix-method(Matrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ANY, sparseVector-method(sparseVector-class)</td>
<td>97</td>
</tr>
<tr>
<td>Ops, atomicVector, sparseVector-method(sparseVector-class)</td>
<td>151</td>
</tr>
<tr>
<td>Ops, ddiMatrix, ANY-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ddiMatrix, dMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ddiMatrix, diagonalMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ddiMatrix, ldMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ddiMatrix, logical-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, dpoMatrix, logical-method(dpoMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, dpoMatrix, numeric-method(dpoMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, dppMatrix, logical-method(dpoMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, dppMatrix, numeric-method(dpoMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, dsparseMatrix, sparseVector-method(sparseVector-class)</td>
<td>151</td>
</tr>
<tr>
<td>Ops, ldMatrix, ldMatrix-method(ldMatrix-class)</td>
<td>84</td>
</tr>
<tr>
<td>Ops, ldMatrix, ANY-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ldMatrix, dMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ldMatrix, diagonalMatrix-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Ops, ldMatrix, logical-method(diagonalMatrix-class)</td>
<td>42</td>
</tr>
<tr>
<td>Term</td>
<td>Page(s)</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>(sparseVector-class)</td>
<td>151</td>
</tr>
<tr>
<td>options</td>
<td>108, 119, 120, 149, 152</td>
</tr>
<tr>
<td>order</td>
<td>76, 94, 161</td>
</tr>
<tr>
<td>outer</td>
<td>98</td>
</tr>
<tr>
<td>over</td>
<td>167</td>
</tr>
<tr>
<td>pack (unpack)</td>
<td>162</td>
</tr>
<tr>
<td>pack, matrix-method (unpack)</td>
<td>162</td>
</tr>
<tr>
<td>pack, symmetricMatrix-method (unpack)</td>
<td>162</td>
</tr>
<tr>
<td>pack, triangularMatrix-method (unpack)</td>
<td>162</td>
</tr>
<tr>
<td>panel.levelplot.raster</td>
<td>70</td>
</tr>
<tr>
<td>paste</td>
<td>139</td>
</tr>
<tr>
<td>PBunchKaufman-class (Cholesky-class)</td>
<td>26</td>
</tr>
<tr>
<td>pCholesky-class (Cholesky-class)</td>
<td>26</td>
</tr>
<tr>
<td>plot.default</td>
<td>70</td>
</tr>
<tr>
<td>pMatrix</td>
<td>73, 74, 76, 150</td>
</tr>
<tr>
<td>pMatrix-class</td>
<td>116</td>
</tr>
<tr>
<td>posdefify</td>
<td>104–106</td>
</tr>
<tr>
<td>print</td>
<td>43, 67, 70, 98, 119, 120, 147</td>
</tr>
<tr>
<td>print, diagonalMatrix-method</td>
<td>42</td>
</tr>
<tr>
<td>print, sparseMatrix-method</td>
<td>146</td>
</tr>
<tr>
<td>print.default</td>
<td>66, 119</td>
</tr>
<tr>
<td>print.Matrix (Matrix-class)</td>
<td>97</td>
</tr>
<tr>
<td>print.sparseMatrix</td>
<td>146</td>
</tr>
<tr>
<td>print.trellis</td>
<td>71</td>
</tr>
<tr>
<td>printSpMatrix</td>
<td>67, 98, 119, 147</td>
</tr>
<tr>
<td>printSpMatrix2</td>
<td>119</td>
</tr>
<tr>
<td>prod, ddiMatrix-method</td>
<td>42</td>
</tr>
<tr>
<td>prod, ldiMatrix-method</td>
<td>42</td>
</tr>
<tr>
<td>qr, 29, 92, 121, 122, 124, 125, 148–150</td>
<td></td>
</tr>
<tr>
<td>qr (qr-methods)</td>
<td>121</td>
</tr>
<tr>
<td>qr, ddenseMatrix-method (qr-methods)</td>
<td>121</td>
</tr>
<tr>
<td>qr, denseMatrix-method (qr-methods)</td>
<td>121</td>
</tr>
<tr>
<td>qr, dgCMatrix-method (qr-methods)</td>
<td>121</td>
</tr>
<tr>
<td>qr, sparseMatrix-method (qr-methods)</td>
<td>121</td>
</tr>
<tr>
<td>qr-methods</td>
<td>121</td>
</tr>
<tr>
<td>qr.coef, 137, 150</td>
<td></td>
</tr>
<tr>
<td>qr.coef, sparseQR, ddenseMatrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.coef, sparseQR, matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.coef, sparseQR, numeric-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.fitted</td>
<td>150</td>
</tr>
<tr>
<td>qr.fitted, sparseQR, ddenseMatrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.fitted, sparseQR, Matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.fitted, sparseQR, matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.fitted, sparseQR, numeric-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.Q, 150</td>
<td></td>
</tr>
<tr>
<td>qr.Q (sparseQR-class)</td>
<td>148</td>
</tr>
<tr>
<td>qr.Q, sparseQR-method (sparseQR-class)</td>
<td>148</td>
</tr>
<tr>
<td>qr.qty, 150</td>
<td></td>
</tr>
<tr>
<td>qr.qty, sparseQR, ddenseMatrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qty, sparseQR, Matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qty, sparseQR, matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qty, sparseQR, numeric-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qy, 150</td>
<td></td>
</tr>
<tr>
<td>qr.qy, sparseQR, ddenseMatrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qy, sparseQR, Matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qy, sparseQR, matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.qy, sparseQR, numeric-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.R, 150</td>
<td></td>
</tr>
<tr>
<td>qr.R, sparseQR-method (sparseQR-class)</td>
<td>148</td>
</tr>
<tr>
<td>qr.resid</td>
<td>150</td>
</tr>
<tr>
<td>qr.resid, sparseQR, ddenseMatrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.resid, sparseQR, Matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.resid, sparseQR, matrix-method</td>
<td>148</td>
</tr>
<tr>
<td>qr.resid, sparseQR, numeric-method</td>
<td>148</td>
</tr>
</tbody>
</table>
(sparseQR-class), 148
qr2rankMatrix (rankMatrix), 123
qrR (qr-methods), 121
range, 46
rankMatrix, 123
rBind (cBind), 16
rbind, 16, 147
rbind2, 16
rbind2, ANY, Matrix-method
   (Matrix-class), 97
rbind2, atomicVector, ddiMatrix-method
   (diagonalMatrix-class), 42
rbind2, atomicVector, ldiMatrix-method
   (diagonalMatrix-class), 42
rbind2, atomicVector, Matrix-method
   (Matrix-class), 97
rbind2, ddiMatrix, atomicVector-method
   (diagonalMatrix-class), 42
rbind2, ddiMatrix, matrix-method
   (diagonalMatrix-class), 42
rbind2, denseMatrix, denseMatrix-method
   (denseMatrix-class), 35
rbind2, denseMatrix, matrix-method
   (denseMatrix-class), 35
rbind2, denseMatrix, numeric-method
   (denseMatrix-class), 35
rbind2, denseMatrix, sparseMatrix-method
   (sparseMatrix-class), 146
rbind2, diagonalMatrix, sparseMatrix-method
   (diagonalMatrix-class), 42
rbind2, indMatrix, indMatrix-method
   (indMatrix-class), 73
rbind2, ldiMatrix, atomicVector-method
   (diagonalMatrix-class), 42
rbind2, ldiMatrix, matrix-method
   (diagonalMatrix-class), 42
rbind2, Matrix, ANY-method
   (Matrix-class), 97
rbind2, Matrix, atomicVector-method
   (Matrix-class), 97
rbind2, matrix, ddiMatrix-method
   (diagonalMatrix-class), 42
rbind2, matrix, denseMatrix-method
   (denseMatrix-class), 35
rbind2, matrix, ldiMatrix-method
   (diagonalMatrix-class), 42
rbind2, Matrix, Matrix-method
   (Matrix-class), 97
rbind2, Matrix, missing-method
   (Matrix-class), 97
rbind2, matrix, NULL-method
   (Matrix-class), 97
rbind2, matrix, sparseMatrix-method
   (sparseMatrix-class), 146
rbind2, NULL, Matrix-method
   (Matrix-class), 97
rbind2, numeric, denseMatrix-method
   (denseMatrix-class), 35
rbind2, numeric, sparseMatrix-method
   (sparseMatrix-class), 146
rbind2, sparseMatrix, denseMatrix-method
   (sparseMatrix-class), 146
rbind2, sparseMatrix, diagonalMatrix-method
   (diagonalMatrix-class), 42
rbind2, sparseMatrix, matrix-method
   (sparseMatrix-class), 146
rbind2, sparseMatrix, numeric-method
   (sparseMatrix-class), 146
rbind2, sparseMatrix, sparseMatrix-method
   (sparseMatrix-class), 146
rcond, 31, 38, 47, 54, 126
rcond, ANY, missing-method (rcond), 126
rcond, ddenseMatrix, character-method
   (ddenseMatrix-class), 33
rcond, ddenseMatrix, missing-method
   (ddenseMatrix-class), 33
rcond, denseMatrix, character-method
   (denseMatrix-class), 35
rcond, denseMatrix, character-method
   (denseMatrix-class), 35
rcond, denseMatrix, missing-method
   (denseMatrix-class), 35
rcond, dgeMatrix, character-method
   (dgeMatrix-class), 37
rcond, dgeMatrix, missing-method
   (dgeMatrix-class), 37
rcond, dpoMatrix, character-method
   (dpoMatrix-class), 46
rcond, dpoMatrix, missing-method
   (dpoMatrix-class), 46
rcond, dppMatrix, character-method
   (dppMatrix-class), 46
rcond, dppMatrix, missing-method
   (dppMatrix-class), 46
rcond, dspMatrix, character-method
   (dsyMatrix-class), 53
rcond, dspMatrix, missing-method
   (dsyMatrix-class), 53
rcond, dsyMatrix, character-method
   (dsyMatrix-class), 53
INDEX

rcond,dsyMatrix,missing-method (dsyMatrix-class), 53
rcond,dtpMatrix,character-method (dtpMatrix-class), 56
rcond,dtpMatrix,missing-method (dtpMatrix-class), 56
rcond,dtrMatrix,character-method (dtrMatrix-class), 59
rcond,dtrMatrix,missing-method (dtrMatrix-class), 59
rcond,ldenseMatrix,character-method (rcond), 126
rcond,Matrix,character-method (rcond), 126
rcond,ndenseMatrix,character-method (rcond), 126
rcond,sparseMatrix,character-method (rcond), 126
rcond,igCMatrix-method (colSums), 27
rowMeans,igCMatris-method (colSums), 27
rowMeans,indMatrix-method (indMatrix-class), 73
rowMeans,lgCMatrix-method (colSums), 27
rowMeans,ngCMatrix-method (colSums), 27
rowMeans,RsparseMatrix-method (colSums), 27
rowMeans,TsparseMatrix-method (colSums), 27
rownames, 122
rowSums (colSums), 27
rowSums,CsparseMatrix-method (colSums), 27
rowSums,ddenseMatrix-method (colSums), 27
rowSums,denseMatrix-method (colSums), 27
rowSums,dgCMatrix-method (colSums), 27
rowSums,dgeMatrix-method (dgeMatrix-class), 37
rowSums,diagonalMatrix-method (colSums), 27
rowSums,indMatrix-method (indMatrix-class), 73
rowSums,lgCMatrix-method (colSums), 27
rowSums,ngCMatrix-method (colSums), 27
rowSums,RsparseMatrix-method (colSums), 27
rowSums,TsparseMatrix-method (colSums), 27
RsparseMatrix, 11, 39, 52, 86, 112, 139, 143, 144
rsparsematrix, 131, 145
RsparseMatrix-class, 132
sample.int, 131
Schur, 29, 62, 133, 134, 135, 159
Schur,dgeMatrix,logical-method (Schur), 133
Schur,dgeMatrix,missing-method (Schur), 133
Schur,diagonalMatrix,logical-method (Schur), 133
Schur,diagonalMatrix,missing-method (Schur), 133
Schur,dsmatrix,ANY-method (Schur), 133
Schur,generalMatrix,logical-method (Schur), 133
Schur, generalMatrix, missing-method (Schur), 133
Schur, matrix, logical-method (Schur), 133
Schur, matrix, missing-method (Schur), 133
Schur, symmetricMatrix, logical-method (Schur), 133
Schur, symmetricMatrix, missing-method (Schur), 133
Schur, triangularMatrix, logical-method (Schur), 133
Schur, triangularMatrix, missing-method (Schur), 133
Schur-class, 134
seq, 6
seqMat-class (abIndex-class), 4
set.seed, 30
setClassUnion, 72, 116, 129, 152
show, 5, 67, 98, 103, 119, 120, 130, 147, 152
show, abIndex-method (abIndex-class), 4
show, BunchKaufman-method (Cholesky-class), 26
show, ddenseMatrix-method (ddenseMatrix-class), 33
show, denseMatrix-method (denseMatrix-class), 35
show, diagonalMatrix-method (diagonalMatrix-class), 42
show, dMatrix-method (dMatrix-class), 45
show, dsyMatrix-method (dsyMatrix-class), 53
show, dtrMatrix-method (dtrMatrix-class), 59
show, Matrix-method (Matrix-class), 97
show, MatrixFactorization-method (MatrixFactorization-class), 102
show, nMatrix-method (nMatrix-class), 108
show, pBunchKaufman-method (Cholesky-class), 26
show, rleDiff-method (rleDiff-class), 130
show, sparseMatrix-method (sparseMatrix-class), 146
show, sparseVector-method (sparseVector-class), 151
showClass, 34, 53, 89, 90, 114, 115
showMethods, 21, 24, 34, 35, 60, 86, 89–91, 108, 114, 115, 126, 158, 163
signif, 45
signif, dgCMatrix-method (dgCMatrix-class), 36
similar.listw, 165, 167
skewpart (symmpart), 157
skewpart, ddenseMatrix-method (symmpart), 157
skewpart, denseMatrix-method (symmpart), 157
skewpart, diagonalMatrix-method (symmpart), 157
skewpart, Matrix-method (symmpart), 157
skewpart, matrix-method (symmpart), 157
skewpart, symmetricMatrix-method (symmpart), 157
skewpart-methods (symmpart), 157
Sobj_SpatialGrid, 167
solve, 23, 24, 32, 37, 47, 54, 117, 127, 135–137, 141
solve (solve-methods), 135
solve, ANY, Matrix-method (solve-methods), 135
solve, CHMfactor, ANY-method (solve-methods), 135
solve, CHMfactor, ddenseMatrix-method (solve-methods), 135
solve, CHMfactor, diagonalMatrix-method (solve-methods), 135
solve, CHMfactor, dMatrix-method (solve-methods), 135
solve, CHMfactor, matrix-method (solve-methods), 135
solve, CHMfactor, missing-method (solve-methods), 135
solve, CHMfactor, numeric-method (solve-methods), 135
solve, ddenseMatrix, ANY-method (solve-methods), 135
solve, ddenseMatrix, Matrix-method (solve-methods), 135
solve, ddenseMatrix, matrix-method (solve-methods), 135
solve, ddenseMatrix, missing-method (solve-methods), 135
solve, ddenseMatrix, numeric-method (solve-methods), 135
solve, denseU, missing-method (solve-methods), 135
solve, dgCMatrix, ddenseMatrix-method
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>solve, dgCMatrix, dsparseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgCMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgCMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgeMatrix, ddenseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgeMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgeMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dgeMatrix, sparseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, diagonalMatrix, Matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, diagonalMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, diagonalMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dpoMatrix, dgeMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dpoMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dpoMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dppMatrix, dgeMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dppMatrix, integer-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dppMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dppMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, ddenseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, denseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, dsparseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dsCMatrix, numeric-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dspMatrix, ddenseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dspMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dspMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtpMatrix, ddenseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtpMatrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtpMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtrMatrix, ddenseMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtrMatrix, dMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtrMatrix, Matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, dtrMatrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, Matrix, ANY-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, Matrix, diagonalMatrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, Matrix, matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, matrix, Matrix-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
<tr>
<td>solve, Matrix, missing-method</td>
<td>(solve-methods), 135</td>
<td></td>
</tr>
</tbody>
</table>
INDEX

(symmpart), 157
symmpart,denseMatrix-method (symmpart), 157
symmpart,diagonalMatrix-method (symmpart), 157
symmpart,Matrix-method (symmpart), 157
symmpart,matrix-method (symmpart), 157
symmpart,symmetricMatrix-method (symmpart), 157
symmpart-methods (symmpart), 157
t, 54, 86, 89, 90, 99, 108, 114–117, 124, 139
t,CsparseMatrix-method (CsparseMatrix-class), 32
t,ddenseMatrix-method (ddenseMatrix-class), 33
t,dgCMatrix-method (dgCMatrix-class), 36
t,dgeMatrix-method (dgeMatrix-class), 37
t,dgRMatrix-method (dgRMatrix-class), 38
t,diagonalMatrix-method (diagonalMatrix-class), 42
t,dppMatrix-method (dppMatrix-class), 46
t,dsCMatrix-method (dsCMatrix-class), 49
t,dspMatrix-method (dspMatrix-class), 53
t,dstMatrix-method (dstMatrix-class), 49
t,dsyMatrix-method (dsyMatrix-class), 53
t,dtCMatrix-method (dtCMatrix-class), 54
t,dtpMatrix-method (dtpMatrix-class), 56
t,dtrMatrix-method (dtrMatrix-class), 59
t,dtMatrix-method (dtMatrix-class), 54
t,indMatrix-method (indMatrix-class), 73
(t,lgCMatrix-method (lgCMatrix-class), 86
t,lgematrix-method (lgematrix-class), 85
t,lgTMatrix-method (lgCMatrix-class), 86
t,lsCMatrix-method (lsCMatrix-class), 86
t,lspMatrix-method (lspMatrix-class), 88
t,lsTMatrix-method (lsTMatrix-class), 86
t,lsyMatrix-method (lsyMatrix-class), 88
t,ltCMatrix-method (ltCMatrix-class), 86
t,ltPMatrix-method (ltPMatrix-class), 86
t,ltRMatrix-method (ltRMatrix-class), 89
t,ltTMatrix-method (ltTMatrix-class), 89
t,lsyMatrix-method (lsyMatrix-class), 86
t,Matrix-method (Matrix-class), 97

t,ngCMatrix-method (nsparseMatrix-classes), 112
t,ngeMatrix-method (ngeMatrix-class), 107
t,ngTMatrix-method (nsparseMatrix-classes), 112
t,nsCMatrix-method (nsparseMatrix-classes), 112
t,nsMatrix-method (nsyMatrix-class), 114
t,nsTMatrix-method (nsparseMatrix-classes), 112
t,nsyMatrix-method (nsyMatrix-class), 114
t,ntCMatrix-method (nsparseMatrix-classes), 112
t,ntMatrix-method (ntMatrix-class), 115
t,ntTMatrix-method (nsparseMatrix-classes), 112
t,pMatrix-method (pMatrix-class), 116
t,RsparseMatrix-method (RsparseMatrix-class), 132
t,sparseVector-method (sparseVector-class), 151
t,TsparseMatrix-method (TsparseMatrix-class), 159
T2graph, 147
T2graph (graph-sparseMatrix), 68
tail,Matrix-method (Matrix-class), 97
tail,sparseVector-method (sparseVector-class), 151
tcrossprod, 99, 101
tcrossprod (matrix-products), 99
tcrossprod,ANY,Matrix-method (matrix-products), 99
tcrossprod,ANY,Matrix-method (matrix-products), 99
tcrossprod,ANY,symmetricMatrix-method (matrix-products), 99
tcrossprod,ANY,TsparseMatrix-method (matrix-products), 99
tcrossprod,CsparseMatrix,CsparseMatrix-method (matrix-products), 99
tcrossprod,CsparseMatrix,ddenseMatrix-method (matrix-products), 99
tcrossprod,CsparseMatrix,diagonalMatrix-method (matrix-products), 99

t,Matrix-method (Matrix-class), 97
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcrossprod, CsparseMatrix, matrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, CsparseMatrix, missing-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, CsparseMatrix, numLike-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, CsparseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, ddenseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, dsCMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, dtrMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, ldenseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, lsCMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, matrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, missing-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, ddenseMatrix, ndenseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, dgeMatrix, dgeMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, dgeMatrix, diagMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, dgeMatrix, dtrMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, dgeMatrix, numLike-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, diagonalMatrix, CsparseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, diagonalMatrix, diagonalMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, diagonalMatrix, sparseMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
<tr>
<td>tcrossprod, dtrMatrix, dtrMatrix-method</td>
<td>(matrix-products), 99</td>
<td></td>
</tr>
</tbody>
</table>
tril, ntRMatrix-method (band), 9
tril, ntrMatrix-method (band), 9
tril, ntTMatrix-method (band), 9
tril, RsparseMatrix-method (band), 9
tril, TsparseMatrix-method (band), 9
tril-methods (band), 9
triu (band), 9
triu, CsparseMatrix-method (band), 9
triu, dDenseMatrix-method (band), 9
triu, dsCMatrix-method (band), 9
triu, dtCMatrix-method (band), 9
triu, dtpMatrix-method (band), 9
triu, dtRMatrix-method (band), 9
triu, dtTMatrix-method (band), 9
triu, itCMatrix-method (band), 9
triu, itRMatrix-method (band), 9
triu, lsCMatrix-method (band), 9
triu, lDenseMatrix-method (band), 9
triu, ltrMatrix-method (band), 9
triu, ltrMMatrix-method (band), 9
triu, lsTMatrix-method (band), 9
triu, RsparseMatrix-method (band), 9
triu, TsparseMatrix-method (band), 9
triu-methods (band), 9
TRUE, 41, 46, 78, 149
TsparseMatrix, 11, 12, 40, 50, 68, 69, 86, 87, 94, 112, 139, 143, 144, 154, 155, 160, 161
TsparseMatrix-class, 159
type, 168
uniqTsparse, 40, 87, 94, 161
unname, Matrix, missing-method (Matrix-class), 97
unname, Matrix-method (Matrix-class), 97
unpack, 162
unpack, dspMatrix-method (unpack), 162
unpack, dtPMatrix-method (unpack), 162
unpack, sparseMatrix-method (unpack), 162
unpack, symmetricMatrix-method (unpack), 162
unpack, triangularMatrix-method (unpack), 162
Unused-classes, 163
update, 19
update, CHMfactor-method (CHMfactor-class), 17
updown, 164
updown, ANY, ANY, ANY-method (updown), 164
updown, character, mMatrix, CHMfactor-method (updown), 164
updown, logical, mMatrix, CHMfactor-method (updown), 164
updown-methods (updown), 164
USCounties, 165
validObject, 33
warning, 91, 108
which, 43, 46, 84, 87, 104, 113
which, ldenseMatrix-method (ldenseMatrix-class), 84
which, ldiMatrix-method (diagonalMatrix-class), 42
which, lgTMatrix-method (lTMatrix-class), 86
which, lMatrix-method (dMatrix-class), 45
which, lSparseMatrix-method (lSparseMatrix-class), 86
which, lSparseVector-method (sparseVector-class), 151
which, lTMatrix-method (lTMatrix-class), 86
which, ltTMatrix-method (ltTMatrix-class), 86
which, ndenseMatrix-method (ndenseMatrix-class), 103
which, ngTMatrix-method (nsparseMatrix-class), 112
which, nsparseMatrix-method (nsparseMatrix-class), 112
which, nsparseVector-method (sparseVector-class), 151
which, nsTMatrix-method (nsparseMatrix-class), 112
which, ntTMatrix-method
   (nsparseMatrix-classes), 112
writeMM, 147
writeMM(externalFormats), 62
writeMM, CsparseMatrix-method
   (externalFormats), 62
writeMM, sparseMatrix-method
   (externalFormats), 62
wrld_1deg, 166

xsparseVector-class
   (sparseVector-class), 151
xtabs, 145, 147

zapsmall, 48, 110
zapsmall, dMatrix-method
   (dMatrix-class), 45
zMatrix-class (Unused-classes), 163
zsparseVector-class
   (sparseVector-class), 151