Package ‘Matrix’

April 26, 2024

Version 1.7-0

VersionNote do also bump src/version.h, inst/include/Matrix/version.h

Date 2024-03-16

Priority recommended

Title Sparse and Dense Matrix Classes and Methods

Description A rich hierarchy of sparse and dense matrix classes, including general, symmetric, triangular, and diagonal matrices with numeric, logical, or pattern entries. Efficient methods for operating on such matrices, often wrapping the 'BLAS', 'LAPACK', and 'SuiteSparse' libraries.

License GPL (>= 2) | file LICENCE


BugReports https://R-forge.R-project.org/tracker/?atid=294&group_id=61

Contact Matrix-authors@R-project.org

Depends R (>= 4.4.0), methods

Imports grDevices, graphics, grid, lattice, stats, utils

Suggests MASS, datasets, sfsmisc, tools

Enhances SparseM, graph

LazyData no

LazyDataNote not possible, since we use data/*.R and our S4 classes

BuildResaveData no

Encoding UTF-8

NeedsCompilation yes

Author Douglas Bates [aut] (<https://orcid.org/0000-0001-8316-9503>), Martin Maechler [aut, cre] (<https://orcid.org/0000-0002-8685-9910>), Mikael Jagan [aut] (<https://orcid.org/0000-0002-3542-2938>), Timothy A. Davis [ctb] (<https://orcid.org/0000-0001-7614-6899>), SuiteSparse libraries, collaborators listed in dir(system.file("`doc`, `SuiteSparse", package="`Matrix`"),
R topics documented:

abIndex-class .................................................. 5
abIseq .......................................................... 6
all.equal-methods .............................................. 7
asUniqueT ......................................................... 8
band-methods ..................................................... 9
bandSparse ....................................................... 11
bdiag ............................................................. 13
boolmatmult-methods ........................................... 15
BunchKaufman-class ........................................... 16
BunchKaufman-methods ....................................... 19
CAex ............................................................. 20
cbind2-methods ................................................ 21
CHMfactor-class ............................................... 22
chol-methods .................................................... 26
chol2inv-methods .............................................. 30
Cholesky-class .................................................. 31
Cholesky-methods .............................................. 34
coerce-methods-graph ......................................... 40
coerce-methods-SparseM ..................................... 41
colSums-methods .............................................. 42
condest .......................................................... 43
CsparseMatrix-class ......................................... 45
denseMatrix-class ............................................ 47
ddiMatrix-class ............................................... 48
denseLU-class .................................................. 49
denseMatrix-class ............................................. 51
dgCMatrix-class ................................................ 52
dgeMatrix-class ................................................ 53
dgRMatrix-class ............................................... 54
dgTMatrix-class ................................................ 55
Diagonal .......................................................... 56
diagonalMatrix-class ......................................... 58
diagU2N .......................................................... 59
R topics documented:

dimScale ......................................................... 61
dMatrix-class ................................................ 62
dmperm .......................................................... 63
dpoMatrix-class .............................................. 65
drop0 ............................................................. 67
dscMatrix-class .............................................. 68
dsparMatrix-class ........................................... 70
dsRMatrix-class .............................................. 70
dsyMatrix-class .............................................. 72
dtCMatrix-class .............................................. 73
dtpMatrix-class .............................................. 75
dtRMatrix-class .............................................. 77
dtrMatrix-class .............................................. 78
expand-methods .............................................. 79
expm-methods .................................................. 82
externalFormats .............................................. 83
facmul-methods .............................................. 85
fastMisc ........................................................ 86
forceSymmetric-methods .................................... 90
formatSparseM .................................................. 91
genericMatrix-class ......................................... 92
Hilbert .......................................................... 93
image-methods ................................................... 94
index-class ..................................................... 96
indMatrix-class .............................................. 97
invertPerm ..................................................... 99
is.na-methods .................................................. 101
is.null.DN ..................................................... 102
isSymmetric-methods ....................................... 103
isTriangular-methods ....................................... 105
KhatriRao ...................................................... 106
KNex ............................................................ 108
kronecker-methods .......................................... 109
ldenseMatrix-class ......................................... 110
ldiMatrix-class ............................................. 111
lgeMatrix-class ............................................. 112
lsparseMatrix-class ....................................... 113
lsyMatrix-class ............................................. 115
ltrMatrix-class ............................................. 116
lu-methods .................................................... 117
mat2triplet .................................................... 119
matmult-methods ............................................ 120
Matrix .......................................................... 122
Matrix-class .................................................. 124
Matrix-notyet ................................................ 126
MatrixClass ................................................... 127
MatrixFactorization-class ................................. 128
ndenseMatrix-class ......................................... 129
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nearPD</td>
<td>130</td>
</tr>
<tr>
<td>ngeMatrix-class</td>
<td>133</td>
</tr>
<tr>
<td>nMatrix-class</td>
<td>134</td>
</tr>
<tr>
<td>nnzero-methods</td>
<td>135</td>
</tr>
<tr>
<td>norm-methods</td>
<td>137</td>
</tr>
<tr>
<td>nspARSEMatrix-class</td>
<td>138</td>
</tr>
<tr>
<td>nSyMatrix-class</td>
<td>140</td>
</tr>
<tr>
<td>ntrMatrix-class</td>
<td>141</td>
</tr>
<tr>
<td>pack-methods</td>
<td>142</td>
</tr>
<tr>
<td>packedMatrix-class</td>
<td>143</td>
</tr>
<tr>
<td>pMatrix-class</td>
<td>145</td>
</tr>
<tr>
<td>printSpMatrix</td>
<td>147</td>
</tr>
<tr>
<td>qr-methods</td>
<td>149</td>
</tr>
<tr>
<td>rankMatrix</td>
<td>152</td>
</tr>
<tr>
<td>rcond-methods</td>
<td>155</td>
</tr>
<tr>
<td>rep2abi</td>
<td>158</td>
</tr>
<tr>
<td>rleDiff-class</td>
<td>158</td>
</tr>
<tr>
<td>rsparseMatrix</td>
<td>159</td>
</tr>
<tr>
<td>RsparseMatrix-class</td>
<td>161</td>
</tr>
<tr>
<td>Schur-class</td>
<td>162</td>
</tr>
<tr>
<td>Schur-methods</td>
<td>163</td>
</tr>
<tr>
<td>solve-methods</td>
<td>165</td>
</tr>
<tr>
<td>sparse.model.matrix</td>
<td>168</td>
</tr>
<tr>
<td>sparseLU-class</td>
<td>171</td>
</tr>
<tr>
<td>sparseMatrix</td>
<td>173</td>
</tr>
<tr>
<td>sparseMatrix-class</td>
<td>176</td>
</tr>
<tr>
<td>sparseQR-class</td>
<td>178</td>
</tr>
<tr>
<td>sparseVector</td>
<td>182</td>
</tr>
<tr>
<td>sparseVector-class</td>
<td>183</td>
</tr>
<tr>
<td>spMatrix</td>
<td>186</td>
</tr>
<tr>
<td>subassign-methods</td>
<td>188</td>
</tr>
<tr>
<td>subscript-methods</td>
<td>189</td>
</tr>
<tr>
<td>symmetricMatrix-class</td>
<td>190</td>
</tr>
<tr>
<td>symmput-methods</td>
<td>191</td>
</tr>
<tr>
<td>triangularMatrix-class</td>
<td>192</td>
</tr>
<tr>
<td>TsparseMatrix-class</td>
<td>193</td>
</tr>
<tr>
<td>unpackedMatrix-class</td>
<td>194</td>
</tr>
<tr>
<td>updown-methods</td>
<td>195</td>
</tr>
<tr>
<td>USCounties</td>
<td>196</td>
</tr>
<tr>
<td>wrld_1deg</td>
<td>198</td>
</tr>
</tbody>
</table>
**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 2:1000000 or c(0:1e5, 1000:1e6).

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

**Objects from the Class**

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

**Slots**

- kind: a character string, one of ("int32", "double", "rleDiff"), denoting the internal structure of the abIndex object.
- x: Object of class "numLike"; is used (i.e., not of length 0) only iff the object is not compressed, i.e., currently exactly when kind != "rleDiff".
- rleD: object of class "rleDiff", used for compression via rle.

**Methods**

- as.numeric, as.integer, as.vector signature(x = "abIndex"): ...
- [ signature(x = "abIndex", i = "index", j = "ANY", drop = "ANY"): ...
- coerce signature(from = "numeric", to = "abIndex"): ...
- coerce signature(from = "abIndex", to = "numeric"): ...
- coerce signature(from = "abIndex", to = "integer"): ...
- length signature(x = "abIndex"): ...
- Ops signature(e1 = "numeric", e2 = "abIndex"): These and the following arithmetic and logic operations are not yet implemented; see Ops for a list of these (S4) group methods.
- Ops signature(e1 = "abIndex", e2 = "abIndex"): ...
- Ops signature(e1 = "abIndex", e2 = "numeric"): ...
- Summary signature(x = "abIndex"): ...
- show ("abIndex"): simple show method, building on show(<rleDiff>).
- is.na ("abIndex"): works analogously to regular vectors.
- is.finite, is.infinite ("abIndex"): ditto.
Note

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

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

See Also

rle (base) which is used here; numeric

Examples

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

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

abIseq

Sequence Generation of "abIndex", Abstract Index Vectors

Description

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

abIseq() is designed to work entirely like seq, but producing "abIndex" vectors.
abIseq1() is its basic building block, where abIseq1(n,m) corresponds to n:m.
c(x, ...) will return an "abIndex" vector, when x is one.

Usage

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

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

Arguments

from, to the starting and (maximal) end value of the sequence.
by number: increment of the sequence.
length.out  desired length of the sequence. A non-negative number, which for seq and
textual("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

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

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

Description

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

Methods

\[
\begin{align*}
\text{target} &= \text{"Matrix"}, \text{current} = \text{"Matrix"} \\
\text{target} &= \text{"ANY"}, \text{current} = \text{"Matrix"} \\
\text{target} &= \text{"Matrix"}, \text{current} = \text{"ANY"} \quad \text{these three methods are simply using} \text{ all.equal.numeric} \\
& \quad \text{directly and work via as.vector().}
\end{align*}
\]

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 %*% t(Q), "CsparseMatrix"),
                         L %*% U)),
       with(ex, all.equal(as(A, "CsparseMatrix"),
                         t(P) %*% L %*% U %*% Q)))

asUniqueT

Standardize a Sparse Matrix in Triplet Format

Description

Detect or standardize a TsparseMatrix with unsorted or duplicated (i,j) pairs.

Usage

anyDuplicatedT(x, ...)
isUniqueT(x, byrow = FALSE, isT = is(x, "TsparseMatrix"))
asUniqueT(x, byrow = FALSE, isT = is(x, "TsparseMatrix"))
aggregateT(x)

Arguments

x an R object. anyDuplicatedT and aggregateT require x inheriting from TsparseMatrix.
asUniqueT requires x inheriting from Matrix and coerces x to TsparseMatrix if necessary.
... optional arguments passed to the default method for generic function anyDuplicated.
byrow a logical indicating if x should be sorted by row then by column.
isT a logical indicating if x inherits from virtual class TsparseMatrix.

Value

anyDuplicatedT(x) returns the index of the first duplicated (i,j) pair in x (0 if there are no duplicated pairs).
isUniqueT(x) returns TRUE if x is a TsparseMatrix with sorted, nonduplicated (i,j) pairs and FALSE otherwise.
asUniqueT(x) returns the unique TsparseMatrix representation of x with sorted, nonduplicated (i,j) pairs. Values corresponding to identical (i,j) pairs are aggregated by addition, where in the logical case “addition” refers to logical OR.
aggregateT(x) aggregates without sorting.
See Also

Virtual class `TsparseMatrix`.

Examples

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

isUniqueT(T2 ) # FALSE
isUniqueT(T2u) # TRUE

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

## Logical l.TMatrix and n.TMatrix :
(L2 <- T2 > 0)
validObject(L2u <- asUniqueT(L2))
(N2 <- as(L2, "nMatrix"))
validObject(N2u <- asUniqueT(N2))
stopifnot(N2u@i == L2u@i, L2u@i == T2u@i, N2@i == L2@i, L2@i == T2@i,
  N2@j == L2@j, L2@j == T2@j)
# now with a nasty NA [partly failed in Matrix 1.1-5]:
L.0N <- L.1N <- L2
L.0N@x[1:2] <- c(FALSE, NA)
L.1N@x[1:2] <- c(TRUE, NA)
validObject(L.0N)
validObject(L.1N)
(m.0N <- as.matrix(L.0N))
(m.1N <- as.matrix(L.1N))
stopifnot(identical(10L, which(is.na(m.0N))), !anyNA(m.1N))
symnum(m.0N)
symnum(m.1N)
Description

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

Usage

band(x, k1, k2, 
triu(x, k = 0L, 
tril(x, k = 0L, 

Arguments

x a matrix-like object
k, k1, k2 integers specifying the diagonals that are not set to zero, k1 <= k2. These are interpreted relative to the main diagonal, which is k = 0. Positive and negative values of k indicate diagonals above and below the main diagonal, respectively.

... optional arguments passed to methods, currently unused by package Matrix.

Details

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

Value

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

Methods

x = "CsparseMatrix" method for compressed, sparse, column-oriented matrices.

x = "RsparseMatrix" method for compressed, sparse, row-oriented matrices.

x = "TsparseMatrix" method for sparse matrices in triplet format.

x = "diagonalMatrix" method for diagonal matrices.

x = "denseMatrix" method for dense matrices in packed or unpacked format.

x = "matrix" method for traditional matrices of implicit class matrix.

See Also

bandSparse for the construction of a banded sparse matrix directly from its non-zero diagonals.

Examples

## A random sparse matrix :
set.seed(7)
m <- matrix(0, 5, 5)
m[sample(length(m), size = 14)] <- rep(1:9, length=14)
(mm <- as(m, "CsparseMatrix"))
tril(mm)  # lower triangle
tril(mm, -1)  # strict lower triangle
triu(mm, 1)  # strict upper triangle
band(mm, -1, 2)  # general band
(m5 <- Matrix(rnorm(25), ncol = 5))
tril(m5)  # lower triangle
tril(m5, -1)  # strict lower triangle
triu(m5, 1)  # strict upper triangle
band(m5, -1, 2)  # general band
(m65 <- Matrix(rnorm(30), ncol = 5))  # not square
triu(m65)  # result not "dtrMatrix" unless square
(sm5 <- crossprod(m65))  # symmetric
  band(sm5, -1, 1)# "dsyMatrix": symmetric band preserves symmetry property
  as(band(sm5, -1, 1), "sparseMatrix")# often preferable
(sm <- round(crossprod(triu(mm/2))))  # sparse symmetric ("dsCv")
  band(sm, -1,1)  # remains "dsC", *however*
  band(sm, -2,1)  # -> "dgC"

---

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

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

**Usage**

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

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

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

giveCsparse (deprecated, replaced with repr): logical indicating if the result should be a CsparseMatrix or a TsparseMatrix, where the default was TRUE, and now is determined from repr; very often Csparse matrices are more efficient subsequently, but not always.

Value

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

See Also

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

Examples

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

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

## If you want a pattern matrix, but with "sparse"-diagonals,
## you currently need to go via logical sparse:
llis <- lapply(list(rpois(20, 2), rpois(20, 1), rpois(20, 3))[c(1:3, 2:3, 3:2)], as.logical)
(s4 <- bandSparse(20, k = bk, symm = TRUE, diag = llis))
(s4. <- as(drop0(s4), "nsparseMatrix"))

n <- 1e4
bk <- c(0:5, 7,11)
bMat <- matrix(1:8, n, 8, byrow=TRUE)
blis <- as.data.frame(bMat)
B <- bandSparse(n, k = bk, diag = blis)
Bs <- bandSparse(n, k = bk, diag = blis, symmetric=TRUE)
B[1:15, 1:30]
Bs[1:15, 1:30]

## can use a list *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!
        )
### Construct a Block Diagonal Matrix

**Description**

Build a block diagonal matrix given several building block matrices.

**Usage**

```r
bdiag(...) 
.bdiag(lst)
```

**Arguments**

- `...`: individual matrices or a list of matrices.
- `lst`: non-empty list of matrices.

**Details**

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

**Value**

A sparse matrix obtained by combining the arguments into a block diagonal matrix.

The value of `bdiag()` inherits from class `CsparseMatrix`, whereas `.bdiag()` returns a `TsparseMatrix`.

**Note**

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

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

**Author(s)**

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

**See Also**

- `Diagonal` for constructing matrices of class `diagonalMatrix`, or `kronecker` which also works for "Matrix" inheriting matrices.
- `bandSparse` constructs a banded sparse matrix from its non-zero sub-/super - diagonals.

Note that other CRAN R packages have own versions of `bdiag()` which return traditional matrices.
bdiag(matrix(1:4, 2), diag(3))
## combine "Matrix" class and traditional matrices:
bdiag(Diagonal(2), matrix(1:3, 3, 4), diag(3:2))

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

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

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

## 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{\linkS4class{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=M)[, rep(seq_len(N), each=k)]),
        p = k * 0L:M,
        x = as.double(unlist(lmat, recursive=FALSE, use.names=FALSE)))
}

l12 <- replicate(12, matrix(rpois(16, lambda = 6.4), 4, 4),
    simplify=FALSE)
dim(T12 <- bdiag_m(l12))# 48 x 48
boolmatmult-methods  Boolean Arithmetic Matrix Products: %&% and Methods

Description

For boolean or "pattern" matrices, i.e., \( \mathbb{R} \) objects of class \( \text{nMatrix} \), it is natural to allow matrix products using boolean instead of numerical arithmetic.

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

Value

a pattern matrix, i.e., inheriting from "\text{nMatrix}" or an "\text{ldiMatrix}" in case of a diagonal matrix.

Methods

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

\begin{verbatim}
signature(x = "ANY", y = "ANY") signature(x = "ANY", y = "Matrix") signature(x = "Matrix", y = "ANY") signature(x = "nMatrix", y = "nMatrix") signature(x = "nMatrix", y = "nsparseMatrix") signature(x = "nsparseMatrix", y = "nMatrix") signature(x = "nsparseMatrix", y = "nsparseMatrix") signature(x = "sparseVector", y = "sparseVector")
\end{verbatim}

Note

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

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

Equivalently, for all matrices \( M \), boolean arithmetic should work as if applied to \( M \neq 0 \) (or \( M \neq \text{FALSE} \)).

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

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

Examples

```r
set.seed(7)
L <- Matrix(rnorm(20) > 1, 4,5)
(N <- as(L, "nMatrix"))
L. <- L; L.[1:2,1] <- TRUE; L.@x[1:2] <- FALSE; L. # has "zeros" to drop0()
D <- Matrix(round(rnorm(30)), 5,6) # -> values in -1:1 (for this seed)
L %&% D
stopifnot(identical(L %&% D, N %&% D),
  all(L %&% D == as((L %*% abs(D)) > 0, "sparseMatrix")))
## cross products , possibly with boolArith = TRUE :
crossprod(N) # -> sparse patter'n' (TRUE/FALSE : boolean arithmetic)
crossprod(N +0) # -> numeric Matrix (with same "pattern")
stopifnot(all(crossprod(N) == t(N) %&% N),
  identical(crossprod(N), crossprod(N +0, boolArith=TRUE)),
  identical(crossprod(L), crossprod(N , boolArith=FALSE)))
crossprod(D, boolArith = TRUE) # pattern: "nsMatrix"
crossprod(L, boolArith = TRUE) # ditto
crossprod(L, boolArith = FALSE) # numeric: "dsMatrix"
```

BunchKaufman-class

Dense Bunch-Kaufman Factorizations

Description

Classes BunchKaufman and pBunchKaufman represent Bunch-Kaufman factorizations of \( n \times n \) real, symmetric matrices \( A \), having the general form

\[
A = UD_U U' = LD_L L'
\]

where \( D_U \) and \( D_L \) are symmetric, block diagonal matrices composed of \( b_U \) and \( b_L \) \( 1 \times 1 \) or \( 2 \times 2 \) diagonal blocks; \( U = \prod_{k=1}^{b_U} P_k U_k \) is the product of \( b_U \) row-permuted unit upper triangular matrices, each having nonzero entries above the diagonal in 1 or 2 columns; and \( L = \prod_{k=1}^{b_L} P_k L_k \) is the product of \( b_L \) row-permuted unit lower triangular matrices, each having nonzero entries below the diagonal in 1 or 2 columns.

These classes store the nonzero entries of the \( 2b_U + 1 \) or \( 2b_L + 1 \) factors, which are individually sparse, in a dense format as a vector of length \( nn \) (BunchKaufman) or \( n(n+1)/2 \) (pBunchKaufman), the latter giving the “packed” representation.

Slots

- Dim, Dimnames inherited from virtual class MatrixFactorization.
uplo a string, either "U" or "L", indicating which triangle (upper or lower) of the factorized symmetric matrix was used to compute the factorization and in turn how the x slot is partitioned.

x a numeric vector of length n*n (BunchKaufman) or n*(n+1)/2 (pBunchKaufman), where n=Dim[1]. The details of the representation are specified by the manual for LAPACK routines dsytrf and dsptrf.

perm an integer vector of length n=Dim[1] specifying row and column interchanges as described in the manual for LAPACK routines dsytrf and dsptrf.

Extends

Class BunchKaufmanFactorization, directly. Class MatrixFactorization, by class BunchKaufmanFactorization, distance 2.

Instantiation

Objects can be generated directly by calls of the form new("BunchKaufman", ...) or new("pBunchKaufman", ...), but they are more typically obtained as the value of BunchKaufman(x) for x inheriting from dsysMatrix or dspMatrix.

Methods

coerce signature(from = "BunchKaufman", to = "dtrMatrix"): returns a dtrMatrix, useful for inspecting the internal representation of the factorization; see ‘Note’.

coerce signature(from = "pBunchKaufman", to = "dtpMatrix"): returns a dtpMatrix, useful for inspecting the internal representation of the factorization; see ‘Note’.

determinant signature(from = "p?BunchKaufman", logarithm = "logical"): computes the determinant of the factorized matrix A or its logarithm.
expand1 signature(x = "pBunchKaufman"): see expand1-methods.
expand2 signature(x = "pBunchKaufman"): see expand2-methods.
solve signature(a = "pBunchKaufman", b = .): see solve-methods.

Note

In Matrix < 1.6-0, class BunchKaufman extended dtrMatrix and class pBunchKaufman extended dtpMatrix, reflecting the fact that the internal representation of the factorization is fundamentally triangular: there are n(n + 1)/2 “parameters”, and these can be arranged systematically to form an n × n triangular matrix. Matrix 1.6-0 removed these extensions so that methods would no longer be inherited from dtrMatrix and dtpMatrix. The availability of such methods gave the wrong impression that BunchKaufman and pBunchKaufman represent a (singular) matrix, when in fact they represent an ordered set of matrix factors.

The coercions as(. , "dtrMatrix") and as(. , "dtpMatrix") are provided for users who understand the caveats.
References

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


See Also

Class dsyMatrix and its packed counterpart.

Generic functions BunchKaufman, expand1, and expand2.

Examples

```
showClass("BunchKaufman")
set.seed(1)

n <- 6L
(A <- forceSymmetric(Matrix(rnorm(n * n), n, n)))

## With dimnames, to see that they are propagated :
dimnames(A) <- rep.int(list(paste0("x", seq_len(n))), 2L)

(bk.A <- BunchKaufman(A))
str(e.bk.A <- expand2(bk.A, complete = FALSE), max.level = 2L)
str(E.bk.A <- expand2(bk.A, complete = TRUE), max.level = 2L)

## Underlying LAPACK representation
(m.bk.A <- as(bk.A, "dtrMatrix"))
stopifnot(identical(as(m.bk.A, "matrix"), 'dim<-'(bk.A@x, bk.A@Dim)))

## Number of factors is 2*b+1, b <= n, which can be nontrivial ...
(b <- (length(E.bk.A) - 1L) %/% 2L)
ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), ...)
ae2 <- function(a, b, ...) ae1(unname(a), unname(b), ...)

## A ~ U DU U', U := prod(Pk Uk) in floating point
stopifnot(exprs = {

  identical(names(e.bk.A), c("U", "DU", "U."))
  identical(e.bk.A[["U" ]], Reduce("%*", E.bk.A[seq_len(b)]))
  identical(e.bk.A[["U."]], t(e.bk.A[["U"]]))
  ae1(A, with(e.bk.A, U %*% DU %*% U.))
})

## Factorization handled as factorized matrix
b <- rnorm(n)
stopifnot(identical(det(A), det(bk.A)),
  identical(solve(A, b), solve(bk.A, b)))
```
BunchKaufman-methods

Methods for Bunch-Kaufman Factorization

Description

Computes the Bunch-Kaufman factorization of an \( n \times n \) real, symmetric matrix \( A \), which has the general form

\[
A = U D_U U' = L D_L L'
\]

where \( D_U \) and \( D_L \) are symmetric, block diagonal matrices composed of \( b_U \) and \( b_L \) \( 1 \times 1 \) or \( 2 \times 2 \) diagonal blocks; \( U = \prod_{k=1}^{b_U} P_k U_k \) is the product of \( b_U \) row-permuted unit upper triangular matrices, each having nonzero entries above the diagonal in 1 or 2 columns; and \( L = \prod_{k=1}^{b_L} P_k L_k \) is the product of \( b_L \) row-permuted unit lower triangular matrices, each having nonzero entries below the diagonal in 1 or 2 columns.

Methods are built on LAPACK routines \( \text{dsytrf} \) and \( \text{dsptrf} \).

Usage

\[
\text{BunchKaufman}(x, \ldots)
\]

## S4 method for signature 'dsyMatrix'

\[
\text{BunchKaufman}(x, \text{warnSing} = \text{TRUE}, \ldots)
\]

## S4 method for signature 'dspMatrix'

\[
\text{BunchKaufman}(x, \text{warnSing} = \text{TRUE}, \ldots)
\]

## S4 method for signature 'matrix'

\[
\text{BunchKaufman}(x, \text{uplo} = "U", \ldots)
\]

Arguments

- \( x \): a finite symmetric matrix or \text{Matrix} to be factorized. If \( x \) is square but not symmetric, then it will be treated as symmetric; see \text{uplo}.
- \text{warnSing}: a logical indicating if a warning should be signaled for singular \( x \).
- \text{uplo}: a string, either "U" or "L", indicating which triangle of \( x \) should be used to compute the factorization.
- \ldots: further arguments passed to or from methods.

Value

An object representing the factorization, inheriting from virtual class \text{BunchKaufmanFactorization}. The specific class is \text{BunchKaufman} unless \( x \) inherits from virtual class \text{packedMatrix}, in which case it is \text{pBunchKaufman}.

References

The LAPACK source code, including documentation; see \text{https://netlib.org/lapack/double/dsytrf.f} and \text{https://netlib.org/lapack/double/dsptrf.f}.

See Also

Classes `BunchKaufman` and `pBunchKaufman` and their methods.
Classes `dsyMatrix` and `dspMatrix`.
Generic functions `expand1` and `expand2`, for constructing matrix factors from the result.
Generic functions `Cholesky`, `Schur`, `lu`, and `qr`, for computing other factorizations.

Examples

```r
showMethods("BunchKaufman", inherited = FALSE)
set.seed(0)
data(CAex, package = "Matrix")
class(CAex) # dgCMatrix
isSymmetric(CAex) # symmetric, but not formally
A <- as(CAex, "symmetricMatrix")
class(A) # dsCMatrix

## Have methods for denseMatrix (unpacked and packed),
## but not yet sparseMatrix ...
## Not run:
(bk.A <- BunchKaufman(A))

## End(Not run)
(bk.A <- BunchKaufman(as(A, "unpackedMatrix")))

## A ~ U DU U' in floating point
str(e.bk.A <- expand2(bk.A), max.level = 2L)
stopifnot(all.equal(as(A, "matrix"), as(Reduce("%*%", e.bk.A), "matrix")))
```

---

### 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](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, \texttt{eigen(CAex)} fell into an infinite loop whereas \texttt{eigen(CAex, EISPACK=TRUE)} had been okay.

Examples

\begin{verbatim}
data(CAex, package = "Matrix") 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"))
\end{verbatim}

Usage

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

## S4 method for signature 'Matrix,Matrix'
cbind2(x, y, ...)
## S4 method for signature 'Matrix,Matrix'
rbind2(x, y, ...)
\end{verbatim}

Arguments

\begin{verbatim}
... for [cr]bind, vector- or matrix-like R objects to be bound together; for [cr]bind2, further arguments passed to or from methods; see \texttt{cbind} and \texttt{cbind2}.
deparse.level integer controlling the construction of labels in the case of non-matrix-like arguments; see \texttt{cbind}.
x, y vector- or matrix-like R objects to be bound together.
\end{verbatim}
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 than non-zero ones (as the default sparse in Matrix()).

Author(s)

Martin Maechler

See Also

cbind, cbind2.

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

Examples

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

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

D <- Diagonal(2)
(M2 <- cbind(4, a, D, -1, D, 0)) # a sparse Matrix

stopifnot(validObject(M2), inherits(M2, "sparseMatrix"),
          dim(M2) == c(2,9))

---

**CHMfactor-class**  
Sparse Cholesky Factorizations

Description

CHMfactor is the virtual class of sparse Cholesky factorizations of \( n \times n \) real, symmetric matrices \( A \), having the general form

\[
P_1 A P_1' = L_1 D L_1' \quad \text{with } D_{ij} \geq 0 \quad LL'
\]

or (equivalently)

\[
A = P_1' L_1 D L_1' P_1 \quad \text{with } D_{ij} \geq 0 \quad P_1' LL' P_1
\]

where \( P_1 \) is a permutation matrix, \( L_1 \) is a unit lower triangular matrix, \( D \) is a diagonal matrix, and \( L = L_1 \sqrt{D} \). The second equalities hold only for positive semidefinite \( A \), for which the diagonal entries of \( D \) are non-negative and \( \sqrt{D} \) is well-defined.

The implementation of class CHMfactor is based on CHOLMOD’s C-level cholmod_factor_struct. Virtual subclasses CHMsimpl and CHMsuper separate the simplicial and supernodal variants. These have nonvirtual subclasses [dn]CHMsimpl and [dn]CHMsuper, where prefix ‘d’ and prefix ‘n’ are reserved for numeric and symbolic factorizations, respectively.
Usage

\texttt{isLDL(x)}

Arguments

\(x\) an object inheriting from virtual class CHMfactor, almost always the result of a call to generic function \texttt{Cholesky}.

Value

\texttt{isLDL(x)} returns \texttt{TRUE} or \texttt{FALSE}: \texttt{TRUE} if \(x\) stores the lower triangular entries of \(L_1 - I + D\), \texttt{FALSE} if \(x\) stores the lower triangular entries of \(L\).

Slots

Of \texttt{CHMfactor}:

- \texttt{Dim,Dimnames} inherited from virtual class \texttt{MatrixFactorization}.
- \texttt{colcount} an integer vector of length \texttt{Dim[1]} giving an estimate of the number of nonzero entries in each column of the lower triangular Cholesky factor. If symbolic analysis was performed prior to factorization, then the estimate is exact.
- \texttt{perm} a 0-based integer vector of length \texttt{Dim[1]} specifying the permutation applied to the rows and columns of the factorized matrix. \texttt{perm} of length 0 is valid and equivalent to the identity permutation, implying no pivoting.
- \texttt{type} an integer vector of length 6 specifying details of the factorization. The elements correspond to members \texttt{ordering, is_ll, is_super, is_monotonic, maxcsize, and maxesize} of the original \texttt{cholmod_factor_struct}. Simplicial and supernodal factorizations are distinguished by \texttt{is_super}. Simplicial factorizations do not use \texttt{maxcsize} or \texttt{maxesize}. Supernodal factorizations do not use \texttt{is_ll} or \texttt{is_monotonic}.

Of \texttt{CHMsimpl} (all unused by \texttt{nCHMsimpl}):

- \texttt{nz} an integer vector of length \texttt{Dim[1]} giving the number of nonzero entries in each column of the lower triangular Cholesky factor. There is at least one nonzero entry in each column, because the diagonal elements of the factor are stored explicitly.
- \texttt{p} an integer vector of length \texttt{Dim[1]+1}. Row indices of nonzero entries in column \(j\) of the lower triangular Cholesky factor are obtained as \(i[p[j]+\text{seq_len(nz[j])}]+1\).
- \texttt{i} an integer vector of length greater than or equal to \texttt{sum(nz)} containing the row indices of nonzero entries in the lower triangular Cholesky factor. These are grouped by column and sorted within columns, but the columns themselves need not be ordered monotonically. Columns may be overallocated, i.e., the number of elements of \(i\) reserved for column \(j\) may exceed \(nz[j]\).
- \texttt{prv, nxt} integer vectors of length \texttt{Dim[1]+2} indicating the order in which the columns of the lower triangular Cholesky factor are stored in \(i\) and \(x\). Starting from \(j<-\text{Dim[1]}+2\), the recursion \(j<-\text{nxt[j+1]}+1\) traverses the columns in forward order and terminates when \(\text{nxt[j+1]}=-1\). Starting from \(j<-\text{Dim[1]}+1\), the recursion \(j<-\text{prv[j+1]}+1\) traverses the columns in backward order and terminates when \(\text{prv[j+1]}=-1\).

Of \texttt{dCHMsimpl}:
CHMfactor-class

x a numeric vector parallel to i containing the corresponding nonzero entries of the lower triangular Cholesky factor \( L \) or (if and only if type[2] is 0) of the lower triangular matrix \( L_1 - I + D \).

Of CHMsuper:

super, pi, px integer vectors of length nsuper+1, where nsuper is the number of supernodes. super[j]+1 is the index of the leftmost column of supernode j. The row indices of supernode j are obtained as s[pi[j]+seq_len(pi[j+1]-pi[j])]+1. The numeric entries of supernode j are obtained as x[px[j]+seq_len(px[j+1]-px[j])]+1 (if slot x is available).

s an integer vector of length greater than or equal to Dim[1] containing the row indices of the supernodes. s may contain duplicates, but not within a supernode, where the row indices must be increasing.

Of dCHMsuper:

x a numeric vector of length less than or equal to prod(Dim) containing the numeric entries of the supernodes.

Extends

Class MatrixFactorization, directly.

Instantiation

Objects can be generated directly by calls of the form new("dCHMsimp", ...), etc., but dCHMsimp and dCHMsuper are more typically obtained as the value of Cholesky(x, ...) for x inheriting from sparseMatrix (often dsCMatrix).

There is currently no API outside of calls to new for generating nCHMsimp and nCHMsuper. These classes are vestigial and may be formally deprecated in a future version of Matrix.

Methods

coerce signature(from = "CHMsimp", to = "dtCMatrix"): returns a dtCMatrix representing the lower triangular Cholesky factor \( L \) or the lower triangular matrix \( L_1 - I + D \), the latter if and only if from@type[2] is 0.

coerce signature(from = "CHMsuper", to = "dgCMatrix"): returns a dgCMatrix representing the lower triangular Cholesky factor \( L \). Note that, for supernodes spanning two or more columns, the supernodal algorithm by design stores non-structural zeros above the main diagonal, hence dgCMatrix is indeed more appropriate than dtCMatrix as a coercion target.

determinant signature(from = "CHMfactor", logarithm = "logical"): behaves according to an optional argument sqrt. If sqrt = FALSE, then this method computes the determinant of the factorized matrix \( A \) or its logarithm. If sqrt = TRUE, then this method computes the determinant of the factor \( L = L_1 \sqrt{D} \) or its logarithm, giving NaN for the modulus when \( D \) has negative diagonal elements. For backwards compatibility, the default value of sqrt is TRUE, but that can be expected change in a future version of Matrix, hence defensive code will always set sqrt (to TRUE, if the code must remain backwards compatible with Matrix < 1.6-0). Calls to this method not setting sqrt may warn about the pending change. The warnings can be disabled with options(Matrix.warnSqrtDefault = 0).
diag signature \((x = \text{"CHMfactor"})\): returns a numeric vector of length \(n\) containing the diagonal elements of \(D\), which (if they are all non-negative) are the squared diagonal elements of \(L\).

expand signature \((x = \text{"CHMfactor"})\): see expand-methods.

expand1 signature \((x = \text{"CHMsimpl"})\): see expand1-methods.

expand1 signature \((x = \text{"CHMsuper"})\): see expand1-methods.

expand2 signature \((x = \text{"CHMsimpl"})\): see expand2-methods.

expand2 signature \((x = \text{"CHMsuper"})\): see expand2-methods.

image signature \((x = \text{"CHMfactor"})\): see image-methods.

nnzero signature \((x = \text{"CHMfactor"})\): see nnzero-methods.

solve signature \((a = \text{"CHMfactor"}, b = .)\): see solve-methods.

update signature \((\text{object} = \text{"CHMfactor"})\): returns a copy of object with the same nonzero pattern but with numeric entries updated according to additional arguments parent and mult, where parent is (coercible to) a dsCMatrix or a dgCMatrix and mult is a numeric vector of positive length.

The numeric entries are updated with those of the Cholesky factor of \(F(\text{parent}) + \text{mult}[1] \times I\), i.e., \(F(\text{parent})\) plus \text{mult}[1] times the identity matrix, where \(F = \text{identity}\) for symmetric parent and \(F = \text{tcrossprod}\) for other parent. The nonzero pattern of \(F(\text{parent})\) must match that of \(S\) if \text{object} = \text{Cholesky}(S, \ldots).

updown signature \((\text{update} = ., C = ., \text{object} = \text{"CHMfactor"})\): see updown-methods.

References

The CHOLMOD source code; see https://github.com/DrTimothyAldenDavis/SuiteSparse, notably the header file `CHOLMOD/Include/cholmod.h` defining cholmod_factor_struct.


See Also

Class dsCMatrix.

Generic functions Cholesky, updown, expand1 and expand2.

Examples

```r
showClass("dCHMsimpl")
showClass("dCHMsuper")
set.seed(2)
```
m <- 1000L
n <- 200L
M <- rsparsematrix(m, n, 0.01)
A <- crossprod(M)

## With dimnames, to see that they are propagated :
dimnames(A) <- dn <- rep.int(list(paste0("x", seq_len(n))), 2L)

(ch.A <- Cholesky(A)) # pivoted, by default
str(e.ch.A <- expand2(ch.A, LDL = TRUE), max.level = 2L)
str(E.ch.A <- expand2(ch.A, LDL = FALSE), max.level = 2L)

ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), ...)
ae2 <- function(a, b, ...) ae1(unname(a), unname(b), ...)

## A ~ P1' L1 D L1' P1 ~ P1' L L' P1 in floating point
stopifnot(exprs = {
  identical(names(e.ch.A), c("P1.", "L1", "D", "L1." , "P1"))
  identical(names(E.ch.A), c("P1.", "L", "L.", "P1"))
  identical(e.ch.A["P1."],
    new("pMatrix", Dim = c(n, n), Dimnames = c(list(NULL), dn[2L]),
      margin = 2L, perm = invertPerm(ch.A@perm, 0L, 1L)))
  identical(e.ch.A["L1."], t(e.ch.A["L1"]))
  identical(e.ch.A["L."], t(e.ch.A["L" ]))
  identical(e.ch.A["D"], Diagonal(x = diag(ch.A)))
  all.equal(E.ch.A["L"], with(e.ch.A, L1 %*% sqrt(D)))
  ae1(A, with(e.ch.A, P1. %*% L1 %*% D %*% L1. %*% P1))
  ae1(A, with(E.ch.A, P1. %*% L %*% L. %*% P1))
  ae2(A[ch.A@perm + 1L, ch.A@perm + 1L], with(e.ch.A, L1 %*% D %*% L1.))
  ae2(A[ch.A@perm + 1L, ch.A@perm + 1L], with(E.ch.A, L %*% L. ))
})

## Factorization handled as factorized matrix
## (in some cases only optionally, depending on arguments)
b <- rnorm(n)

Stopifnot(identical(det(A), det(ch.A, sqrt = FALSE)),
  identical(solve(A, b), solve(ch.A, b, system = "A"))
)

u1 <- update(ch.A, A , mult = sqrt(2))
u2 <- update(ch.A, t(M), mult = sqrt(2)) # updating with crossprod(M), not M

stopifnot(all.equal(u1, u2, tolerance = 1e-14))

---

**chol-methods**

**Compute the Cholesky Factor of a Matrix**

**Description**

Computes the upper triangular Cholesky factor of an n × n real, symmetric, positive semidefinite matrix A, optionally after pivoting. That is the factor L' in

\[ P_1 A P_1' = L L' \]
or (equivalently)\[ A = P_1' L L' P_1 \]

where \( P_1 \) is a permutation matrix.

Methods for \texttt{denseMatrix} are built on LAPACK routines \texttt{dpstrf}, \texttt{dpotrf}, and \texttt{dpptrf}. The latter two do not permute rows or columns, so that \( P_1 \) is an identity matrix.

Methods for \texttt{sparseMatrix} are built on CHOLMOD routines \texttt{cholmod_analyze} and \texttt{cholmod_factorize_p}.

**Usage**

```r
chol(x, ...) # S4 method for signature 'dsyMatrix'
chol(x, pivot = FALSE, tol = -1, ...)
# S4 method for signature 'dspMatrix'
chol(x, ...)
# S4 method for signature 'dsCMatrix'
chol(x, pivot = FALSE, ...)
# S4 method for signature 'ddiMatrix'
chol(x, ...)
# S4 method for signature 'generalMatrix'
chol(x, uplo = "U", ...)
# S4 method for signature 'triangularMatrix'
chol(x, uplo = "U", ...)
```

**Arguments**

- \( x \) a finite, symmetric, positive semidefinite matrix or \texttt{Matrix} to be factorized. If \( x \) is square but not symmetric, then it will be \textit{treated} as symmetric; see \texttt{uplo}. Methods for dense \( x \) require positive definiteness when \texttt{pivot = FALSE}. Methods for sparse (but not diagonal) \( x \) require positive definiteness unconditionally.

- \texttt{pivot} a logical indicating if the rows and columns of \( x \) should be pivoted. Methods for sparse \( x \) employ the approximate minimum degree (AMD) algorithm in order to reduce fill-in, i.e., without regard for numerical stability.

- \texttt{tol} a finite numeric tolerance, used only if \texttt{pivot = TRUE}. The factorization algorithm stops if the pivot is less than or equal to \texttt{tol}. Negative \texttt{tol} is equivalent to \( nrow(x) \times \text{.Machine}$double\.eps \times \max(diag(x)) \).

- \texttt{uplo} a string, either "U" or "L", indicating which triangle of \( x \) should be used to compute the factorization. The default is "U", even for lower triangular \( x \), to be consistent with \texttt{chol} from \texttt{base}.

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

**Details**

For \( x \) inheriting from \texttt{diagonalMatrix}, the diagonal result is computed directly and without pivoting, i.e., bypassing CHOLMOD.

For all other \( x \), \texttt{chol(x, pivot = value)} calls \texttt{Cholesky(x, perm = value, ...)} under the hood. If you must know the permutation \( P_1 \) in addition to the Cholesky factor \( L' \), then call \texttt{Cholesky} directly, as the result of \texttt{chol(x, pivot = TRUE)} specifies \( L' \) but not \( P_1 \).
Value

A matrix, `triangularMatrix`, or `diagonalMatrix` representing the upper triangular Cholesky factor \( L' \). The result is a traditional matrix if \( x \) is a traditional matrix, dense if \( x \) is dense, and sparse if \( x \) is sparse.

References


The CHOLMOD source code; see https://github.com/DrTimothyAldenDavis/SuiteSparse, notably the header file ‘CHOLMOD/Include/cholmod.h’ defining cholmod_factor_struct.


See Also

The default method from `base, chol`, called for traditional matrices \( x \).

Generic function `Cholesky`, for more flexibility notably when computing the Cholesky factorization and not only the factor \( L' \).

Examples

```r
showMethods("chol", inherited = FALSE)
set.seed(0)

## ---- Dense ----------------------------------------------------------
## chol(x, pivot = value) wrapping Cholesky(x, perm = value)
selectMethod("chol", "dsyMatrix")

## Except in packed cases where pivoting is not yet available
selectMethod("chol", "dspMatrix")

## .... Positive definite ..............................................

(A1 <- new("dsyMatrix", Dim = c(2L, 2L), x = c(1, 2, 2, 5)))
(R1.nopivot <- chol(A1))

(R1 <- chol(A1, pivot = TRUE))

## In 2-by-2 cases, we know that the permutation is 1:2 or 2:1,
## even if in general 'chol' does not say ...
```
### chol-methods

```
stopifnot(exprs = {
  all.equal( A1 , as(crossprod(R1.nopivot), "dsyMatrix"))
  all.equal(t(A1[2:1, 2:1]), as(crossprod(R1 ), "dsyMatrix"))
  identical(Cholesky(A1)@perm, 2:1) # because 5 > 1
})

## .... Positive semidefinite but not positive definite .................

(A2 <- new("dpoMatrix", Dim = c(2L, 2L), x = c(1, 2, 2, 4)))
try(R2.nopivot <- chol(A2)) # fails as not positive definite
(R2 <- chol(A2, pivot = TRUE)) # returns, with a warning and ...

stopifnot(exprs = {
  all.equal(t(A2[2:1, 2:1]), as(crossprod(R2), "dsyMatrix"))
  identical(Cholesky(A2)@perm, 2:1) # because 4 > 1
})

## .... Not positive semidefinite ......................................

(A3 <- new("dsyMatrix", Dim = c(2L, 2L), x = c(1, 2, 2, 3)))
try(R3.nopivot <- chol(A3)) # fails as not positive definite
(R3 <- chol(A3, pivot = TRUE)) # returns, with a warning and ...

## _Not_ equal: see details and examples in help("Cholesky")
all.equal(t(A3[2:1, 2:1]), as(crossprod(R3), "dsyMatrix"))

## ---- Sparse ---------------------------------------------------------

## chol(x, pivot = value) wrapping
## Cholesky(x, perm = value, LDL = FALSE, super = FALSE)
selectMethod("chol", "dsCMatrix")

## Except in diagonal cases which are handled "directly"
selectMethod("chol", "ddiMatrix")

(A4 <- toeplitz(as(c(10, 0, 1, 0, 3), "sparseVector")))
(ch.A4.nopivot <- Cholesky(A4, perm = FALSE, LDL = FALSE, super = FALSE))
(ch.A4 <- Cholesky(A4, perm = TRUE, LDL = FALSE, super = FALSE))
(R4.nopivot <- chol(A4))
(R4 <- chol(A4, pivot = TRUE))

det4 <- det(A4)
b4 <- rnorm(5L)
x4 <- solve(A4, b4)

stopifnot(exprs = {
  identical(R4.nopivot, expand1(ch.A4.nopivot, "L."))
  identical(R4, expand1(ch.A4, "L."))
  all.equal(A4, crossprod(R4.nopivot))
  all.equal(A4[ch.A4@perm + 1L, ch.A4@perm + 1L], crossprod(R4))
  all.equal(diag(R4.nopivot), sqrt(diag(ch.A4.nopivot)))
  all.equal(diag(R4), sqrt(diag(ch.A4)))
})
```
chol2inv-methods  Inverse from Cholesky Factor

Description

Given formally upper and lower triangular matrices \( U \) and \( L \), compute \((U'U)^{-1}\) and \((LL')^{-1}\), respectively.

This function can be seen as way to compute the inverse of a symmetric positive definite matrix given its Cholesky factor. Equivalently, it can be seen as a way to compute \((X'X)^{-1}\) given the \( R \) part of the QR factorization of \( X \), if \( R \) is constrained to have positive diagonal entries.

Usage

```r
chol2inv(x, uplo = "U", ...)  
## S4 method for signature 'dtrMatrix'
chol2inv(x, uplo = "U", ...)  
## S4 method for signature 'dtCMatrix'
chol2inv(x, uplo = "U", ...)  
## S4 method for signature 'generalMatrix'
chol2inv(x, uplo = "U", ...)  
```

Arguments

- `x`: a square matrix or `Matrix`, typically the result of a call to `chol`. If \( x \) is square but not (formally) triangular, then only the upper or lower triangle is considered, depending on optional argument `uplo` if \( x \) is a `Matrix`. 
- `uplo`: a string, either "U" or "L", indicating which triangle of \( x \) contains the Cholesky factor. The default is "U", to be consistent with `chol2inv` from `base`. 
- `...`: further arguments passed to or from methods.

Value

A matrix, `symmetricMatrix`, or `diagonalMatrix` representing the inverse of the positive definite matrix whose Cholesky factor is \( x \). The result is a traditional matrix if \( x \) is a traditional matrix, dense if \( x \) is dense, and sparse if \( x \) is sparse.
See Also

The default method from base, chol2inv, called for traditional matrices x.

Generic function chol, for computing the upper triangular Cholesky factor $L'$ of a symmetric positive semidefinite matrix.

Generic function solve, for solving linear systems and (as a corollary) for computing inverses more generally.

Examples

(A <- Matrix(cbind(c(1, 1, 1), c(1, 2, 4), c(1, 4, 16))))
(R <- chol(A))
(L <- t(R))
(R2i <- chol2inv(R))
(L2i <- chol2inv(R))
stopifnot(exprs = {
  all.equal(R2i, tcrossprod(solve(R))))
  all.equal(L2i, crossprod(solve(L))))
  all.equal(as(R2i %*% A, "matrix"), diag(3L)) # the identity
  all.equal(as(L2i %*% A, "matrix"), diag(3L)) # ditto
})

---

**Cholesky-class**

**Dense Cholesky Factorizations**

Description

Classes Cholesky and pCholesky represent dense, pivoted Cholesky factorizations of $n \times n$ real, symmetric, positive semidefinite matrices $A$, having the general form

$$P_1 A P_1' = L_1 D L_1' = LL'$$

or (equivalently)

$$A = P_1' L_1 D L_1' P_1 = P_1' L L' P_1$$

where $P_1$ is a permutation matrix, $L_1$ is a unit lower triangular matrix, $D$ is a non-negative diagonal matrix, and $L = L_1 \sqrt{D}$.

These classes store the entries of the Cholesky factor $L$ or its transpose $L'$ in a dense format as a vector of length $nn$ (Cholesky) or $n(n+1)/2$ (pCholesky), the latter giving the “packed” representation.

Slots

Dim, Dimnames inherited from virtual class MatrixFactorization.

uplo a string, either "U" or "L", indicating which triangle (upper or lower) of the factorized symmetric matrix was used to compute the factorization and in turn whether x stores $L'$ or $L$.

x a numeric vector of length $n \times n$ (Cholesky) or $n(n+1)/2$ (pCholesky), where $n=\text{Dim}[1]$, listing the entries of the Cholesky factor $L$ or its transpose $L'$ in column-major order.
perm a 1-based integer vector of length \texttt{Dim}[1] specifying the permutation applied to the rows and columns of the factorized matrix. \texttt{perm} of length 0 is valid and equivalent to the identity permutation, implying no pivoting.

**Extends**

Class \texttt{CholeskyFactorization}, directly. Class \texttt{MatrixFactorization}, by class \texttt{CholeskyFactorization}, distance 2.

**Instantiation**

Objects can be generated directly by calls of the form \texttt{new("Cholesky", ...)} or \texttt{new("pCholesky", ...)}, but they are more typically obtained as the value of \texttt{Cholesky(x)} for \texttt{x} inheriting from \texttt{dsyMatrix} or \texttt{dspMatrix} (often the subclasses of those reserved for positive semidefinite matrices, namely \texttt{dpoMatrix} and \texttt{dppMatrix}).

**Methods**

- \texttt{coerce signature(from = "Cholesky", to = "dtrMatrix")}: returns a \texttt{dtrMatrix} representing the Cholesky factor \( L \) or its transpose \( L' \); see ‘Note’.
- \texttt{coerce signature(from = "pCholesky", to = "dtpMatrix")}: returns a \texttt{dtpMatrix} representing the Cholesky factor \( L \) or its transpose \( L' \); see ‘Note’.
- \texttt{determinant signature(from = "p?Cholesky", logarithm = "logical")}: computes the determinant of the factorized matrix \( A \) or its logarithm.
- \texttt{diag signature(x = "p?Cholesky")}: returns a numeric vector of length \( n \) containing the diagonal elements of \( D \), which are the squared diagonal elements of \( L \).
- \texttt{expand1 signature(x = "p?Cholesky")}: see expand1-methods.
- \texttt{expand2 signature(x = "p?Cholesky")}: see expand2-methods.
- \texttt{solve signature(a = "p?Cholesky", b = .)}: see solve-methods.

**Note**

In \texttt{Matrix} < 1.6-0, class \texttt{Cholesky} extended \texttt{dtrMatrix} and class \texttt{pCholesky} extended \texttt{dtpMatrix}, reflecting the fact that the factor \( L \) is indeed a triangular matrix. \texttt{Matrix} 1.6-0 removed these extensions so that methods would no longer be inherited from \texttt{dtrMatrix} and \texttt{dtpMatrix}. The availability of such methods gave the wrong impression that \texttt{Cholesky} and \texttt{pCholesky} represent a (singular) matrix, when in fact they represent an ordered set of matrix factors.

The coercions \texttt{as(., "dtrMatrix")} and \texttt{as(., "dtpMatrix")} are provided for users who understand the caveats.

**References**

The \texttt{LAPACK} source code, including documentation; see \url{https://netlib.org/lapack/double/dpstrf.f}, \url{https://netlib.org/lapack/double/dpotrf.f}, and \url{https://netlib.org/lapack/double/dpptrf.f}.

Cholesky-class


See Also

Class `CHMfactor` for sparse Cholesky factorizations.

Classes `dpoMatrix` and `dppMatrix`.

Generic functions `Cholesky`, `expand1` and `expand2`.

Examples

```r
showClass("Cholesky")
set.seed(1)

m <- 30L
n <- 6L
(A <- crossprod(Matrix(rnorm(m * n), m, n)))

## With dimnames, to see that they are propagated :
dimnames(A) <- dn <- rep.int(list(paste0("x", seq_len(n))), 2L)

(ch.A <- Cholesky(A)) # pivoted, by default
str(e.ch.A <- expand2(ch.A, LDL = TRUE), max.level = 2L)
str(E.ch.A <- expand2(ch.A, LDL = FALSE), max.level = 2L)

## Underlying LAPACK representation
(m.ch.A <- as(ch.A, "dtrMatrix")) # which is L', not L, because
A@uplo == "U"
stopifnot(identical(as(m.ch.A, "matrix"), 'dim<-'(ch.A@x, ch.A@Dim)))

ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), ...)
ae2 <- function(a, b, ...) ae1(unname(a), unname(b), ...)

## A ~ P1' L1 D L1' P1 ~ P1' L L' P1 in floating point
stopifnot(exprs = {
  identical(names(e.ch.A), c("P1.", "L1", "D", "L1.", "P1"))
  identical(names(E.ch.A), c("P1.", "L", "L.", "P1"))
  identical(e.ch.A[["P1."]],
    new("pMatrix", Dim = c(n, n), Dimnames = c(list(NULL), dn[2L]),
      margin = 2L, perm = invertPerm(ch.A@perm)))
  identical(e.ch.A[["P1."]], t(e.ch.A[["P1."]]))
  identical(e.ch.A[["L1."]], t(e.ch.A[["L1."]]))
  identical(E.ch.A[["L." ]], t(E.ch.A[["L" ]]))
  identical(e.ch.A[["D." ]], Diagonal(x = diag(ch.A)))
  all.equal(E.ch.A[["L" ]], with(e.ch.A, L1 %*% sqrt(D)))
  ae1(A, with(e.ch.A, P1. %*% L1 %*% D %*% L1. %*% P1))
  ae1(A, with(E.ch.A, P1. %*% L %*% L. %*% P1))
  ae2(A[ch.A@perm, ch.A@perm], with(e.ch.A, L1 %*% D %*% L1.))
  ae2(A[ch.A@perm, ch.A@perm], with(E.ch.A, L %*% L.))
})
```
## Factorization handled as factorized matrix

b <- rnorm(n)
all.equal(det(A), det(ch.A), tolerance = 0)
all.equal(solve(A, b), solve(ch.A, b), tolerance = 0)

## For identical results, we need the _unpivoted_ factorization
## computed by det(A) and solve(A, b)
(ch.A.nopivot <- Cholesky(A, perm = FALSE))
stopifnot(identical(det(A), det(ch.A.nopivot)),
identical(solve(A, b), solve(ch.A.nopivot, b)))

---

### Description

Computes the pivoted Cholesky factorization of an $n \times n$ real, symmetric matrix $A$, which has the general form

$$P_1 A P_1^T = L_1 D L_1^T \begin{bmatrix} D_{jj} \geq 0 \end{bmatrix} LL'$$

or (equivalently)

$$A = P_1^T L_1 D L_1^T P_1 \begin{bmatrix} D_{jj} \geq 0 \end{bmatrix} P_1^T LL' P_1$$

where $P_1$ is a permutation matrix, $L_1$ is a unit lower triangular matrix, $D$ is a diagonal matrix, and $L = L_1\sqrt{D}$. The second equalities hold only for positive semidefinite $A$, for which the diagonal entries of $D$ are non-negative and $\sqrt{D}$ is well-defined.

Methods for `denseMatrix` are built on LAPACK routines `dpstrf`, `dpotrf`, and `dpptrf`. The latter two do not permute rows or columns, so that $P_1$ is an identity matrix.

Methods for `sparseMatrix` are built on CHOLMOD routines `cholmod_analyze` and `cholmod_factorize_p`.

### Usage

```r
Cholesky(A, ...)
## S4 method for signature 'dsyMatrix'
Cholesky(A, perm = TRUE, tol = -1, ...)
## S4 method for signature 'dspMatrix'
Cholesky(A, ...)
## S4 method for signature 'dsCMatrix'
Cholesky(A, perm = TRUE, LDL = !super, super = FALSE,
          Imult = 0, ...)
## S4 method for signature 'ddiMatrix'
Cholesky(A, ...)
## S4 method for signature 'generalMatrix'
Cholesky(A, uplo = "U", ...)
## S4 method for signature 'triangularMatrix'
Cholesky(A, uplo = "U", ...)
## S4 method for signature 'matrix'
Cholesky(A, uplo = "U", ...)
```
Arguments

A a finite, symmetric matrix or Matrix to be factorized. If A is square but not symmetric, then it will be treated as symmetric; see uplo. Methods for dense A require positive definiteness when perm = FALSE and positive semidefiniteness when perm = TRUE. Methods for sparse A require positive definiteness when LDL = TRUE and nonzero leading principal minors (after pivoting) when LDL = FALSE. Methods for sparse, diagonal A are an exception, requiring positive semidefiniteness unconditionally.

perm a logical indicating if the rows and columns of A should be pivoted. Methods for sparse A employ the approximate minimum degree (AMD) algorithm in order to reduce fill-in, i.e., without regard for numerical stability. Pivoting for sparsity may introduce nonpositive leading principal minors, causing the factorization to fail, in which case it may be necessary to set perm = FALSE.

tol a finite numeric tolerance, used only if perm = TRUE. The factorization algorithm stops if the pivot is less than or equal to tol. Negative tol is equivalent to nrow(A) * .Machine$double.eps * max(diag(A)).

LDL a logical indicating if the simplicial factorization should be computed as $P_1' L_1 D L_1' P_1$, such that the result stores the lower triangular entries of $L_1 - I + D$. The alternative is $P_1' L L' P_1$, such that the result stores the lower triangular entries of $L = L_1 \sqrt{D}$. This argument is ignored if super = TRUE (or if super = NA and the supernodal algorithm is chosen), as the supernodal code does not yet support the LDL = TRUE variant.

super a logical indicating if the factorization should use the supernodal algorithm. The alternative is the simplicial algorithm. Setting super = NA leaves the choice to a CHOLMOD-internal heuristic.

Imult a finite number. The matrix that is factorized is $A + \text{Imult} \cdot \text{diag}(\text{nrow}(A))$, i.e., $A$ plus Imult times the identity matrix. This argument is useful for symmetric, indefinite $A$, as $\text{Imult} > \max(\text{rowSums}(\text{abs}(A)) - \text{diag}(\text{abs}(A)))$ ensures that $A + \text{Imult} \cdot \text{diag}(\text{nrow}(A))$ is diagonally dominant. (Symmetric, diagonally dominant matrices are positive definite.)

uplo a string, either "U" or "L", indicating which triangle of A should be used to compute the factorization. The default is "U", even for lower triangular A, to be consistent with chol from base.

... further arguments passed to or from methods.

Details

Note that the result of a call to Cholesky inherits from CholeskyFactorization but not Matrix. Users who just want a matrix should consider using chol, whose methods are simple wrappers around Cholesky returning just the upper triangular Cholesky factor $L'$, typically as a triangularMatrix. However, a more principled approach would be to construct factors as needed from the CholeskyFactorization object, e.g., with expand1(x, "L"), if x is the object.

The behaviour of Cholesky(A, perm = TRUE) for dense A is somewhat exceptional, in that it expects without checking that A is positive semidefinite. By construction, if $A$ is positive semidefinite and the exact algorithm encounters a zero pivot, then the unfactorized trailing submatrix is the zero matrix, and there is nothing left to do. Hence when the finite precision algorithm encounters a pivot
less than to1, it signals a warning instead of an error and zeros the trailing submatrix in order to
guarantee that $P'LL'P$ is positive semidefinite even if $A$ is not. It follows that one way to test for
positive semidefiniteness of $A$ in the event of a warning is to analyze the error

$$\frac{\|A - P'LL'P\|}{\|A\|}.$$

See the examples and LAPACK Working Note ("LAWN") 161 for details.

Value

An object representing the factorization, inheriting from virtual class CholeskyFactorization. For a traditional matrix $A$, the specific class is Cholesky. For $A$ inheriting from unpackedMatrix, packedMatrix, and sparseMatrix, the specific class is Cholesky, pCholesky, and dCHMsimpl or dCHMsuper, respectively.

References


The CHOLMOD source code; see https://github.com/DrTimothyAldenDavis/SuiteSparse, notably the header file ‘CHOLMOD/Include/cholmod.h’ defining cholmod_factor_struct.


See Also

Classes Cholesky, pCholesky, dCHMsimpl and dCHMsuper and their methods.

Classes dpoMatrix, dppMatrix, and dsCMatrix.

Generic function chol, for obtaining the upper triangular Cholesky factor $L'$ as a matrix or Matrix.

Generic functions expand1 and expand2, for constructing matrix factors from the result.

Generic functions BunchKaufman, Schur, lu, and qr, for computing other factorizations.

Examples

showMethods("Cholesky", inherited = FALSE)
set.seed(0)
Cholesky-methods

```r
## ---- Dense ----------------------------------------------------------
## .... Positive definite ..............................................

n <- 6L
(A1 <- crossprod(Matrix(rnorm(n * n), n, n)))
(ch.A1.nopivot <- Cholesky(A1, perm = FALSE))
(ch.A1 <- Cholesky(A1))
stopifnot(exprs = {
  length(ch.A1@perm) == ncol(A1)
  isPerm(ch.A1@perm)
  is.unsorted(ch.A1@perm)  # typically not the identity permutation
  length(ch.A1.nopivot@perm) == 0L
})

## A ~ P1' L D L' P1 ~ P1' L L P1 in floating point
str(e.ch.A1 <- expand2(ch.A1, LDL = TRUE), max.level = 2L)
str(E.ch.A1 <- expand2(ch.A1, LDL = FALSE), max.level = 2L)
stopifnot(exprs = {
  all.equal(as(A1, "matrix"), as(Reduce("%*%", e.ch.A1), "matrix"))
  all.equal(as(A1, "matrix"), as(Reduce("%*%", E.ch.A1), "matrix"))
})

## .... Positive semidefinite but not positive definite ..............

A2 <- A1
A2[1L, ] <- A2[, 1L] <- 0
A2
try(Cholesky(A2, perm = FALSE))  # fails as not positive definite
ch.A2 <- Cholesky(A2)  # returns, with a warning and ...
A2.hat <- Reduce("%*%", expand2(ch.A2, LDL = FALSE))
norm(A2 - A2.hat, "2") / norm(A2, "2") # 7.670858e-17

## .... Not positive semidefinite ......................................

A3 <- A1
A3[1L, ] <- A3[, 1L] <- -1
A3
try(Cholesky(A3, perm = FALSE))  # fails as not positive definite
ch.A3 <- Cholesky(A3)  # returns, with a warning and ...
A3.hat <- Reduce("%*%", expand2(ch.A3, LDL = FALSE))
norm(A3 - A3.hat, "2") / norm(A3, "2") # 1.781568

## Indeed, 'A3' is not positive semidefinite, but 'A3.hat' is ...
ch.A3.hat <- Cholesky(A3.hat)
A3.hat.hat <- Reduce("%*%", expand2(ch.A3.hat, LDL = FALSE))

## ---- Sparse ---------------------------------------------------------
## Really just three cases modulo permutation :
##
```
data(KNex, package = "Matrix")
A4 <- crossprod(KNex[["mm"]])

ch.A4 <-
 list(pivoted =
   list(simpl1 = Cholesky(A4, perm = TRUE, super = FALSE, LDL = TRUE),
        simpl0 = Cholesky(A4, perm = TRUE, super = FALSE, LDL = FALSE),
        super0 = Cholesky(A4, perm = TRUE, super = TRUE)),
    unpivoted =
   list(simpl1 = Cholesky(A4, perm = FALSE, super = FALSE, LDL = TRUE),
        simpl0 = Cholesky(A4, perm = FALSE, super = FALSE, LDL = FALSE),
        super0 = Cholesky(A4, perm = FALSE, super = TRUE)))

ch.A4

s <- simplify2array
rapply2 <- function(object, f, ...) rapply(object, f, , how = "list", ...)
s(rapply2(ch.A4, isLDL))
s(m.ch.A4 <- rapply2(ch.A4, expand1, "L")) # giving L = L1 sqrt(D)

## By design, the pivoted and simplicial factorizations
## are more sparse than the unpivoted and supernodal ones ... 
s(rapply2(m.ch.A4, object.size))

## Which is nicely visualized by lattice-based methods for 'image'
inm <- c("pivoted", "unpivoted")
jnm <- c("simpl1", "simpl0", "super0")
for(i in 1:2)
   for(j in 1:3)
      print(image(m.ch.A4[[c(i, j)]], main = paste(inm[i], jnm[j])),
            split = c(j, i, 3L, 2L), more = i * j < 6L)

simpl1 <- ch.A4[[c("pivoted", "simpl1")]]
stopifnot(exprs = {
   length(simpl1@perm) == ncol(A4)
   isPerm(simpl1@perm, 0L)
   is.unsorted(simpl1@perm) # typically not the identity permutation
})

## One can expand with and without D regardless of isLDL(.),
## but "without" requires L = L1 sqrt(D), which is conditional
## on min(diag(D)) >= 0, hence "with" is the default
isLDL(simpl1)
stopifnot(min(diag(simpl1)) >= 0)
str(e.ch.A4 <- expand2(simpl1, LDL = TRUE), max.level = 2L) # default
str(E.ch.A4 <- expand2(simpl1, LDL = FALSE), max.level = 2L)
stopifnot(exprs = {
   all.equal(E.ch.A4[["L" ]], e.ch.A4[["L1" ]]) %*% sqrt(e.ch.A4[["D"]]))
all.equal(ch.A4["L"]), sqrt(ch.A4["D"])) \%\% e.ch.A4["L1"]) all.equal(A4, as(Reduce("%*", e.ch.A4), "symmetricMatrix")) all.equal(A4, as(Reduce("%*", e.ch.A4), "symmetricMatrix"))

}## The "same" permutation matrix with "alternate" representation## [i, perm[i]] (margin=1) \leftrightarrow [invertPerm(perm)[j], j] (margin=2)
alt <- function(P) {
  P@margin <- 1L + !(P@margin - 1L) # 1 \leftrightarrow 2
  P@perm <- invertPerm(P@perm)
  P
}

## Expansions are elegant but inefficient (transposes are redundant)## hence programmers should consider methods for 'expand1' and 'diag'## chol(A, pivot = value) is a simple wrapper around## Cholesky(A, perm = value, LDL = FALSE, super = FALSE),## returning L' = sqrt(D) L1' _but_ giving no information## about the permutation P1
stopifnot(exprs = {
  identical(expand1(simpl1, "P1"), alt(e.ch.A4["P1"])))
  identical(expand1(simpl1, "L"), E.ch.A4["L"])
  identical(Diagonal(x = diag(simpl1)), e.ch.A4["D"])
})

## chol(A, pivot = value) is a simple wrapper around## Cholesky(A, perm = value, LDL = FALSE, super = FALSE),## returning L' = sqrt(D) L1' _but_ giving no information## about the permutation P1
selectMethod("chol", "dsCMatrix")
stopifnot(all.equal(chol(A4, pivot = TRUE), E.ch.A4["L"])))

## Now a symmetric matrix with positive _and_ negative eigenvalues,## hence _not_ positive semidefinite
A5 <- new("dsCMatrix",
  Dim = c(7L, 7L),
  p = c(0:1, 3L, 6:7, 10:11, 15L),
  i = c(0L, 0:1, 0:3, 2:5, 3:6),
  x = c(1, 6, 38, 10, 60, 103, -4, 6, -32, -247, -2, -16, -128, -2, -67))
(ev <- eigen(A5, only.values = TRUE)$values)
(t.ev <- table(factor(sign(ev), -1:1))) # the matrix "inertia"

ch.A5 <- Cholesky(A5)
isLDL(ch.A5)
(d.A5 <- diag(ch.A5)) # diag(D) is partly negative

## Sylvester's law of inertia holds here, but not in general## in finite precision arithmetic
stopifnot(identical(table(Factor(sign(d.A5), -1:1)), t.ev))

try(expand1(ch.A5, "L")) # unable to compute L = L1 sqrt(D)
try(expand2(ch.A5, LDL = FALSE)) # ditto
try(chol(A5, pivot = TRUE)) # ditto

## The default expansion is "square root free" and still works here
str(e.ch.A5 <- expand2(ch.A5, LDL = TRUE), max.level = 2L)
stopifnot(all.equal(A5, as(Reduce("%*", e.ch.A5), "symmetricMatrix")))
## Version of the SuiteSparse library, which includes CHOLMOD

```r
Mv <- Matrix.Version()
Mv[["suitesparse"]]
```

### coerce-methods-graph

Conversions "graph" <- (sparse) Matrix

**Description**

Since 2005, package **Matrix** has supported coercions to and from class **graph** from package **graph**. Since 2013, this functionality has been exposed via functions T2graph and graph2T, which, unlike methods for `as(from, "<Class>")`, support optional arguments.

**Usage**

```r
graph2T(from, use.weights = )
T2graph(from, need.uniq = !isUniqueT(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**

Package **igraph**, which provides similar coercions to and from its class `igraph` via functions `graph_from_adjacency_matrix` and `as_adjacency_matrix`. 

```r
```
Examples

```r
if(requireNamespace("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(graph::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), repr="T") # no 'x'
  show(a.) # "ngTMatrix"
  show(g. <- as(a., "graph"))
}
```

---

**Sparse Matrix Coercion from and to those from package SparseM**

**Description**

Methods for coercion from and to sparse matrices from package **SparseM** are provided here, for ease of porting functionality to the **Matrix** package, and comparing functionality of the two packages. All these work via the usual `as(. , "<class>")` coercion.

```r
as(from, Class)
```

**Methods**

```r
from = "matrix.csr", to = "dgRMatrix" ... 
from = "matrix.csc", to = "dgCMatrix" ... 
from = "matrix.coo", to = "dgTMatrix" ... 
from = "dgRMatrix", to = "matrix.csr" ... 
from = "dgCMatrix", to = "matrix.csc" ... 
from = "dgTMatrix", to = "matrix.coo" ... 
from = "Matrix", to = "matrix.csr" ... 
from = "matrix.csr", to = "dgCMatrix" ... 
from = "matrix.coo", to = "dgCMatix" ... 
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.

colSums-methods  Form Row and Column Sums and Means

Description

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

Usage

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

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

Arguments

- **x**: a Matrix, i.e., inheriting from Matrix.
- **na.rm**: logical. Should missing values (including NaN) be omitted from the calculations?
- **dims**: completely ignored by the Matrix methods.
- **...**: potentially further arguments, for method \(<\leftrightarrow\) 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

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

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

## dimnames(x) --> names(<value>):
dimnames(M) <- list(paste0("r", 1:7), paste0("V",1:8))
M
colSums(M)
rowMeans(M)
## Assertions:
stopifnot(exprs = {
  all.equal(colSums(M),
    structure((1,1,6,6,6,6,2), names = colnames(M)))
  all.equal(rowMeans(M),
    structure(c(1,1,4,8,12,3,2)/8, names = paste0("r", 1:7)))
})

condest

**condest**

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

**Description**

“Estimate”, i.e. compute approximately the CONDition number of a (potentially large, often sparse) matrix A. It works by apply a fast randomized approximation of the 1-norm, norm(A,"1"), through onenormest(.).
condest(A, t = min(n, 5), normA = norm(A, "1"),
   silent = FALSE, quiet = TRUE)

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

Arguments

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

   t      number of columns to use in the iterations.

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

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

   quiet   logical indicating if convergence messages should be displayed.

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

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

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

   eps      the relative change that is deemed irrelevant.

Details

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

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

Value

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

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

   v    the maximal Ax column, scaled to norm(v) = 1. Consequently, norm(Av) =
          norm(A)/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 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 \texttt{R} and adaption by Martin Maechler.

References


See Also

\texttt{norm}, \texttt{rcond}.

Examples

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

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

one <- onenormest(mtm)
str(one) # est = 12.3
## the maximal column:
which(one$v == 1) # mostly 4, rarely 1, depending on random seed
```

CsparseMatrix-class

\textit{Class \texttt{CsparseMatrix} of Sparse Matrices in Column-compressed Form}

Description

The "\texttt{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 \texttt{showClass("CsparseMatrix")} for its subclasses.
CsparseMatrix-class

Slots

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

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

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

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

Extends

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

Methods

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

\[
\text{signature(e1 = "CsparseMatrix", e2 = "numeric"): ...}
\]

\texttt{Arith} signature(e1 = "numeric", e2 = "CsparseMatrix"): ...

\texttt{Math} signature(x = "CsparseMatrix"): ...

\texttt{band} signature(x = "CsparseMatrix"): ...

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

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

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

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

\texttt{coerce} signature(from = "CsparseMatrix", to = "TsparseMatrix"): ...

\texttt{coerce} signature(from = "CsparseMatrix", to = "denseMatrix"): ...

\texttt{coerce} signature(from = "CsparseMatrix", to = "matrix"): ...

\texttt{coerce} signature(from = "TsparseMatrix", to = "CsparseMatrix"): ...

\texttt{coerce} signature(from = "denseMatrix", to = "CsparseMatrix"): ...

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

\texttt{gamma} signature(x = "CsparseMatrix"): ...

\texttt{lgamma} signature(x = "CsparseMatrix"): ...

\texttt{log} signature(x = "CsparseMatrix"): ...

\texttt{t} signature(x = "CsparseMatrix"): ...

\texttt{tril} signature(x = "CsparseMatrix"): ...

\texttt{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

getClass("CsparseMatrix")

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


denseMatrix-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.

Extends

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

Slots

the same slots at its subclass dgeMatrix, see there.

Methods

Most methods are implemented via as(*, "generalMatrix") and are mainly used as “fallbacks” when the subclass doesn’t need its own specialized method.

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

See Also

The virtual classes Matrix, dMatrix, and dsparseMatrix.
Examples

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

---

**ddiMatrix-class**  
Class "ddiMatrix" of Diagonal Numeric Matrices

**Description**

The class "ddiMatrix" of numerical diagonal matrices.

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

**Objects from the Class**

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

**Slots**

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

**Extends**

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

**Methods**

\%

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

**See Also**

Class diagonalMatrix and function Diagonal.

**Examples**

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

M <- Matrix(cbind(1,2:4))
denseLU-class

M %*% d2 #> `fast' multiplication
chol(d2) # trivial
stopifnot(is(cd2 <- chol(d2), "ddiMatrix"),
  all.equal(cd2@x, c(sqrt(10),1)))

denseLU-class Dense LU Factorizations

Description
denseLU is the class of dense, row-pivoted LU factorizations of \( m \times n \) real matrices \( A \), having the general form
\[
P_1 A = LU
\]
or (equivalently)
\[
A = P_1'LU
\]
where \( P_1 \) is an \( m \times m \) permutation matrix, \( L \) is an \( m \times \min(m,n) \) unit lower trapezoidal matrix, and \( U \) is a \( \min(m,n) \times n \) upper trapezoidal matrix. If \( m = n \), then the factors \( L \) and \( U \) are triangular.

Slots
- `Dim`, `Dimnames` inherited from virtual class `MatrixFactorization`.
- `x` a numeric vector of length \( \prod(\text{Dim}) \) storing the triangular \( L \) and \( U \) factors together in a packed format. The details of the representation are specified by the manual for LAPACK routine `dgetrf`.
- `perm` an integer vector of length \( \min(\text{Dim}) \) specifying the permutation \( P_1 \) as a product of transpositions. The corresponding permutation vector can be obtained as `asPerm(perm)`.

Extends
Class `LU`, directly. Class `MatrixFactorization`, by class `LU`, distance 2.

Instantiation
Objects can be generated directly by calls of the form `new("denseLU", ...)` but they are more typically obtained as the value of `lu(x)` for \( x \) inheriting from `denseMatrix` (often `dgeMatrix`).

Methods
- `coerce` signature(from = "denseLU", to = "dgeMatrix"): returns a `dgeMatrix` with the dimensions of the factorized matrix \( A \), equal to \( L \) below the diagonal and equal to \( U \) on and above the diagonal.
- `determinant` signature(from = "denseLU", logarithm = "logical"): computes the determinant of the factorized matrix \( A \) or its logarithm.
- `expand` signature(x = "denseLU"): see `expand-methods`.
- `expand1` signature(x = "denseLU"): see `expand1-methods`.
- `expand2` signature(x = "denseLU"): see `expand2-methods`.
- `solve` signature(a = "denseLU", b = "missing"): see `solve-methods`. 
References

The LAPACK source code, including documentation; see https://netlib.org/lapack/double/dgetrf.f.


See Also

Class `sparseLU` for sparse LU factorizations.

Class `dgeMatrix`.

Generic functions `lu, expand1` and `expand2`.

Examples

```r
showClass("denseLU")
set.seed(1)

n <- 3L
(A <- Matrix(round(rnorm(n * n), 2L), n, n))

## With dimnames, to see that they are propagated :
dimnames(A) <- dn <- list(paste0("r", seq_len(n)),
paste0("c", seq_len(n)))

(1u.A <- lu(A))
str(e.lu.A <- expand2(lu.A), max.level = 2L)

## Underlying LAPACK representation
(m.lu.A <- as(lu.A, "dgeMatrix")) # which is L and U interlaced
stopifnot(identical(as(m.lu.A, "matrix"), `dim<~`(lu.A@x, lu.A@Dim)))

ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), 
    ae2 <- function(a, b, ...) ae1(unname(a), unname(b), 

## A ~ P1' L U in floating point
stopifnot(exprs = {
    identical(names(e.lu.A), c("P1.", "L", "U"))
    identical(e.lu.A[["P1."]],
        new("pMatrix", Dim = c(n, n), Dimnames = c(dn[1L], list(NULL)),
            margin = 1L, perm = invertPerm(asPerm(lu.A@perm))))
    identical(e.lu.A[["L"]],
        new("dtrMatrix", Dim = c(n, n), Dimnames = list(NULL, NULL),
            uplo = "L", diag = "U", x = lu.A@x))
    identical(e.lu.A[["U"]],
        new("dtrMatrix", Dim = c(n, n), Dimnames = c(list(NULL), dn[2L]),
            uplo = "U", diag = "N", x = lu.A@x))
    ae1(A, with(e.lu.A, P1. %*% L %*% U))
    ae2(A[asPerm(lu.A@perm), ], with(e.lu.A, L %*% U))
})
```
## Factorization handled as factorized matrix
b <- rnorm(n)
stopifnot(identical(det(A), det(lu.A)),
          identical(solve(A, b), solve(lu.A, b)))

---

denseMatrix-class  Virtual Class "denseMatrix" of All Dense Matrices

Description

This is the virtual class of all dense (S4) matrices. It partitions into two subclasses \texttt{packedMatrix} and \texttt{unpackedMatrix}. Alternatively into the (currently) three subclasses \texttt{ddenseMatrix}, \texttt{ldenseMatrix}, and \texttt{ndenseMatrix}.
denseMatrix is (hence) the direct superclass of these \((2 + 3 = 5)\) classes.

Extends

class "Matrix" directly.

Slots

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

Methods

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

Extraction \((\texttt{[})\) 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

\texttt{showClass("denseMatrix")}
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")
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]
dgeMatrix-class

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.

colMeans signature(x = "dgeMatrix"): columnwise means (averages)

colSums signature(x = "dgeMatrix"): columnwise sums

diag signature(x = "dgeMatrix"): ...

dim signature(x = "dgeMatrix"): ...

dimnames signature(x = "dgeMatrix"): ...

eigen signature(x = "dgeMatrix", only.values = "logical"): ...
eigen signature(x = "dgeMatrix", only.values = "missing"): ...
norm signature(x = "dgeMatrix", type = "character"): ...
norm signature(x = "dgeMatrix", type = "missing"): ...
rcond signature(x = "dgeMatrix", norm = "character") or norm = "missing": the reciprocal condition number, rcond().
rowMeans signature(x = "dgeMatrix"): rowwise means (averages)
rowSums signature(x = "dgeMatrix"): rowwise sums
t signature(x = "dgeMatrix"): matrix transpose

See Also
Classes Matrix, dtrMatrix, and dsyMatrix.

dgRMatrix-class Sparse Compressed, Row-oriented Numeric Matrices

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

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

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

Slots
j: Object of class "integer" of length nnzero (number of non-zero elements). These are the column numbers for each non-zero element in the matrix.
p: Object of class "integer" of pointers, one for each row, to the initial (zero-based) index of elements in the row.
x: Object of class "numeric" - the non-zero elements of the matrix.
Dim: Object of class "integer" - the dimensions of the matrix.

Methods
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.
**Description**

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

**Objects from the Class**

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

**Slots**

- **i**: `integer` row indices of non-zero entries in 0-base, i.e., must be in $0:(nrow(.)-1)$.
- **j**: `integer` column indices of non-zero entries. Must be the same length as slot `i` and 0-based as well, i.e., in $0:(ncol(.)-1)$.
- **x**: `numeric` vector - the (non-zero) entry at position $(i,j)$. Must be the same length as slot `i`. If an index pair occurs more than once, the corresponding values of slot `x` are added to form the element of the matrix.
- **Dim**: Object of class "integer" of length 2 - the dimensions of the matrix.

**Methods**

- `+` signature(e1 = "dgTMatrix", e2 = "dgTMatrix")
- `image` signature(x = "dgTMatrix"): plots an image of x using the `levelplot` function
- `t` signature(x = "dgTMatrix"): returns the transpose of x

**Note**

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

Note that both `new(.)` and `spMatrix` constructors for "dgTMatrix" (and other "TsparseMatrix" classes) implicitly add $x_k$'s that belong to identical $(i_k, j_k)$ pairs.

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

**See Also**

Class `dgCMatrix` or the superclasses `dparsesMatrix` and `TsparseMatrix`; `asUniqueT`. 

---

**Sparse matrices in triplet form**
Diagonal

Construct a Diagonal Matrix

Description

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

Usage

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

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

Arguments

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

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

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

uplo

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

shape

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

unitri

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

kind

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

cols

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

Value

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

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

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

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

Author(s)

Martin Maechler

See Also

the generic function diag for extraction of the diagonal from a matrix works for all “Matrices”.

bandSparse constructs a banded sparse matrix from its non-zero sub-/super - diagonals. band(A) returns a band matrix containing some sub-/super - diagonals of A.

Matrix for general matrix construction; further, class diagonalMatrix.

Examples

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

## Use Diagonal() + kronecker() for "repeated-block" matrices:
```r
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: character string, either "U" or "N", where "U" means ‘unit-diagonal’.
- Dim: matrix dimension, and
- Dimnames: the dimnames, a list, see the Matrix class description. Typically list(NULL,NULL) for diagonal matrices.

### Extends

Class "sparseMatrix", directly.

### Methods

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

- **coerce** signature(from = "matrix", to = "diagonalMatrix"): ...
- **coerce** signature(from = "Matrix", to = "diagonalMatrix"): ...
- **coerce** signature(from = "diagonalMatrix", to = "generalMatrix"): ...
- **coerce** signature(from = "diagonalMatrix", to = "triangularMatrix"): ...
- **coerce** signature(from = "diagonalMatrix", to = "nMatrix"): ...
- **coerce** signature(from = "diagonalMatrix", to = "matrix"): ...
- **coerce** signature(from = "diagonalMatrix", to = "sparseVector"): ...
- **t** signature(x = "diagonalMatrix"): ...

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

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

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

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

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

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

See Also

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

Examples

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

# an unusual way to construct a band matrix:  
rbind2(cbind2(I5, D5),  
       cbind2(D5, I5))
Description

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

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

Usage

\[
\begin{align*}
\text{diagU2N}(x, \text{cl} = \text{getClassDef}(\text{class}(x)), \text{checkDense} = \text{FALSE}) \\
\text{diagN2U}(x, \text{cl} = \text{getClassDef}(\text{class}(x)), \text{checkDense} = \text{FALSE})
\end{align*}
\]

Arguments

- \(x\) a `triangularMatrix`, often sparse.
- \(\text{cl}\) (optional, for speedup only:) class (definition) of \(x\).
- \(\text{checkDense}\) 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.

See Also

"triangularMatrix", "dtCMatrix".

Examples

\[
\begin{align*}
(T &\leftarrow \text{Diagonal}(7) + \text{triu}(\text{Matrix}(\text{rpois}(49, 1/4), 7, 7), k = 1)) \\
(uT &\leftarrow \text{diagN2U}(T)) \quad \text{"unitriangular"} \\
(t.u &\leftarrow \text{diagN2U}(10*T)) \# \text{changes the diagonal!} \\
\text{stopifnot}(\text{all}(T == uT), \text{diag}(t.u) == 1, \\
\text{identical}(T, \text{diagU2N}(uT)))
\end{align*}
\]
dimScale

T[upper.tri(T)] <- 5 # still "dtC"
T <- diagN2U(as(T,"triangularMatrix"))
dT <- as(T, "denseMatrix") # (unitriangular)
dT.n <- diagU2N(dT, checkDense = TRUE)
sT.n <- diagU2N(dT)
stopifnot(is(dT.n, "denseMatrix"), is(sT.n, "sparseMatrix"),
  dT@diag == "U", dT.n@diag == "N", sT.n@diag == "N",
  all(dT == dT.n), all(dT == sT.n))

---

**dimScale**

*Scale the Rows and Columns of a Matrix*

**Description**

dimScale, rowScale, and colScale implement $D_1 \frac{x}{D_1} \times \frac{x}{D_2} D_2$, $D_1 x$, and $x D_2$, and $D$, respectively. Unlike the explicit products, these functions preserve dimnames(x) and symmetry where appropriate.

**Usage**

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

**Arguments**

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

**Details**

dimScale(x) (with d1 and d2 unset) is only roughly equivalent to `cov2cor(x)`. `cov2cor` sets the diagonal entries of the result to 1 (exactly); `dimScale` does not.

**Value**

The result of scaling `x`, currently always inheriting from virtual class `dMatrix`.

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

**Author(s)**

Mikael Jagan

**See Also**

cov2cor
Examples

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

(a <- Diagonal(x = a))

D <- Diagonal(x = d)

(scx <- dimScale(x, d)) # symmetry and 'dimnames' kept

(D <- Diagonal(x = d)) # symmetry and 'dimnames' lost

stopifnot(identical(unname(as(scx, "generalMatrix")), mmx))

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

---

**dMatrix-class**

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

---

### Description

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

### Slots

Common to all matrix object in the package:

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

### Methods

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

- **Arith** signature(e1 = "dMatrix", e2 = "dMatrix"): ...
- **Arith** signature(e1 = "dMatrix", e2 = "numeric"): ...
- **Arith** signature(e1 = "numeric", e2 = "dMatrix"): ...
- **Math** signature(x = "dMatrix"): ...
- **Math2** signature(x = "dMatrix", digits = "numeric"): this group contains `round()` and `signif()`.
- **Compare** signature(e1 = "numeric", e2 = "dMatrix"): ...
- **Compare** signature(e1 = "dMatrix", e2 = "numeric"): ...
- **Compare** signature(e1 = "dMatrix", e2 = "dMatrix"): ...
- **Summary** signature(x = "dMatrix"): The "Summary" group contains the seven functions `max()`, `min()`, `range()`, `prod()`, `sum()`, `any()`, and `all()`.
The following methods are also defined for all double matrices:

**expm** signature(x = "dMatrix"): computes the “Matrix Exponential”. see expm.

The following methods are defined for all logical matrices:

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

See Also

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

The numeric matrix classes dgeMatrix, dgCMatrix, and Matrix.

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

Examples

```r
showClass("dMatrix")

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

---

**dmperm** *Dulmage-Mendelsohn Permutation / Decomposition*

Description

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

Usage

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

Arguments

- **x**: a typically sparse matrix; internally coerced to either "dgCMatrix" or "dtCMatrix".
- **nAns**: an integer specifying the length of the resulting list. Must be 2, 4, or 6.
- **seed**: an integer code in -1,0,1; determining the (initial) permutation; by default, seed = 0, no (or the identity) permutation; seed = -1 uses the “reverse” permutation \( k:1 \); for seed = 1, it is a random permutation (using R’s RNG, seed, etc).
Details

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

Value

a named list with (by default) 6 components,

- \( p \) integer vector with the permutation \( p \), of length \( nrow(x) \).
- \( q \) integer vector with the permutation \( q \), of length \( ncol(x) \).
- \( r \) integer vector of length \( nb+1 \), where block \( k \) is rows \( r[k] \to r[k+1] - 1 \) in \( A[p,q] \).
- \( s \) integer vector of length \( nb+1 \), where block \( k \) is cols \( s[k] \to s[k+1] - 1 \) in \( A[p,q] \).
- \( rr5 \) integer vector of length 5, defining the coarse row decomposition.
- \( cc5 \) integer vector of length 5, defining the coarse column decomposition.

Author(s)

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

References

Section 7.4 Dulmage-Mendelsohn decomposition, pp. 122 ff of
of Algorithms”.

See Also

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

Examples

```r
set.seed(17)
(S9 <- rsparsematrix(9, 9, nnz = 10, symmetric=TRUE)) # dsCMatrix
str(dm9 <- dmperm(S9))
(S9p <- with(dm9, S9[p, q]))
## looks good, but *not* quite upper triangular; these, too:
str(dm9.0 <- dmperm(S9, seed=-1)) # non-random too.
str(dm9_1 <- dmperm(S9, seed= 1)) # a random one
## The last two permutations differ, but have the same effect!
(S9p0 <- with(dm9.0, S9[p, q])) # .. hmm ..
stopifnot(all.equal(S9p0, S9p))# same as as default, but different from the random one

set.seed(11)
(M <- triu(rsparsematrix(9,11, 1/4)))
(dM <- dmperm(M); with(dM, M[p, q])
(Mp <- M[sample.int(nrow(M)), sample.int(ncol(M))])
(dMp <- dmperm(Mp); with(dMp, Mp[p, q])
```
set.seed(7)
(n7 <- rsparsematrix(5, 12, nnz = 10, rand.x = NULL))
str( dm.7 <- dmperm(n7) )
stopifnot(exprs = {
  lengths(dm.7[1:2]) == dim(n7)
  identical(dm.7, dmperm(as(n7, "dMatrix")))
  identical(dm.7[1:4], dmperm(n7, nAns=4))
  identical(dm.7[1:2], dmperm(n7, nAns=2))
})

---

dpoMatrix-class  Positive Semi-definite Dense (Packed | Non-packed) Numeric Matrices

Description

- The "dpoMatrix" class is the class of positive-semidefinite symmetric matrices in nonpacked storage.
- The "dppMatrix" class is the same except in packed storage. Only the upper triangle or the lower triangle is required to be available.
- The "corMatrix" and "copMatrix" classes represent correlation matrices. They extend "dpoMatrix" and "dppMatrix", respectively, with an additional slot sd allowing restoration of 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" and "copMatrix") a numeric vector of length n containing the (original) \( \sqrt{\text{var}(.)} \) entries which allow reconstruction of a covariance matrix from the correlation matrix.
dpoMatrix-class

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).

Note

Currently the validity methods for these classes such as getValidity(getClass("dpoMatrix")) for efficiency reasons only check the diagonal entries of the matrix – they may not be negative. This is only necessary but not sufficient for a symmetric matrix to be positive semi-definite.
A more reliable (but often more expensive) check for positive semi-definiteness would look at the signs of diag(BunchKaufman(.)) (with some tolerance for very small negative values), and for (strict) positive definiteness at something like !inherits(tryCatch(chol(.), error=identity), "error"). Indeed, when coercing to these classes, a version of Cholesky() or chol() is typically used, e.g., see selectMethod("coerce", c(from="dsyMatrix", to="dpoMatrix")).

See Also

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

Examples

h6 <- Hilbert(6)
rcond(h6)
str(h6)
h6 * 27720 # is `integer'
solve(h6)
str(h6 <- pack(h6))

### Note that as(*, "corMatrix") *scales* the matrix
(ch6 <- as(h6, "corMatrix"))
drop0

Drop Non-Structural Zeros from a Sparse Matrix

**Description**

Deletes "non-structural" zeros (i.e., zeros stored explicitly, in memory) from a sparse matrix and returns the result.

**Usage**

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

**Arguments**

- `x` a Matrix, typically inheriting from virtual class `sparseMatrix`. denseMatrix and traditional vectors and matrices are coerced to `CsparseMatrix`, with zeros dropped automatically, hence users passing such `x` should consider `as(x, "CsparseMatrix")` instead, notably in the `tol = 0` case.
- `tol` a non-negative number. If `x` is numeric, then entries less than or equal to `tol` in absolute value are deleted.
- `is.Csparse` a logical used only if `give.Csparse` is `TRUE`, indicating if `x` already inherits from virtual class `CsparseMatrix`, in which case coercion is not attempted, permitting some (typically small) speed-up.
- `give.Csparse` a logical indicating if the result must inherit from virtual class `CsparseMatrix`. If FALSE and `x` inherits from `RsparseMatrix`, `TsparseMatrix`, or `indMatrix`, then the result preserves the class of `x`. The default value is `TRUE` only for backwards compatibility.

**Value**

A `sparseMatrix`, the result of deleting non-structural zeros from `x`, possibly after coercion.

**Note**

`drop0` is sometimes called in conjunction with `zapsmall`, e.g., when dealing with sparse matrix products; see the example.

**See Also**

Function `sparseMatrix`, for constructing objects inheriting from virtual class `sparseMatrix`; `nnzero`. 
Examples

```
(m <- sparseMatrix(i = 1:8, j = 2:9, x = c(0:2, 3:-1),
    dims = c(10L, 20L)))

drop0(m)
```

```
## A larger example:
t5 <- new("dtCMatrix", Dim = c(5L, 5L), uplo = "L",
    x = c(10, 1, 3, 10, 1, 10, 10, 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.
dsCMatrix-class

Extends

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

Methods

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

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

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

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

determinant signature(x = "dsCMa trix", 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 = "dsCMa trix"): 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 "dsCMatrix", see above.

See Also

Classes dgCMatrix, dgTMatrix, dgeMatrix and those mentioned above.

Examples

```r
mm <- Matrix(toeplitz(c(10, 0, 1, 0, 3)), sparse = TRUE)
mm # automatically dsCMatrix
str(mm)
mT <- as(mm, "generalMatrix"), "TsparseMatrix")

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

# or
sT <- Matrix(mT, sparse=TRUE, forceCheck=TRUE) # dsT

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

- `showClass("dsparseMatrix")`

---

dsRMatrix-class

Symmetric Sparse Compressed Row Matrices

Description

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

Objects from the Class

These "..RMatrix" classes are currently still mostly unimplemented!
Objects can be created by calls of the form `new("dsRMatrix", ...)`. 

---
Slots

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

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

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

defactors: 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 dsparseMatrix, symmetricMatrix, and RsparseMatrix, directly.

Class "dMatrix", by class "dsparseMatrix"; class "sparseMatrix", by classes "dsparseMatrix" and "RsparseMatrix".

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).

See Also

the classes dgCMatrix, dgTMatrix, and dgeMatrix.

Examples

```
(m0 <- new("dsRMATRIX"))
m2 <- new("dsRMATRIX", Dim = c(2L,2L),
       x = c(3,1), j = c(1L,1L), p = 0:2)
m2
stopifnot(colSums(as(m2, "TsparseMatrix")) == 3:4)
str(m2)
(ds2 <- forceSymmetric(diag(2))) # dsy*
dR <- as(ds2, "RsparseMatrix")
dR # dsRMATRIX
```
The "dsyMatrix" class is the class of symmetric, dense matrices in non-packed storage and "dspMatrix" is the class of symmetric dense matrices in packed storage, see `pack()`. Only the upper triangle or the lower triangle is stored.

Objects from the Class

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

Slots

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

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

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

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

Extends

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

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

Methods

`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.
dtCMatrix-class

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).

cho(sy2) #-> "Cholesky" matrix
(s2p2 <- pack(sy2)) # a "dspMatrix"

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

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

---

dtCMatrix-class  

Triangular, (compressed) sparse column matrices

Description

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

The "dtTMatrix" class is a class of triangular, sparse matrices in triplet format.

Objects from the Class

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

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

solve signature(a = "dtCMatrix", b = "..."): sparse triangular solve (aka “backsolve” or “for-
wardsolve”), 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, dtTMatrix, dgeMatrix, and dtrMatrix.

Examples

showClass("dtCMatrix")
showClass("dtTMatrix")
t1 <- new("dtTMatrix", x = c(3,7), i = 0:1, j=3:2, Dim= as.integer(c(4,4)))
t1
## from 0-diagonal to unit-diagonal {low-level step}:
tu <- t1 ; tu@diag <- "U"
tu
(cu <- as(tu, "CsparseMatrix"))
str(cu)# only two entries in @i and @x
stopifnot(cu@i == 1:0,
      all(2 * symmpart(cu) == Diagonal(4) + forceSymmetric(cu)))
t1[1,2:3] <- -1:-2
diag(t1) <- 10*c(1:2,3:2)
t1 # still triangular
(it1 <- solve(t1))
t1. <- solve(it1)
all(abs(t1 - t1.) < 10 * .Machine$double.eps)

## 2nd example
U5 <- new("dtCMatrix", i= c(1L, 0:3), p=c(0L, 0L, 0:2, 5L), Dim = c(5L, 5L),
        x = rep(1, 5), diag = "U")
U5
(iu <- solve(U5)) # contains one '0'
validObject(iu2 <- solve(U5, Diagonal(5)))# failed in earlier versions
I5 <- iu %*% 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 \( d \times d \), 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.
**dtpMatrix-class**

**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").

determinant signature(x = "dtpMatrix", logarithm = "logical"): the determinant(x) triv-
ially is prod(diag(x)), but computed on log scale to prevent over- and underflow.

diag signature(x = "dtpMatrix"): ...

norm signature(x = "dtpMatrix", type = "character"): ...

rcond signature(x = "dtpMatrix", norm = "character"): ...

solve signature(a = "dtpMatrix", b = "..."): efficiently using internal backsolve or forw-
ard-solve, see solve-methods.

t signature(x = "dtpMatrix"): t(x) remains a "dtpMatrix", lower triangular if x is upper tri-
angular, and vice versa.
```

**See Also**

Class dtrMatrix

**Examples**

```r
class(dtrMatrix)

dexample("dtrMatrix-class", echo=FALSE)
p1 <- pack(T2)
str(p1)
(pp <- pack(T))
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)))
```
**dtRMatrix-class**

Triangular Sparse Compressed Row Matrices

**Description**

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

**Objects from the Class**

This class is currently still mostly unimplemented!

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

**Slots**

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

**Extends**

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

**Methods**

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

**See Also**

Classes `dgCMatrix, dgTMatrix, dgeMatrix`
Examples

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

---

dtrMatrix-class

Triangular, dense, numeric matrices

Description

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

Objects from the Class

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

Slots

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

Extends

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

Methods

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

- `norm` signature(x = "dtrMatrix", type = "character"):
- `rcond` signature(x = "dtrMatrix", norm = "character"):
- `solve` signature(a = "dtrMatrix", b = ". . ."): efficiently use a “forwardsolve” or backsolve for a lower or upper triangular matrix, respectively, see also `solve-methods`.

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

See Also

Classes ddenseMatrix, dtpMatrix, triangularMatrix

Examples

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

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

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

Description

expand1 and expand2 construct matrix factors from objects specifying matrix factorizations. Such objects typically do not store the factors explicitly, employing instead a compact representation to save memory.

Usage

expand1(x, which, ...)
expand2(x, ...)

expand (x, ...)

Arguments

x a matrix factorization, typically inheriting from virtual class MatrixFactorization.
which a character string indicating a matrix factor.
... further arguments passed to or from methods.
Details

Methods for expand are retained only for backwards compatibility with Matrix < 1.6-0. New code should use expand1 and expand2, whose methods provide more control and behave more consistently. Notably, expand2 obeys the rule that the product of the matrix factors in the returned list should reproduce (within some tolerance) the factorized matrix, including its dimnames.

Hence if x is a matrix and y is its factorization, then

\[
\text{all.equal(as(x, "matrix"), as(Reduce("%*%", expand2(y)), "matrix"))}
\]

should in most cases return TRUE.

Value

expand1 returns an object inheriting from virtual class Matrix, representing the factor indicated by which, always without row and column names.

expand2 returns a list of factors, typically with names using conventional notation, as in list(L=, U=). The first and last factors get the row and column names of the factorized matrix, which are preserved in the Dimnames slot of x.

Methods

The following table lists methods for expand1 together with allowed values of argument which.

<table>
<thead>
<tr>
<th>class(x)</th>
<th>which</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schur</td>
<td>c(&quot;Q&quot;, &quot;T&quot;, &quot;Q.&quot;</td>
</tr>
<tr>
<td>denseLU</td>
<td>c(&quot;P1&quot;, &quot;P1.&quot;, &quot;L&quot;, &quot;U&quot;)</td>
</tr>
<tr>
<td>sparseLU</td>
<td>c(&quot;P1&quot;, &quot;P1.&quot;, &quot;P2.&quot;, &quot;L&quot;, &quot;U&quot;)</td>
</tr>
<tr>
<td>sparseQR</td>
<td>c(&quot;P1&quot;, &quot;P1.&quot;, &quot;P2.&quot;, &quot;Q&quot;, &quot;Q1&quot;, &quot;R&quot;, &quot;R1&quot;)</td>
</tr>
<tr>
<td>BunchKaufman, pBunchKaufman</td>
<td>c(&quot;U&quot;, &quot;DU&quot;, &quot;U.&quot;, &quot;L&quot;, &quot;DL&quot;, &quot;L.&quot;)</td>
</tr>
<tr>
<td>Cholesky, pCholesky</td>
<td>c(&quot;P1&quot;, &quot;P1.&quot;, &quot;L1&quot;, &quot;D&quot;, &quot;L1.&quot;, &quot;L&quot;, &quot;L.&quot;)</td>
</tr>
<tr>
<td>CHMsimpl, CHMsimpl</td>
<td>c(&quot;P1&quot;, &quot;P1.&quot;, &quot;L1&quot;, &quot;D&quot;, &quot;L1.&quot;, &quot;L&quot;, &quot;L.&quot;)</td>
</tr>
</tbody>
</table>

Methods for expand2 and expand are described below. Factor names and classes apply also to expand1.

expand2 signature(x = "CHMsimpl"): expands the factorization \( A = P' L_1 D L_1' P_1 = P' L L' P_1 \) as list(P1, L1, D, L1, P1) (the default) or as list(P1, L, L, P1), depending on optional logical argument LDL. P1 and P1 are pMatrix, L1, L1, L, and L are dtCMatrix, and D is a ddiMatrix.

expand2 signature(x = "CHMsuper"): as CHMsimpl, but the triangular factors are stored as dgCMatrix.

expand2 signature(x = "p?Cholesky"): expands the factorization \( A = L_1 D L_1' = LL' \) as list(L1, L, L1) (the default) or as list(L, L), depending on optional logical argument LDL. L1, L1, L, and L are dtrMatrix or dtpMatrix, and D is a ddiMatrix.

expand2 signature(x = "p?BunchKaufman"): expands the factorization \( A = U D_1 U' = LD L' \) where

\[
U = \prod_{k=1}^{b_U} P_k U_k \text{ and } L = \prod_{k=1}^{b_L} P_k L_k
\]

as list(U, DU, U) or list(L, DL, L), depending on x@uplo. If optional argument complete is TRUE, then an unnamed list giving the full expansion with \(2b_U + 1 \) or \(2b_L + 1 \) matrix factors is returned instead. P_k are represented...
as pMatrix. $U_k$ and $L_k$ are represented as dtCMatrix, and $D_U$ and $D_L$ are represented as dsCMatrix.

expand2 signature(x = "Schur") expands the factorization $A = QTQ'$ as list(Q, T, Q.). Q and T are x@Q and t(x@Q) modulo Dimnames, and T is x@T.

expand2 signature(x = "sparseLU") expands the factorization $A = P_1^T L U P_2$ as list(P1., L, U, P2.). P1. and P2. are pMatrix, and L and U are dtCMatrix.

expand2 signature(x = "denseLU") expands the factorization $A = P_1^T L U$ as list(P1., L, U).

P1. is a pMatrix, and L and U are dtrMatrix if square and dgeMatrix otherwise.

expand2 signature(x = "sparseQR") expands the factorization $A = P_1^T Q R P_2^T = P_1^T Q_1 R_1 P_2^T$ as list(P1., Q, R, P2.) or list(P1., Q1, R1, P2.) depending on optional logical argument complete. P1. and P2. are pMatrix, Q and Q1 are dgeMatrix, R is a dgCMatrix, and R1 is a dtCMatrix.

expand signature(x = "CHMfactor") as expand2, but returning list(P, L).

expand2(x)[["P"]]

and expand2(x)[["P1."]] represent the same permutation matrix $P_1$ but have opposite margin slots and inverted perm slots. The components of expand2(x) do not preserve x@Dimnames.

expand signature(x = "sparseLU") as expand2, but returning list(P, L, U, Q).

expand2(x)[["Q"]]

and expand2(x)[["P2."]] represent the same permutation matrix $P_2$ but have opposite margin slots and inverted perm slots. expand2(x)[["P"]]

represents the permutation matrix $P_1$ rather than its transpose $P_1^T$; it is expand2(x)[["P1."]] with an inverted perm slot.

expand2(x)[["L"]]

and expand2(x)[["U"]]

represent the same unit lower triangular matrix $L$, but with diag slot equal to "N" and "U", respectively. expand2(x)[["L"]]

and expand2(x)[["U"]]

store the permuted first and second components of x@Dimnames in their Dimnames slots.

expand signature(x = "denseLU") as expand2, but returning list(L, U, P).

expand2(x)[["P"]]

and expand2(x)[["P1."]] are identical modulo Dimnames. The components of expand2(x) do not preserve x@Dimnames.

See Also

The virtual class MatrixFactorization of matrix factorizations.

Generic functions Cholesky, BunchKaufman, Schur, lu, and qr for computing factorizations.

Examples

showMethods("expand1", inherited = FALSE)
showMethods("expand2", inherited = FALSE)
set.seed(0)

(A <- Matrix(rnorm(9L, 0, 10), 3L, 3L))
(lu.A <- lu(A))
(e.lu.A <- expand2(lu.A))
stopifnot(exprs = {
  is.list(e.lu.A)
  identical(names(e.lu.A), c("P1.", "L", "U"))
  all(sapply(e.lu.A, is, "Matrix"))
  all.equal(as(A, "matrix"), as(Reduce(`%*%`, e.lu.A), "matrix"))
})
expm-methods

Matrix Exponential

Description

Compute the exponential of a matrix.

Usage

expm(x)

Arguments

x

a matrix, typically inheriting from the dMatrix class.

Details

The exponential of a matrix is defined as the infinite Taylor series \( \expm(A) = I + A + A^2/2! + A^3/3! + \ldots \) (although this is definitely not the way to compute it). The method for the dgeMatrix class uses Ward’s diagonal Pade’ approximation with three step preconditioning, a recommendation from Moler & Van Loan (1978) “Nineteen dubious ways…”.

Value

The matrix exponential of x.

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


for historical reference mostly:

See Also

Package `expm`, which provides newer (in some cases faster, more accurate) algorithms for computing the matrix exponential via its own (non-generic) function `expm()`.

Generic function `Schur`.

Examples

```r
(m1 <- Matrix(c(1,0,1,1), ncol = 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), ncol = 2))
(e2 <- expm(m2))
(m3 <- Matrix(cbind(0,rbind(6*diag(3),0))))# sparse!
(e3 <- expm(m3)) # upper triangular
```

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

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

Value

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

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

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

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

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")))
## https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/counterx/counterx.htm
str(jgl <- readMM(system.file("external/jgl009.mtx", package = "Matrix")))
```

## 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 untar step:
  u. <- url("https://www.cise.ufl.edu/research/sparse/RB/Boeing/msc00726.tar.gz")
  str(sm <- readHB(gzcon(u.)))
}

data(KNex, package = "Matrix")
## 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") ] # (some confirmation of the file's)

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

Multiplies a matrix or vector on the left or right by a factor from a matrix factorization or its transpose.

Usage

\[
\text{facmul}(x, \text{factor}, y, \text{trans} = \text{FALSE}, \text{left} = \text{TRUE}, \ldots)
\]

Arguments

- **x**: a \texttt{MatrixFactorization} object.
- **factor**: a character string indicating a factor in the factorization represented by \(x\), typically an element of \texttt{names(expand2(x, \ldots))}.
- **y**: a matrix or vector to be multiplied on the left or right by the factor or its transpose.
- **trans**: a logical indicating if the transpose of the factor should be used, rather than the factor itself.
- **left**: a logical indicating if the \(y\) should be multiplied on the left by the factor, rather than on the right.
- **\ldots**: further arguments passed to or from methods.

Details

\texttt{facmul} is experimental and currently no methods are exported from \texttt{Matrix}.

Value

The value of \texttt{op(M) \%*% y} or \texttt{y \%*% op(M)}, depending on \texttt{left}, where \(M\) is the factor (always \texttt{without} dimnames) and \texttt{op(M)} is \(M\) or \(t(M)\), depending on \texttt{trans}.

Examples

```
## Conceptually, methods for 'facmul' _would_ behave as follows ...
## Not run:
n <- 3L
x <- lu(Matrix(rnorm(n * n), n, n))
y <- rnorm(n)
L <- unname(expand2(x)[[nm <- "L"]])
stopifnot(exprs = {
  all.equal(facmul(x, nm, y, trans = FALSE, left = TRUE), L %*% y)
  all.equal(facmul(x, nm, y, trans = FALSE, left = FALSE), y %*% L)
  all.equal(facmul(x, nm, y, trans = TRUE, left = TRUE), crossprod(L, y))
})
```
all.equal(facmul(x, nm, y, trans = TRUE, left = FALSE), tcrossprod(y, L))
}

## End(Not run)

---

**fastMisc**

"Low Level" Coercions and Methods

**Description**

"Semi-API" functions used internally by `Matrix`, often to bypass S4 dispatch and avoid the associated overhead. These are exported to provide this capability to expert users. Typical users should continue to rely on S4 generic functions to dispatch suitable methods, by calling, e.g., `as(., <class>))` for coercions.

**Usage**

`.M2kind(from, kind = ".", sparse = NA)

`.M2gen(from, kind = ".")
.M2sym(from, ...)  
.M2tri(from, ...)  
.M2diag(from)

.M2v(from)  
.M2m(from)  
.M2unpacked(from)  
.M2packed(from)  
.M2C(from)  
.M2R(from)  
.M2T(from)

.M2V(from)  
m2V(from, kind = ".")

.sparse2dense(from, packed = FALSE)  
.diag2dense(from, kind = ".", shape = "t", packed = FALSE, uplo = "U")  
.ind2dense(from, kind = "n")  
.m2dense(from, class = ".ge", uplo = "U", diag = "N", trans = FALSE)

.dense2sparse(from, repr = "C")  
.diag2sparse(from, kind = ".", shape = "t", repr = "C", uplo = "U")  
.ind2sparse(from, kind = "n", repr = ".")  
.m2sparse(from, class = ".gC", uplo = "U", diag = "N", trans = FALSE)

.tCRT(x, lazy = TRUE)
.diag.dsC(x, Chx = Cholesky(x, LDL = TRUE), res.kind = "diag")
.solve.dgC.lu (a, b, tol = .Machine$double.eps, check = TRUE)
.solve.dgC.qr (a, b, order = 3L, check = TRUE)
.solve.dgC.chol(a, b, check = TRUE)
.updateCHMfactor(object, parent, mult = 0)

Arguments

from, x, a, b  a Matrix, matrix, or vector.

kind  a string ("."", "n", "l", or "d") specifying the “kind” of the result. "." indicates that the kind of from should be preserved. "," is equivalent to "z" if from is complex and to "d" otherwise. "n" indicates that the result should inherit from nMatrix or nsparseVector (and so on).

shape  a string ("."", "g", "s", or "t") specifying the “shape” of the result. "." indicates that the shape of from should be preserved. "g" indicates that the result should inherit from generalMatrix (and so on).

repr  a string ("."", "C", "R", or "T") specifying the sparse representation of the result. "." is accepted only by .ind2sparse and indicates the most efficient representation, which is "C" ("R") for margin = 2 (1). "C" indicates that the result should inherit from CsparseMatrix (and so on).

packed  a logical indicating if the result should inherit from packedMatrix rather than from unpackedMatrix. It is ignored for from inheriting from generalMatrix.

sparse  a logical indicating if the result should inherit from sparseMatrix rather than from denseMatrix. If NA, then the result will be formally sparse if and only if from is.

uplo  a string ("U" or "L") indicating whether the result (if symmetric or triangular) should store the upper or lower triangle of from. The elements of from in the opposite triangle are ignored.

diag  a string ("N" or "U") indicating whether the result (if triangular) should be formally nonunit or unit triangular. In the unit triangular case, the diagonal elements of from are ignored.

trans  a logical indicating if the result should be a 1-row matrix rather than a 1-column matrix where from is a vector but not a matrix.

class  a string whose first three characters specify the class of the result. It should match the pattern "^[.nld][ge|sy|tr|sp|tp]" for .m2dense and "^[.nld][gst][CRT]" for .m2sparse, where "." in the first position is equivalent to "l" for logical arguments and "d" for numeric arguments.

...  optional arguments passed to isSymmetric or isTriangular.

lazy  a logical indicating if the transpose should be constructed with minimal allocation, but possibly without preserving representation.

Chx  optionally, the Cholesky(x, ...) factorization of x. If supplied, then x is unused.
res.kind  a string in c("trace", "sumLog", "prod", "min", "max", "range", "diag", "diagBack").
tol  see lu-methods.
order  see qr-methods.
check  a logical indicating if the first argument should be tested for inheritance from
dgCMatrix and coerced if necessary. Set to FALSE for speed only if it is known
to already inherit from dgCMatrix.
object  a Cholesky factorization inheriting from virtual class CHMfactor, almost always
the result of a call to generic function Cholesky.
parent  an object of class dsCMat or class dgCMatrix.
mult  a numeric vector of positive length. Only the first element is used, and that must
be finite.

Details

Functions with names of the form .<A>2<B> implement coercions from virtual class A to the “nearest” non-virtual subclass of virtual class B, where the virtual classes are abbreviated as follows:

M Matrix
V sparseVector
m matrix
v vector
dense denseMatrix
unpacked unpackedMatrix
packed packedMatrix
sparse CsparseMatrix, RsparseMatrix, or TsparseMatrix
C CsparseMatrix
R RsparseMatrix
T TsparseMatrix
gen generalMatrix
sym symmetricMatrix
tri triangularMatrix
diag diagonalMatrix
ind indMatrix

Abbreviations should be seen as a guide, rather than as an exact description of behaviour. Notably,
.m2dense, .m2sparse, and .m2V accept vectors that are not matrices.

.tCRT(x): If lazy = TRUE, then .tCRT constructs the transpose of x using the most efficient
representation, which for ‘CRT’ is ‘RCT’. If lazy = FALSE, then .tCRT preserves the representation
of x, behaving as the corresponding methods for generic function t.
.diag.dsC(x): .diag.dsC computes (or uses if Ch is supplied) the Cholesky factorization of x as \(LDL'\) in order to calculate one of several possible statistics from the diagonal entries of \(D\). See res.kind under ‘Arguments’.

.solve.dgC.*(a, b): .solve.dgC.lu(a, b) needs a square matrix a. .solve.dgC.qr(a, b) needs a “long” matrix a, with nrow(a) >= ncol(a). .solve.dgC.chol(a, b) needs a “wide” matrix a, with nrow(a) <= ncol(a).

All three may be used to solve sparse linear systems directly. Only .solve.dgC.qr and .solve.dgC.chol be used to solve sparse least squares problems.

.updateCHMfactor(object, parent, mult): .updateCHMfactor updates object with the result of Cholesky factorizing \(F(parent) + mult[1] \times diag(nrow(parent))\), i.e., \(F(parent)\) plus \(mult[1]\) times the identity matrix, where \(F = \text{id}\) if \(parent\) is a dsCMatrix and \(F = \text{tcrossprod}\) if \(parent\) is a dgCMatrix. The nonzero pattern of \(F(parent)\) must match that of \(S\) if object = Cholesky(S, ...).

Examples

```r
d. <- diag(x = c(1, 1, 2, 3, 5, 8))
d.0 <- Diagonal(x = c(0, 0, 0, 3, 5, 8))
s. <- toeplitz(as.double(1:6))
c. <- new("dgCMatrix", Dim = c(3L, 4L),
     p = c(0L, 1L, 1L, 1L, 3L), i = c(1L, 0L, 2L), x = c(-8, 2, 3))
stopifnot(exprs = {
  identical(.M2tri (d.), as(d., "triangularMatrix"))
  identical(.M2sym (d.), as(d., "symmetricMatrix"))
  identical(.M2diag(d.), as(d., "diagonalMatrix"))
  identical(.M2kind(c., "l"),
            as(c., "lMatrix"))
  identical(.M2kind(.sparse2dense(c.), "l"),
            as(as(c., "denseMatrix"), "lMatrix"))
  identical(.diag2sparse(.diag2dense(D.0, ".", "t", "C"),
                         .dense2sparse(.diag2dense(D.0, ".", "m", TRUE), "C"))
  identical(.M2gen(.diag2dense(D.0, ".", "s", FALSE)),
            .sparse2dense(.M2gen(.diag2sparse(D.0, ".", "s", "T"))))
  identical(s.,
            .M2m(.m2sparse(s., ".R")))
  identical(s. * lower.tri(s.) + diag(1, 6L),
            .M2m(.m2dense (s., ".tr", "L", "U")))
  identical(.M2R(c.), .M2R(.M2T(c.)))
  identical(.tCRT(c.), .M2R(t(c.)))
})
```

```r
a <- tcrossprod(c.)/6 + Diagonal(3, 1/3); A[1,2] <- 3; A
stopifnot(exprs = {
  is.numeric( x. <- c(2.2, 0, -1.2 ) )
  all.equal(x., .solve.dgC.lu(A, c(1,0,0), check=FALSE))
  all.equal(x., .solve.dgC.qr(A, c(1,0,0), check=FALSE))
})
```
## Solving sparse least squares:

X <- rbind(A, Diagonal(3)) # design matrix X (for L.S.)
Xt <- t(X) # *transposed* X (for L.S.)
(y <- drop(crossprod(Xt, 1:3)) + c(-1,1)/1000) # small rand.err.
str(solveCh <- .solve.dgC.chol(Xt, y, check=FALSE)) # Xt *is* dgC..
stopifnot(exprs = {
  all.equal(solveCh$coef, 1:3, tol = 1e-3)# rel.err ~ 1e-4
  all.equal(solveCh$coef, drop(solve(tcrossprod(Xt), Xt %*% y)))
  all.equal(solveCh$coef, .solve.dgC.qr(X, y, check=FALSE))
})

---

forceSymmetric-methods

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

### Description

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

### Usage

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

### Arguments

- **x**
  
  any square matrix (of numbers), either `"traditional"` (matrix) or inheriting from `Matrix`.

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

### Value

a square matrix inheriting from class `symmetricMatrix`.

### See Also

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

### Examples

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

## result can be pretty surprising:
(M <- Matrix(1:36, 6))
forceSymmetric(M) # symmetric, hence very different in lower triangle
(tm <- tril(M))
forceSymmetric(tm)

formatSparseM

Formatting Sparse Numeric Matrices Utilities

Description

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

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

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

Usage

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

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

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 "1" instead of "1" (to emphasize that the 1’s are "structural").
digits
- significant digits to use for printing, see print.default.

cx
- (optional) character matrix; a formatted version of x, still with strings such as "0.00" for the zeros.

iN0
- (optional) integer vector, specifying the location of the non-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
- 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))
**Hilbert**

**Objects from the Class**

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

**Slots**

- **Dim, Dimnames** inherited from virtual class `Matrix`.
- **factors** a list of `MatrixFactorization` objects caching factorizations of the matrix. Typically, it is initialized as an empty list and updated “automagically” whenever a factorization is computed.

**Extends**

Class "Matrix", directly.

**See Also**

Virtual classes `symmetricMatrix`, `triangularMatrix`, and `diagonalMatrix`.

---

**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**

Hilbert(n)

**Arguments**

- **n** a non-negative integer.

**Value**

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

**See Also**

the class `dpoMatrix`

**Examples**

Hilbert(6)
Methods for image() in Package 'Matrix'

Description

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

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

Usage

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

Arguments

- **x**: a Matrix object, i.e., fulfilling is(x, "Matrix").
- **xlim, ylim**: x- and y-axis limits; may be used to “zoom into” matrix. Note that $x, y$ “feel reversed”: ylim is for the rows (= 1st index) and xlim for the columns (= 2nd index). For convenience, when the limits are integer valued, they are both extended by 0.5; also, ylim is always used decreasingly.
- **aspect**: aspect ratio specified as number (y/x) or string; see levelplot.
- **sub, xlab, ylab**: axis annotation with sensible defaults; see plot.default.
- **cuts**: number of levels the range of matrix values would be divided into.
- **useRaster**: logical indicating if raster graphics should be used (instead of the tradition rectangle vector drawing). If true, panel.levelplot.raster (from lattice package) is used, and the colorkey is also done via rasters, see also levelplot and possibly grid.raster.
  Note that using raster graphics may often be faster, but can be slower, depending on the matrix dimensions and the graphics device (dimensions).
- **useAbs**: logical indicating if 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.
image-methods

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

```r
showMethods(image)
## And if you want to see the method definitions:
showMethods(image, includeDefs = TRUE, inherited = FALSE)

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

cCA <- Cholesky(crossprod(CAex), Imult = .01)
## See ?print.trellis --- place two image() plots side by side:
print(image(cCA, main="Cholesky(crossprod(CAex), Imult = .01)")
  , split=c(x=1,y=1,nx=2, ny=1), more=TRUE)
print(image(cCA, useAbs=TRUE)
  , split=c(x=2,y=1,nx=2, ny=1))

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

## Drawing borders around each rectangle;
  # again, viewing depends very much on the device:
image(USCounties[1:400,1:200], lwd=.1)
## Using (xlim,ylim) has advantage : matrix dimension and (col/row) indices:
image(USCounties, c(1,200), c(1,400), lwd=.1)
image(USCounties, c(1,300), c(1,200), lwd=.5)
image(USCounties, c(1,300), c(1,200), lwd=.01)
## These 3 are all equivalent :
(I1 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE))
I2 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE, border.col=NA)
I3 <- image(USCounties, c(1,100), c(1,100), useAbs=FALSE, lwd=2, border.col=NA)
stopifnot(all.equal(I1, I2, check.environment=FALSE),
  all.equal(I2, I3, check.environment=FALSE))
## using an opaque border color
image(USCounties, c(1,100), c(1,100), useAbs=FALSE, lwd=3, border.col = adjustcolor("skyblue", 1/2))

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

---

### index-class

Virtual Class “index” of Index Vectors

#### Description

Class `index` is a virtual class designating index vectors, or “subscripts”, for (possibly named) vectors and arrays. It is typically used in signatures of methods for the subscript and subassignment operators, namely `[` and `<-`. It is implemented as a union of the atomic vector classes `numeric`, `logical`, and `character`.

#### See Also

`[`, `[-methods`, and `[-<--methods`.

#### Examples

```r
showClass("index")
```
Description

The `indMatrix` class is the class of row and column *index* matrices, stored as 1-based integer index vectors. A row (column) index matrix is a matrix whose rows (columns) are standard unit vectors. Such matrices are useful when mapping observations to discrete sets of covariate values.

Multiplying a matrix on the left by a row index matrix is equivalent to indexing its rows, i.e., sampling the rows “with replacement”. Analogously, multiplying a matrix on the right by a column index matrix is equivalent to indexing its columns. Indeed, such products are implemented in `Matrix` as indexing operations; see ‘Details’ below.

A matrix whose rows and columns are standard unit vectors is called a *permutation* matrix. This special case is designated by the `pMatrix` class, a direct subclass of `indMatrix`.

Details

The transpose of an index matrix is an index matrix with identical `perm` but opposite `margin`. Hence the transpose of a row index matrix is a column index matrix, and vice versa.

The cross product of a row index matrix `R` and itself is a diagonal matrix whose diagonal entries are the number of entries in each column of `R`.

Given a row index matrix `R` with `perm` slot `p`, a column index matrix `C` with `perm` slot `q`, and a matrix `M` with conformable dimensions, we have

- $RM = R \times M = M[p, ]$
- $MC = M \times C = M[, q]$
- $C'M = \text{crossprod}(C, M) = M[q, ]$
- $MR' = \text{tcrossprod}(M, R) = M[, p]$
- $R'R = \text{crossprod}(R) = \text{Diagonal}(x=\text{tabulate}(p, \text{ncol}(R)))$
- $CC' = \text{tcrossprod}(C) = \text{Diagonal}(x=\text{tabulate}(q, \text{nrow}(C)))$

Operations on index matrices that result in index matrices will accordingly return an `indMatrix`. These include products of two column index matrices and (equivalently) column-indexing of a column index matrix (when dimensions are not dropped). Most other operations on `indMatrix` treat them as sparse nonzero pattern matrices (i.e., inheriting from virtual class `nsparseMatrix`). Hence vector-valued subsets of `indMatrix`, such as those given by `diag`, are always of type "logical".

Objects from the Class

Objects can be created explicitly with calls of the form `new("indMatrix", ...)`, but they are more commonly created by coercing 1-based integer index vectors, with calls of the form `as(. , "indMatrix")`; see ‘Methods’ below.

Slots

- `margin` an integer, either 1 or 2, specifying whether the matrix is a row (1) or column (2) index.
perm a 1-based integer index vector, i.e., a vector of length Dim[margin] with elements taken from 1:Dim[1+margin%%2].

Dim,Dimnames inherited from virtual superclass Matrix.

Extends

Classes "sparseMatrix" and "generalMatrix", directly.

Methods

%% signature(x = "indMatrix", y = "Matrix") and others listed by showMethods("%%", classes = "indMatrix"): matrix products implemented where appropriate as indexing operations.

coerce signature(from = "numeric", to = "indMatrix"): supporting typical indMatrix construction from a vector of positive integers. Row indexing is assumed.

coerce signature(from = "list", to = "indMatrix"): supporting indMatrix construction for row and column indexing, including index vectors of length 0 and index vectors whose maximum is less than the number of rows or columns being indexed.

coerce signature(from = "indMatrix", to = "matrix"): coercion to a traditional matrix of logical type, with FALSE and TRUE in place of 0 and 1.

t signature(x = "indMatrix"): the transpose, which is an indMatrix with identical perm but opposite margin.

rowSums,rowMeans,colSums,colMeans signature(x = "indMatrix"): row and column sums and means.

rbind2,cbind2 signature(x = "indMatrix", y = "indMatrix"): row-wise catenation of two row index matrices with equal numbers of columns and column-wise catenation of two column index matrices with equal numbers of rows.

kroncker signature(X = "indMatrix", Y = "indMatrix"): Kronecker product of two row index matrices or two column index matrices, giving the row or column index matrix corresponding to their “interaction”.

Author(s)

Fabian Scheipl at ‘uni-muenchen.de’, building on the existing class pMatrix after a nice hike’s conversation with Martin Maechler. Methods for crossprod(x, y) and kroncker(x, y) with both arguments inheriting from indMatrix were made considerably faster thanks to a suggestion by Boris Vaillant. Diverse tweaks by Martin Maechler and Mikael Jagan, notably the latter’s implementation of margin, prior to which the indMatrix class was designated only for row index matrices.

See Also

Subclass pMatrix of permutation matrices, a special case of index matrices; virtual class nMatrix of nonzero pattern matrices, 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, "nMatrix")
s10[1:7, 1:4] # gives an "ngTMatrix" (most economic!)
s10[1:4, ] # preserves "indMatrix"-class

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

invertPerm

Utilities for Permutation Vectors

Description

invertPerm and signPerm compute the inverse and sign of a length-\( n \) permutation vector. isPerm tests if a length-\( n \) integer vector is a valid permutation vector. asPerm coerces a length-\( m \) transposition vector to a length-\( n \) permutation vector, where \( m \leq n \).
Usage

invertPerm(p, off = 1L, ioff = 1L)
signPerm(p, off = 1L)
isPerm(p, off = 1L)
asPerm(pivot, off = 1L, ioff = 1L, n = length(pivot))

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

Arguments

p an integer vector of length n.
pivot an integer vector of length m.
off an integer offset, indicating that p is a permutation of off+0:(n-1) or that pivot contains m values sampled with replacement from off+0:(n-1).
ioff an integer offset, indicating that the result should be a permutation of ioff+0:(n-1).
n an integer greater than or equal to m, indicating the length of the result. Transpositions are applied to a permutation vector initialized as seq_len(n).
zero.p a logical. Equivalent to off=0 if TRUE and off=1 if FALSE.
zero.res a logical. Equivalent to ioff=0 if TRUE and ioff=1 if FALSE.

Details

invertPerm(p, off, ioff=1) is equivalent to order(p) or sort.list(p) for all values of off. For the default value off=1, it returns the value of p after p[p] <- seq_along(p).

invPerm is a simple wrapper around invertPerm, retained for backwards compatibility.

Value

By default, i.e., with off=1 and ioff=1:

invertPerm(p) returns an integer vector of length length(p) such that p[invertPerm(p)] and invertPerm(p)[p] are both seq_along(p), i.e., the identity permutation.

signPerm(p) returns 1 if p is an even permutation and -1 otherwise (i.e., if p is odd).

isPerm(p) returns TRUE if p is a permutation of seq_along(p) and FALSE otherwise.

asPerm(pivot) returns the result of transposing elements i and pivot[i] of a permutation vector initialized as seq_len(n), for i in seq_along(pivot).

See Also

Class pMatrix of permutation matrices.

Examples

p <- sample(10L) # a random permutation vector
ip <- invertPerm(p)
s <- signPerm(p)
## 'p' and 'ip' are indeed inverses:
stopifnot(exprs = {
  isPerm(p)
  isPerm(ip)
  identical(s, 1L) || identical(s, -1L)
  identical(s, signPerm(ip))
  identical(p[ip], 1:10)
  identical(ip[p], 1:10)
  identical(invertPerm(ip), p)
})

## Product of transpositions (1 2)(2 1)(4 3)(6 8)(10 1) = (3 4)(6 8)(1 10)
pivot <- c(2L, 1L, 3L, 3L, 5L, 8L, 7L, 8L, 9L, 1L)
q <- asPerm(pivot)
stopifnot(exprs = {
  identical(q, c(10L, 2L, 4L, 3L, 5L, 8L, 7L, 6L, 9L, 1L))
  identical(q[q], seq_len(10L)) # because the permutation is odd:
  signPerm(q) == -1L
})

invPerm # a less general version of 'invertPerm'

---

**is.na-methods**

*is.na(), is.finite() Methods for 'Matrix' Objects*

### Description
Methods for generic functions `anyNA()`, `is.na()`, `is.nan()`, `is.infinite()`, and `is.finite()`, for objects inheriting from virtual class `Matrix` or `sparseVector`.

### Usage
```r
## S4 method for signature 'denseMatrix'
is.na(x)
## S4 method for signature 'sparseMatrix'
is.na(x)
## S4 method for signature 'diagonalMatrix'
is.na(x)
## S4 method for signature 'indMatrix'
is.na(x)
## S4 method for signature 'sparseVector'
is.na(x)
## ...
## and likewise for anyNA, is.nan, is.infinite, is.finite
```

### Arguments
- `x` an R object, here a sparse or dense matrix or vector.
is.null.DN

Value

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

See Also

`NA`, `NaN`, `Inf`

Examples

```r
(M <- Matrix(1:6, nrow = 4, ncol = 3,
    dimnames = list(letters[1:4], LETTERS[1:3])))
stopifnot(!anyNA(M), !any(is.na(M)))
M[2:3, 2] <- NA
(inM <- is.na(M))
stopifnot(anyNA(M), sum(inM) == 2)

(A <- spMatrix(nrow = 10, ncol = 20,
    i = c(1, 3:8), j = c(2, 9, 6:10), x = 7 * (1:7)))
stopifnot(!anyNA(A), !any(is.na(A)))
(inA <- is.na(A))
stopifnot(anyNA(A), sum(inA) == 1 + 1 + 5)
```

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 \texttt{Matrix},
as the latter will always have \texttt{dimnames} \texttt{list(NULL, NULL)} exactly, in such a case.

Author(s)

Martin Maechler

See Also

\texttt{is.null}, \texttt{dimnames}, \texttt{matrix}.

Examples

\begin{verbatim}
m1 <- m2 <- m3 <- m4 <- m <- matrix(round(100 * rnorm(6)), 2, 3)
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
c.o <- capture.output
cm <- c.o(m)
stopifnot(exprs = {
m == m1; m == m2; m == m3; m == m4
identical(cm, c.o(m1)); identical(cm, c.o(m2))
identical(cm, c.o(m3)); identical(cm, c.o(m4))
})

hasNoDimnames <- function(.) is.null.DN(dimnames(.))
stopifnot(exprs = {
hasNoDimnames(m)
hasNoDimnames(m1); hasNoDimnames(m2)
hasNoDimnames(m3); hasNoDimnames(m4)
hasNoDimnames(Matrix(m) -> M)
hasNoDimnames(as(M, "sparseMatrix"))
})
\end{verbatim}
of implicit class "matrix". Methods are defined here for various proper and virtual classes in Matrix, so that isSymmetric works for all objects inheriting from virtual class "Matrix".

Usage

```r
## S4 method for signature 'denseMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'CsparseMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'RsparseMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'TsparseMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'diagonalMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'indMatrix'
isSymmetric(object, checkDN = TRUE, ...)
## S4 method for signature 'dgeMatrix'
isSymmetric(object, checkDN = TRUE, tol = 100 * .Machine$double.eps, tol1 = 8 * tol, ...)
## S4 method for signature 'dgCMatrix'
isSymmetric(object, checkDN = TRUE, tol = 100 * .Machine$double.eps, ...)
```

Arguments

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

Details

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

- `dn[1]` and `dn[2]` are identical or one is NULL; and
- `ndn <- names(dn)` is NULL or `ndn[1]` and `ndn[2]` are identical or one is the empty string "".

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

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

Value

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

See Also

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

Examples

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

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

---

isTriangular-methods  Test whether a Matrix is Triangular or Diagonal

Description

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

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

Usage

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

Arguments

object
  an R object, typically a matrix.
upper
  a logical, either TRUE or FALSE, in which case TRUE is returned only for upper or lower triangular object: or otherwise NA (the default), in which case TRUE is returned for any triangular object.
...
  further arguments passed to methods (currently unused by Matrix).

Value

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

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

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

Examples

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

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

## Look at implementations:
showMethods("isTriangular", includeDefs = TRUE)
showMethods("isDiagonal", includeDefs = TRUE)

### Khatri-Rao Matrix Product

Description

Computes Khatri-Rao products for any kind of matrices.

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

Usage

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

Arguments

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

Value

a "CsparseMatrix", say R, the Khatri-Rao product of X (n × k) and Y (m × k), is of dimension (n · m) × k, where the j-th column, R[,j] is the kronecker product kronecker(X[,j], Y[,j]).
Note
The current implementation is efficient for large sparse matrices.

Author(s)
Original by Michael Cysouw, Univ. Marburg; minor tweaks, bug fixes etc, by Martin Maechler.

References


See Also
kronecker.

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

nm <- as(m, "nsparseMatrix")
nm <- as(d, "nsparseMatrix")
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[] <- FALSE # all FALSE matrix
stopifnot(all(K1 <- KhatriRao(nd, zm) == 0), identical(dim(K1), c(12L, 4L)),
    all(K2 <- KhatriRao(zm, nd) == 0), identical(dim(K2), c(12L, 4L)))
d0 <- d; d0[] <- 0; m0 <- Matrix(d0[-1,])
stopifnot(all(K3 <- KhatriRao(d0, m) == 0), identical(dim(K3), dim(Kdm)),
    all(K4 <- KhatriRao(m, d0) == 0), identical(dim(K4), dim(Kmd)),
    all(KhatriRao(d0, d0) == 0), all(KhatriRao(m0, d0) == 0),
```

all(KhatriRao(d0, m0) == 0), all(KhatriRao(m0, m0) == 0),
identical(dimnames(KhatriRao(m, d0, make.dimnames=TRUE)), dimnames(Kmd)))

## a matrix with "structural" and non-structural zeros:
m01 <- new("dgCMatrix", i = c(0L, 2L, 0L, 1L), p = c(0L, 0L, 0L, 2L, 4L),
          Dim = 3:4, x = c(1, 0, 1, 0))
D4 <- Diagonal(4, x=1:4) # "as" d
DU <- Diagonal(4)# unit-diagonal: uplo="U"
(K5 <- KhatriRao( d, m01))
K5d <- KhatriRao( d, m01, sparseY=FALSE)
K5Dd <- KhatriRao(D4, m01, sparseY=FALSE)
K5Ud <- KhatriRao(DU, m01, sparseY=FALSE)
(K6 <- KhatriRao(diag(3), t(m01)))
K6D <- KhatriRao(Diagonal(3), t(m01))
K6d <- KhatriRao(diag(3), t(m01), sparseY=FALSE)
K6Dd <- KhatriRao(Diagonal(3), t(m01), sparseY=FALSE)
stopifnot(exprs = {
  all(K5 == K5d)
  identical(cbind(c(7L, 10L), c(3L, 4L)),
            which(K5 != 0, arr.ind = TRUE, useNames=FALSE))
  identical(K5d, K5Dd)
  identical(K6, K6D)
  all(K6 == K6d)
  identical(cbind(3:4, 1L),
            which(K6 != 0, arr.ind = TRUE, useNames=FALSE))
  identical(K6d, K6Dd)
})

---

**KNex**

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

**Description**

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

**Usage**

data(KNex)

**References**

Roger Koenker and Pin Ng (2003). SparseM: A sparse matrix package for R; *J. of Statistical Software,* 8 (6), doi:10.18637/jss.v008.i06
Examples

data(KNex, package = "Matrix")
class(KNex$mm)
dim(KNex$mm)
image(KNex$mm)
str(KNex)

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

head(round(sparse.sol,3))

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

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

kronecker-methods

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

Description

Computes Kronecker products for objects inheriting from "Matrix".
In order to preserved sparseness, we treat 0 * NA as 0, not as 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

(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(13)
(N <- as(M, "nspMatrix")) # "ntCMtMatrix" (upper triangular)
N2 <- as(N, "generalMatrix") # (lost "t"riangularity)
MM <- kronecker(M, M)
NN <- kronecker(N, N) # "dtTMtMatrix" i.e. did keep
NN2 <- kronecker(N2, N2)
stopifnot(identical(NN, MM),

is(NN2, "sparseMatrix"), all(NN2 == NN),

is(NN, "triangularMatrix"))

ldenseMatrix-class

Virtual Class "ldenseMatrix" of Dense Logical Matrices

Description

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

Slots

x: logical vector containing the entries of the matrix.

Dim, Dimnames: see Matrix.

Extends

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

Methods

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")
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"))
crossprod(nM) # pattern sparse
```
Class "lgeMatrix" of General Dense Logical Matrices

Description

This is the class of general dense logical matrices.

Slots

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

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

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

Extends

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

Methods

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

See Also

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

Examples

showClass("lgeMatrix")
str(new("lgeMatrix"))
set.seed(1)
(lM <- Matrix(matrix(rnorm(28), 4,7) > 0))# a simple random lgeMatrix
set.seed(11)
(lC <- Matrix(matrix(rnorm(28), 4,7) > 0))# a simple random lgCMatrix
as(lM, "CsparseMatrix")
lsparseMatrix-class

Sparse logical matrices

Description

The lsparseMatrix class is a virtual class of logical sparse matrices, i.e., sparse matrices with entries TRUE, FALSE, or NA.

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 \(\rightarrow\) TRUE” and “TRUE + FALSE \(\rightarrow\) TRUE”. Note the use of asUniqueT() 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 lMatrix class documentation.

See Also

the class dgCMatrix and dgTMatrix

Examples

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

data(KNex, package = "Matrix")
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")
stopifnot(identical(dmG.1, as((KNex $ mm) != 0,"nMatrix")),
          validObject(lmG.1),
          identical(lmG.1, mmG.1))

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

Symmetric Dense Logical Matrices

**Description**

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

**Objects from the Class**

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

**Slots**

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

**Extends**

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

**Methods**

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

**See Also**

`lgeMatrix, Matrix, t`

**Examples**

```r
(M2 <- Matrix(c(TRUE, NA, FALSE, FALSE), 2, 2)) # logical dense (ltr)
str(M2)
# can
(sM <- M2 | t(M2)) # "lge"
as(sM, "symmetricMatrix")
str(sM <- as(sM, "packedMatrix")) # packed symmetric
```
### Description

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

### Slots

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

### Extends

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

### Methods

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

### See Also

Classes `lgeMatrix, Matrix`; function `t`

### Examples

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

Description

Computes the pivoted LU factorization of an $m \times n$ real matrix $A$, which has the general form

$$P_1AP_2 = LU$$

or (equivalently)

$$A = P'_1LUP'_2$$

where $P_1$ is an $m \times m$ permutation matrix, $P_2$ is an $n \times n$ permutation matrix, $L$ is an $m \times \min(m, n)$ unit lower trapezoidal matrix, and $U$ is a $\min(m, n) \times n$ upper trapezoidal matrix.

Methods for denseMatrix are built on LAPACK routine dgetrf, which does not permute columns, so that $P_2$ is an identity matrix.

Methods for sparseMatrix are built on CXSparse routine cs_lu, which requires $m = n$, so that $L$ and $U$ are triangular matrices.

Usage

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

Arguments

- **x** a finite matrix or Matrix to be factorized, which must be square if sparse.
- **warnSing** a logical indicating if a warning should be signaled for singular x. Used only by methods for dense matrices.
- **errSing** a logical indicating if an error should be signaled for singular x. Used only by methods for sparse matrices.
- **order** an integer in 0:3 passed to CXSparse routine cs_sqr, indicating a strategy for choosing the column permutation $P_2$. 0 means no column permutation. 1, 2, and 3 indicate a fill-reducing ordering of $A + A'$, $A'\tilde{A}$, and $A'\tilde{A}$, where $\tilde{A}$ is $A$ with “dense” rows removed. NA (the default) is equivalent to 2 if tol == 1 and 1 otherwise. Do not set to 0 unless you know that the column order of $A$ is already sensible.
tol: a number. The original pivot element is used if its absolute value exceeds \( \text{tol} \times a \), where \( a \) is the maximum in absolute value of the other possible pivot elements. Set \( \text{tol} < 1 \) only if you know what you are doing.

cache: a logical indicating if the result should be cached in \code{x@factors[["LU"]]}\footnote{Note that caching is experimental and that only methods for classes extending \code{generalMatrix} or \code{symmetricMatrix} will have this argument.}\footnote{Further arguments passed to or from methods.}

Details

What happens when \( x \) is determined to be near-singular differs by method. The method for class \code{dgeMatrix} completes the factorization, warning if \code{warnSing = TRUE} and in any case returning a valid \code{denseLU} object. Users of this method can detect singular \( x \) with a suitable warning handler; see \code{tryCatch}. In contrast, the method for class \code{dgCMatrix} abandons further computation, throwing an error if \code{errSing = TRUE} and otherwise returning \code{NA}. Users of this method can detect singular \( x \) with an error handler or by setting \code{errSing = FALSE} and testing for a formal result with \code{is(., "sparseLU")}.

Value

An object representing the factorization, inheriting from virtual class \code{LU}. The specific class is \code{denseLU} unless \( x \) inherits from virtual class \code{sparseMatrix}, in which case it is \code{sparseLU}.

References

The LAPACK source code, including documentation; see \url{https://netlib.org/lapack/double/dgetrf.f}.


See Also

Classes \code{denseLU} and \code{sparseLU} and their methods.

Classes \code{dgeMatrix} and \code{dgCMatrix}.

Generic functions \code{expand1} and \code{expand2}, for constructing matrix factors from the result.

Generic functions \code{Cholesky}, \code{BunchKaufman}, \code{Schur}, and \code{qr}, for computing other factorizations.

Examples

```r
showMethods("lu", inherited = FALSE)
set.seed(8)

#---- Dense -----------------------------------------------
(A1 <- Matrix(rnorm(9L), 3L, 3L))
```
(lu.A1 <- lu(A1))

(A2 <- round(10 * A1[, -3L]))
(lu.A2 <- lu(A2))

## A ~ P1’ L U in floating point
str(e.lu.A2 <- expand2(lu.A2), max.level = 2L)
stopifnot(all.equal(A2, Reduce("%*%", e.lu.A2)))

## ---- Sparse ---------------------------------------------------------
A3 <- as(readMM(system.file("external/pores_1.mtx", package = "Matrix")),
  "CsparseMatrix")
(lu.A3 <- lu(A3))

## A ~ P1’ L U P2’ in floating point
str(e.lu.A3 <- expand2(lu.A3), max.level = 2L)
stopifnot(all.equal(A3, Reduce("%*%", e.lu.A3)))

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
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 asUniqueT(byrow=FALSE).

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
mat2triplet() is conceptually the inverse function of spMatrix and (one case of) sparseMatrix.

Examples

mat2triplet # simple definition
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))

matmult-methods  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 'CsparseMatrix,diagonalMatrix'
crossprod(x, y = NULL, boolArith = NA, ...)
## .... and for many more signatures

crossprod(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 `dgeMatrix`, it is much faster to calculate the cross-product of the transpose directly instead of calculating the transpose first and then its cross-product.
boolArith = TRUE for regular ("non cross") matrix products, %*% cannot be specified. Instead, we provide the %&% operator for boolean matrix products.

Value

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

Methods

%*% signature(x = "dgeMatrix", y = "dgeMatrix"): Matrix multiplication; ditto for several other signature combinations, see `showMethods("%*%, class = "dgeMatrix")`.
%*% signature(x = "dtrMatrix", y = "matrix") and other signatures (use `showMethods("%*%, class="dtrMatrix")`): matrix multiplication. Multiplication of (matching) triangular matrices now should remain triangular (in the sense of class `triangularMatrix`).
crossprod signature(x = "dgeMatrix", y = "dgeMatrix"): ditto for several other signatures, use `showMethods("crossprod", class = "dgeMatrix")`, matrix crossproduct, an efficient version of t(x) %*% y.
crossprod signature(x = "CsparseMatrix", y = "missing") returns t(x) %*% x as an dsCMatrix object.
crossprod signature(x = "TsparseMatrix", y = "missing") returns t(x) %*% x as an dsCMatrix object.
crossprod,tcrossprod signature(x = "dtrMatrix", y = "matrix") and other signatures, see "%*%" above.
Note

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

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

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

Examples

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

## tcrossprod() is very fast:
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]
```

Matrix

Construct a Classed Matrix

Description

Construct a Matrix of a class that inherits from Matrix.

Usage

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

- **data**: an optional numeric data vector or matrix.
- **nrow**: when data is not a matrix, the desired number of rows
- **ncol**: when data is not a matrix, the desired number of columns
- **byrow**: logical. If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.
- **dimnames**: a **dimnames** attribute for the matrix: a list of two character components. They are set if not **NULL** (as per default).
- **sparse**: logical or **NULL**, specifying if the result should be sparse or not. By default, it is made sparse when more than half of the entries are 0.
- **doDiag**: logical indicating if a **diagonalMatrix** object should be returned when the resulting matrix is diagonal (mathematically). As class **diagonalMatrix** extends **sparseMatrix**, this is a natural default for all values of **sparse**. Otherwise, if doDiag is false, a dense or sparse (depending on **sparse**) **symmetric** matrix will be returned.
- **forceCheck**: logical indicating if the checks for structure should even happen when data is already a "Matrix" object.

Details

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

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

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

Value

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

See Also

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

Examples

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

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

Matrix(1:9, nrow=3,
     dimnames = list(c("a", "b", "c"), c("A", "B", "C")))
(I3 <- Matrix(diag(3)))# identity, i.e., unit "diagonalMatrix"
str(I3) # note 'diag = "U"' and the empty 'x' slot

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

---

### Matrix-class

**Virtual Class "Matrix" of Matrices**

#### Description

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

#### Slots

- **Dim** an integer vector of length 2 giving the dimensions of the matrix.
- **Dimnames** a list of length 2. Each element must be NULL or a character vector of length equal to the corresponding element of Dim.
Methods

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

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

dim signature(x = "Matrix"): extract matrix dimensions dim.
dim<- signature(x = "Matrix", value = "ANY"): where value is integer of length 2. Allows to reshape Matrix objects, but only when prod(value) == prod(dim(x)).
dimnames signature(x = "Matrix"): extract dimnames.
dimnames<- signature(x = "Matrix", value = "list"): set the dimnames to a list of length 2, see dimnames<-.

length signature(x = "Matrix"): simply defined as prod(dim(x)) (and hence of mode "double").
show signature(object = "Matrix"): show method for printing. For printing sparse matrices, see printSpMatrix.
zapsmall signature(x = "Matrix"): typically used for "dMatrix": round() matrix entries such that (relatively) very small entries become zero exactly.
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.
Virtual Classes Not Yet Really Implemented and Used

Description

\texttt{iMatrix} is the virtual class of all integer (S4) matrices. It extends the \texttt{Matrix} class directly.

\texttt{zMatrix} is the virtual class of all \texttt{complex} (S4) matrices. It extends the \texttt{Matrix} class directly.

Examples

\begin{verbatim}
showClass("iMatrix")
showClass("zMatrix")
\end{verbatim}
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.

dropVirtual logical indicating if virtual classes are included or not.

... further arguments are passed to selectSuperClasses().

Value

a character string

Author(s)

Martin Maechler, 24 Mar 2009

See Also

Matrix, the mother of all Matrix classes.

Examples

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

Virtual Class "MatrixFactorization" of Matrix Factorizations

Description

MatrixFactorization is the virtual class of factorizations of $m \times n$ matrices $A$, having the general form

$$P_1 A P_2 = A_1 \cdots A_p$$

or (equivalently)

$$A = P_1' A_1 \cdots A_p P_2'$$

where $P_1$ and $P_2$ are permutation matrices. Factorizations requiring symmetric $A$ have the constraint $P_2 = P_1'$, and factorizations without row or column pivoting have the constraints $P_1 = I_m$ and $P_2 = I_n$, where $I_m$ and $I_n$ are the $m \times m$ and $n \times n$ identity matrices.

CholeskyFactorization, BunchKaufmanFactorization, SchurFactorization, LU, and QR are the virtual subclasses of MatrixFactorization containing all Cholesky, Bunch-Kaufman, Schur, LU, and QR factorizations, respectively.

Slots

- Dim an integer vector of length 2 giving the dimensions of the factorized matrix.
- Dimnames a list of length 2 preserving the dimnames of the factorized matrix. Each element must be NULL or a character vector of length equal to the corresponding element of Dim.

Methods

determinant signature(x = "MatrixFactorization", logarithm = "missing"): sets logarithm = TRUE and recalls the generic function.
dim signature(x = "MatrixFactorization"): returns x@Dim.
dimnames signature(x = "MatrixFactorization"): returns x@Dimnames.
dimnames<- signature(x = "MatrixFactorization", value = "NULL"): returns x with x@Dimnames set to list(NULL, NULL).
dimnames<- signature(x = "MatrixFactorization", value = "list"): returns x with x@Dimnames set to value.
length signature(x = "MatrixFactorization"): returns prod(x@Dim).
show signature(object = "MatrixFactorization"): prints the internal representation of the factorization using str.
solve signature(a = "MatrixFactorization", b = .): see solve-methods.
unname signature(obj = "MatrixFactorization"): returns obj with obj@Dimnames set to list(NULL, NULL).
See Also
Classes extending CholeskyFactorization, namely \texttt{Cholesky}, \texttt{pCholesky}, and \texttt{CHMfactor}.
Classes extending BunchKaufmanFactorization, namely \texttt{BunchKaufman} and \texttt{pBunchKaufman}.
Classes extending SchurFactorization, namely \texttt{Schur}.
Classes extending LU, namely \texttt{denseLU} and \texttt{sparseLU}.
Classes extending QR, namely \texttt{sparseQR}.
Generic functions \texttt{Cholesky}, \texttt{BunchKaufman}, \texttt{Schur}, \texttt{lu}, and \texttt{qr} for computing factorizations.
Generic functions \texttt{expand1} and \texttt{expand2} for constructing matrix factors from \texttt{MatrixFactorization} objects.

Examples

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

---

\textbf{ndenseMatrix-class}

Virtual Class "ndenseMatrix" of Dense Logical Matrices

Description

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

Slots

- \texttt{x}: logical vector containing the entries of the matrix.
- \texttt{Dim, Dimnames}: see \texttt{Matrix}.

Extends

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

Methods

- \texttt{\%\% \% signature(x = "nsparseMatrix", y = "ndenseMatrix")}: ...
- \texttt{\%\% \% signature(x = "ndenseMatrix", y = "nsparseMatrix")}: ...
- \texttt{crossprod signature(x = "nsparseMatrix", y = "ndenseMatrix"): ...}
- \texttt{crossprod signature(x = "ndenseMatrix", y = "nsparseMatrix"): ...}
- \texttt{as.vector signature(x = "ndenseMatrix", mode = "missing"): ...}
- \texttt{diag signature(x = "ndenseMatrix"): extracts the diagonal as for all matrices, see the generic \texttt{diag}().}
- \texttt{which signature(x = "ndenseMatrix"), semantically equivalent to base function \texttt{which(x, arr.ind); for details, see the \texttt{lMatrix} class documentation.}
nearPD

Nearest Positive Definite Matrix

Description

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

Usage

nearPD(x, corr = FALSE, keepDiag = FALSE, 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.

See Also

Class `ngeMatrix` and the other subclasses.

Examples

```
showClass("ndenseMatrix")

as(diag(3) > 0, "ndenseMatrix") # -> "nge"
```
ensureSymmetry: logical; by default, \texttt{symmpart(x)} is used whenever \texttt{isSymmetric(x)} is not true. The user can explicitly set this to \texttt{TRUE} or \texttt{FALSE}, saving the symmetry test. \textit{Beware} however that setting it \texttt{FALSE} for an asymmetric input \texttt{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 \texttt{posdefify} step when \texttt{do2eigen} is \texttt{TRUE}).

maxit: maximum number of iterations allowed.

conv.norm.type: convergence norm type (\texttt{norm(*, type)}) used for Higham algorithm. The default is "I" (infinity), for reasons of speed (and back compatibility); using "F" is more in line with Higham’s proposal.

trace: logical or integer specifying if convergence monitoring should be traced.

Details

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

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

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

Value

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

- \texttt{mat}: a matrix of class \texttt{dpoMatrix}, the computed positive-definite matrix.
- \texttt{eigenvalues}: numeric vector of eigenvalues of \texttt{mat}.
- \texttt{corr}: logical, just the argument \texttt{corr}.
- \texttt{normF}: the Frobenius norm (\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 Oehlschlägel 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) # 1.102 / 1.08

if(requireNamespace("sfsmisc")) {
  m2 <- sfsmisc::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, 0.477, 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, 0.477, 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), ncol=9, byrow=TRUE)
```

```r
near.m <- nearPD(pr, corr=TRUE, base.matrix=TRUE)$mat
near.m
```

```r
round(near.m, 7)
```

```r
near.m.mat <- nearPD(pr, corr=TRUE, do2eigen=FALSE, base.matrix=TRUE)$mat
diff <- near.m.mat - near.m
diff
```

```r
near.m.mat
```

```r
round(near.m.mat, 7)
```

```r
norm(diff) # 0.007 / 0.008
```

```r
if(requireNamespace("sfsmisc")) {
  m2 <- sfsmisc::posdefify(pr) # a simpler approach
  norm(diff) # 0.011, i.e., slightly "less near"
}

round(near.m.mat, 7)
```
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.

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

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

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

Examples

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

dMatrix-class

Class "dMatrix" of Double Precision Matrices

Description
The dMatrix class is the virtual “mother” class of all numeric 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

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

Additional methods contain group methods, such as
**Ops** signature(e1 = "nMatrix", e2 = "...."),...
**Arith** signature(e1 = "nMatrix", e2 = "...."),...
**Compare** signature(e1 = "nMatrix", e2 = "...."),...
**Logic** signature(e1 = "nMatrix", e2 = "...."),...
**Summary** signature(x = "nMatrix", "...."),...

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

**Examples**

```r
getClass("nMatrix")
```

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

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

---

**nnzero-methods**

*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**

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

**Arguments**

- `x` an R object, typically inheriting from class `Matrix` or `numeric`.
- `na.counted` a `logical` describing how `NA`s should be counted. There are three possible settings for `na.counted`:
  - **TRUE** `NA`s 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** `NA`s are omitted from `x` before the non-zero entries are counted.

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

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

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

Methods

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

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

signature(x = "diagonalMatrix") and signature(x = "indMatrix") fast simple methods for these special "sparseMatrix" classes.

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

See Also

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

drop0, zapsmall.

Examples

m <- Matrix(0+1:28, nrow = 4)
m[-3,c(2,4:5,7)] <- m[ 3, 1:4] <- m[1:3, 6] <- 0
(mT <- as(m, "TsparseMatrix"))

nnzero(mT)

(S <- crossprod(mT))

nnzero(S)

str(S) # slots are smaller than nnzero()

stopifnot(nnzero(S) == sum(as.matrix(S) != 0))# failed earlier

data(KNex, package = "Matrix")

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

length(M); stopifnot(length(M) == prod(dim(M)))
nnzero(M) # more relevant than length

## the above are also visible from

str(M)
**Matrix Norms**

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

### Usage
```
norm(x, type, ...)  
```

### 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" aka "2-norm", which is the largest singular value (`svd`) of \( x \).
  The default is "O". Only the first character of type[1] is used.
- **...**: further arguments passed to or from other methods.

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

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

### References

### See Also
- `onenormest()`, an approximate randomized estimate of the 1-norm condition number, efficient for large sparse matrices.
- 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(x) {
  ## norm(NA, "2") did not work until R 4.0.0
  do2 <- getRversion() >= "4.0.0" || !anyNA(x)
  vapply(c("1", "I", "F", "M", if(do2) "2"), norm, 0, x = x)
}
allnorms(x)
allnorms(Hilbert(10))

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

Description

The `nsparseMatrix` class is a virtual class of sparse “pattern” matrices, i.e., binary matrices conceptually with TRUE/FALSE entries. Only the positions of the elements that are TRUE are stored.

These can be stored in the “triplet” form (TsparseMatrix, subclasses ngTMatrix, nsTMatrix, and ntTMatrix which really contain pairs, not triplets) or in compressed column-oriented form (class CsparseMatrix, subclasses ngCMatrix, nsCMatrix, and ntCMatrix) or – rarely – in compressed row-oriented form (class RsparseMatrix, subclasses ngRMatrix, nsRMatrix, and ntRMatrix). The second letter in the name of these non-virtual classes indicates general, symmetric, or triangular.

Objects from the Class

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

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

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

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

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

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

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

Methods

coerce signature(from = "dgCMatrix", to = "ngCMatrix"), and many similar ones; typically you should coerce to "n sparseMatrix" (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, "n sparseMatrix") ## \-> will be a "ngCMatrix"
str(mm) # no \x\' slot
mm <- !mm # no longer sparse
## consistency check:
stopifnot(xor(as( mm, "matrix"),
            as(mm, "matrix")))

## low-level way of adding "non-structural zeros" :
mm <- as(mm, "lsparseMatrix") # "lgCMatrix"
mm@x[2:4] <- c(FALSE, NA, NA)
mm
as(mm, "nMatrix") # NAs *and* non-structural 0 |---\rightarrow \"TRUE\"
data(KNex, package = "Matrix")
nmm <- as(KNex $ mm, "nMatrix")
str(xlx <- crossprod(nmm)) # "nsCMatrix"
stopifnot(isSymmetric(xlx))
image(xlx, main=paste("crossprod(nmm) : Sparse", class(xlx)))

### nsysMatrix-class

**Symmetric Dense Nonzero-Pattern Matrices**

#### Description

The "nsyMatrix" class is the class of symmetric, dense nonzero-pattern matrices in non-packed storage and "nspMatrix" is the class of 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="nsyMatrix")` for details.

#### See Also

`ngeMatrix`, `Matrix`, `t`
Examples

```r
(s0 <- new("nsyMatrix"))

(M2 <- Matrix(c(TRUE, NA, FALSE, FALSE), 2, 2)) # logical dense (ltr)
(sM <- M2 & t(M2)) # -> "lge"
class(sM <- as(sM, "nMatrix")) # -> "nge"
(sM <- as(sM, "symmetricMatrix")) # -> "nsy"
str(sM <- as(sM, "packedMatrix")) # -> "nsp", i.e., 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="ntrMatrix")` for details.

**See Also**

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

showClass("ntMMatrix")

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

pack-methods

Representation of Packed and Unpacked Dense Matrices

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

Usage

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

unpack(x, ...)

Arguments

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

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

symmetric logical (including NA) optionally indicating whether x is symmetric (or triangular).

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

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

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

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

Value

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

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

Examples

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

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

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

Description

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

Slots

uplo: "character"; either "U", for upper triangular, and "L", for lower.

Dim, Dimnames: as all Matrix objects.

Extends


Methods

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

Author(s)

Mikael Jagan

See Also

pack and unpack; its virtual "complement" "unpackedMatrix"; its proper subclasses "dspMatrix", "ltpMatrix", etc.

Examples

showClass("packedMatrix")
showMethods(classes = "packedMatrix")
Description

The `pMatrix` class is the class of permutation matrices, stored as 1-based integer permutation vectors. A permutation matrix is a square matrix whose rows and columns are all standard unit vectors. It follows that permutation matrices are a special case of index matrices (hence `pMatrix` is defined as a direct subclass of `indMatrix`).

Multiplying a matrix on the left by a permutation matrix is equivalent to permuting its rows. Analogously, multiplying a matrix on the right by a permutation matrix is equivalent to permuting its columns. Indeed, such products are implemented in `Matrix` as indexing operations; see ‘Details’ below.

Details

By definition, a permutation matrix is both a row index matrix and a column index matrix. However, the `perm` slot of a `pMatrix` cannot be used interchangeably as a row index vector and column index vector. If `margin=1`, then `perm` is a row index vector, and the corresponding column index vector can be computed as `invPerm(perm)`, i.e., by inverting the permutation. Analogously, if `margin=2`, then `perm` and `invPerm(perm)` are column and row index vectors, respectively.

Given an \( n \times n \) row permutation matrix \( P \) with `perm` slot \( p \) and a matrix \( M \) with conformable dimensions, we have

\[
\begin{align*}
PM & = P \times M = M[p, ] \\
MP & = M \times P = M[, i(p)] \\
P'M & = \text{crossprod}(P, M) = M[i(p), ] \\
MP' & = \text{tcrossprod}(M, P) = M[, p] \\
P'P & = \text{crossprod}(P) = \text{Diagonal}(n) \\
PP' & = \text{tcrossprod}(P) = \text{Diagonal}(n)
\end{align*}
\]

where \( i := \text{invPerm} \).

Objects from the Class

Objects can be created explicitly with calls of the form `new("pMatrix", ...)` , but they are more commonly created by coercing 1-based integer index vectors, with calls of the form `as(. , "pMatrix")`; see ‘Methods’ below.

Slots

`margin,perm` inherited from superclass `indMatrix`. Here, `perm` is an integer vector of length \( \text{Dim}[1] \) and a permutation of \( 1: \text{Dim}[1] \).

`Dim,Ddimnames` inherited from virtual superclass `Matrix`. 
**pMatrix-class**

**Extends**

Class "indMatrix", directly.

**Methods**

%*% signature(x = "pMatrix", y = "Matrix") and others listed by showMethods("%*%", classes = "pMatrix"): matrix products implemented where appropriate as indexing operations.

coerce signature(from = "numeric", to = "pMatrix"): supporting typical pMatrix construction from a vector of positive integers, specifically a permutation of 1:n. Row permutation is assumed.

t signature(x = "pMatrix"): the transpose, which is a pMatrix with identical perm but opposite margin. Coincides with the inverse, as permutation matrices are orthogonal.

solve signature(a = "pMatrix", b = "missing"): the inverse permutation matrix, which is a pMatrix with identical perm but opposite margin. Coincides with the transpose, as permutation matrices are orthogonal. See showMethods("solve", classes = "pMatrix") for more signatures.

determinant signature(x = "pMatrix", logarithm = "logical"): always returning 1 or -1, as permutation matrices are orthogonal. In fact, the result is exactly the sign of the permutation.

**See Also**

Superclass indMatrix of index matrices, for many inherited methods; invPerm, for computing inverse permutation vectors.

**Examples**

```r
(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 %*% pml
pml %*% mm
try(as(as.integer(c(3,3,1)), "pMatrix"))# Error: not a permutation

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

## row-indexing of a <pMatrix> keeps it as an <indMatrix>:
p10[1:3, ]
```
**Description**

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

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

**Usage**

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

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

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

**Arguments**

- `x` an R object inheriting from class `sparseMatrix`.
- `digits` significant digits to use for printing, see `print.default`, the default, `NULL`, corresponds to using `getOption("digits")`.
- `maxp` integer, default from `options(max.print)`, influences how many entries of large matrices are printed at all. Typically should not be smaller than around 1000; values smaller than 100 are silently “rounded up” to 100.
- `cld` the class definition of `x`; must be equivalent to `getClassDef(class(x))` and exists mainly for possible speedup.
- `zero.print` character which should be printed for structural zeroes. The default "." may occasionally be replaced by " " (blank); using "0" would look almost like `print()`ing of non-sparse matrices.
col.names logical or string specifying if and how column names of x should be printed, possibly abbreviated. The default is taken from `options("sparse.colnames")` if that is set, otherwise FALSE unless there are less than ten columns. When TRUE the full column names are printed. When `col.names` is a string beginning with "abb" or "sub" and ending with an integer n (i.e., of the form "abb...<n>"), the column names are `abbreviate()`d or `substring()`ed to (target) length n, see the examples.

note.dropping.colnames logical specifying, when `col.names` is FALSE if the dropping of the column names should be noted, TRUE by default.

uniDiag logical indicating if the diagonal entries of a sparse unit triangular or unit-diagonal matrix should be formatted as "1" instead of "1" (to emphasize that the 1’s are "structural").

col.trailer a string to be appended to the right of each column; this is typically made use of by `show(<sparseMatrix>)` only, when suppressing columns.

suppRows, suppCols logicals or NULL, for `printSpMatrix2()` specifying if rows or columns should be suppressed in printing. If NULL, sensible defaults are determined from `dim(x)` and `options(c("width", "max.print"))`. Setting both to FALSE may be a very bad idea.

align a string specifying how the zero.print codes should be aligned, i.e., padded as strings. The default, "fancy", takes some effort to align the typical zero.print = "." with the position of 0, i.e., the first decimal (one left of decimal point) of the numbers printed, whereas align = "right" just makes use of `print(*, right = TRUE)`.

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

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

... unused optional arguments.

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

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

Examples

```r
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")
```

Description

Computes the pivoted QR factorization of an \( m \times n \) real matrix \( A \), which has the general form

\[
P_1 A P_2 = QR
\]

or (equivalently)

\[
A = P'_1 Q R P'_{2}
\]

where \( P_1 \) and \( P_2 \) are permutation matrices, \( Q = \prod_{j=1}^{n} H_j \) is an \( m \times m \) orthogonal matrix equal to the product of \( n \) Householder matrices \( H_j \), and \( R \) is an \( m \times n \) upper trapezoidal matrix.

denseMatrix use the default method implemented in base, namely qr.default. It is built on LINPACK routine dqrdc and LAPACK routine dgeqp3, which do not pivot rows, so that \( P_1 \) is an identity matrix.

Methods for \texttt{spMatrix} are built on CXSparse routines \texttt{cs_sqr} and \texttt{cs_qr}, which require \( m \geq n \).
Usage

```r
qr(x, ...) # S4 method for signature 'dgCMatrix'
qr(x, order = 3L, ...)
```

Arguments

- `x`: a finite matrix or `Matrix` to be factorized, satisfying `nrow(x) >= ncol(x)` if sparse.
- `order`: an integer in `0:3` passed to CXSparse routine `cs_sqr`, indicating a strategy for choosing the column permutation `P_2`. `0` means no column permutation. `1`, `2`, and `3` indicate a fill-reducing ordering of `A + A', A' A`, and `A'A`, where `A` with “dense” rows removed. Do not set to `0` unless you know that the column order of `A` is already sensible.
- `...`: further arguments passed to or from methods.

Details

If `x` is sparse and structurally rank deficient, having structural rank `r < n`, then `x` is augmented with `(n - r)` rows of (partly non-structural) zeros, such that the augmented matrix has structural rank `n`. This augmented matrix is factorized as described above:

```
P_1 A P_2 = P_1 \begin{bmatrix} A_0 \\ \hline 0 \end{bmatrix} P_2 = QR
```

where `A_0` denotes the original, user-supplied `(m - (n - r)) \times n` matrix.

Value

An object representing the factorization, inheriting from virtual S4 class `QR` or S3 class `qr`. The specific class is `qr` unless `x` inherits from virtual class `sparseMatrix`, in which case it is `sparseQR`.

References


See Also

- Class `sparseQR` and its methods.
- Class `dgCMatrix`.
- Generic function `qr` from `base`, whose default method `qr.default` “defines” the S3 class `qr` of dense QR factorizations.
- Generic functions `expand1` and `expand2`, for constructing matrix factors from the result.
- Generic functions `Cholesky`, `BunchKaufman`, `Schur`, and `lu`, for computing other factorizations.
Examples

showMethods("qr", inherited = FALSE)

## Rank deficient: columns 3 {b2} and 6 {c3} are "extra"
M <- as(cbind(a1 = 1,
        b1 = rep(c(1, 0), each = 3L),
        b2 = rep(c(0, 1), each = 3L),
        c1 = rep(c(1, 0, 0), 2L),
        c2 = rep(c(0, 1, 0), 2L),
        c3 = rep(c(0, 0, 1), 2L)),
        CsparseMatrix")
rownames(M) <- paste0("r", seq_len(nrow(M)))
b <- 1:6
eps <- .Machine$double.eps

## .... [1] full rank ..................................................
## ===> a least squares solution of  \( A x = b \) exists
## and is unique _in exact arithmetic_
(A1 <- M[, -c(3L, 6L)])
(qr.A1 <- qr(A1))
stopifnot(exprs = {
  rankMatrix(A1) == ncol(A1)
  { d1 <- abs(diag(qr.A1@R)); sum(d1 < max(d1) * eps) == 0L }
  rcond(crossprod(A1)) >= eps
  all.equal(qr.coef(qr.A1, b), drop(solve(crossprod(A1), crossprod(A1, b))))
  all.equal(qr.fitted(qr.A1, b) + qr.resid(qr.A1, b))
})

## .... [2] numerically rank deficient with full structural rank .......
## ===> a least squares solution of  \( A x = b \) does not
## exist or is not unique _in exact arithmetic_
(A2 <- M)
(qr.A2 <- qr(A2))
stopifnot(exprs = {
  rankMatrix(A2) == ncol(A2) - 2L
  { d2 <- abs(diag(qr.A2@R)); sum(d2 < max(d2) * eps) == 2L }
  rcond(crossprod(A2)) < eps
  all.equal(qr.fitted(qr.A2, b) + qr.resid(qr.A2, b))
})

## .... [3] numerically and structurally rank deficient ..............
## ===> factorization of _augmented_ matrix with
## full structural rank proceeds as in [2]
rankMatrix

Description

Compute ‘the’ matrix rank, a well-defined functional in theory(*), somewhat ambiguous in practice. We provide several methods, the default corresponding to Matlab’s definition.
(*) The rank of a $n \times m$ matrix $A$, $rk(A)$, is the maximal number of linearly independent columns (or rows); hence $rk(A) \leq \min(n, m)$.

Usage

```r
rankMatrix(x, tol = NULL,
method = c("tolNorm2", "qr.R", "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, $\max(\dim(x)) \cdot \Machine$double.eps is according to Matlab’s default (for its only method which is our method="tolNorm2").
- **method**: a character string specifying the computational method for the rank, can be abbreviated:
  - "tolNorm2": the number of singular values $\geq tol \cdot \max(sval)$;
  - "qrLINPACK": for a dense matrix, this is the rank of $\text{qr}(x, tol, \text{LAPACK}=\text{FALSE})$ (which is $\text{qr}(\ldots)$’$\text{srank}$);
    - 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 $\text{qr-methods}$) to compute the decomposition $QR$.
    - The rank of $R$ is then defined as the number of “non-zero” diagonal entries $d_i$ of $R$, and “non-zero”’s fulfill $|d_i| \geq tol \cdot \max(|d_i|)$.
  - "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(..., do.warn = warn.qr), see below.

qr an R object resulting from qr(x,..), 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 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 di (see above), end with non-finite, typically NaN entries. Then, a warning is signalled (unless warn.qr / do.warn is not true) and NA (specifically, 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 svd() which currently coerces x to a denseMatrix which may be very slow or impossible, depending on the matrix dimensions.

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

Author(s)

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

See Also

qr, svd.

Examples

```r
rankMatrix(cbind(1, 0, 1:3)) # 2

(meths <- eval(formals(rankMatrix)$method))
```

## a "border" case:
```r
H12 <- Hilbert(12)
rankMatrix(H12, tol = 1e-20) # 12; but 11 with default method & tol.
```
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)`.
rcond-methods

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.
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 \(rcond(qr(R(qr(X))),\ldots)\) where \(X\) is \(x\) or \(t(x)\).

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

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

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


See Also

`norm`, `kappa()` from package `base` computes an approximate condition number of a “traditional” matrix, even non-square ones, with respect to the $p = 2$ (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))))

## Large sparse example ---------
m <- Matrix(c(3,0:2), 2,2)
M <- bdiag(kronecker(Diagonal(2), m), kronecker(m,m))
36*(iM <- solve(M)) # still sparse
MM <- kronecker(Diagonal(10), kronecker(Diagonal(5),kronecker(m,M)))
dim(M3 <- kronecker(bdiag(M,M),MM)) # 12'800 ^ 2
if(interactive()) ## takes about 2 seconds if you have >= 8 GB RAM
  system.time(r <- rcond(M3))

## whereas this is *fast* even though it computes solve(M3)
  system.time(r. <- rcond(M3, useInv=TRUE))
if(interactive()) ## the values are not the same
  c(r, r.) # 0.05555 0.013888

## for all 4 norms available for sparseMatrix :
cbind(rr <- sapply(c("1","I","F","M"),
  function(N) rcond(M3, norm=N, useInv=TRUE)))
```

rep2abI

Description
rep2abI(x, times) conceptually computes rep.int(x, times) but with an abIndex class result.

Usage
rep2abI(x, times)

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

Value
a vector of class abIndex

See Also
rep.int(), the base function; abIseq, abIndex.

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

rleDiff-class

Class "rleDiff" of rle(diff(.)) Stored Vectors

Description
Class "rleDiff" is for compactly storing long vectors which mainly consist of linear stretches. For such a vector x, diff(x) consists of constant stretches and is hence well compressable via rle().

Objects from the Class
Objects can be created by calls of the form new("rleDiff", ...).
Currently experimental, see below.
**Slots**

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

**Methods**

There is a simple **show** method only.

**Note**

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

**See Also**

**rle**, **abIndex**.

**Examples**

```r
showClass("rleDiff")

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

---

**rsparsematrix**

**Random Sparse Matrix**

**Description**

Generate a random sparse matrix efficiently. The default has rounded gaussian non-zero entries, and **rand.x = NULL** generates random pattern matrices, i.e. inheriting from **nsparseMatrix**.

**Usage**

```r
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 \texttt{symmetricMatrix}. Note that in the symmetric case, \texttt{nnz} denotes the number of non zero entries of the upper (or lower) part of the matrix, including the diagonal.

\texttt{rand.x} \texttt{NULL} or the random number generator for the \texttt{x} slot, a \texttt{function} such that \texttt{rand.x(n)} generates a numeric vector of length \texttt{n}. Typical examples are \texttt{rand.x = rnorm}, or \texttt{rand.x = runif}; the default is nice for didactical purposes.

... optionally further arguments passed to \texttt{sparseMatrix()}, notably \texttt{repr}.

\textbf{Details}

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

\textbf{Value}

A \texttt{sparseMatrix}, say \texttt{M} of dimension \((nrow, ncol)\), i.e., with \texttt{dim(M) == c(nrow, ncol)}, if symmetric is not true, with \texttt{nzM <- nnzero(M)} fulfills \texttt{nzM <= nnz} and typically, \texttt{nzM == nnz}.

\textbf{Author(s)}

Martin Maechler

\textbf{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 patter*n* 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 Row-compressed Form

Description

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

Originally, few methods were defined on purpose, as we rather use the `CsparseMatrix` in `Matrix`. Then, more methods were added but beware that these typically do not return "RsparseMatrix" results, but rather Csparse* or Tsparse* ones; e.g., `R[i, j] <- v` for an "RsparseMatrix" R works, but after the assignment, R is a (triplet) "TsparseMatrix".

`t` signature(x = "RsparseMatrix"): ...

`coerce` signature(from = "RsparseMatrix", to = "CsparseMatrix"): ...

`coerce` signature(from = "RsparseMatrix", to = "TsparseMatrix"): ...

See Also

its superclass, `sparseMatrix`, and, e.g., class `dgRMatrix` for the links to other classes.

Examples

`showClass("RsparseMatrix")`
Schur-class  

Schur Factorizations

Description

Schur is the class of Schur factorizations of $n \times n$ real matrices $A$, having the general form

$$A = QTQ'$$

where $Q$ is an orthogonal matrix and $T$ is a block upper triangular matrix with $1 \times 1$ or $2 \times 2$ diagonal blocks specifying the real and complex conjugate eigenvalues of $A$. The column vectors of $Q$ are the Schur vectors of $A$, and $T$ is the Schur form of $A$.

The Schur factorization generalizes the spectral decomposition of normal matrices $A$, whose Schur form is block diagonal, to arbitrary square matrices.

Details

The matrix $A$ and its Schur form $T$ are similar and thus have the same spectrum. The eigenvalues are computed trivially as the eigenvalues of the diagonal blocks of $T$.

Slots

- Dim, Dimnames inherited from virtual class MatrixFactorization.
- Q an orthogonal matrix, inheriting from virtual class Matrix.
- T a block upper triangular matrix, inheriting from virtual class Matrix. The diagonal blocks have dimensions 1-by-1 or 2-by-2.
- EValues a numeric or complex vector containing the eigenvalues of the diagonal blocks of $T$, which are the eigenvalues of $T$ and consequently of the factorized matrix.

Extends

Class SchurFactorization, directly. Class MatrixFactorization, by class SchurFactorization, distance 2.

Instantiation

Objects can be generated directly by calls of the form `new("Schur", ...)`, but they are more typically obtained as the value of `Schur(x)` for $x$ inheriting from Matrix (often dgeMatrix).

Methods

determinant signature(from = "Schur", logarithm = "logical"): computes the determinant of the factorized matrix $A$ or its logarithm.
expand1 signature(x = "Schur"): see expand1-methods.
expand2 signature(x = "Schur"): see expand2-methods.
solve signature(a = "Schur", b = .): see solve-methods.
Schur-methods

References

The LAPACK source code, including documentation; see https://netlib.org/lapack/double/dgees.f.


See Also

Class `dgeMatrix`.

Generic functions `Schur`, `expand1` and `expand2`.

Examples

```r
showClass("Schur")
set.seed(0)

n <- 4L
(A <- Matrix(rnorm(n * n), n, n))

## With dimnames, to see that they are propagated :
dimnames(A) <- list(paste0("r", seq_len(n)),
                   paste0("c", seq_len(n)))

(sch.A <- Schur(A))
str(e.sch.A <- expand2(sch.A), max.level = 2L)

## A ~ Q T Q'
stopifnot(exprs = {
  identical(names(e.sch.A), c("Q", "T", "Q."))
  all.equal(A, with(e.sch.A, Q %*% T %*% Q.))
})

## Factorization handled as factorized matrix
b <- rnorm(n)
stopifnot(all.equal(det(A), det(sch.A)),
          all.equal(solve(A, b), solve(sch.A, b)))

## One of the non-general cases:
Schur(Diagonal(6L))
```

Methods for Schur Factorization
Description

Computes the Schur factorization of an $n \times n$ real matrix $A$, which has the general form

$$A = QTQ'$$

where $Q$ is an orthogonal matrix and $T$ is a block upper triangular matrix with $1 \times 1$ and $2 \times 2$ diagonal blocks specifying the real and complex conjugate eigenvalues of $A$. The column vectors of $Q$ are the Schur vectors of $A$, and $T$ is the Schur form of $A$.

Methods are built on LAPACK routine dgees.

Usage

Schur(x, vectors = TRUE, ...)

Arguments

x a finite square matrix or Matrix to be factorized.

vectors a logical. If TRUE (the default), then Schur vectors are computed in addition to the Schur form.

... further arguments passed to or from methods.

Value

An object representing the factorization, inheriting from virtual class SchurFactorization if vectors = TRUE. Currently, the specific class is always Schur in that case.

An exception is if x is a traditional matrix, in which case the result is a named list containing Q, T, and EValues slots of the Schur object.

If vectors = FALSE, then the result is the same named list but without Q.

References

The LAPACK source code, including documentation; see https://netlib.org/lapack<double/dgees.f.


See Also

Class Schur and its methods.

Class dgeMatrix.

Generic functions expand1 and expand2, for constructing matrix factors from the result.

Generic functions Cholesky, BunchKaufman, lu, and qr, for computing other factorizations.
Examples

showMethods("Schur", inherited = FALSE)
set.seed(0)

Schur(Hilbert(9L)) # real eigenvalues

(A <- Matrix(round(rnorm(25L, sd = 100)), 5L, 5L))
(sch.A <- Schur(A)) # complex eigenvalues

## A ~ Q T Q'
str(e.sch.A <- expand2(sch.A), max.level = 2L)
stopifnot(all.equal(A, Reduce("%*%", e.sch.A)))

(e1 <- eigen(sch.A@T, only.values = TRUE)$values)
(e2 <- eigen(A, only.values = TRUE)$values)
(e3 <- sch.A@EValues)

stopifnot(exprs = {
  all.equal(e1, e2, tolerance = 1e-13)
  all.equal(e1, e3[order(Mod(e3), decreasing = TRUE)], tolerance = 1e-13)
  identical(Schur(A, vectors = FALSE),
            list(T = sch.A@T, EValues = e3))
  identical(Schur(as(A, "matrix"),
                  list(Q = as(sch.A@Q, "matrix"),
                       T = as(sch.A@T, "matrix"), EValues = e3))
})

solve-methods

Methods in Package Matrix for Function solve

Description

Methods for generic function solve for solving linear systems of equations, i.e., for $X$ in $AX = B$, where $A$ is a square matrix and $X$ and $B$ are matrices with dimensions consistent with $A$.

Usage

solve(a, b, ...)

## S4 method for signature 'dgeMatrix,ANY'
solve(a, b, tol = .Machine$double.eps, ...)

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

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

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

## S4 method for signature 'denseLU,dgeMatrix'
solve(a, b, ...)
## S4 method for signature 'BunchKaufman,dgeMatrix'
solve(a, b, ...)
## S4 method for signature 'Cholesky,dgeMatrix'
solve(a, b, ...)
## S4 method for signature 'sparseLU,dgCMatrix'
solve(a, b, tol = .Machine$double.eps, ...)
## S4 method for signature 'sparseQR,dgCMatrix'
solve(a, b, ...)
## S4 method for signature 'CHMfactor,dgCMatrix'
solve(a, b, system = c("A", "LDLt", "LD", "DLt", "L", "Lt", "D", "P", "Pt"), ...)

### Arguments

**a**
- A finite square matrix or a Matrix containing the coefficients of the linear system, or otherwise a MatrixFactorization, in which case methods behave (by default) as if the factorized matrix were specified.

**b**
- A vector, sparseVector, matrix, or a Matrix satisfying NROW(b) == nrow(a), giving the right-hand side(s) of the linear system. Vectors b are treated as length(b)-by-1 matrices. If b is missing, then methods take b to be an identity matrix.

**tol**
- A non-negative number. For a inheriting from denseMatrix, an error is signaled if the reciprocal one-norm condition number (see rcond) of a is less than tol, indicating that a is near-singular. For a of class sparseLU, an error is signaled if the ratio min(d)/max(d) is less than tol, where d = abs(diag(a@U)). (Interpret with care, as this ratio is a cheap heuristic and not in general equal to or even proportional to the reciprocal one-norm condition number.) Setting tol = 0 disables the test.

**sparse**
- A logical indicating if the result should be formally sparse, i.e., if the result should inherit from virtual class sparseMatrix. Only methods for sparse a and missing or matrix b have this argument. Methods for missing or sparse b use sparse = TRUE by default. Methods for dense b use sparse = FALSE by default.

**system**
- A string specifying a linear system to be solved. Only methods for a inheriting from CHMfactor have this argument. See ‘Details’.

### Details

Methods for general and symmetric matrices a compute a triangular factorization (LU, Bunch-Kaufman, or Cholesky) and call the method for the corresponding factorization class. The factorization is sparse if a is. Methods for sparse, symmetric matrices a attempt a Cholesky factorization and perform an LU factorization only if that fails (typically because a is not positive definite).

Triangular, diagonal, and permutation matrices do not require factorization (they are already “factors”), hence methods for those are implemented directly. For triangular a, solutions are obtained
by forward or backward substitution; for diagonal \( a \), they are obtained by scaling the rows of \( b \); and for permutations \( a \), they are obtained by permuting the rows of \( b \).

Methods for dense \( a \) are built on 14 LAPACK routines: class \( d..Matrix \), where \(..=(ge|tr|tp|sy|sp|po|pp)\), uses routines \(d..tri\) and \(d..trs\) for missing and non-missing \( b \), respectively. A corollary is that these methods always give a dense result.

Methods for sparse \( a \) are built on CXSparse routines \(cs_lsolve\), \(cs_usolve\), and \(cs_spsolve\) and CHOLMOD routines \(cholmod_solve\) and \(cholmod_spsolve\). By default, these methods give a vector result if \( b \) is a vector, a sparse matrix result if \( b \) is missing or a sparse matrix, and a dense matrix result if \( b \) is a dense matrix. One can override this behaviour by setting the \( sparse\) argument, where available, but that should be done with care. Note that a sparse result may be sparse only in the formal sense and not at all in the mathematical sense, depending on the nonzero patterns of \( a \) and \( b \). Furthermore, whereas dense results are fully preallocated, sparse results must be “grown” in a loop over the columns of \( b \).

Methods for \( a \) of class \( sparseQR \) are simple wrappers around \(qr.coef\), giving the least squares solution in overdetermined cases.

Methods for \( a \) inheriting from \( CHMfactor \) can solve systems other than the default one \( AX = B \). The correspondence between its system argument the system actually solved is outlined in the table below. See \(CHMfactor-class\) for a definition of notation.

<table>
<thead>
<tr>
<th>system</th>
<th>isLDL((a))=TRUE</th>
<th>isLDL((a))=FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;</td>
<td>(AX = B)</td>
<td>(AX = B)</td>
</tr>
<tr>
<td>&quot;LDLt&quot;</td>
<td>(L_1 DL_1'X = B)</td>
<td>(LL'X = B)</td>
</tr>
<tr>
<td>&quot;LD&quot;</td>
<td>(L_1 DX = B)</td>
<td>(LX = B)</td>
</tr>
<tr>
<td>&quot;DLt&quot;</td>
<td>(DL_1'X = B)</td>
<td>(L'X = B)</td>
</tr>
<tr>
<td>&quot;L&quot;</td>
<td>(L_1 X = B)</td>
<td>(LX = B)</td>
</tr>
<tr>
<td>&quot;Lt&quot;</td>
<td>(L_1'X = B)</td>
<td>(L'X = B)</td>
</tr>
<tr>
<td>&quot;D&quot;</td>
<td>(DX = B)</td>
<td>(X = B)</td>
</tr>
<tr>
<td>&quot;P&quot;</td>
<td>(X = P_1 B)</td>
<td>(X = P_1B)</td>
</tr>
<tr>
<td>&quot;Pt&quot;</td>
<td>(X = P_1' B)</td>
<td>(X = P_1'B)</td>
</tr>
</tbody>
</table>

See Also

Virtual class \(MatrixFactorization\) and its subclasses.

Generic functions \( Cholesky\), \(BunchKaufman\), \(Schur\), \(lu\), and \(qr\) for computing factorizations.

Generic function \(solve\) from \(base\).

Function \(qr.coef\) from \(base\) for computing least squares solutions of overdetermined linear systems.

Examples

```r
## A close to symmetric example with "quite sparse" inverse:
n1 <- 7; n2 <- 3
dd <- data.frame(a = gl(n1,n2), b = gl(n2,1,n1*n2))# balanced 2-way
X <- sparse.model.matrix(~ -1+ a + b, dd)# no intercept --> even sparser
XXt <- tcrossprod(X)
diag(XXt) <- rep(c(0,0,1,0), length.out = nrow(XXt))
```

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

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

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

```
fac2Sparse2(from, to = c("d", "l", "n"),
            drop.unused.levels = TRUE, repr = c("C", "R", "T"), giveCsparse,
            factorPatt12, contrasts.arg = NULL)
```

Arguments

- `object` an object of an appropriate class. For the default method, a model formula or terms object.
sparse.model.matrix

- **data**: a data frame created with `model.frame`. If another sort of object, `model.frame` is called first.
- **contrasts.arg** for `sparse.model.matrix()`: A list, whose entries are contrasts suitable for input to the `contrasts` replacement function and whose names are the names of columns of data containing factors.
  - **for** `fac2Sparse()`: character string or NULL or (coercable to) "sparseMatrix", specifying the contrasts to be applied to the factor levels.
- **xlev** to be used as argument of `model.frame` if data has no "terms" attribute.
- **transpose** logical indicating if the `transpose` should be returned; if the transposed is used anyway, setting `transpose = TRUE` is more efficient.
- **drop.unused.levels** should factors have unused levels dropped? The default for `sparse.model.matrix` has been changed to `FALSE`, 2010-07, for compatibility with R’s standard (dense) `model.matrix()`.
- **row.names** logical indicating if row names should be used.
- **sep** character string passed to `paste()` when constructing column names from the variable name and its levels.
- **verbose** logical or integer indicating if (and how much) progress output should be printed.
- **...** further arguments passed to or from other methods.
- **from** (for `fac2sparse()`): a `factor`.
- **to** a character indicating the “kind” of sparse matrix to be returned. The default, "d" is for `double`.
- **giveCsparse** deprecated, replaced with `repr`; logical indicating if the result must be a `CsparseMatrix`.
- **repr** character string, one of "C", "T", or "R", specifying the sparse representation to be used for the result, i.e., one from the super classes `CsparseMatrix`, `TsparseMatrix`, or `RsparseMatrix`.
- **factorPatt12** logical vector, say `fp`, of length two; when `fp[1]` is true, return “contrasted” `t(X)`; when `fp[2]` is true, the original ("dummy") `t(X)`, i.e, the result of `fac2sparse()`.

**Value**

- a sparse matrix, extending `CsparseMatrix` (for `fac2sparse()` if `repr = "C"` as per default; a `TsparseMatrix` or `RsparseMatrix`, otherwise).
- For `fac2Sparse()`, a `list` of length two, both components with the corresponding transposed model matrix, where the corresponding `factorPatt12` is true.
- `fac2sparse()`, the basic workhorse of `sparse.model.matrix()`, returns the `transpose` (`t`) of the model matrix.

**Note**

`model.Matrix(sparse = TRUE)` from package **MatrixModels** may be nowadays be preferable to `sparse.model.matrix`, as `model.Matrix` returns an object of class `modelMatrix` with additional slots `assign` and `contrasts` relating to the model variables.
Author(s)

Doug Bates and Martin Maechler, with initial suggestions from Tim Hesterberg.

See Also

model.matrix in package stats, part of base R.

model.Matrix in package MatrixModels; see ‘Note’.

as(f, "sparseMatrix") (see coerce(from = "factor", ..) in the class doc sparseMatrix) produces the transposed sparse model matrix for a single factor f (and no contrasts).

Examples

dd <- data.frame(a = gl(3,4), b = gl(4,1,12))# balanced 2-way
options(\"contrasts\") \# the default: \"contr.treatment\"
sparse.model.matrix(~ a + b, dd)
sparse.model.matrix(~ -1+ a + b, dd)# no intercept --> even sparser
sparse.model.matrix(~ a + b, dd, contrasts = list(a="contr.sum"))
sparse.model.matrix(~ a + b, dd, contrasts = list(b="contr.SAS"))

## Sparse method is equivalent to the traditional one:
stopifnot(all(sparse.model.matrix(~ a + b, dd) ==
          Matrix(model.matrix(~ a + b, dd), sparse=TRUE)),
          all(sparse.model.matrix(~0 + a + b, dd) ==
          Matrix(model.matrix(~0 + a + b, dd), sparse=TRUE)))

(ff <- gl(3,4,, c("X","Y", "Z")))
fac2sparse(ff) \# 3 x 12 sparse Matrix of class \"dgCMatrix\"

## X 1 1 1 1 . . . . . . .
## Y . . . . 1 1 1 1 . . . .
## Z . . . . . . . . 1 1 1 1

## can also be computed via sparse.model.matrix():
f30 <- gl(3,0)
f12 <- gl(3,0, 12)

stopifnot(
          all.equal(t(fac2sparse(ff) ),
              sparse.model.matrix(~ 0+ff),
              tolerance = 0, check.attributes=FALSE),
          is(M <- fac2sparse(f30, drop= TRUE),"CsparseMatrix"), dim(M) == c(0, 0),
          is(M <- fac2sparse(f30, drop=FALSE),"CsparseMatrix"), dim(M) == c(3, 0),
          is(M <- fac2sparse(f12, drop= TRUE),"CsparseMatrix"), dim(M) == c(0,12),
          is(M <- fac2sparse(f12, drop=FALSE),"CsparseMatrix"), dim(M) == c(3,12) )

sparseLU-class

Sparse LU Factorizations

Description

sparseLU is the class of sparse, row- and column-pivoted LU factorizations of \( n \times n \) real matrices \( A \), having the general form

\[
P_1 A P_2 = LU
\]

or (equivalently)

\[
A = P'_1 L U P'_2
\]

where \( P_1 \) and \( P_2 \) are permutation matrices, \( L \) is a unit lower triangular matrix, and \( U \) is an upper triangular matrix.

Slots

\( \text{Dim, Dimnames} \) inherited from virtual class \text{MatrixFactorization}.

\( L \) an object of class \text{dtrMatrix}, the unit lower triangular \( L \) factor.

\( U \) an object of class \text{dtrMatrix}, the upper triangular \( U \) factor.

\( p, q \) 0-based integer vectors of length \( \text{Dim}[1] \), specifying the permutations applied to the rows and columns of the factorized matrix. \( q \) of length 0 is valid and equivalent to the identity permutation, implying no column pivoting. Using \( R \) syntax, the matrix \( P_1 A P_2 \) is precisely \( A[p+1, q+1] \) (\( A[p+1, \cdot] \) when \( q \) has length 0).

Extends

Class \text{LU}, directly. Class \text{MatrixFactorization}, by class \text{LU}, distance 2.

Instantiation

Objects can be generated directly by calls of the form \text{new("sparseLU", ...)}, but they are more typically obtained as the value of \text{lu(x)} for \( x \) inheriting from \text{sparseMatrix} (often \text{dgCMatrix}).

Methods

determinant signature(from = "sparseLU", logarithm = "logical"): computes the determinant of the factorized matrix \( A \) or its logarithm.

expand signature(x = "sparseLU"): see \text{expand-methods}.

expand1 signature(x = "sparseLU"): see \text{expand1-methods}.

expand2 signature(x = "sparseLU"): see \text{expand2-methods}.

solve signature(a = "sparseLU", b = .): see \text{solve-methods}.
References


See Also

Class `denseLU` for dense LU factorizations.

Class `dgCMatrix`.

Generic functions `lu`, `expand1` and `expand2`.

Examples

```r
showClass("sparseLU")
set.seed(2)
A <- as(readMM(system.file("external", "pores_1.mtx", package = "Matrix"),
  "CsparseMatrix")
(n <- A@Dim[1L])

## With dimnames, to see that they are propagated :
dimnames(A) <- dn <- list(paste0("r", seq_len(n)),
paste0("c", seq_len(n)))

(lu.A <- lu(A))
str(e.lu.A <- expand2(lu.A), max.level = 2L)

ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), ...)
ae2 <- function(a, b, ...) ae1(unname(a), unname(b), ...)

## A ~ P1' L U P2 in floating point
stopifnot(exprs = {
  identical(names(e.lu.A), c("P1.", "L", "U", "P2."))
  identical(e.lu.A[["P1."]],
    new("pMatrix", Dim = c(n, n), Dimnames = c(dn[1L], list(NULL)),
      margin = 1L, perm = invertPerm(lu.A@p, 0L, 1L)))
  identical(e.lu.A[["P2."]],
    new("pMatrix", Dim = c(n, n), Dimnames = c(list(NULL), dn[2L]),
      margin = 2L, perm = invertPerm(lu.A@q, 0L, 1L)))
  identical(e.lu.A[["L"]], lu.A@L)
  identical(e.lu.A[["U"]], lu.A@U)
  ae1(A, with(e.lu.A, P1. %*% L %*% U %*% P2.))
  ae2(A[lu.A@p + 1L, lu.A@q + 1L], with(e.lu.A, L %*% U))
})

## Factorization handled as factorized matrix
b <- rnorm(n)
stopifnot(identical(det(A), det(lu.A)),
  identical(solve(A, b), solve(lu.A, b)))
```
sparseMatrix

General Sparse Matrix Construction from Nonzero Entries

Description

User-friendly construction of sparse matrices (inherting from virtual class CsparseMatrix, R sparseMatrix, or TsparseMatrix) from the positions and values of their nonzero entries.

This interface is recommended over direct construction via calls such as new("[CRT]Matrix", ...).

Usage

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

Arguments

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

p  integer vector of pointers, one for each column (or row), to the initial (zero-based) index of elements in the column (or row). Exactly one of i, j, and p must be missing.

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

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

dimnames  optional list of dimnames; if missing, then NULL ones are used.

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

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

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

repr  character string, one of "C", "R", and "T", specifying the representation of the sparse matrix result, i.e., specifying one of the virtual classes CsparseMatrix, RsparseMatrix, and TsparseMatrix.
sparseMatrix

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

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

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

Details

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

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

If \( i \) or \( j \) is missing then \( p \) must be a non-decreasing integer vector whose first element is zero. It provides the compressed, or “pointer” representation of the row or column indices, whichever is missing. The expanded form of \( p \), \( \text{rep}(\text{seq_along}(dp), dp) \) 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 \( \text{repr} = "C" \), the CsparseMatrix derived from this triplet form is returned, where \( \text{repr} = "R" \) now allows to directly get an RsparseMatrix and \( \text{repr} = "T" \) 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 in compressed sparse column format and (formally) without symmetric or triangular structure, i.e., by default 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(*)`, 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, A.[1,2] == 2) # 2 was *after* 7

## for a pattern matrix, of course there is no "summing":
(nA <- do.call(sparseMatrix, args[c("i","j")]))

dn <- list(LETTERS[1:3], letters[1:5])
## pointer vectors can be used, and the (i,x) slots are sorted if necessary:
m <- sparseMatrix(i = c(3,1, 3:2, 2:1), p= c(0:2, 4,4,6), x = 1:6, dimnames = dn)
m
```
str(m)
stopifnot(identical(dimnames(m), dn))

sparseMatrix(x = 2.72, i=1:3, j=2:4) # recycling x
sparseMatrix(x = TRUE, i=1:3, j=2:4) # recycling x, |-- "lgCMatrix"

## no 'x' --> pattern* matrix:
(n <- sparseMatrix(i=1:6, j=rev(2:7))) # -> ngCMatrix

## an empty sparse matrix:
(e <- sparseMatrix(dims = c(4,6), i=[], j=[]) )

## a symmetric one:
(sy <- sparseMatrix(i= c(2,4,3:5), j= c(4,7:5,5), x = 1:5,
    dims = c(7,7), symmetric=TRUE))
stopifnot(isSymmetric(sy),
    identical(sy, ## switch i <-> j (and transpose )
    t( sparseMatrix(j= c(2,4,3:5), i= c(4,7:5,5), x = 1:5,
        dims = c(7,7), symmetric=TRUE))))

## rsparsematrix() calls sparseMatrix() :
M1 <- rsparsematrix(1000, 20, nnz = 200)
summary(M1)

## pointers example in converting from other sparse matrix representations.
if(requireNamespace("SparseM") &&
    packageVersion("SparseM") >= "0.87" &&
    nzchar(dfil <- system.file("extdata", "rua_32_ax.rua", package = "SparseM"))) {
    X <- SparseM::model.matrix(SparseM::read.matrix.hb(dfil))
    XX <- sparseMatrix(j = X@ja, p = X@ia - 1L, x = X@ra, dims = X@dimension)
    validObject(XX)

    ## Alternatively, and even more user friendly :
    X. <- as(X, "Matrix") # or also
    X2 <- as(X, "sparseMatrix")
    stopifnot(identical(XX, X.), identical(X., X2))
}

---

**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

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

print signature(x = "sparseMatrix"), ....

The print method for sparse matrices by default is the same as show() but can be called with extra optional arguments, see printSpMatrix().

format signature(x = "sparseMatrix"), ....

The format method for sparse matrices, see formatSpMatrix() for details such as the extra optional arguments.

summary (object = "sparseMatrix", uniqT=FALSE): Returns an object of S3 class "sparseSummary" which is basically a data.frame with columns (i,j,x) (or just (i,j) for nsparseMatrix class objects) with the stored (typically non-zero) entries. The print method resembles Matlab’s way of printing sparse matrices, and also the MatrixMarket format, see writeMM.

cbind2 (x = *, y = *): several methods for binding matrices together, column-wise, see the basic cbind and rbind functions.

Note that the result will typically be sparse, even when one argument is dense and larger than the sparse one.

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

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

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

dim<- signature(x = "sparseMatrix", value = "ANY"): allows to reshape a sparse matrix to a sparse matrix with the same entries but different dimensions. value must be of length two and fulfill prod(value) == prod(dim(x)).

coerce signature(from = "factor", to = "sparseMatrix"): Coercion of a factor to "sparseMatrix" produces the matrix of indicator rows stored as an object of class "dgCMatrix". To obtain columns representing the interaction of the factor and a numeric covariate, replace the "x" slot of the result by the numeric covariate then take the transpose. Missing values (NA) from the factor are translated to columns of all 0s.

See also colSums, norm, ... for methods with separate help pages.

Note

In method selection for multiplication operations (i.e. %*% and the two-argument form of crossprod) the sparseMatrix class takes precedence in the sense that if one operand is a sparse matrix and the other is any type of dense matrix then the dense matrix is coerced to a dgeMatrix and the appropriate sparse matrix method is used.

See Also

sparseMatrix, and its references, such as xtabs(*, sparse=TRUE), or sparse.model.matrix(), for constructing sparse matrices.

T2graph for conversion of "graph" objects (package graph) to and from sparse matrices.
Examples

showClass("sparseMatrix") ## and look at the help() of its subclasses
M <- Matrix(0, 10000, 100)
M[1,1] <- M[2,3] <- 3.14
M ## show(.) method suppresses printing of the majority of rows

data(CAex, package = "Matrix")
dim(CAex) # 72 x 72 matrix
determinant(CAex) # works via sparse lu(.)

## factor -> t( <sparse design matrix> ) :
(fact <- gl(5, 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 Factorizations

Description

sparseQR is the class of sparse, row- and column-pivoted QR factorizations of \( m \times n \) (\( m \geq n \)) real matrices, having the general form

\[
P_1 A P_2 = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1
\]

or (equivalently)

\[
A = P'_1 Q R P'_2 = P'_1 \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} P'_2 = P'_1 Q_1 R_1 P'_2
\]

where \( P_1 \) and \( P_2 \) are permutation matrices, \( Q = \prod_{j=1}^n H_j \) is an \( m \times m \) orthogonal matrix (\( Q_1 \) contains the first \( n \) column vectors) equal to the product of \( n \) Householder matrices \( H_j \), and \( R \) is an \( m \times n \) upper trapezoidal matrix (\( R_1 \) contains the first \( n \) row vectors and is upper triangular).

Usage

\( qrR(qr, complete = FALSE, backPermute = TRUE, row.names = TRUE) \)
sparseQR-class

Arguments

- **qr**: an object of class `sparseQR`, almost always the result of a call to generic function `qr` with sparse x.
- **complete**: a logical indicating if \( R \) should be returned instead of \( R_1 \).
- **backPermute**: a logical indicating if \( R \) or \( R_1 \) should be multiplied on the right by \( P_2' \).
- **row.names**: a logical indicating if `dimnames(qr)[1]` should be propagated unpermuted to the result. If `complete = FALSE`, then only the first \( n \) names are kept.

Details

The method for \( qr.Q \) does not return \( Q \) but rather the (also orthogonal) product \( P_1'Q \). This behaviour is algebraically consistent with the base implementation (see `qr`), which can be seen by noting that `qr.default` in base does not pivot rows, constraining \( P_1 \) to be an identity matrix. It follows that `qr.Q(qr.default(x))` also returns \( P_1'Q \).

Similarly, the methods for `qr.qy` and `qr.qty` multiply on the left by \( P_1'Q \) and \( Q'P_1 \) rather than \( Q \) and \( Q' \).

It is wrong to expect the values of `qr.Q` (or `qr.R`, `qr.qy`, `qr.qty`) computed from “equivalent” sparse and dense factorizations (say, `qr(x)` and `qr(as(x, "matrix"))`) for \( x \) of class `dgCMatrix` to compare equal. The underlying factorization algorithms are quite different, notably as they employ different pivoting strategies, and in general the factorization is not unique even for fixed \( P_1 \) and \( P_2 \).

On the other hand, the values of `qr.X`, `qr.coef`, `qr.fitted`, and `qr.resid` are well-defined, and in those cases the sparse and dense computations should compare equal (within some tolerance).

The method for `qr.R` is a simple wrapper around `qrR`, but not back-permuting by default and never giving row names. It did not support `backPermute = TRUE` until Matrix 1.6-0, hence code needing the back-permuted result should call `qrR` if `Matrix >= 1.6-0` is not known.

Slots

- **Dim, Dimnames**: inherited from virtual class `MatrixFactorization`.
- **beta**: a numeric vector of length `Dim[2]`, used to construct Householder matrices; see \( V \) below.
- **V**: an object of class `dgCMatrix` with `Dim[2]` columns. The number of rows `nrow(V)` is at least `Dim[1]` and at most `Dim[1]+Dim[2]`. \( V \) is lower trapezoidal, and its column vectors generate the Householder matrices \( H_j \) that compose the orthogonal \( Q \) factor. Specifically, \( H_j \) is constructed as `diag(Dim[1]) - beta[j] * tcrossprod(V[, j])`.
- **R**: an object of class `dgCMatrix` with `nrow(V)` rows and `Dim[2]` columns. \( R \) is the upper trapezoidal \( R \) factor.
- **p, q**: 0-based integer vectors of length `nrow(V)` and `Dim[2]`, respectively, specifying the permutations applied to the rows and columns of the factorized matrix. `q` of length 0 is valid and equivalent to the identity permutation, implying no column pivoting. Using \( R \) syntax, the matrix \( P_1AP_2 \) is precisely \( A[p+1, q+1] \) (\( A[p+1, ] \) when \( q \) has length 0).

Extends

Class `QR`, directly. Class `MatrixFactorization`, by class `QR`, distance 2.
**Instantiation**

Objects can be generated directly by calls of the form `new("sparseQR", ...)` but they are more typically obtained as the value of `qr(x)` for `x` inheriting from `sparseMatrix` (often `dgCMatrix`).

**Methods**

determinant signature(from = "sparseQR", logarithm = "logical"): computes the determinant of the factorized matrix $A$ or its logarithm.

expand1 signature(x = "sparseQR"): see `expand1-methods`.

expand2 signature(x = "sparseQR"): see `expand2-methods`.

qr.Q signature(qr = "sparseQR"): returns as a `dgeMatrix` either $P'_1Q$ or $P'_1Q_1$, depending on optional argument `complete`. The default is `FALSE`, indicating $P'_1Q_1$.

qr.R signature(qr = "sparseQR"): `qrR` returns $R$, $R_1$, $RP'_2$, or $R_1P'_2$, depending on optional arguments `complete` and `backPermute`. The default in both cases is `FALSE`, indicating $R_1$, for compatibility with `base`. The class of the result in that case is `dtCMatrix`. In the other three cases, it is `dgCMatrix`.

qr.X signature(qr = "sparseQR"): returns $A$ as a `dgeMatrix`, by default. If $m > n$ and optional argument `ncol` is greater than $n$, then the result is augmented with $P'_1QJ$, where $J$ is composed of columns $(n + 1)$ through `ncol` of the $m \times m$ identity matrix.

qr.coef signature(qr = "sparseQR", y = .): returns as a `dgeMatrix` or vector the result of multiplying $y$ on the left by $P_2R_1^{-1}Q'_1P_1$.

qr.fitted signature(qr = "sparseQR", y = .): returns as a `dgeMatrix` or vector the result of multiplying $y$ on the left by $P'_1Q_1Q'_1P_1$.

qr.resid signature(qr = "sparseQR", y = .): returns as a `dgeMatrix` or vector the result of multiplying $y$ on the left by $P'_1Q_1Q'_2P_1$.

qr.qty signature(qr = "sparseQR", y = .): returns as a `dgeMatrix` or vector the result of multiplying $y$ on the left by $Q'P_1$.

qr.qy signature(qr = "sparseQR", y = .): returns as a `dgeMatrix` or vector the result of multiplying $y$ on the left by $P'_1Q$.

solve signature(a = "sparseQR", b = .): see `solve-methods`.

**References**


**See Also**

Class `dgCMatrix`.

Generic function `qr` from `base`, whose default method `qr.default` “defines” the S3 class `qr` of dense QR factorizations.

`qr-methods` for methods defined in `Matrix`.
Generic functions `expand1` and `expand2`.


**Examples**

```
showClass("sparseQR")
set.seed(2)

m <- 300L
n <- 60L
A <- rsparsematrix(m, n, 0.05)

## With dimnames, to see that they are propagated:
dimnames(A) <- dn <- list(paste0("r", seq_len(m)),
paste0("c", seq_len(n)))

(qr.A <- qr(A))
str(e.qr.A <- expand2(qr.A, complete = FALSE), max.level = 2L)
str(E.qr.A <- expand2(qr.A, complete = TRUE), max.level = 2L)

t(sapply(e.qr.A, dim))
t(sapply(E.qr.A, dim))
```

```
## Horribly inefficient, but instructive:
slowQ <- function(V, beta) {
  d <- dim(V)
  Q <- diag(d[1L])
  if(d[2L] > 0L) {
    for(j in d[2L]:1L) {
      cat(j, 
      Q <- Q - (beta[j] * tcrossprod(V[, j])) %*% Q
    }
  }
  Q
}

ae1 <- function(a, b, ...) all.equal(as(a, "matrix"), as(b, "matrix"), ...)
ae2 <- function(a, b, ...) ae1(unname(a), unname(b), ...)

## A ~ P1' Q R P2' ~ P1' Q1 R1 P2' in floating point
stopifnot(exprs = {
  identical(names(e.qr.A), c("P1.", "Q1", "R1", "P2."))
  identical(names(E.qr.A), c("P1.", "Q", "R", "P2."))
  identical(e.qr.A[["P1."]],
            new("pMatrix", Dim = c(m, m), Dimnames = c(dn[1L], list(NULL)),
                 margin = 1L, perm = invertPerm(qr.A@p, 0L, 1L)))
  identical(e.qr.A[["P2."]],
            new("pMatrix", Dim = c(n, n), Dimnames = c(list(NULL), dn[2L]),
                 margin = 2L, perm = invertPerm(qr.A@q, 0L, 1L)))
  identical(e.qr.A[["R1"]], triu(E.qr.A[["R"]][seq_len(n), ]))
})
```
identical(e.qr.A[["Q1"]], E.qr.A[["Q"]][, seq_len(n)])
identical(E.qr.A[["R"]], qr.A@R)

```r
## ae1(E.qr.A[["Q"]], slowQ(qr.A@V, qr.A@beta))
ae1(crossprod(E.qr.A[["Q"]]), diag(m))
ae1(A, with(e.qr.A, P1. %*% Q1 %*% R1 %*% P2.))
ae1(A, with(E.qr.A, P1. %*% Q %*% R %*% P2.))
ae2(A.perm <- A[qr.A@p + 1L, qr.A@q + 1L], with(e.qr.A, Q1 %*% R1))
ae2(A.perm, with(E.qr.A, Q %*% R))
```

```r
## More identities
b <- rnorm(m)
stopifnot(exprs = {
  ae1(qrX <- qr.X (qr.A), A)
ae2(qrQ <- qr.Q (qr.A), with(e.qr.A, P1. %*% Q1))
ae2(qr.R (qr.A), with(e.qr.A, R1))
ae2(qrc <- qr.coef (qr.A, b), with(e.qr.A, solve(R1 %*% P2., t(qrQ)) %*% b))
ae2(qrf <- qr.fitted(qr.A, b), with(e.qr.A, tcrossprod(qrQ) %*% b))
ae2(qrr <- qr.resid (qr.A, b - qrf)
ae2(qr.qy <- qr.qy(qr.A, b), with(E.qr.A, P1. %*% Q %*% b))
ae2(qrq <- qr.qty(qr.A, b))
})
```

```r
## Sparse and dense computations should agree here
qr.Am <- qr(as(A, "matrix")) # <=> qr.default(A)
stopifnot(exprs = {
  ae2(qrX, qr.X (qr.Am))
ae2(qrc, qr.coef (qr.Am, b))
ae2(qrf, qr.fitted(qr.Am, b))
ae2(qrr, qr.resid (qr.Am, b))
})
```

---

**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.
sparseVector-class

Details
	one 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,-3:1,0,0,0,5,0,0)
ss <- as(sx, "sparseVector")
stopifnot(identical(ss,
  sparseVector(x = c(2, -1, -2, 3, 1, -3, 5, 3.2),
               i = c(15L, 10:9, 3L,12L,8L,18L, 4L), length = 20L)))

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

---

sparseVector-class  
Sparse Vector Classes

Description

Sparse Vector Classes: The virtual mother class "sparseVector" has the five actual daughter classes "dsparseVector", "isparseVector", "lsparseVector", "nsparseVector", and "zsparseVector", where we've mainly implemented methods for the d*, l* and n* ones.

Slots

length: class "numeric" - the length of the sparse vector. Note that "numeric" can be considerably larger than the maximal "integer", .Machine$integer.max, on purpose.

i: class "numeric" - the (1-based) indices of the non-zero entries. Must not be NA and strictly sorted increasingly.
  Note that "integer" is part of "numeric", and can (and often will) be used for non-huge sparseVectors.

x: (for all but "nsparseVector"): the non-zero entries. This is of class "numeric" for class "dsparseVector", "logical" for class "lsparseVector", etc.
Methods

length signature(x = "sparseVector"): simply extracts the length slot.

show signature(object = "sparseVector"): The show method for sparse vectors prints “structural” zeroes as "." using the non-exported prSpVector function which allows further customization such as replacing "." by " " (blank).

Note that options(max.print) will influence how many entries of large sparse vectors are printed at all.

as.vector signature(x = "sparseVector", mode = "character") coerces sparse vectors to “regular”, i.e., atomic vectors. This is the same as as(x, "vector").

as ..: see coerce below

coerce signature(from = "sparseVector", to = "sparseMatrix"). and

coerce signature(from = "sparseMatrix", to = "sparseVector"), etc: coercions to and from sparse matrices (sparseMatrix) are provided and work analogously as in standard R, i.e., a vector is coerced to a 1-column matrix.

dim<-. signature(x = "sparseVector", value = "integer") coerces a sparse vector to a sparse Matrix, i.e., an object inheriting from sparseMatrix, of the appropriate dimension.

head signature(x = "sparseVector"): as with R's (package util) head, head(x, n) (for n >= 1) is equivalent to x[1:n], but here can be much more efficient, see the example.

tail signature(x = "sparseVector"): analogous to head, see above.

toeplitz signature(x = "sparseVector"): as toeplitz(x), produce the n x n Toeplitz matrix from x, where n = length(x).

rep signature(x = "sparseVector") repeat x, with the same argument list (x, times, length.out, each,...) as the default method for rep().

which signature(x = "nsparseVector") and

which signature(x = "lsparseVector") return the indices of the non-zero entries (which is trivial for sparse vectors).

Ops signature(e1 = "sparseVector", e2 = "."): define arithmetic, compare and logic operations, (see Ops).

Summary signature(x = "sparseVector"): define all the Summary methods.

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.

zapsmall signature(x = "sparseVectors"): typically used for numeric sparse vector: round() entries such that (relatively) very small entries become zero exactly.

c.sparseVector() is an S3 method for all "sparseVector"s, but automatic dispatch only happens for the first argument, so it is useful also as regular R function, see the examples.

See Also

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

getClass("sparseVector")
getClass("dsparseVector")

sx <- c(0,0,3, 3.2, 0,0,0,-3:1,0,2,0,0,5,0,0)
(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:
## 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), last(0@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(exprs = {
  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,
            check.class = FALSE)
  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(0L), j = integer(0L), x = double(0L))
```

**Arguments**

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

A sparse matrix in triplet form, as an \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 :

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

M1 <- rSpMatrix(100000, 20, \text{nnz} = 200)
summary(M1)
```
Methods for "\[\textless-\]" - Assigning to Subsets for 'Matrix'

**Description**

Methods for "\[\textless-\]", i.e., extraction or subsetting mostly of matrices, in package **Matrix**.

**Note**: Contrary to standard matrix assignment in base R, in \texttt{x[...]} \textless\textasciitilde \texttt{val} it is typically an error (see **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 "\texttt{lsparseMatrix}" and \texttt{val} is numeric. In other cases, e.g., when \texttt{x} is a "\texttt{nsparseMatrix}" and \texttt{val} is not \texttt{TRUE} or \texttt{FALSE}, a warning is signalled, and \texttt{val} is “interpreted” as \texttt{logical}; and (logical) \texttt{NA} is interpreted as \texttt{TRUE}.

**Methods**

There are many many more than these:

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

\texttt{x = "Matrix", i = "ANY", j = "ANY", value = "ANY"} currently gives an error

\texttt{x = "denseMatrix", i = "index", j = "missing", value = "numeric"} ...

\texttt{x = "denseMatrix", i = "index", j = "index", value = "numeric"} ...

**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)
```
m

## rows or columns only:
m[1,] <- 10
m[,2] <- 1:7
m[-(1:6), ] <- 3:0 # not the first 6 rows, i.e. only the 7th
as(m, "sparseMatrix")

---

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

\texttt{[<--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)
```
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

Dim, Dimnames inherited from virtual class Matrix. See comments below about symmetry of Dimnames.

factors a list of MatrixFactorization objects caching factorizations of the matrix. Typically, it is initialized as an empty list and updated “automagically” whenever a factorization is computed.

uplo a character string, either "U" or "L" indicating that only entries in the upper or lower triangle are referenced.

Extends

Class "Matrix", directly.

Methods

dimnames signature(object = "symmetricMatrix"): returns symmetric dimnames, even when the Dimnames slot only has row or column names. This allows to save storage for large (typically sparse) symmetric matrices.

isSymmetric signature(object = "symmetricMatrix"): returns TRUE trivially.

There’s a C function symmetricMatrix_validate() called by the internal validity checking functions, and also from getValidity(getClass("symmetricMatrix")).

Validity and dimnames

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

See Also

isSymmetric which has efficient methods (isSymmetric-methods) for the Matrix classes. Classes triangularMatrix, and, e.g., dsyMatrix for numeric dense matrices, or IsCMatrix for a logical sparse matrix class.
Examples

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

```r
symmpart(x)
skewpart(x)
```

Arguments

- `x` a `square` matrix; either “traditional” of class "matrix", or typically, inheriting from the `Matrix` class.

Details

These are generic functions with several methods for different matrix classes, use e.g., `showMethods(symmpart)` to see them.

If the row and column names differ, the result will use the column names unless they are (partly) `NULL` where the row names are non-`NULL` (see also the examples).
triangularMatrix-class

Value

symmpart(x) returns a symmetric matrix, inheriting from symmetricMatrix or diagonalMatrix if x inherits from Matrix.

skewpart(x) returns a skew-symmetric matrix, inheriting from generalMatrix, symmetricMatrix or diagonalMatrix if x inherits from Matrix.

See Also

isSymmetric.

Examples

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, such that the length of the x slot does not depend on diag.

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

Class "Matrix", directly.

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.
unpackedMatrix-class

Extends

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

Methods

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

Note

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

Note that all new(.), spMatrix and sparseMatrix(*, 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 CsparseMatrix 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 !

unpackedMatrix-class

Virtual Class "unpackedMatrix" of Unpacked Dense Matrices

Description

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

Slots

Dim, Dimnames: as all Matrix objects.
Extends


Methods

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

Author(s)

Mikael Jagan

See Also

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

Examples

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

Description

Computes a rank-\(k\) update or downdate of a sparse Cholesky factorization

\[ P_1 A P_1' = L_1 D L_1' = LL' \]

which for some \(k\)-column matrix \(C\) is the factorization

\[ P_1 (A + sCC') P_1' = \tilde{L}_1 \tilde{D} \tilde{L}_1' = \tilde{L} \tilde{L}' \]

Here, \(s = 1\) for an update and \(s = -1\) for a downdate.

Usage

updown(update, C, L)
USCounties

Arguments

update a logical (TRUE or FALSE) or character ("+" or "-") indicating if L should be updated (or otherwise downdated).

C a finite matrix or Matrix such that tcrossprod(C) has the dimensions of L.

L an object of class dCHMsimpl or dCHMsuper specifying a sparse Cholesky factorization.

Value

A sparse Cholesky factorization with dimensions matching L, typically of class dCHMsimpl.

Author(s)

Initial implementation by Nicholas Nagle, University of Tennessee.

References


See Also

Classes dCHMsimpl and dCHMsuper and their methods, notably for generic function update, which is not equivalent to updown(update = TRUE).

Generic function Cholesky.

Examples

```r
m <- sparseMatrix(i = c(3, 1, 3:2, 2:1), p = c(0:2, 4, 4, 6), x = 1:6,
    dimnames = list(LETTERS[1:3], letters[1:5]))
uc0 <- Cholesky(A <- crossprod(m) + Diagonal(5))
uc1 <- updown("+", Diagonal(5, 1), uc0)
uc2 <- updown("-", Diagonal(5, 1), uc1)
stopifnot(all.equal(uc0, uc2))
```

Description

This matrix gives the contiguities of 3111 U.S. counties, using the queen criterion of at least one shared vertex or edge.

Usage

data(USCounties)
**Format**

A 3111 × 3111 sparse, symmetric matrix of class `dsCMatrix`, with 9101 nonzero entries.

**Source**

GAL lattice file `usc_q.GAL` (retrieved in 2008 from `http://sal.uiuc.edu/weights/zips/usc.zip` with permission from Luc Anselin for use and distribution) was read into R using function `read.gal` from package `spdep`.

Neighbour lists were augmented with row-standardized (and then symmetrized) spatial weights, using functions `nb2listw` and `similar.listw` from packages `spdep` and `spatialreg`. The resulting listw object was coerced to class `dsTMatrix` using `as_dsTMatrix_listw` from `spatialreg`, and subsequently to class `dsCMatrix`.

**References**


**Examples**

data(USCounties, package = "Matrix")
(n <- ncol(USCounties))
I <- .symDiagonal(n)

set.seed(1)
r <- 50L
rho <- 1 / runif(r, 0, 0.5)

system.time(MJ0 <- sapply(rho, function(mult)
  determinant(USCounties + mult * I, logarithm = TRUE)$modulus))

## Can be done faster by updating the Cholesky factor:

C1 <- Cholesky(USCounties, Imult = 2)
system.time(MJ1 <- sapply(rho, function(mult)
  determinant(update(C1, USCounties, mult), sqrt = FALSE)$modulus))
stopifnot(all.equal(MJ0, MJ1))

C2 <- Cholesky(USCounties, super = TRUE, Imult = 2)
system.time(MJ2 <- sapply(rho, function(mult)
  determinant(update(C2, USCounties, mult), sqrt = FALSE)$modulus))
stopifnot(all.equal(MJ0, MJ2))
Contiguity Matrix of World One-Degree Grid Cells

Description
This matrix gives the continuiities of 15260 one-degree grid cells of world land areas, using a criterion based on the great-circle distance between centers.

Usage
data(wrld_1deg)

Format
A $15260 \times 15260$ sparse, symmetric matrix of class `dsCMatrix`, with 55973 nonzero entries.

Source
Shoreline data were read into R from the GSHHS database using function `Rgshhs` from package `maptools`. Antarctica was excluded. An approximately one-degree grid was generated using function `Sobj_SpatialGrid`, also from `maptools`. Grid cells with centers on land were identified using the over method for classes `SpatialPolygons` and `SpatialGrid`, defined in package `sp`. Neighbours of these were identified by passing the resulting `SpatialPixels` object to function `dnearneigh` from package `spdep`, using as a cut-off a great-circle distance of $\sqrt{2}$ kilometers between centers.

Neighbour lists were augmented with row-standardized (and then symmetrized) spatial weights, using functions `nb2listw` and `similar.listw` from packages `spdep` and `spatialreg`. The resulting `listw` object was coerced to class `dsTMatrix` using `as_dsTMatrix_listw` from `spatialreg`, and subsequently to class `dsCMatrix`.

References

Examples
data(wrld_1deg, package = "Matrix")
(n <- ncol(wrld_1deg))
I <- .symDiagonal(n)

doExtras <- interactive() || nzchar(Sys.getenv("R_MATRIX_CHECK_EXTRA"))
set.seed(1)
r <- if(doExtras) 20L else 3L
rho <- 1 / runif(r, 0, 0.5)

system.time(MJ0 <- sapply(rho, function(mult)
determinant(wrld_1deg + mult * I, logarithm = TRUE)$modulus))

## Can be done faster by updating the Cholesky factor:

C1 <- Cholesky(wrld_1deg, Imult = 2)
system.time(MJ1 <- sapply(rho, function(mult)
  determinant(update(C1, wrld_1deg, mult), sqrt = FALSE)$modulus))
stopifnot(all.equal(MJ0, MJ1))

C2 <- Cholesky(wrld_1deg, super = TRUE, Imult = 2)
system.time(MJ2 <- sapply(rho, function(mult)
  determinant(update(C2, wrld_1deg, mult), sqrt = FALSE)$modulus))
stopifnot(all.equal(MJ0, MJ2))
ddiMatrix-class, 48
denseLU-class, 49
denseMatrix-class, 51
dgCMatrix-class, 52
dgeMatrix-class, 53
dgRMatrix-class, 54
dgTMatrix-class, 55
diagonalMatrix-class, 58
dMatrix-class, 62
dpoMatrix-class, 65
dsdMatrix-class, 68
dsparsesMatrix-class, 70
dsymMatrix-class, 72
dtMatrix-class, 73
dtpMatrix-class, 75
dtrMatrix-class, 77
dtrMatrix-class, 78
generalMatrix-class, 92
index-class, 96
indMatrix-class, 97
ldenseMatrix-class, 110
ldiMatrix-class, 111
lgeMatrix-class, 112
lsparseMatrix-class, 113
lsyMatrix-class, 115
ltriMatrix-class, 116
Matrix-class, 124
Matrix-notyet, 126
MatrixFactorization-class, 128
ndenseMatrix-class, 129
ngeMatrix-class, 133
nMatrix-class, 134
nsparseMatrix-class, 138
nMatrix-class, 140
ntriMatrix-class, 141
packedMatrix-class, 143
pMatrix-class, 145
rleDiff-class, 158
RsparseMatrix-class, 161
Schur-class, 162
sparseLU-class, 171
sparseMatrix-class, 176
sparseQR-class, 178
sparseVector-class, 183
symmetricMatrix-class, 190
triangularMatrix-class, 192
TsparseMatrix-class, 193
unpackedMatrix-class, 194
* connection
  externalFormats, 83
* datasets
  CAex, 20
  KNex, 108
  USCounties, 196
  wrld_1deg, 198
* distribution
  rsparsematrix, 159
* file
  externalFormats, 83
* hplot
  image-methods, 94
* logic
  all.equal-methods, 7
  asUniqueT, 8
  boolmatmult-methods, 15
  nnzero-methods, 135
* manip
  abIseq, 6
  asUniqueT, 8
  cbind2-methods, 21
  drop0, 67
  rep2abI, 158
  sparseVector-class, 183
* math
  condest, 43
  expm-methods, 82
  is.na-methods, 101
  norm-methods, 137
  rcond-methods, 155
* methods
  all.equal-methods, 7
  band-methods, 9
  boolmatmult-methods, 15
  BunchKaufman-methods, 19
  cbind2-methods, 21
  chol-methods, 26
  chol2inv-methods, 30
  Cholesky-methods, 34
  coerce-methods-graph, 40
  coerce-methods-SparseM, 41
  colSums-methods, 42
  expand-methods, 79
  expm-methods, 82
  externalFormats, 83
  facmul-methods, 85
forceSymmetric-methods, 90
image-methods, 94
is.na-methods, 101
issymmetric-methods, 103
isTriangular-methods, 105
kronecker-methods, 109
lu-methods, 117
nnzero-methods, 135
norm-methods, 137
pack-methods, 142
qr-methods, 149
rcond-methods, 155
schur-methods, 163
solve-methods, 165
subassign-methods, 188
subscript-methods, 189
symmpart-methods, 191
updown-methods, 195

* models
  sparse.model.matrix, 168

* print
  formatSparseM, 91
  printSpMatrix, 147

* programming
  all.equal-methods, 7
  CHMfactor-class, 22
  is.na-methods, 101
  is.null.DN, 102
  isSymmetric-methods, 103
  isTriangular-methods, 105
* utilities
  abIseq, 6
  asUniqueT, 8
  bandSparse, 11
  bdiag, 13
  CHMfactor-class, 22
  coerce-methods-graph, 40
  condest, 43
  diagonal, 56
  diag2U2N, 59
  dimScale, 61
  dmperm, 63
  drop0, 67
  externalFormats, 83
  fastMisc, 86
  formatSparseM, 91
  hilbert, 93
  invertPerm, 99
  is.null.DN, 102
  khatriRao, 106
  mat2triplet, 119
  matrix, 122
  MatrixClass, 127
  nearPD, 130
  printSpMatrix, 147
  rankMatrix, 152
  rep2abI, 158
  sparse.model.matrix, 168
  sparseMatrix, 173
  sparseQR-class, 178
  sparseVector, 182
  spMatrix, 186
  *,Matrix,ddiMatrix-method
    (Matrix-class), 124
  *,Matrix,ldiMatrix-method
    (Matrix-class), 124
  *,Matrix,ndiMatrix-method
    (Matrix-class), 124
  *,ddenseMatrix,ddiMatrix-method
    (ddenseMatrix-class), 47
  *,ddenseMatrix,ldiMatrix-method
    (ddenseMatrix-class), 47
  *,ddenseMatrix,ndiMatrix-method
    (ddenseMatrix-class), 47
  *,ldenseMatrix,ldiMatrix-method
    (ldenseMatrix-class), 110
  *,ldenseMatrix,ldiMatrix-method
    (ldenseMatrix-class), 110
  *,ldenseMatrix,ndiMatrix-method
    (ldenseMatrix-class), 110
  *,ldiMatrix,Matrix-method
    (ldiMatrix-class), 111
  *,ldiMatrix,ddenseMatrix-method
    (ldiMatrix-class), 111
  *,ldiMatrix,ldenseMatrix-method
    (ldiMatrix-class), 111
  *,ldiMatrix,ndenseMatrix-method
    (ldiMatrix-class), 111
(ldiMatrix-class), 111
*, ndenseMatrix, ddiMatrix-method
   (ndenseMatrix-class), 129
*, ndenseMatrix, ldiMatrix-method
   (ndenseMatrix-class), 129
*, ndenseMatrix, ndiMatrix-method
   (ndenseMatrix-class), 129
*, ndiMatrix, Matrix-method
   (ldiMatrix-class), 111
*, ndiMatrix, ddenseMatrix-method
   (ldiMatrix-class), 111
*, ndiMatrix, ldenseMatrix-method
   (ldiMatrix-class), 111
*, ndiMatrix, ndenseMatrix-method
   (ldiMatrix-class), 111
+, Matrix, missing-method (Matrix-class), 124
+, dgTMatrix, dgTMatrix-method
   (dgTMatrix-class), 55
-, Matrix, missing-method (Matrix-class), 124
-, denseMatrix, missing-method
   (denseMatrix-class), 51
-, diagonalMatrix, missing-method
   (diagonalMatrix-class), 58
-, dsparseVector, missing-method
   (dsparseVector-class), 183
-, indMatrix, missing-method
   (indMatrix-class), 97
-, nsparseMatrix, missing-method
   (nsparseMatrix-class), 138
-, sparseMatrix, missing-method
   (sparseMatrix-class), 176
.CR2RC (fastMisc), 86
.CR2T (fastMisc), 86
.M2C (fastMisc), 86
.M2R (fastMisc), 86
.M2T (fastMisc), 86
.M2V (fastMisc), 86
.M2diag (fastMisc), 86
.M2gen (fastMisc), 86
.M2kind (fastMisc), 86
.M2m (fastMisc), 86
.M2packed (fastMisc), 86
.M2sym (fastMisc), 86
.M2tri (fastMisc), 86
.M2unpack (fastMisc), 86
.M2v (fastMisc), 86
Machine, 153, 183
.T2CR (fastMisc), 86
.bdiag (bdiag), 13
dense2g (fastMisc), 86
dense2kind (fastMisc), 86
dense2m (fastMisc), 86
dense2sparse (fastMisc), 86
dense2v (fastMisc), 86
diag.dsC (fastMisc), 86
diag2dense (fastMisc), 86
diag2sparse (fastMisc), 86
diagN2U (diagU2N), 59
diagU2N (diagU2N), 59
.formatSparseSimple, 148
.formatSparseSimple (formatSparseM), 91
.ind2dense (fastMisc), 86
.ind2sparse (fastMisc), 86
.m2V (fastMisc), 86
.m2dense (fastMisc), 86
.m2sparse (fastMisc), 86
.selectSuperClasses, 127
.solve dgC.chol (fastMisc), 86
.solve dgC.lu (fastMisc), 86
.solve dgC.qr (fastMisc), 86
.sparse2dense (fastMisc), 86
.sparse2g (fastMisc), 86
.sparse2kind (fastMisc), 86
.sparse2m (fastMisc), 86
.sparse2v (fastMisc), 86
.sparseDiagonal (Diagonal), 56
.symDiagonal (Diagonal), 56
tCR2RC (fastMisc), 86
tCRT (fastMisc), 86
tDiagonal (Diagonal), 56
.updateCHMfactor (fastMisc), 86
.validateCsparse (CsparseMatrix-class), 45
/, ddiMatrix, Matrix-method
   (ddiMatrix-class), 48
/, ddiMatrix, ddenseMatrix-method
   (ddiMatrix-class), 48
/, ddiMatrix, ldenseMatrix-method
   (ddiMatrix-class), 48
/, ddiMatrix, ndenseMatrix-method
   (ddiMatrix-class), 48
/, ldiMatrix, Matrix-method
   (ldiMatrix-class), 111
/, ldiMatrix, ddenseMatrix-method
(subassign-methods), 188

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]

[, , replValue-method[, , replValue-method]
(matmult-methods), 120 %*,CsparseMatrix,diagonalMatrix-method
(matmult-methods), 120 %*,CsparseMatrix,matrix-method
(matmult-methods), 120 %*,CsparseMatrix,vector-method
(matmult-methods), 120 %*,Matrix,ANY-method
(matmult-methods), 120 %*,Matrix,indMatrix-method
(matmult-methods), 120 %*,Matrix,pMatrix-method
(matmult-methods), 120 %*,Matrix,sparseVector-method
(matmult-methods), 120 %*,RsparseMatrix,CsparseMatrix-method
(matmult-methods), 120 %*,RsparseMatrix,RsparseMatrix-method
(matmult-methods), 120 %*,RsparseMatrix,TsparseMatrix-method
(matmult-methods), 120 %*,RsparseMatrix,denseMatrix-method
(matmult-methods), 120 %*,RsparseMatrix,diagonalMatrix-method
(matmult-methods), 120 %*,RsparseMatrix,matrix-method
(matmult-methods), 120 %*,RsparseMatrix,vector-method
(matmult-methods), 120 %*,TsparseMatrix,CsparseMatrix-method
(matmult-methods), 120 %*,TsparseMatrix,RsparseMatrix-method
(matmult-methods), 120 %*,TsparseMatrix,TsparseMatrix-method
(matmult-methods), 120 %*,TsparseMatrix,denseMatrix-method
(matmult-methods), 120 %*,TsparseMatrix,diagonalMatrix-method
(matmult-methods), 120 %*,TsparseMatrix,matrix-method
(matmult-methods), 120 %*,TsparseMatrix,vector-method
(matmult-methods), 120 %*,denseMatrix,CsparseMatrix-method
(matmult-methods), 120 %*,denseMatrix,denseMatrix-method
(matmult-methods), 120 %*,denseMatrix,diagonalMatrix-method
(matmult-methods), 120 %*,denseMatrix,matrix-method
(matmult-methods), 120 %*,denseMatrix,vector-method
(matmult-methods), 120 %*,diagonalMatrix,CsparseMatrix-method
(matmult-methods), 120 %*,diagonalMatrix,RsparseMatrix-method
(matmult-methods), 120 %*,diagonalMatrix,TsparseMatrix-method
(matmult-methods), 120 %*,diagonalMatrix,denseMatrix-method
(matmult-methods), 120 %*,diagonalMatrix,diagonalMatrix-method
(matmult-methods), 120 %*,diagonalMatrix,matrix-method
(matmult-methods), 120 %*,diagonalMatrix,vector-method
(matmult-methods), 120 %*,indMatrix,Matrix-method
(matmult-methods), 120 %*,indMatrix,indMatrix-method
(matmult-methods), 120 %*,indMatrix,matrix-method
(matmult-methods), 120 %*,indMatrix,pMatrix-method
(matmult-methods), 120 %*,matrix,CsparseMatrix-method
(matmult-methods), 120 %*,matrix,denseMatrix-method
(matmult-methods), 120 %*,matrix,diagonalMatrix-method
(matmult-methods), 120 %*,matrix,indMatrix-method
(matmult-methods), 120 %*,matrix,pMatrix-method
(matmult-methods), 120 %*,matrix,sparseVector-method
(matmult-methods), 120
(boolmatmult-methods), 15
%&%, CsparseMatrix, CsparseMatrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, RsparseMatrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, TsparseMatrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, denseMatrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, diagonalMatrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, matrix-method
(boolmatmult-methods), 15
%&%, CsparseMatrix, vector-method
(boolmatmult-methods), 15
%&%, Matrix, ANY-method
(boolmatmult-methods), 15
%&%, Matrix, indMatrix-method
(boolmatmult-methods), 15
%&%, Matrix, pMatrix-method
(boolmatmult-methods), 15
%&%, Matrix, sparseVector-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, CsparseMatrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, RsparseMatrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, TsparseMatrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, denseMatrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, diagonalMatrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, matrix-method
(boolmatmult-methods), 15
%&%, RsparseMatrix, vector-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, CsparseMatrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, RsparseMatrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, TsparseMatrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, denseMatrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, diagonalMatrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, matrix-method
(boolmatmult-methods), 15
%&%, TsparseMatrix, vector-method
(boolmatmult-methods), 15
%&%, matrix, ANY-method
(boolmatmult-methods), 15
%&%, matrix, CsparseMatrix-method
(boolmatmult-methods), 15
%&%, matrix, RsparseMatrix-method
(boolmatmult-methods), 15
%&%, matrix, TsparseMatrix-method
&ndenseMatrix,ndiMatrix-method (ndenseMatrix-class), 129
&ndiMatrix,Matrix-method (ldiMatrix-class), 111
&ndiMatrix,ddenseMatrix-method (ldiMatrix-class), 111
&ndiMatrix,ldenseMatrix-method (ldiMatrix-class), 111
&ndiMatrix,ndenseMatrix-method (ldiMatrix-class), 111
\%,%, 16, 46, 53, 122
\%,%, 120–122
\^,Matrix,ddiMatrix-method (Matrix-class), 124
\^,Matrix,ldiMatrix-method (Matrix-class), 124
\^,Matrix,ndiMatrix-method (Matrix-class), 124
\^,ddenseMatrix,ddiMatrix-method (ddenseMatrix-class), 47
\^,ddenseMatrix,ldiMatrix-method (ddenseMatrix-class), 47
\^,ddenseMatrix,ndiMatrix-method (ddenseMatrix-class), 47
\^,ldenseMatrix,ddiMatrix-method (ldenseMatrix-class), 110
\^,ldenseMatrix,ldiMatrix-method (ldenseMatrix-class), 110
\^,ldenseMatrix,ndiMatrix-method (ldenseMatrix-class), 110
\^,ndenseMatrix,ddiMatrix-method (ndenseMatrix-class), 129
\^,ndenseMatrix,ldiMatrix-method (ndenseMatrix-class), 129
\^,ndenseMatrix,ndiMatrix-method (ndenseMatrix-class), 129
all.equal,abIndex, numLike-method (all.equal-methods), 7
all.equal, Matrix, Matrix-method (all.equal-methods), 7
all.equal, Matrix, sparseVector-method (all.equal-methods), 7
all.equal, Matrix, vector-method (all.equal-methods), 7
all.equal, numLike, abIndex-method (all.equal-methods), 7
all.equal, sparseVector, Matrix-method (all.equal-methods), 7
all.equal, sparseVector, sparseVector-method (all.equal-methods), 7
all.equal, vector, Matrix-method (all.equal-methods), 7
all.equal, vector, sparseVector-method (all.equal-methods), 7
all.equal-methods, 7
all.equal.numeric, 7
any, 62
anyDuplicated, 8
anyDuplicatedT (asUniqueT), 8
anyNA, 101
anyNA (is.na-methods), 101
anyNA, denseMatrix-method (is.na-methods), 101
anyNA, diagonalMatrix-method (is.na-methods), 101
anyNA, indMatrix-method (is.na-methods), 101
anyNA, sparseMatrix-method (is.na-methods), 101
anyNA, sparseVector-method (is.na-methods), 101
anyNA-methods (is.na-methods), 101
apply, 125
Arith, 53, 62
Arith, abIndex, abIndex-method (abIndex-class), 5
Arith, abIndex, numLike-method (abIndex-class), 5
Arith, CsparseMatrix, CsparseMatrix-method (CsparseMatrix-class), 45
Arith, CsparseMatrix, numeric-method (CsparseMatrix-class), 45
abbreviate, 148
abIndex, 6, 7, 158, 159
abIndex-class, 5
abIseq, 5, 6, 158
abIseq1 (abIseq), 6
abs, 94
aggregateT (asUniqueT), 8
all, 62
all.equal, 7, 104
all.equal (all.equal-methods), 7
all.equal, abIndex, abIndex-method (all.equal-methods), 7
Arith,ddenseMatrix,logical-method (ddenseMatrix-class), 47
Arith,ddenseMatrix,numeric-method (ddenseMatrix-class), 47
Arith,ddenseMatrix,sparseVector-method (ddenseMatrix-class), 47
Arith,ddiMatrix,logical-method (ddiMatrix-class), 48
Arith,ddiMatrix,numeric-method (ddiMatrix-class), 48
Arith,dgCMatrix,dgCMatrix-method (dgCMatrix-class), 52
Arith,dgCMatrix,logical-method (dgCMatrix-class), 52
Arith,dgCMatrix,numeric-method (dgCMatrix-class), 52
Arith,dgeMatrix,dgeMatrix-method (dgeMatrix-class), 53
Arith,dgeMatrix,logical-method (dgeMatrix-class), 53
Arith,dgeMatrix,numeric-method (dgeMatrix-class), 53
Arith,dgeMatrix,sparseVector-method (dgeMatrix-class), 53
Arith,dlgCMatrix,lgCMatrix-method (lgCMatrix-class), 113
Arith,lgeMatrix,lgMatrix-method (lgMatrix-class), 112
Arith,lgTMatrix,lgTMatrix-method (lgTMatrix-class), 113
Arith,lMatrix,logical-method (lMatrix-class), 62
Arith,lMatrix,numeric-method (lMatrix-class), 62
Arith,logical,ddenseMatrix-method (ddenseMatrix-class), 47
Arith,logical,ddiMatrix-method (ddiMatrix-class), 48
Arith,logical,dgCMatrix-method (dgCMatrix-class), 52
Arith,logical,dgeMatrix-method (dgeMatrix-class), 53
Arith,logical,dpoMatrix-method (dpoMatrix-class), 65
Arith,logical,dsparseMatrix-method (dsparseMatrix-class), 70
Arith,logical,ldiMatrix-method (ldiMatrix-class), 70
Arith,logical,lMatrix-method (lMatrix-class), 62
Arith,logical,nMatrix-method (nMatrix-class), 134
Arith,lsparseMatrix,Matrix-method (lsparseMatrix-class), 113
Arith,lsparseMatrix,nsparseMatrix-method (nsparseMatrix-class), 138
Arith,Matrix,lsparseMatrix-method (Matrix-class), 124
Arith,Matrix,nMatrix-method (Matrix-class), 124
Arith,ngeMatrix,ngeMatrix-method (ngeMatrix-class), 133
Arith,nMatrix,logical-method (nMatrix-class), 134
Arith,nMatrix,numeric-method (nMatrix-class), 134
Arith,nsparseMatrix,nsparseMatrix-method (nsparseMatrix-class), 138
Arith,nsparseMatrix,dsparseMatrix-method (nsparseMatrix-class), 138
Arith, nssparseMatrix, lssparseMatrix-method
  (nsparseMatrix-class), 138
Arith, nssparseMatrix, Matrix-method
  (nsparseMatrix-class), 138
Arith, numeric, CsparseMatrix-method
  (CsparseMatrix-class), 45
Arith, numeric, ddmatrix-method
  (ddenseMatrix-class), 47
Arith, numeric, ddimatrix-method
  (ddiMatrix-class), 48
Arith, numeric, dgeMatrix-method
  (dgeMatrix-class), 53
Arith, numeric, dgeMatrix-method
  (dpeMatrix-class), 53
Arith, numeric, dpoMatrix-method
  (dpoMatrix-class), 53
Arith, numeric, dpoMatrix-method
  (dppMatrix-class), 53
Arith, numeric, dsparseMatrix-method
  (dsparseMatrix-class), 70
Arith, numeric, ldiMatrix-method
  (ldiMatrix-class), 111
Arith, numeric, Matrix-method
  (Matrix-class), 124
Arith, numeric, nMatrix-method
  (nMatrix-class), 134
Arith, numeric, nMatrix-method
  (nMatrix-class), 134
Arith, nnumeric, abIndex-method
  (abIndex-class), 5
Arith, sparseVector, ddmatrix-method
  (sparseVector-class), 183
Arith, sparseVector, dgeMatrix-method
  (sparseVector-class), 183
Arith, sparseVector, sparseVector-method
  (sparseVector-class), 183
Arith, triangularMatrix, diagonalMatrix-method
  (triangularMatrix-class), 192
as, 40, 41, 112, 115, 116, 134, 140, 141
as.array, 125
as.array, Matrix-method
  (Matrix-class), 124
as.array, sparseVector-method
  (sparseVector-class), 183
as.complex, Matrix-method
  (Matrix-class), 124
as.complex, sparseVector-method
  (sparseVector-class), 183
as.integer, abIndex-method
  (abIndex-class), 5
as.integer, Matrix-method
  (Matrix-class), 124
as.integer, sparseVector-method
  (sparseVector-class), 183
as.logical, Matrix-method
  (Matrix-class), 124
as.logical, sparseVector-method
  (sparseVector-class), 183
as.matrix, 125
as.matrix, Matrix-method (Matrix-class), 124
as.matrix, sparseVector-method
  (sparseVector-class), 183
as.numeric, abIndex-method
  (abIndex-class), 5
as.numeric, Matrix-method
  (Matrix-class), 124
as.numeric, sparseVector-method
  (sparseVector-class), 183
as.vector, 7
as.vector, abIndex-method
  (abIndex-class), 5
as.vector, Matrix-method (Matrix-class), 124
as.vector, sparseVector-method
  (sparseVector-class), 183
asPerm, 49
asPerm (invertPerm), 99
asUniqueT, 8, 55, 113, 119
attribute, 104, 105
band, 11, 12, 57
band (band-methods), 9
band, CsparseMatrix-method
  (band-methods), 9
band, ddmatrix-method (band-methods), 9
band, diagonalMatrix-method
  (band-methods), 9
band, indMatrix-method (band-methods), 9
band, matrix-method (band-methods), 9
band, RsparseMatrix-method
  (band-methods), 9
band, TsparseMatrix-method
  (band-methods), 9
band-methods, 9
bandSparse, 10, 11, 13, 57, 123, 175
bdig, 12, 13, 123, 175, 187
boolmatmul-methods, 15
BunchKaufman, 17–20, 36, 66, 80, 81, 118, 129, 150, 164, 167
BunchKaufman (BunchKaufman-methods), 19
BunchKaufman, dspMatrix-method (BunchKaufman-methods), 19
BunchKaufman, dsyMatrix-method (BunchKaufman-methods), 19
BunchKaufman, matrix-method (BunchKaufman-methods), 19
BunchKaufman-class, 16
BunchKaufman-methods, 19
BunchKaufmanFactorization, 17, 19
BunchKaufmanFactorization-class (MatrixFactorization-class), 128

chol (chol-methods), 26
chol, ddiMatrix-method (chol-methods), 26
chol, diagonalMatrix-method (chol-methods), 26
chol, dsCMatrix-method (chol-methods), 26
chol, dspMatrix-method (chol-methods), 26
chol, dsRMatrix-method (chol-methods), 26
chol, dsyMatrix-method (chol-methods), 26
chol, generalMatrix-method (chol-methods), 26
chol, symmetricMatrix-method (chol-methods), 26
chol, triangularMatrix-method (chol-methods), 26
chol-methods, 26
chol2inv, 30, 31
chol2inv (chol2inv-methods), 30
chol2inv, ANY-method (chol2inv-methods), 30
chol2inv, ddiMatrix-method (chol2inv-methods), 30
chol2inv, diagonalMatrix-method (chol2inv-methods), 30
chol2inv, dtCMatrix-method (chol2inv-methods), 30
chol2inv, dtRMatrix-method (chol2inv-methods), 30
chol2inv, dtTMatrix-method (chol2inv-methods), 30
chol2inv, generalMatrix-method (chol2inv-methods), 30
chol2inv, symmetricMatrix-method (chol2inv-methods), 30
chol2inv, triangularMatrix-method (chol2inv-methods), 30
chol2inv-methods, 30

Cholesky, 20, 23–25, 27, 28, 32, 33, 36, 66, 69, 80, 81, 87, 88, 118, 129, 150, 164, 167, 177, 196
Cholesky (Cholesky-methods), 34
Cholesky, ddiMatrix-method (Cholesky-methods), 34
Cholesky, diagonalMatrix-method (Cholesky-methods), 34
(Cholesky-methods), 34
Cholesky,dsCMatrix-method
(Cholesky-methods), 34
Cholesky,dsMatrix-method
(Cholesky-methods), 34
Cholesky,dsRMatrix-method
(Cholesky-methods), 34
Cholesky,dsTMatrix-method
(Cholesky-methods), 34
Cholesky,dsyMatrix-method
(Cholesky-methods), 34
Cholesky,generalMatrix-method
(Cholesky-methods), 34
Cholesky,matrix-method
(Cholesky-methods), 34
Cholesky,symmetricMatrix-method
(Cholesky-methods), 34
Cholesky,triangularMatrix-method
(Cholesky-methods), 34
Cholesky-class, 31
Cholesky-methods, 34
CholeskyFactorization-class
(MatrixFactorization-class), 128
class, 5, 12, 22, 60, 102, 104, 105, 127, 131, 154, 158, 173, 182, 183, 188
coerce, 193
coerce,abIndex,integer-method
(abIndex-class), 5
coerce,abIndex,numeric-method
(abIndex-class), 5
coerce,abIndex,seqMat-method
(abIndex-class), 5
coerce,abIndex,vector-method
(abIndex-class), 5
coerce,ANY,denseMatrix-method
(denseMatrix-class), 51
coerce,ANY,Matrix-method
(Matrix-class), 124
coerce,ANY,sparseMatrix-method
(sparseMatrix-class), 176
coerce,ANY,sparseVector-method
(sparseVector-class), 183
coerce,BunchKaufman,dtrMatrix-method
(BunchKaufman-class), 16
coerce,CHMsimply,dtCMatrix-method
(CHMfactor-class), 22
coerce,Cholesky,cholMatrix-method
(Cholesky-class), 22
coerce,Cholesky,dtrMatrix-method
(Cholesky-class), 31
coerce,Cholesky,dsyMatrix-method
(Cholesky-class), 31
coopMatrix,corMatrix-method
(dpoMatrix-class), 65
coe,corMatrix,copMatrix-method
(dpoMatrix-class), 65
coe,corMatrix,dpoMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dpoMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dppMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dppMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
coe,Cholesky,dsyMatrix-method
(dpoMatrix-class), 65
```
coerce, graph, TsparseMatrix-method
(coerce-methods-graph), 40
coerce, graphAM, TsparseMatrix-method
(coerce-methods-graph), 40
coerce, graphNEL, TsparseMatrix-method
(coerce-methods-graph), 40
coerce, indMatrix, pMatrix-method
(indMatrix-class), 97
coerce, list, indMatrix-method
(indMatrix-class), 97
coerce, logical, abIndex-method
(abIndex-class), 5
coerce, Matrix, copMatrix-method
(Matrix-class), 124
coerce, matrix, copMatrix-method
(dpoMatrix-class), 65
coerce, Matrix, corMatrix-method
(Matrix-class), 124
coerce, matrix, corMatrix-method
(dpoMatrix-class), 65
coerce, Matrix, CsparseMatrix-method
(Matrix-class), 124
coerce, matrix, CsparseMatrix-method
(CsparseMatrix-class), 45
coerce, Matrix, ddenseMatrix-method
(Matrix-class), 124
coerce, matrix, ddenseMatrix-method
(ddenseMatrix-class), 47
coerce, Matrix, denseMatrix-method
(Matrix-class), 124
coerce, matrix, denseMatrix-method
(denseMatrix-class), 51
coerce, matrix, dgCMatrix-method
(dgCMatrix-class), 52
coerce, Matrix, diagonalMatrix-method
(Matrix-class), 124
coerce, matrix, diagonalMatrix-method
(diagonalMatrix-class), 58
coerce, Matrix, dMatrix-method
(Matrix-class), 124
coerce, matrix, dMatrix-method
(dMatrix-class), 62
coerce, Matrix, dpoMatrix-method
(Matrix-class), 124
coerce, matrix, dpoMatrix-method
(dpoMatrix-class), 65
coerce, Matrix, dsparseMatrix-method
(Matrix-class), 124
coerce, matrix, dsparseMatrix-method
(dsparseMatrix-class), 70
coerce, Matrix, generalMatrix-method
(Matrix-class), 124
coerce, matrix, generalMatrix-method
(generalMatrix-class), 92
coerce, Matrix, graph-method
(coerce-methods-graph), 40
coerce, Matrix, graphNEL-method
(coerce-methods-graph), 40
coerce, Matrix, indMatrix-method
(Matrix-class), 124
coerce, matrix, indMatrix-method
(indMatrix-class), 97
coerce, Matrix, ldenseMatrix-method
(ldenseMatrix-class), 110
coerce, Matrix, lMatrix-method
(Matrix-class), 124
coerce, matrix, lMatrix-method
(lMatrix-class), 134
coerce, Matrix, ndenseMatrix-method
(ndenseMatrix-class), 129
coerce, Matrix, nMatrix-method
(nMatrix-class), 134
coerce, Matrix, nsparsesMatrix-method (Matrix-class), 124
coerce, matrix, nsparsesMatrix-method (nsparsesMatrix-class), 138
coerce, Matrix, packedMatrix-method (Matrix-class), 124
coerce, matrix, packedMatrix-method (packedMatrix-class), 143
coerce, Matrix, pMatrix-method (Matrix-class), 124
coerce, matrix, pMatrix-method (pMatrix-class), 145
coerce, Matrix, RsparseMatrix-method (Matrix-class), 124
coerce, matrix, RsparseMatrix-method (RsparseMatrix-class), 161
coerce, Matrix, sparseMatrix-method (Matrix-class), 124
coerce, matrix, sparseMatrix-method (sparseMatrix-class), 176
coerce, Matrix, sparseVector-method (Matrix-class), 124
coerce, matrix, sparseVector-method (sparseVector-class), 183
coerce, Matrix, symmetricMatrix-method (Matrix-class), 124
coerce, matrix, symmetricMatrix-method (symmetricMatrix-class), 190
coerce, Matrix, triangularMatrix-method (Matrix-class), 124
coerce, matrix, triangularMatrix-method (triangularMatrix-class), 192
coerce, Matrix, TsparseMatrix-method (Matrix-class), 124
coerce, matrix, TsparseMatrix-method (TsparseMatrix-class), 193
coerce, Matrix, unpackedMatrix-method (Matrix-class), 124
coerce, matrix, unpackedMatrix-method (unpackedMatrix-class), 194
coerce, matrix.coo, CsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, dgCMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, dgTMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, Matrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, RsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, sparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.coo, TsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csc, CsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csc, CsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csc, sparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csc, TsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csr, CsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csr, CsparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csr, sparseMatrix-method (coerce-methods-SparseM), 41
coerce, matrix.csr, TsparseMatrix-method (coerce-methods-SparseM), 41
coerce, Matrix, indMatrix-method (Matrix-class), 124
coerce, matrix, indMatrix-method (indMatrix-class), 97
coerce, numeric, abIndex-method (abIndex-class), 5
coerce, numeric, indMatrix-method (indMatrix-class), 97
coerce, numeric, pMatrix-method (pMatrix-class), 145
coerce, numeric, seqMat-method (abIndex-class), 5
coerce, pBunchKaufman, dtpMatrix-method (BunchKaufman-class), 16
coerce, pCholesky, dtpMatrix-method (Cholesky-class), 31
coerce, seqMat, abIndex-method
  (abIndex-class), 5
coerce, seqMat, numeric-method
  (abIndex-class), 5
coerce, sparseVector, CsparseMatrix-method
  (sparseVector-class), 183
coerce, sparseVector, denseMatrix-method
  (sparseVector-class), 183
coerce, sparseVector, dsparseVector-method
  (sparseVector-class), 183
coerce, sparseVector, generalMatrix-method
  (sparseVector-class), 183
coerce, sparseVector, isparseVector-method
  (sparseVector-class), 183
coerce, sparseVector, Matrix-method
  (sparseVector-class), 183
coerce, sparseVector, ldenseMatrix-method
  (ldenseMatrix-class), 110
coerce, vector, ldenseMatrix-method
  (dMatrix-class), 62
coerce, vector, lMatrix-method
  (dMatrix-class), 62
coerce, vector, lsparseMatrix-method
  (lsparseMatrix-class), 113
coerce, vector, lsparseVector-method
  (sparseVector-class), 183
coerce, vector, Matrix-method
  (Matrix-class), 124
coerce, vector, ndenseMatrix-method
  (ndenseMatrix-class), 129
coerce, vector, nMatrix-method
  (nMatrix-class), 134
coerce, vector, nsparseMatrix-method
  (nsparseMatrix-class), 138
coerce, vector, nsparseVector-method
  (sparseVector-class), 183
coerce, vector, RsparseMatrix-method
  (RsparseMatrix-class), 161
coerce, vector, sparseMatrix-method
  (sparseMatrix-class), 176
coerce, vector, sparseVector-method
  (sparseVector-class), 183
coerce, vector, TsparseMatrix-method
  (TsparseMatrix-class), 193
coerce, vector, unpackedMatrix-method
  (unpackedMatrix-class), 194
coerce, vector, zsparseVector-method
  (sparseVector-class), 183
coerce-methods-graph, 40
colMeans(colSums-methods), 42
colMeans(CsparseMatrix-method)
  (colSums-methods), 42
colMeans(denseMatrix-method)
  (colSums-methods), 42
colMeans(dMatrix-method)
  (dMatrix-class), 51
colMeans(diagonalMatrix-method)
  (colSums-methods), 42
colMeans(indMatrix-method)
  (colSums-methods), 42
colMeans(RsparseMatrix-method)
  (colSums-methods), 42
colMeans(TsparseMatrix-method)
  (colSums-methods), 42
colMeans-methods (colSums-methods), 42
colScale (dimScale), 61
colSums, 42, 43, 47, 51, 125, 177
INDEX

219

colSums (colSums-methods), 42
colSums, CsparseMatrix-method
(colSums-methods), 42
colSums, denseMatrix-method
(colSums-methods), 42
colSums, diagonalMatrix-method
(colSums-methods), 42
colSums, indMatrix-method
(colSums-methods), 42
colSums, RsparseMatrix-method
(colSums-methods), 42
colSums-methods, 42

Compare, CsparseMatrix, CsparseMatrix-method
(CsparseMatrix-class), 45
Compare, dMatrix, logical-method
(dMatrix-class), 62
Compare, dMatrix, numeric-method
(dMatrix-class), 62
Compare, lgeMatrix, lgeMatrix-method
(lgeMatrix-class), 112
Compare, lMatrix, logical-method
(dMatrix-class), 62
Compare, lMatrix, numeric-method
(dMatrix-class), 62
Compare, logical, dMatrix-method
(dMatrix-class), 62
Compare, logical, nMatrix-method
(nMatrix-class), 134
Compare, ngeMatrix, ngeMatrix-method
(ngeMatrix-class), 133
Compare, nMatrix, logical-method
(nMatrix-class), 134
Compare, nMatrix, nMatrix-method
(nMatrix-class), 134
Compare, nMatrix, numeric-method
(nMatrix-class), 134
Compare, numeric, dMatrix-method
(dMatrix-class), 62
Compare, numeric, lMatrix-method
(dMatrix-class), 62
Compare, numeric, nMatrix-method
(nMatrix-class), 134
Compare, triangularMatrix, diagonalMatrix-method
(triangularMatrix-class), 192

complex, 126
condest, 43, 44, 157
contrasts, 169
copMatrix-class (dpoMatrix-class), 65
corMatrix, 73
corMatrix-class (dpoMatrix-class), 65
cov2cor, 61
cov2cor, packedMatrix-method
(packedMatrix-class), 143
cov2cor, sparseMatrix-method
(sparseMatrix-class), 176
cov2cor, unpackedMatrix-method
(unpackedMatrix-class), 194
crossprod, 46, 53, 66, 98, 120, 122, 177
crossprod (matmult-methods), 120
crossprod, ANY, Matrix-method
(matmult-methods), 120
crossprod, ANY, sparseVector-method
(matmult-methods), 120
crossprod, CsparseMatrix, CsparseMatrix-method
(matmult-methods), 120
crossprod, CsparseMatrix, denseMatrix-method
(matmult-methods), 120
crossprod, CsparseMatrix, diagonalMatrix-method
(matmult-methods), 120
crossprod, CsparseMatrix, matrix-method
(matmult-methods), 120
crossprod, CsparseMatrix, missing-method
(matmult-methods), 120
crossprod, CsparseMatrix, RsparseMatrix-method
(matmult-methods), 120
crossprod, CsparseMatrix, TsparseMatrix-method
(matmult-methods), 120
crossprod, denseMatrix, vector-method
(matmult-methods), 120
crossprod, denseMatrix, CsparseMatrix-method
(matmult-methods), 120
crossprod, denseMatrix, denseMatrix-method
(matmult-methods), 120
crossprod, denseMatrix, diagonalMatrix-method
(matmult-methods), 120
crossprod, denseMatrix, matrix-method
(matmult-methods), 120
crossprod, denseMatrix, missing-method
(matmult-methods), 120
crossprod, denseMatrix, RsparseMatrix-method
(matmult-methods), 120
crossprod, denseMatrix, TsparseMatrix-method
(matmult-methods), 120
(matmult-methods), 120

crossprod,denseMatrix,vector-method
(matmult-methods), 120

crossprod,diagonalMatrix,CsparseMatrix-method
(matmult-methods), 120

crossprod,diagonalMatrix,denseMatrix-method
(matmult-methods), 120

crossprod,diagonalMatrix,diagonalMatrix-method
(matmult-methods), 120

crossprod,diagonalMatrix,matrix-method
(matmult-methods), 120

crossprod,diagonalMatrix,missing-method
(matmult-methods), 120

crossprod,diagonalMatrix,RsparseMatrix-method
(matmult-methods), 120

crossprod,diagonalMatrix,TsparseMatrix-method
(matmult-methods), 120

crossprod,indMatrix,Matrix-method
(matmult-methods), 120

crossprod,matrix,CsparseMatrix-method
(matmult-methods), 120

crossprod,matrix,denseMatrix-method
(matmult-methods), 120

crossprod,matrix,diagonalMatrix-method
(matmult-methods), 120

crossprod,matrix,matrix-method
(matmult-methods), 120

crossprod,matrix,missing-method
(matmult-methods), 120

crossprod,matrix,RsparseMatrix-method
(matmult-methods), 120

crossprod,matrix,sparseVector-method
(matmult-methods), 120

crossprod,matrix,TsparseMatrix-method
(matmult-methods), 120

crossprod,matrix,sparseVector-method
(matmult-methods), 120

crossprod,RsparseMatrix,CsparseMatrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,denseMatrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,diagonalMatrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,matrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,missing-method
(matmult-methods), 120

crossprod,RsparseMatrix,RsparseMatrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,TsparseMatrix-method
(matmult-methods), 120

crossprod,RsparseMatrix,vector-method
(matmult-methods), 120

crossprod,sparseVector,ANY-method
(matmult-methods), 120

crossprod,sparseVector,Matrix-method
(matmult-methods), 120

crossprod,sparseVector,matrix-method
(matmult-methods), 120

crossprod,sparseVector,missing-method
(matmult-methods), 120

crossprod,sparseVector,sparseVector-method
(matmult-methods), 120

crossprod,TsparseMatrix,CsparseMatrix-method
(matmult-methods), 120

crossprod,TsparseMatrix,denseMatrix-method
(matmult-methods), 120

crossprod,TsparseMatrix,diagonalMatrix-method
(matmult-methods), 120

crossprod,TsparseMatrix,matrix-method
(matmult-methods), 120

crossprod,TsparseMatrix,missing-method
(matmult-methods), 120
determinant, Cholesky, logical-method
(Cholesky-class), 31
determinant, denseLU, logical-method
(denseLU-class), 49
determinant, dgCMat, logical-method
(dgCMat-class), 52
determinant, dgeMatrix, logical-method
(dgeMatrix-class), 53
determinant, dgRMat, logical-method
(dgRMat-class), 54
determinant, dgTMat, logical-method
(dgTMat-class), 55
determinant, diagonalMatrix, logical-method
(diagonalMatrix-class), 58
determinant, dpoMat, logical-method
(dpoMat-class), 65
determinant, dppMat, logical-method
(dppMat-class), 65
determinant, dsCMat, logical-method
(dsCMat-class), 68
determinant, dspMat, logical-method
(dspMat-class), 72
determinant, dsRMat, logical-method
(dsRMat-class), 70
determinant, dsTMat, logical-method
(dsTMat-class), 70
determinant, dsysMat, logical-method
(dsysMat-class), 72
determinant, indMat, logical-method
(indMat-class), 97
determinant, Matrix, logical-method
(Matrix-class), 124
determinant, MatrixFactorization, missing-method
(MatrixFactorization-class), 128
determinant, pBunchKaufman, logical-method
(pBunchKaufman-class), 16
determinant, pCholesky, logical-method
(pCholesky-class), 16
determinant, pMatrix, logical-method
(pMatrix-class), 145
determinant, Schur, logical-method
(Schur-class), 162
determinant, sparseLU, logical-method
(sparseLU-class), 171
determinant, sparseQR, logical-method
(sparseQR-class), 171
INDEX

determinant, triangularMatrix, logical-method (triangularMatrix-class), 192
dgCMatrix, 20, 24, 25, 47, 54, 55, 63, 69–71, 74, 77, 80, 81, 88, 114, 118, 121, 126, 139, 150, 171, 172, 179, 180
dgCMatrix-class, 52
dgeMatrix, 47, 49, 50, 63, 66, 69, 71, 73, 74, 77, 81, 118, 126, 162–164, 180
dgeMatrix-class, 53
dgRMatrix, 70, 161
dgRMatrix-class, 54
dgTMatrix, 40, 69–71, 74, 77, 95, 113, 114, 193, 194
dgTMatrix-class, 55
diag, 12, 57, 97, 129, 130
diag, ChMfactor-method (ChMfactor-class), 22
diag, Cholesky-method (Cholesky-class), 31
diag, CsparseMatrix-method (CsparseMatrix-class), 45
diag, denseMatrix-method (denseMatrix-class), 51
diag, diagonalMatrix-method (diagonalMatrix-class), 58
diag, indMatrix-method (indMatrix-class), 97
diag, pCholesky-method (Cholesky-class), 31
diag, RsparseMatrix-method (RsparseMatrix-class), 161
diag, TsparseMatrix-method (TsparseMatrix-class), 193
diag<-, CsparseMatrix-method (CsparseMatrix-class), 45
diag<-, denseMatrix-method (denseMatrix-class), 51
diag<-, diagonalMatrix-method (diagonalMatrix-class), 58
diag<-, indMatrix-method (indMatrix-class), 97
diag<-, RsparseMatrix-method (RsparseMatrix-class), 161
diag<-, TsparseMatrix-method (TsparseMatrix-class), 193
diagN2U (diagU2N), 59
Diagonal, 13, 48, 56, 59, 111, 123, 174, 175, 187
diagonalMatrix, 13, 22, 27, 28, 30, 48, 56, 57, 88, 93, 106, 111, 120, 123, 192
diagonalMatrix-class, 58
diagU2N, 59
diff, 125, 158
diff, denseMatrix-method (denseMatrix-class), 51
diff, sparseMatrix-method (sparseMatrix-class), 176
diff, sparseVector-method (sparseVector-class), 183
dim, 125, 148, 159
dim, Matrix-method (Matrix-class), 124
dim, MatrixFactorization-method (MatrixFactorization-class), 128
dim<-, denseMatrix-method (denseMatrix-class), 51
dim<-, sparseMatrix-method (sparseMatrix-class), 176
dim<-, sparseVector-method (sparseVector-class), 183
dimnames, 42, 48, 58, 63, 84, 92, 102, 103, 106, 111, 123, 125, 173, 190
dimnames, Matrix-method (Matrix-class), 124
dimnames, MatrixFactorization-method (MatrixFactorization-class), 128
dimnames, symmetricMatrix-method (symmetricMatrix-class), 190
dimnames<-, generalMatrix, list-method (generalMatrix-class), 92
dimnames<-, generalMatrix, NULL-method (generalMatrix-class), 92
dimnames<-, Matrix, list-method (Matrix-class), 124
dimnames<-, Matrix, NULL-method (Matrix-class), 124
dimnames<-, MatrixFactorization, list-method (MatrixFactorization-class), 128
dimnames<-, MatrixFactorization, NULL-method (MatrixFactorization-class), 128
dimnames<-, symmetricMatrix, list-method (symmetricMatrix-class), 190
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimnames&lt;- symmetricMatrix, NULL-method</td>
<td>190</td>
</tr>
<tr>
<td>dimScale</td>
<td>61</td>
</tr>
<tr>
<td>dMatrix</td>
<td>15, 47, 48, 61, 82, 125</td>
</tr>
<tr>
<td>dMatrix-class</td>
<td>62</td>
</tr>
<tr>
<td>dmperm</td>
<td>63</td>
</tr>
<tr>
<td>double</td>
<td>159, 169</td>
</tr>
<tr>
<td>dpoMatrix</td>
<td>32, 33, 36, 73, 93, 130, 131, 194</td>
</tr>
<tr>
<td>dpoMatrix-class</td>
<td>65</td>
</tr>
<tr>
<td>dppMatrix</td>
<td>32, 33, 36, 73, 144</td>
</tr>
<tr>
<td>dppMatrix-class (dpoMatrix-class)</td>
<td>65</td>
</tr>
<tr>
<td>drop,abIndex-method (abIndex-class)</td>
<td>5</td>
</tr>
<tr>
<td>drop,Matrix-method (Matrix-class)</td>
<td>124</td>
</tr>
<tr>
<td>drop0</td>
<td>15, 63, 67, 136, 183</td>
</tr>
<tr>
<td>dsCMatrix</td>
<td>24, 25, 36, 52, 81, 88, 197, 198</td>
</tr>
<tr>
<td>dsCMatrix-class</td>
<td>68</td>
</tr>
<tr>
<td>dsparseMatrix, 47, 55, 57, 69, 71</td>
<td></td>
</tr>
<tr>
<td>dsparseMatrix-class</td>
<td>70</td>
</tr>
<tr>
<td>dsparseVector-class</td>
<td></td>
</tr>
<tr>
<td>(sparseVector-class)</td>
<td>183</td>
</tr>
<tr>
<td>dspMatrix</td>
<td>17, 20, 32, 144</td>
</tr>
<tr>
<td>dspMatrix-class (dsyMatrix-class)</td>
<td>72</td>
</tr>
<tr>
<td>dsyMatrix-class</td>
<td>70</td>
</tr>
<tr>
<td>dtCMatrix</td>
<td>197, 198</td>
</tr>
<tr>
<td>dtCMatrix-class (dsCMatrix-class)</td>
<td>68</td>
</tr>
<tr>
<td>dsysMatrix-class</td>
<td>17, 18, 20, 32, 54, 66, 190, 195</td>
</tr>
<tr>
<td>dsysMatrix-class</td>
<td>72</td>
</tr>
<tr>
<td>dtMatrix</td>
<td>24, 52, 60, 63, 80, 81, 171, 180</td>
</tr>
<tr>
<td>dtMatrix-class</td>
<td>73</td>
</tr>
<tr>
<td>dtpMatrix</td>
<td>17, 32, 79, 80</td>
</tr>
<tr>
<td>dtpMatrix-class</td>
<td>75</td>
</tr>
<tr>
<td>dtRMatrix</td>
<td>17, 32, 54, 74, 76, 80, 81, 193</td>
</tr>
<tr>
<td>dtRMatrix-class</td>
<td>77</td>
</tr>
<tr>
<td>dtrMatrix</td>
<td>78</td>
</tr>
<tr>
<td>dtTMatrix-class (dtCMatrix-class)</td>
<td>73</td>
</tr>
<tr>
<td>eigen</td>
<td>20, 21, 125</td>
</tr>
<tr>
<td>error</td>
<td>117</td>
</tr>
<tr>
<td>expand (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand, CHMfactor-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand, denseLU-method (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand, sparseLU-method (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand-methods</td>
<td>79</td>
</tr>
<tr>
<td>expand1, BunchKaufman-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, CHMsimpl1-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, CHMsuper-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, Cholesky-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, denseLU-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, pBunchKaufman-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, pCholesky-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, Schur-method (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, sparseLU-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand1, sparseQR-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2 (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, BunchKaufman-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, CHMsimpl1-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, CHMsuper-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, Cholesky-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, denseLU-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, pBunchKaufman-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, pCholesky-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, Schur-method (expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, sparseLU-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2, sparseQR-method</td>
<td></td>
</tr>
<tr>
<td>(expand-methods)</td>
<td>79</td>
</tr>
<tr>
<td>expand2-methods</td>
<td>79</td>
</tr>
<tr>
<td>expm</td>
<td>63</td>
</tr>
<tr>
<td>expm (expm-methods)</td>
<td>82</td>
</tr>
<tr>
<td>expm, ddiMatrix-method (expm-methods)</td>
<td>82</td>
</tr>
</tbody>
</table>
expm,dgeMatrix-method (expm-methods), 82
expm,dMatrix-method (expm-methods), 82
expm,dspMatrix-method (expm-methods), 82
expm,dmmatrix-method (expm-methods), 82
expm,dsyMatrix-method (expm-methods), 82
expm,dtpMatrix-method (expm-methods), 82
expm,Matrix-method (expm-methods), 82
expm,matrix-method (expm-methods), 82
expm-methods, 82
extends, 123
externalFormats, 83
Extract, 188, 189
fac2Sparse (sparse.model.matrix), 168
fac2sparse, 169
fac2sparse (sparse.model.matrix), 168
facmul (facmul-methods), 85
facmul-methods, 85
factor, 169
FALSE, 102
fastMisc, 86
finite, 19, 27, 35, 117, 150, 164, 166, 196
forceSymmetric, 105
forceSymmetric (forceSymmetric-methods), 90
forceSymmetric, CsparseMatrix, character-method
(forceSymmetric-methods), 90
forceSymmetric, CsparseMatrix, missing-method
(forceSymmetric-methods), 90
forceSymmetric, denseMatrix, character-method
(forceSymmetric-methods), 90
forceSymmetric, denseMatrix, missing-method
(forceSymmetric-methods), 90
forceSymmetric, diagonalMatrix, character-method
identity, 25
(forceSymmetric-methods), 90
forceSymmetric, diagonalMatrix, missing-method
(forceSymmetric-methods), 90
forceSymmetric, indMatrix, character-method
(forceSymmetric-methods), 90
forceSymmetric, indMatrix, missing-method
(forceSymmetric-methods), 90
forceSymmetric, matrix, character-method
(forceSymmetric-methods), 90
forceSymmetric, matrix, missing-method
(forceSymmetric-methods), 90
forceSymmetric, RsparseMatrix, character-method
index-class, 92
(forceSymmetric-methods), 90
forceSymmetric, RsparseMatrix, character-method
image (image-methods), 94
forceSymmetric, RsparseMatrix, character-method
image, ANY-method (image-methods), 94
forceSymmetric, RsparseMatrix, character-method
image, CHMfactor-method (image-methods), 94
forceSymmetric, RsparseMatrix, character-method
image, dgTMatrix-method (image-methods), 94
forceSymmetric, RsparseMatrix, character-method
image, Matrix-method (image-methods), 94
forceSymmetric, RsparseMatrix, character-method
image-methods, 94
forceSymmetric, RsparseMatrix, character-method
indMatrix-class (Matrix-notyet), 126
forceSymmetric, RsparseMatrix, character-method
index, 188
forceSymmetric, RsparseMatrix, character-method
indMatrix, 22, 67, 88, 145, 146
indMatrix-class, 97
Inf, 102
initialize, Matrix-method
(Matrix-class), 124
initialize, sparseVector-method
(sparseVector-class), 183
integer, 46, 55, 68, 74, 136
invertPerm, 99
invisible, 148
invPerm, 145, 146
invPerm(invertPerm), 99
is, 94
is.finite, 101
is.finite(is.na-methods), 101
is.finite, abIndex-method
(is.na-methods), 101
is.finite, denseMatrix-method
(is.na-methods), 101
is.finite, diagonalMatrix-method
(is.na-methods), 101
is.finite, indMatrix-method
(is.na-methods), 101
is.finite, sparseMatrix-method
(is.na-methods), 101
is.finite, sparseVector-method
(is.na-methods), 101
is.finite-methods(is.na-methods), 101
is.infinite, 101
is.infinite(is.na-methods), 101
is.infinite, abIndex-method
(is.na-methods), 101
is.infinite, denseMatrix-method
(is.na-methods), 101
is.infinite, diagonalMatrix-method
(is.na-methods), 101
is.infinite, indMatrix-method
(is.na-methods), 101
is.infinite, sparseMatrix-method
(is.na-methods), 101
is.infinite, sparseVector-method
(is.na-methods), 101
is.infinite-methods(is.na-methods), 101
is.na, 101
is.na(is.na-methods), 101
is.na, abIndex-method(is.na-methods), 101
is.na, denseMatrix-method
(is.na-methods), 101
is.na, diagonalMatrix-method
(is.na-methods), 101
is.na, indMatrix-method(is.na-methods), 101
is.na, sparseMatrix-method
(is.na-methods), 101
is.na, sparseVector-method
(is.na-methods), 101
is.na-methods, 101
is.nan, 101
is.nan(is.na-methods), 101
is.nan, denseMatrix-method
(is.na-methods), 101
is.nan, diagonalMatrix-method
(is.na-methods), 101
is.nan, indMatrix-method
(is.na-methods), 101
is.nan, sparseMatrix-method
(is.na-methods), 101
is.nan, sparseVector-method
(is.na-methods), 101
is.nan-methods(is.na-methods), 101
is.null, 102, 103
is.null.DN, 102
isDiagonal, 59
isDiagonal(isTriangular-methods), 105
isDiagonal, CsparseMatrix-method
(isTriangular-methods), 105
isDiagonal, denseMatrix-method
(isTriangular-methods), 105
isDiagonal, diagonalMatrix-method
(isTriangular-methods), 105
isDiagonal, indMatrix-method
(isTriangular-methods), 105
isDiagonal, matrix-method
(isTriangular-methods), 105
isDiagonal, RsparseMatrix-method
(isTriangular-methods), 105
isDiagonal, TsparseMatrix-method
(isTriangular-methods), 105
isDiagonal-methods
(isTriangular-methods), 105
isLDL, 167
isLDL(CHMfactor-class), 22
isparseVector-class
(sparseVector-class), 183
isPerm(invertPerm), 99
isSymmetric, 87, 105, 106, 131, 143, 190.
isSymmetric (isSymmetric-methods), 103
isSymmetric,CsparseMatrix-method (isSymmetric-methods), 103
isSymmetric,denseMatrix-method (isSymmetric-methods), 103
isSymmetric,dgCMatrix-method (isSymmetric-methods), 103
isSymmetric,dgeMatrix-method (isSymmetric-methods), 103
isSymmetric,dgRMatrix-method (isSymmetric-methods), 103
isSymmetric,dgTMatrix-method (isSymmetric-methods), 103
isSymmetric,diagonalMatrix-method (isSymmetric-methods), 103
isSymmetric,dtCMatrix-method (isSymmetric-methods), 103
isSymmetric,dtpMatrix-method (isSymmetric-methods), 103
isSymmetric,dtRMatrix-method (isSymmetric-methods), 103
isSymmetric,dtrMatrix-method (isSymmetric-methods), 103
isSymmetric,dtTMatrix-method (isSymmetric-methods), 103
isSymmetric,indMatrix-method (isSymmetric-methods), 103
isSymmetric,matrix-method (isSymmetric-methods), 103
isSymmetric,RsparseMatrix-method (isSymmetric-methods), 103
isSymmetric,TsparseMatrix-method (isSymmetric-methods), 103
isSymmetric-methods, 103, 190
isSymmetric.matrix, 104
isTriangular, 87, 143, 193
isTriangular (isTriangular-methods), 105
isTriangular,CsparseMatrix-method (isTriangular-methods), 105
isTriangular,denseMatrix-method (isTriangular-methods), 105
isTriangular,diagonalMatrix-method (isTriangular-methods), 105
isTriangular,indMatrix-method (isTriangular-methods), 105
isTriangular,matrix-method (isTriangular-methods), 105
isTriangular,RsparseMatrix-method (isTriangular-methods), 105
isTriangular,TsparseMatrix-method (isTriangular-methods), 105
isTriangular-methods, 105
isUniqueT (asUniqueT), 8
kappa, 125, 157
KhatriRao, 106
KNex, 108
kronecker, 13, 47, 51, 98, 106, 107, 109, 126
kronecker (kronecker-methods), 109
kronecker,CsparseMatrix,CsparseMatrix-method (kronecker-methods), 109
kronecker,CsparseMatrix,diagonalMatrix-method (kronecker-methods), 109
kronecker,CsparseMatrix,Matrix-method (kronecker-methods), 109
kronecker,denseMatrix,denseMatrix-method (kronecker-methods), 109
kronecker,denseMatrix,Matrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,CsparseMatrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,diagonalMatrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,Matrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,RsparseMatrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,RsparseMatrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,indMatrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,matrix-method (kronecker-methods), 109
kronecker,diagonalMatrix,RsparseMatrix-method (kronecker-methods), 109
kronecker,indMatrix,diagonalMatrix-method (kronecker-methods), 109
kronecker,indMatrix,matrix-method (kronecker-methods), 109
kronecker,Matrix,matrix-method (kronecker-methods), 109
kronecker,matrix,Matrix-method (kronecker-methods), 109
kronecker,Matrix,vector-method (kronecker-methods), 109
kronecker,RsparseMatrix,diagonalMatrix-method (kronecker-methods), 109
kronecker,RsparseMatrix,Matrix-method (kronecker-methods), 109
kronecker,RsparseMatrix,matrix-method (kronecker-methods), 109
kronecker,RsparseMatrix,VectorXd-method (kronecker-methods), 109
kronecker,RsparseMatrix,VectorXd-method (kronecker-methods), 109
kronecker,RsparseMatrix,VectorXd-method (kronecker-methods), 109
kronecker,RsparseMatrix,VectorXd-method (kronecker-methods), 109
INDEX

kroncker,RsparseMatrix,RsparseMatrix-method Logic,ANY,Matrix-method (Matrix-class), 124
(kroncker-methods), 109
kroncker,TsparseMatrix,diagonalMatrix-method Logic,CSparseMatrix,CSparseMatrix-method
(kroncker-methods), 109
kroncker,TsparseMatrix,Matrix-method Logic,DMatrix,logical-method
(kroncker-methods), 109
kroncker,TsparseMatrix,TsparseMatrix-method Logic,DMatrix,numeric-method
(kroncker-methods), 109
kroncker,vector,Matrix-method Logic,DMatrix,sparseVector-method
(kroncker-methods), 109
kroncker-methods, 109
ldenseMatrix, 51, 115, 116
ldenseMatrix-class, 110
ldiMatrix, 15, 59
ldiMatrix-class, 111
length, 57, 63, 136, 183
length,abIndex-method (abIndex-class), 5
length,Matrix-method (Matrix-class), 124
length,MatrixFactorization-method (MatrixFactorization-class), 128
length,sparseVector-method (sparseVector-class), 183
levelplot, 52, 54, 55, 94, 95, 125
lgCMatrix, 112
lgCMatrix-class (lsparseMatrix-class), 113
lgeMatrix, 110, 115, 116
lgeMatrix-class, 112
lgRMatrix-class (lsparseMatrix-class), 113
lgTMatrix-class (lsparseMatrix-class), 113
list, 13, 44, 58, 63, 64, 95, 119, 125, 159, 169
lMatrix, 15, 110, 111, 114, 116, 121, 123, 129, 135, 139
lMatrix-class (dMatrix-class), 62
log,denseMatrix-method (denseMatrix-class), 51
log,diagonalMatrix-method (diagonalMatrix-class), 58
log,indMatrix-method (indMatrix-class), 97
log,sparseMatrix-method (sparseMatrix-class), 176
log,sparseVector-method (sparseVector-class), 183

Logic,ANY,Matrix-method (Matrix-class), 124
Logic,CSparseMatrix,CSparseMatrix-method (kroncker-methods), 109
Logic,DMatrix,logical-method (dMatrix-class), 62
Logic,DMatrix,numeric-method (dMatrix-class), 62
Logic,DMatrix,sparseVector-method (dMatrix-class), 62
Logic,ldenseMatrix,lsparseMatrix-method (ldenseMatrix-class), 110
Logic,lgeMatrix,lgeMatrix-method (lgeMatrix-class), 112
Logic,lgTMatrix,lgTMatrix-method (lsparseMatrix-class), 113
Logic,lMatrix,logical-method (dMatrix-class), 62
Logic,lMatrix,numeric-method (dMatrix-class), 62
Logic,lMatrix,sparseVector-method (dMatrix-class), 62
Logic,logical,dMatrix-method (dMatrix-class), 62
Logic,logical,lMatrix-method (dMatrix-class), 62
Logic,logical,nMatrix-method (nMatrix-class), 134
Logic,lsCMatrix,lsCMatrix-method (lsparseMatrix-class), 113
Logic,lsparseMatrix,ldenseMatrix-method (lsparseMatrix-class), 113
Logic,lsparseMatrix,lsparseMatrix-method (lsparseMatrix-class), 113
Logic,lsparseVector,lsparseVector-method (sparseVector-class), 183
Logic,ltCMatrix,ltCMatrix-method (lsparseMatrix-class), 113
Logic,Matrix,ANY-method (Matrix-class), 124
Logic,Matrix,nMatrix-method (Matrix-class), 124
Logic,ngeMatrix,ngeMatrix-method (ngeMatrix-class), 133
Logic,nMatrix,logical-method (nMatrix-class), 134
Logic, nMatrix, Matrix-method
  (nMatrix-class), 134
Logic, nMatrix, nMatrix-method
  (nMatrix-class), 134
Logic, nMatrix, numeric-method
  (nMatrix-class), 134
Logic, nMatrix, sparseVector-method
  (nMatrix-class), 134
Logic, numeric, dMatrix-method
  (dMatrix-class), 62
Logic, numeric, lMatrix-method
  (dMatrix-class), 62
Logic, numeric, nMatrix-method
  (nMatrix-class), 134
Logic, sparseVector, dMatrix-method
  (sparseVector-class), 183
Logic, sparseVector, lMatrix-method
  (sparseVector-class), 183
Logic, sparseVector, nMatrix-method
  (sparseVector-class), 183
Logic, sparseVector, sparseVector-method
  (sparseVector-class), 183
Logic, triangularMatrix, diagonalMatrix-method
  (triangularMatrix-class), 192
logical, 57, 63, 66, 96–98, 102, 104, 105,
  112, 119, 121, 123, 127, 135, 154,
  184, 188
lsCMatrix, 190
lsCMatrix-class (lsparseMatrix-class), 113
lsCMatrix-class (lsparseMatrix-class), 113
lsCMatrix-class
  (lsparseMatrix-class), 113
lsparseVector-class
  (sparseVector-class), 183
lsCMatrix-class (lsyMatrix-class), 115
lsCMatrix-class (lsyMatrix-class), 113
lsTMatrix-class (lsyMatrix-class), 113
lsyMatrix, 112
lsyMatrix-class, 115
ltCMatrix, 193
ltCMatrix-class (ltrMatrix-class), 113
ltCMatrix-class (ltrMatrix-class), 116
ltCMatrix-class (ltpMatrix-class), 116
ltrMatrix, 112, 195
ltrMatrix-class (lsparseMatrix-class), 113
ltrMatrix-class, 116
ltrMatrix-class (lsparseMatrix-class), 113
LU, 20, 36, 44, 49, 50, 52, 81, 129, 150, 164,
  167, 171, 172, 177
lu (lu-methods), 117
lu, denseMatrix-method (lu-methods), 117
lu, dgCMatrix-method (lu-methods), 117
lu, dgeMatrix-method (lu-methods), 117
lu, dgRMatrix-method (lu-methods), 117
lu, dGMatrix-method (lu-methods), 117
lu, diagonalMatrix-method (lu-methods), 117
lu, dsCMatrix-method (lu-methods), 117
lu, dspMatrix-method (lu-methods), 117
lu, dsRMatrix-method (lu-methods), 117
lu, dsTMatrix-method (lu-methods), 117
lu, dsyMatrix-method (lu-methods), 117
lu, dtCMatrix-method (lu-methods), 117
lu, dtRMatrix-method (lu-methods), 117
lu, dtRMatrix-method (lu-methods), 117
lu, dtTMatrix-method (lu-methods), 117
lu, matrix-method (lu-methods), 117
lu, sparseMatrix-method (lu-methods), 117
LU-class (MatrixFactorization-class), 128
lu-methods, 117
mat2triplet, 119
Math, denseMatrix-method
  (denseMatrix-class), 51
Math, diagonalMatrix-method
  (diagonalMatrix-class), 58
Math, indMatrix-method
  (indMatrix-class), 97
Math, sparseMatrix-method
  (sparseMatrix-class), 176
Math, sparseVector-method
  (sparseVector-class), 183
Math2, Matrix-method (Matrix-class), 124
Math2, sparseVector-method
  (sparseVector-class), 183
matmult-methods, 120
INDEX

	nnzero, 67, 77, 160

Matrix-class, 124

Matrix-notyet, 126

matrix, 10, 15, 59, 90, 91, 98, 102–105, 119, 123, 125, 130, 136, 188

Matrix-class, 124

Matrix-namespace, 126

Matrix.csr, 42

Matrix.Version (Matrix-class), 124

MatrixFactorization, 16, 17, 23, 24, 31, 32, 49, 79, 81, 85, 93, 162, 166, 167, 171, 179, 190

MatrixFactorization-class, 128

max, 62

mean, denseMatrix-method
  (denseMatrix-class), 51

mean, sparseMatrix-method
  (sparseMatrix-class), 176

mean, sparseVector-method
  (sparseVector-class), 183

method, 103

min, 62

model.frame, 169

model.matrix, 169, 170

NA, 59, 63, 102, 104, 105, 109, 113, 134, 135, 177, 184, 188, 191

NA.logical, 154

names, 42, 57

NaN, 102, 154

nCHMsimpl-class (CHMfactor-class), 22

nCHMsuper-class (CHMfactor-class), 22

ncol, 192

ndenseMatrix, 51

ndenseMatrix-class, 129

ndiMatrix-class (ldiMatrix-class), 111

nearcor, 132

nearPD, 130

new, 24, 47, 173

ngCMatrix, 134

ngCMatrix-class (nsparseMatrix-class), 138

ngeMatrix, 130, 140, 141

ngeMatrix-class, 133

ngRMatrix-class (nsparseMatrix-class), 138

ngTMatrix, 40

ngTMatrix-class (nsparseMatrix-class), 138

nMatrix, 11, 15, 63, 87, 98, 102, 119, 121, 133

nMatrix-class, 134

nnzero, 67, 77, 160

nnzero (nnzero-methods), 135

nnzero, ANY-method (nnzero-methods), 135

nnzero, CHMfactor-method
  (nnzero-methods), 135

nnzero, denseMatrix-method
  (nnzero-methods), 135

nnzero, diagonalMatrix-method
  (nnzero-methods), 135

nnzero, indMatrix-method
  (nnzero-methods), 135

nnzero, sparseMatrix-method
  (nnzero-methods), 135

nnzero, vector-method (nnzero-methods), 135

nnzero-methods, 135

norm, 44, 45, 72, 73, 131, 137, 156, 157, 177

norm (norm-methods), 137

norm, ANY, missing-method (norm-methods), 137

norm, denseMatrix, character-method
  (norm-methods), 137

norm, diagonalMatrix, character-method
  (norm-methods), 137

norm, indMatrix, character-method
  (norm-methods), 137

norm, pMatrix, character-method
  (norm-methods), 137

norm, sparseMatrix, character-method
  (norm-methods), 137

norm-methods, 137

nrow, 192

nsCMatrix-class (nsparseMatrix-class), 138

nsparseMatrix, 15, 57, 97, 119, 121, 134, 135, 159, 160, 173, 177

nsparseMatrix-class, 138

nsparseMatrix-classes
  (nsparseMatrix-class), 138

nsparseVector, 87, 102
index

nsparseVector-class
  (sparseVector-class), 183
nspMatrix-class (nsyMatrix-class), 140
nsRMatrix-class (nsparseMatrix-class), 138
ntTMatrix, 40
ntTMatrix-class (nsparseMatrix-class), 138
nsyMatrix-class, 134
ntCMatrix-class (nsparseMatrix-class), 138
ntpMatrix-class (ntrMatrix-class), 141
ntrMatrix, 134
ntrMatrix-class (nsparseMatrix-class), 138
ntrMatrix-class, 141
ntTMatrix-class (nsparseMatrix-class), 138
NULL, 83, 92, 102, 123, 147, 160, 173
numeric, 5, 6, 42, 55, 65, 70, 96, 108, 135
onenormest, 137
onenormest (condest), 43
Ops, 5, 59, 78, 184
Ops, abIndex, abIndex-method
  (abIndex-class), 5
Ops, abIndex, ANY-method (abIndex-class), 5
Ops, ANY, abIndex-method (abIndex-class), 5
Ops, ANY, ddiMatrix-method
  (ddiMatrix-class), 48
Ops, ANY, ldiMatrix-method
  (ldiMatrix-class), 111
Ops, ANY, Matrix-method (Matrix-class), 124
Ops, ANY, sparseVector-method
  (sparseVector-class), 183
Ops, ddiMatrix, ANY-method
  (ddiMatrix-class), 48
Ops, ddiMatrix, ddiMatrix-method
  (ddiMatrix-class), 48
Ops, ddiMatrix, dMatrix-method
  (ddiMatrix-class), 48
Ops, ddiMatrix, ldiMatrix-method
  (ldiMatrix-class), 48
Ops, ddiMatrix, logical-method
  (ddiMatrix-class), 48
Ops, ddiMatrix, Matrix-method
  (ddiMatrix-class), 48
Ops, ddiMatrix, ndiMatrix-method
  (ldiMatrix-class), 111
Ops, ddiMatrix, nMatrix-method
  (ldiMatrix-class), 111
Ops, ddiMatrix, triangularMatrix-method
  (ddiMatrix-class), 58
Ops, dMatrix, ddiMatrix-method
  (dMatrix-class), 62
Ops, dMatrix, dMatrix-method
  (dMatrix-class), 62
Ops, dMatrix, ldiMatrix-method
  (dMatrix-class), 62
Ops, dMatrix, nMatrix-method
  (dMatrix-class), 62
Ops, dpoMatrix, logical-method
  (dpoMatrix-class), 65
Ops, dpoMatrix, numeric-method
  (dpoMatrix-class), 65
Ops, dppMatrix, logical-method
  (dpoMatrix-class), 65
Ops, dppMatrix, numeric-method
  (dpoMatrix-class), 65
Ops, dsparseMatrix, nsparseMatrix-method
  (dsparseMatrix-class), 70
Ops, ldenseMatrix, ldenseMatrix-method
  (ldenseMatrix-class), 110
Ops, ldiMatrix, ANY-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, ddiMatrix-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, Matrix-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, nMatrix-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, logical-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, Matrix-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, ndiMatrix-method
  (ldiMatrix-class), 111
Ops, ldiMatrix, numeric-method
  (ldiMatrix-class), 111
INDEX

Ops, ldiMatrix, sparseMatrix-method (ldiMatrix-class), 111
Ops, lMatrix, dMatrix-method (dMatrix-class), 62
Ops, lMatrix, lMatrix-method (lMatrix-class), 62
Ops, lMatrix, numeric-method (Matrix-class), 124
Ops, lMatrix, numeric-method (Matrix-class), 124
Ops, matrix, numeric-method (Matrix-class), 124
Ops, matrix, numeric-method (Matrix-class), 124
Ops, Matrix, ANY-method (Matrix-class), 124
Ops, Matrix, ddiMatrix-method (Matrix-class), 124
Ops, Matrix, ldiMatrix-method (Matrix-class), 124
Ops, Matrix, matrix-method (Matrix-class), 124
Ops, Matrix, Matrix-method (Matrix-class), 124
Ops, Matrix, NULL-method (Matrix-class), 124
Ops, Matrix, sparseVector-method (Matrix-class), 124
Ops, ndenseMatrix, ndenseMatrix-method (ndenseMatrix-class), 129
Ops, ndMatrix, dMatrix-method (ldiMatrix-class), 111
Ops, ndMatrix, ldiMatrix-method (ldiMatrix-class), 111
Ops, ndMatrix, ndMatrix-method (Matrix-class), 111
Ops, nMatrix, dMatrix-method (nMatrix-class), 134
Ops, nMatrix, lMatrix-method (nMatrix-class), 134
Ops, nMatrix, nMatrix-method (nMatrix-class), 134
Ops, nMatrix, numeric-method (nMatrix-class), 134

Ops, nMatrix, dMatrix-method (nMatrix-class), 134
Ops, nMatrix, lMatrix-method (nMatrix-class), 134
Ops, nMatrix, numeric-method (nMatrix-class), 134

Ops, numeric, nMatrix-method (nMatrix-class), 134

Ops, numeric, sparseMatrix-method (sparseMatrix-class), 176
Ops, numeric, sparseMatrix-method (sparseMatrix-class), 176
Ops, numeric, sparseMatrix-method (sparseMatrix-class), 176
Ops, numeric, sparseMatrix-method (sparseMatrix-class), 176
Ops, sparseMatrix, ddiMatrix-method (sparseMatrix-class), 176
Ops, sparseMatrix, ddiMatrix-method (sparseMatrix-class), 176
Ops, sparseMatrix, ldiMatrix-method (sparseMatrix-class), 176
Ops, sparseMatrix, ldiMatrix-method (sparseMatrix-class), 176
Ops, sparseMatrix, numeric-method (sparseMatrix-class), 176
Ops, sparseMatrix, numeric-method (sparseMatrix-class), 176

Ops, sparseMatrix, sparseMatrix-method (sparseMatrix-class), 176
Ops, sparseVector, ANY-method (sparseVector-class), 183
Ops, sparseVector, Matrix-method (sparseVector-class), 183
Ops, sparseVector, sparseVector-method (sparseVector-class), 183
Ops, sparseVector, vector-method (sparseVector-class), 183
Ops, vector, sparseVector-method (sparseVector-class), 183

options, 24, 147, 148, 184
order, 100, 119
outer, 125

pack, 72, 78, 144, 195
pack (pack-methods), 142
pack, dgeMatrix-method (pack-methods), 142
pack, lgeMatrix-method (pack-methods), 142
pack, matrix-method (pack-methods), 142
pack, ngeMatrix-method (pack-methods), 142
pack, packedMatrix-method (pack-methods), 142
pack, sparseMatrix-method (pack-methods), 142
pack, unpackedMatrix-method (pack-methods), 142
pack-methods, 142
packedMatrix, 19, 36, 51, 87, 88, 142, 195
packedMatrix-class, 143
panel.levelplot.raster, 94
paste, 169
pBunchKaufman, 19, 20, 80, 129
pBunchKaufman-class (BunchKaufman-class), 16
pCholesky, 36, 80, 129
pCholesky-class (Cholesky-class), 31
plot.default, 94
pMatrix, 64, 80, 81, 97, 98, 100
pMatrix-class, 145
posdefify, 130–132
print, 59, 92, 94, 125, 147, 148, 177
print, diagonalMatrix-method (diagonalMatrix-class), 58
print, sparseMatrix-method (sparseMatrix-class), 176
print.default, 92, 147
print.sparseMatrix (sparseMatrix-class), 176
print.trellis, 95
printSpMatrix, 92, 125, 147, 177
printSpMatrix2 (printSpMatrix), 147
prod, 62
QR, 150, 179
qr, 20, 36, 81, 118, 129, 150, 153, 154, 164, 167, 179, 180
qr (qr-methods), 149
qr, dgCMatrix-method (qr-methods), 149
qr, sparseMatrix-method (qr-methods), 149
QR-class (MatrixFactorization-class), 128
qr-methods, 149
qr.coef, 167, 181
qr.coef, sparseQR, dgeMatrix-method (sparseQR-class), 178
qr.coef, sparseQR, Matrix-method (sparseQR-class), 178
qr.default, 149
qr.fitted, 181
qr.fitted, sparseQR, dgeMatrix-method (sparseQR-class), 178
qr.fitted, sparseQR, Matrix-method (sparseQR-class), 178
qr.fitted, sparseQR, matrix-method (sparseQR-class), 178
qr.fitted, sparseQR, vector-method (sparseQR-class), 178
qr.Q, 181
qr.Q, sparseQR-method (sparseQR-class), 178
qr.qty, 181
qr.qty, sparseQR, dgeMatrix-method (sparseQR-class), 178
qr.qty, sparseQR, Matrix-method (sparseQR-class), 178
qr.qty, sparseQR, matrix-method (sparseQR-class), 178
qr.qty, sparseQR, vector-method (sparseQR-class), 178
qr.qy, 181
qr.qy, sparseQR, dgeMatrix-method (sparseQR-class), 178
qr.qy, sparseQR, Matrix-method (sparseQR-class), 178
qr.qy, sparseQR, matrix-method (sparseQR-class), 178
qr.qy, sparseQR, vector-method (sparseQR-class), 178
qr.R, 181
qr.R, sparseQR-method (sparseQR-class), 178
qr.resid, 181
qr.resid, sparseQR, dgeMatrix-method (sparseQR-class), 178
qr.resid, sparseQR, Matrix-method (sparseQR-class), 178
qr.resid, sparseQR, matrix-method (sparseQR-class), 178
qr.resid, sparseQR, vector-method (sparseQR-class), 178
qr.solve, 181
qr.X, 181
qr.X, sparseQR-method (sparseQR-class), 178
qr2rankMatrix (rankMatrix), 152
qrR (sparseQR-class), 178
range, 62
rankMatrix, 152
rbind, 21, 177
rbind2, 21
rbind2 (cbind2-methods), 21
rbind2, Matrix, Matrix-method (cbind2-methods), 21
rbind2, Matrix, matrix-method (cbind2-methods), 21
rbind2, matrix, Matrix-method (cbind2-methods), 21
rbind2, Matrix, missing-method (cbind2-methods), 21
rbind2, Matrix, NULL-method (cbind2-methods), 21
rbind2, Matrix, vector-method (cbind2-methods), 21
rbind2, NULL, Matrix-method (cbind2-methods), 21
rbind2, vector, Matrix-method (cbind2-methods), 21
rbind2-methods (cbind2-methods), 21
rcond, 45, 54, 66, 72, 73, 166
rcond (rcond-methods), 155
rcond, ANY, missing-method (rcond-methods), 155
rcond, denseMatrix, character-method (rcond-methods), 155
rcond, diagonalMatrix, character-method (rcond-methods), 155
rcond, indMatrix, character-method (rcond-methods), 155
rcond, pMatrix, character-method (rcond-methods), 155
rcond, sparseMatrix, character-method (rcond-methods), 155
rcond-methods, 155
readHB (externalFormats), 83
readMM (externalFormats), 83
rep, denseMatrix-method (denseMatrix-class), 51
rep, sparseMatrix-method (sparseMatrix-class), 176
rep, sparseVector-method (sparseVector-class), 183
rep.int, 158
rep2abI, 7, 158
rle, 5–7, 158, 159
rleDiff, 5
rleDiff-class, 158
round, 62, 125, 184
rowMeans, 125
rowMeans (colSums-methods), 42
rowMeans, CsparseMatrix-method (colSums-methods), 42
rowMeans, denseMatrix-method (colSums-methods), 42
rowMeans, diagonalMatrix-method (colSums-methods), 42
rowMeans, indMatrix-method (colSums-methods), 42
rowMeans, RsparseMatrix-method (colSums-methods), 42
rowMeans, TsparseMatrix-method (colSums-methods), 42
rowMeans-methods (colSums-methods), 42
rowScale (dimScale), 61
rowSums (colSums-methods), 42
rowSums, CsparseMatrix-method (colSums-methods), 42
rowSums, denseMatrix-method (colSums-methods), 42
rowSums, diagonalMatrix-method (colSums-methods), 42
rowSums, indMatrix-method (colSums-methods), 42
rowSums, RsparseMatrix-method (colSums-methods), 42
rowSums, TsparseMatrix-method (colSums-methods), 42
rowSums-methods (colSums-methods), 42
rsparseMatrix, 159, 175
RsparseMatrix-class, 161
sample.int, 160
Schur, 20, 36, 64, 80, 81, 83, 118, 129, 150, 162–164, 167, 193
Schur (Schur-methods), 163
Schur, dgeMatrix-method (Schur-methods), 163
solve,pCholesky,dgeMatrix-method
   (solve-methods), 165
solve,pCholesky,missing-method
   (solve-methods), 165
solve,pMatrix,Matrix-method
   (solve-methods), 165
solve,pMatrix,matrix-method
   (solve-methods), 165
solve,pMatrix,missing-method
   (solve-methods), 165
solve,pMatrix,vector-method
   (solve-methods), 165
solve,RsparseMatrix,ANY-method
   (solve-methods), 165
solve,Schur,ANY-method
   (solve-methods), 165
solve,sparseLU,dgCMatrix-method
   (solve-methods), 165
solve,sparseLU,dgeMatrix-method
   (solve-methods), 165
solve,sparseLU,missing-method
   (solve-methods), 165
solve,sparseQR,dgCMatrix-method
   (solve-methods), 165
solve,sparseQR,dgeMatrix-method
   (solve-methods), 165
solve,sparseQR,missing-method
   (solve-methods), 165
solve,triangularMatrix,CsparseMatrix-method
   (solve-methods), 165
solve,triangularMatrix,denseMatrix-method
   (solve-methods), 165
solve,triangularMatrix,dgCMatrix-method
   (solve-methods), 165
solve,triangularMatrix,dgeMatrix-method
   (solve-methods), 165
solve,triangularMatrix,diagonalMatrix-method
   (solve-methods), 165
solve,triangularMatrix,indMatrix-method
   (solve-methods), 165
solve,triangularMatrix,matrix-method
   (solve-methods), 165
solve,triangularMatrix,RsparseMatrix-method
   (solve-methods), 165
solve,triangularMatrix,TsparseMatrix-method
   (solve-methods), 165
solve,triangularMatrix,vector-method
   (solve-methods), 165
solve,TsparseMatrix,ANY-method
   (solve-methods), 165
solve-methods, 165
sort,sparseVector-method
   (sparseVector-class), 183
sort.list, 100
sparse.model.matrix, 168, 175, 177
sparseLU, 50, 80, 118, 129, 166
sparseLU-class, 171
SparseM.ontology, 42
sparseMatrix, 10, 12, 22, 24, 27, 34, 36, 42,
   46–48, 51, 52, 55, 58, 67, 68, 83, 87,
   91, 98, 106, 111, 117, 118, 120, 123,
   134, 147, 149, 150, 160, 161, 166,
   169–171, 173, 177, 180, 183, 184,
   186, 187, 194
sparseMatrix-class, 176
sparseQR, 80, 129, 150, 154, 167, 179
sparseQR-class, 178
sparseVector, 7, 15, 42, 43, 88, 101, 120,
   166, 182, 182, 183, 184
sparseVector-class, 183
spMatrix, 55, 120, 149, 186, 194
stop, 188
str, 128
subassign-methods, 188
substring-methods, 189
substring, 148
sum, 62
Summary, 184
summary, 120
Summary,abIndex-method (abIndex-class), 5
Summary,denseMatrix-method
   (denseMatrix-class), 51
Summary,diagonalMatrix-method
   (diagonalMatrix-class), 58
summary,diagonalMatrix-method
   (diagonalMatrix-class), 58
Summary,indMatrix-method
   (indMatrix-class), 97
Summary,sparseMatrix-method
   (sparseMatrix-class), 176
summary,sparseMatrix-method
   (sparseMatrix-class), 176
Summary,sparseVector-method
   (sparseVector-class), 183
svd, 125, 137, 153, 154
INDEX

symmetricMatrix, 11, 30, 57, 61, 69, 71, 88, 90, 93, 105, 115, 118, 120, 121, 123, 136, 143, 160, 192, 193

symmetricMatrix-class, 190

symmpart, 90, 105, 130, 131

symmpart(symmpart-methods), 191

symmpart,CsparseMatrix-method (symmpart-methods), 191

symmpart,denseMatrix-method (symmpart-methods), 191

symmpart,diagonalMatrix-method (symmpart-methods), 191

symmpart,indMatrix-method (symmpart-methods), 191

symmpart,matrix-method (symmpart-methods), 191

symmpart,RsparseMatrix-method (symmpart-methods), 191

symmpart-methods, 191

t, 73, 112, 115, 116, 120, 134, 140, 141, 153, 169

t,CsparseMatrix-method (CsparseMatrix-class), 45

t,denseMatrix-method (denseMatrix-class), 51

t,diagonalMatrix-method (diagonalMatrix-class), 58

t,indMatrix-method (indMatrix-class), 97

t,pMatrix-method (pMatrix-class), 145

t,RsparseMatrix-method (RsparseMatrix-class), 161

t,sparseVector-method (sparseVector-class), 183

t,TsparseMatrix-method (TsparseMatrix-class), 193

T2graph, 177

T2graph(coerce-methods-graph), 40
tail,Matrix-method (Matrix-class), 124
tail,sparseVector-method (sparseVector-class), 183
tcrossprod, 16, 25, 120, 122, 196
tcrossprod (matmult-methods), 120
tcrossprod,ANY,Matrix-method (matmult-methods), 120
tcrossprod,ANY,sparseVector-method (matmult-methods), 120
tcrossprod,CsparseMatrix,CsparseMatrix-method (matmult-methods), 120
tcrossprod,CsparseMatrix,denseMatrix-method (matmult-methods), 120
tcrossprod,CsparseMatrix,diagonalMatrix-method (matmult-methods), 120
tcrossprod,CsparseMatrix,matrix-method (matmult-methods), 120
tcrossprod,CsparseMatrix,missing-method (matmult-methods), 120
tcrossprod,CsparseMatrix,RsparseMatrix-method (matmult-methods), 120
tcrossprod,CsparseMatrix,TsparseMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,CsparseMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,denseMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,diagonalMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,matrix-method (matmult-methods), 120
tcrossprod,denseMatrix,missing-method (matmult-methods), 120
tcrossprod,denseMatrix,RsparseMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,TsparseMatrix-method (matmult-methods), 120
tcrossprod,denseMatrix,vector-method (matmult-methods), 120
tcrossprod,diagonalMatrix,CsparseMatrix-method (matmult-methods), 120
tcrossprod,diagonalMatrix,diagonalMatrix-method (matmult-methods), 120
tcrossprod,diagonalMatrix,matrix-method (matmult-methods), 120
tcrossprod,diagonalMatrix,missing-method (matmult-methods), 120
tcrossprod,diagonalMatrix,RsparseMatrix-method (matmult-methods), 120
tcrossprod,diagonalMatrix,TsparseMatrix-method (matmult-methods), 120
tcrossprod,diagonalMatrix,vector-method (matmult-methods), 120
<table>
<thead>
<tr>
<th>Function</th>
<th>Documentation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcrossprod.indMatrix,Matrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,RsparseMatrix,RsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod.indMatrix,matrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,RsparseMatrix,TsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod.indMatrix,missing-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,RsparseMatrix,vector-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod.indMatrix,vector-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,sparseVector,ANY-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,Matrix,ANY-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,sparseVector,Matrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,Matrix,indMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,sparseVector,matrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,Matrix,RsparseMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,sparseVector,missing-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,Matrix,sparseVector-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,sparseVector,sparseVector-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,matrix,CsparseMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,diagonalMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,matrix,denseMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,matrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,matrix,diagonalMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,missing-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,matrix,pMatrix,Matrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,RsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,matrix,pMatrix,matrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,TsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,pMatrix,missing-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,TsparseMatrix,vector-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,pMatrix,pMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,CsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,pMatrix,vector-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,denseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,RsparseMatrix,CsparseMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,diagonalMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,RsparseMatrix,denseMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,indMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,RsparseMatrix,diagonalMatrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,RsparseMatrix-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,RsparseMatrix,matrix-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,sparseVector-method (matmult-methods), 120</td>
</tr>
<tr>
<td>tcrossprod,RsparseMatrix,missing-method</td>
<td>(matmult-methods), 120</td>
<td>tcrossprod,vector,TsparseMatrix-method (matmult-methods), 120</td>
</tr>
</tbody>
</table>
INDEX 239
tcrossprod-methods (matmult-methods), 120
toeplitz, 184
toeplitz, sparseVector-method (sparseVector-class), 183
triangularMatrix, 10, 28, 35, 57, 60, 61, 74, 
75, 77–79, 88, 90, 93, 106, 116, 121, 
123, 141, 143, 190
triangularMatrix-class, 192
tril (band-methods), 9
tril, CsparseMatrix-method (band-methods), 9
tril, denseMatrix-method (band-methods), 9
tril, diagonalMatrix-method (band-methods), 9
tril, indMatrix-method (band-methods), 9
tril, matrix-method (band-methods), 9
tril, RsparseMatrix-method (band-methods), 9
tril, TsparseMatrix-method (band-methods), 9
tril-methods (band-methods), 9
triu (band-methods), 9
triu, CsparseMatrix-method (band-methods), 9
triu, denseMatrix-method (band-methods), 9
triu, diagonalMatrix-method (band-methods), 9
triu, indMatrix-method (band-methods), 9
triu, matrix-method (band-methods), 9
triu, RsparseMatrix-method (band-methods), 9
triu, TsparseMatrix-method (band-methods), 9
triu-methods (band-methods), 9
TRUE, 63, 102
tryCatch, 118
TsparseMatrix, 8, 9, 12, 13, 40, 55, 67, 69, 
88, 113, 119, 138, 169, 173, 174, 
186, 187, 194
TsparseMatrix-class, 193
type, 188
typeof, 57

uniqTsparse (asUniqueT), 8
unnamed, Matrix-method (Matrix-class), 124
unnamed, MatrixFactorization-method 
(MatrixFactorization-class), 128
unpack, 144, 195
unpack (pack-methods), 142
unpack, matrix-method (pack-methods), 142
unpack, packedMatrix-method 
(pack-methods), 142
unpack, sparseMatrix-method 
(pack-methods), 142
unpack, unpackedMatrix-method 
(pack-methods), 142
unpack-methods (pack-methods), 142
unpackedMatrix, 36, 51, 87, 88, 142, 144
unpackedMatrix-class, 194
update, 196
update, CHMfactor-method 
(CHMfactor-class), 22
updown, 25
updown (updown-methods), 195
updown, character, ANY, ANY-method 
(updown-methods), 195
updown, logical, dgCMatrix, CHMfactor-method 
(updown-methods), 195
updown, logical, dsCMatrix, CHMfactor-method 
(updown-methods), 195
updown, logical, dtCMatrix, CHMfactor-method 
(updown-methods), 195
updown, logical, Matrix, CHMfactor-method 
(updown-methods), 195
updown-methods, 195
USCounties, 196

validObject, 47
warning, 19, 117
which, 59, 63, 110, 114, 129, 139
which, indMatrix-method 
(indMatrix-class), 97
which, ldenseMatrix-method 
(ldenseMatrix-class), 110
which, ldiMatrix-method 
(ldiMatrix-class), 111
which, lsparseMatrix-method 
(lsparseMatrix-class), 113
which, lsparseVector-method 
(sparseVector-class), 183
which,ndenseMatrix-method
  (ndenseMatrix-class), 129
which,ndiMatrix-method
  (ldiMatrix-class), 111
which,nsparseMatrix-method
  (nsparseMatrix-class), 138
which,nsparseVector-method
  (sparseVector-class), 183
writeMM, 177
writeMM (externalFormats), 83
writeMM, CsparseMatrix-method
  (externalFormats), 83
writeMM, sparseMatrix-method
  (externalFormats), 83
wrld_1deg, 198

xtabs, 175, 177

zapsmall, 67, 136
zapsmall, Matrix-method (Matrix-class), 124
zapsmall, sparseVector-method
  (sparseVector-class), 183
zMatrix-class (Matrix-notyet), 126
zsparseVector-class
  (sparseVector-class), 183