

Package ‘spam’

January 12, 2012

Version 0.28-0

Date 2012-01-11

Author Reinhard Furrer

Maintainer Reinhard Furrer <reinhard.furrer@math.uzh.ch>

Depends R (>= 2.10), methods

Suggests fields, SparseM (>= 0.72), Matrix

Description Set of function for sparse matrix algebra. Differences with SparseM/Matrix are: (1) we only support (essentially) one sparse matrix format, (2) based on transparent and simple structure(s), (3) tailored for MCMC calculations within GMRF. (4) S3 and S4 like-“compatible” ... and it is fast.

LazyLoad Yes

LazyData Yes

License GPL | file LICENSE

Title SPArse Matrix

URL <http://www.math.uzh.ch/furrer/software/spam/>

Repository CRAN

Date/Publication 2012-01-12 09:24:51

R topics documented:

. SPAM	3
allequal	4
apply	6
bandwidth	7
bdiag	8
cbind	9

chol	11
circulant	13
cleanup	14
complexity	15
det	15
diag	17
dim	18
display	19
foreign	20
germany	22
image	23
import	24
isSymmetric	25
kronecker	26
lower.tri	27
makeprec	28
Math	29
Math2	30
mle	31
nearestdist	34
options	36
Oral	38
ordering	39
permutation	40
powerboost	41
print	42
rmvnorm	43
spam	45
spam operations	46
spam solve	47
spam-class	50
spam.chol.NgPeyton-class	52
Summary	53
toeplitz	54
triplet	55
UScounties	56
USprecip	56
version	57

Description

`spam` is a collection of functions for sparse matrix algebra.

General overview

What is `spam` and what is it not:

While `Matrix` seems an overshoot of classes and `SparseM` focuses mainly on regression type problem, we provide a minimal set of sparse matrix functions fully functional for everyday spatial statistics life. There is however some emphasize on Markov chain Monte Carlo type calculations within the framework of (Gaussian) Markov random fields.

Emphasis is given on a comprehensive, simple, tutorial structure of the code. The code is S4 based but (in a tutorial spirit) the functions are in a S3 structure visible to the user (exported via `NAMESPACE`).

There exist many methods for sparse matrices that work identically as in the case of ordinary matrices. All the methods are discussed in the help and can be accessed directly via a `*.spam` concatenation to the function. For example, `help{chol.spam}` calls the help directly, whereas with `help{chol}` the user has to choose first between the basis help and the help provided by `spam`.

Sparseness is used when handling large matrices. Hence, care has been used to provide efficient and fast routines. Essentially, the functions do not transform the sparse structure into full matrices to use standard (available) functionality, followed by a back transform. We agree, more operators, functions, etc. should eventually be implemented.

The packages `fields` and `spam` are closely linked.

Author(s)

Reinhard Furrer

References

Reinhard Furrer, Stephan R. Sain (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields.", *Journal of Statistical Software*, 36(10), 1-25, <http://www.jstatsoft.org/v36/i10/>.

See Also

See [spam.class](#) for a detailed class description, [spam](#) and [spam.ops](#) for creation, coercion and algebraic operations.

`demo(package='spam')` lists available demos.

Related packages are [fields](#), [Matrix](#) and [SparseM.ontology](#).

Examples

```
## Not run:
## History of changes
file.show(system.file("NEWS", package = "spam"))

## End(Not run)
```

allequal

Test if Two 'spam' Objects are (Nearly) Equal

Description

Utility to compare two spam objects testing 'near equality'. Depending on the type of difference, comparison is still made to some extent, and a report of the differences is returned.

Usage

```
## S3 method for class 'spam'
all.equal(target, current, tolerance = .Machine$double.eps^0.5,
          scale = NULL, check.attributes = FALSE,...)
```

Arguments

<code>target</code>	a spam object.
<code>current</code>	another spam object to be compared with <code>target</code> .
<code>tolerance</code>	numeric ≥ 0 . Differences smaller than <code>tolerance</code> are not considered.
<code>scale</code>	numeric scalar > 0 (or <code>NULL</code>). See 'Details'.
<code>check.attributes</code>	currently not yet implemented.
<code>...</code>	Further arguments for different methods.

Details

Numerical comparisons for `scale = NULL` (the default) are done by first computing the mean absolute difference of the two numerical vectors. If this is smaller than `tolerance` or not finite, absolute differences are used, otherwise relative differences scaled by the mean absolute difference.

If `scale` is positive, absolute comparisons are made after scaling (dividing) by `scale`.

Do not use `all.equal.spam` directly in `if` expressions: either use `isTRUE(all.equal.spam(...))` or `identical` if appropriate.

Cholesky decomposition routines use this function to test for symmetry.

A method for `matrix-spam` objects is defined as well.

There is the additional catch of a zero matrix being represented by one zero element, see ‘Examples’ below.

Value

Either `TRUE` or a vector of ‘mode’ “character” describing the differences between target and current.

Author(s)

Reinhard Furrer

See Also

[isSymmetric.spam](#) and [cleanup](#).

Examples

```
obj <- diag.spam(2)
obj[1,2] <- .Machine$double.eps

all.equal( diag.spam(2), obj)

all.equal( t(obj), obj)

all.equal( t(obj), obj*1.1)

# We can compare a spam to a matrix
all.equal(diag(2),diag.spam(2))

# the opposite does often not make sense,
# hence, it is not implemented.
all.equal(diag.spam(2),diag(2))

# A zero matrix contains one element:
str(spam(0))
# hence
all.equal.spam(spam(0,3,3), diag.spam(0,3) )
norm(spam(0,3,3) - diag.spam(0,3) )
```

 apply

Apply Functions Over Sparse Matrix Margins

Description

Returns a vector or array or list of values obtained by applying a function to margins of a sparse matrix.

Usage

```
apply.spam(X, MARGIN=NULL, FUN, ...)
```

Arguments

X	the spam matrix to be used.
MARGIN	a vector giving the subscripts which the function will be applied over. 1 indicates rows, 2 indicates columns, NULL or c(1, 2) indicates rows and columns.
FUN	the function to be applied.
...	optional arguments to FUN.

Details

This is a handy wrapper to apply a function to the (nonzero) elements of a sparse matrix. For example, it is possible to apply a covariance matrix to a distance matrix obtained by `nearest.dist`, see Examples.

A call to `apply` only coerces the sparse matrix to a regular one.

The basic principle is applying the function to `@entries`, or to the extracted columns or rows (`[, i, drop=F]` or `[i, , drop=F]`). It is important to note that an empty column contains at least one zero value and may lead to non intuitive results.

This function may evolve over the next few releases.

Value

Similar as a call to `apply` with a regular matrix. The most important cases are as follows. The result is a vector (MARGIN is length 1 and FUN is scalar) or a matrix (MARGIN is length 1 and FUN returns fixed length vectors, or MARGIN is length 2 and FUN is scalar) or a list (if FUN returns vectors of different lengths).

Author(s)

Reinhard Furrer

See Also

base:apply for more details on Value.

Examples

```
S <- as.spam(dist(1:5))
S <- apply.spam(S/2, NULL, exp)
# instead of
# S@entries <- exp( S@entries/2)

# Technical detail, a null matrix consists
# of one zero element.
apply.spam(S,c(1,2),pmax)
apply.spam(S,1,range)

# A similar example as for the base apply.
# However, no dimnames else we would get warnings.
x <- as.spam(cbind(x1 = 3, x2 = c(0,0,0, 5:2)))
apply.spam(x, 2, mean, trim = .2)
col.sums <- apply.spam(x, 2, sum)
row.sums <- apply.spam(x, 1, sum)
rbind(cbind(x, row.sums), c(col.sums, sum(col.sums)))

apply.spam(x, 2, is.vector)

# Sort the columns of a matrix
# Notice that the result is a list due to the different
# lengths induced by the nonzero elements
apply.spam(x, 2, sort)

# Function with extra args:
cave <- function(x, c1, c2) c(mean(x[c1]), mean(x[c2]))
apply(x,1, cave, c1=1, c2=c(1,2))

ma <- spam(c(1:4, 0, 0,0, 6), nrow = 2)
ma
apply.spam(ma, 1, table) #--> a list of length 2
apply.spam(ma, 1, stats::quantile)# 5 x n matrix with rownames
```

bandwidth

Bandwidth of a Sparse Matrix

Description

Returns the lower and upper bandwidth of a sparse matrix

Usage

```
bandwidth(A)
```

Arguments

A spam object

Details

The matrix does not need to be diagonal. Values can be negative indicating the the matrix contains a band confined in the upper or lower triangular part.

Value

Integer vector containing the lower and upper bandwidth

Author(s)

Reinhard Furrer

See Also

[diag.spam.](#)

Examples

```
bandwidth(spam(c(0, 1), 3, 2))
```

```
bandwidth(spam(c(0, 0, 1, rep(0, 9)), 4, 3))
```

bdiag

Binds Arrays Corner-to-Corner

Description

Creates a sparse block-diagonal matrix.

Usage

```
bdiag.spam(...)
```

Arguments

... Arrays to be binded together

Details

This is a small helper function to create block diagonal sparse matrices. In the two matrix case, `bdiag.spam(A,B)`, this is equivalent to a complicated `rbind(cbind(A, null), cbind(B, t(null)))`, where `null` is a null matrix of appropriate dimension.

It is recursively defined.

The arrays are coerced to sparse matrices first.

This function is similar to the function `bdiag` from the package `Matrix`. It is also similar to the function `adiag` from the package `magic`. However, here no padding is done and all the dimnames are stripped.

Value

Returns a `spam` matrix as described above.

Author(s)

Reinhard Furrer

See Also

[diag.spam](#).

Examples

```
A <- diag.spam(2, 4)           # 2*I4
B <- matrix(1,3,3)
AB <- bdiag.spam(A,B)

# equivalent to:
ABalt <- rbind(cbind( A, matrix(0,nrow(A),ncol(B))),
               cbind( matrix(0,nrow(B),ncol(A)), B))

norm(AB-ABalt)

# Matrices do not need to be square:
bdiag.spam(1,2:5,6)
```

cbind

Combine spam Matrices by Rows or Columns

Description

Take a sequence of vector, matrix or `spam` object arguments and combine by columns or rows, respectively.

Usage

```
cbind.spam(..., deparse.level = 0)
rbind.spam(..., deparse.level = 0)
```

Arguments

... vectors, matrices or spam objects. See ‘Details’ and ‘Value’

deparse.level for compatibility reason here. Only 0 is implemented.

Details

rbind and cbind are not exactly symmetric in how the objects are processed. The former is essentially an concatenation of the slots due to the sparse storage format. Different types of inputs are handled differently. The latter calls a Fortran routine after the input has been coerced to spam objects.

Only two objects at a time are processed. If more than two are present, a loop concatenates them successively.

A method is defined for a spam object as first argument.

Value

a spam object combining the ... arguments column-wise or row-wise. (Exception: if there are no inputs or all the inputs are NULL, the value is NULL.)

Author(s)

Reinhard Furrer

Examples

```
x <- cbind.spam(1:5,6)

y <- cbind(x, 7)

rbind(x, x)
# for some large matrices t( cbind( t(x), t(x)))
# might be slightly faster:
```

 chol

Cholesky Factorization for Sparse Matrices

Description

chol performs a Cholesky decomposition of a symmetric positive definite sparse matrix `x` of class `spam`.

Usage

```
# chol(x, ...)

## S3 method for class 'spam'
chol(x, pivot = "MMD", method = "NgPeyton", memory =
      list(), eps = .Spam$eps, ...)

# update.spam.chol.NgPeyton(object, x, ...)
## S3 method for class 'spam.chol.NgPeyton'
update(object, x, ...)
```

Arguments

<code>x</code>	symmetric positive definite matrix of class <code>spam</code> .
<code>pivot</code>	should the matrix be permuted, and if, with what algorithm, see ‘Details’ below.
<code>method</code>	Currently, only <code>NgPeyton</code> is implemented.
<code>memory</code>	Parameters specific to the method, see ‘Details’ below.
<code>eps</code>	threshold to test symmetry. Defaults to <code>.Spam\$eps</code> .
<code>...</code>	further arguments passed to or from other methods.
<code>object</code>	an object from a previous call to <code>chol</code> .

Details

`chol` performs a Cholesky decomposition of a symmetric positive definite sparse matrix `x` of class `spam`. Currently, there is only the block sparse Cholesky algorithm of Ng and Peyton (1993) implemented (`method="NgPeyton"`).

To pivot/permute the matrix, you can choose between the multiple minimum degree (`pivot="MMD"`) or reverse Cuthill-McKee (`pivot="RCM"`) from George and Lui (1981). It is also possible to furnish a specific permutation in which case `pivot` is a vector. For compatibility reasons, `pivot` can also take a logical in which for `FALSE` no permutation is done and for `TRUE` is equivalent to `MMD`.

Often the sparsity structure is fixed and does not change, but the entries do. In those cases, we can update the Cholesky factor with `update.spam.chol.NgPeyton` by supplying a Cholesky factor and the updated matrix.

The option `cholupdatesingular` determines how singular matrices are handled by update. The function hands back an error ("error"), a warning ("warning") or the value NULL ("null").

The Cholesky decompositions requires parameters, linked to memory allocation. If the default values are too small the Fortran routine returns an error to \mathbb{R} , which allocates more space and calls the Fortran routine again. The user can also pass better estimates of the allocation sizes to `chol` with the argument `memory=list(nnzR=..., nnzcolindices=...)`. The minimal sizes for a fixed sparsity structure can be obtained from a summary call, see 'Examples'.

The output of `chol` can be used with `forwardsolve` and `backsolve` to solve a system of linear equations.

Notice that the Cholesky factorization of the package `SparseM` is also based on the algorithm of Ng and Peyton (1993). Whereas the Cholesky routine of the package `Matrix` are based on CHOLMOD by Timothy A. Davis (C code).

Value

The function returns the Cholesky factor in an object of class `spam.chol.method`. Recall that the latter is the Cholesky factor of a reordered matrix `x`, see also [ordering](#).

Note

Although the symmetric structure of `x` is needed, only the upper diagonal entries are used. By default, the code does check for symmetry (contrarily to `base::chol`). However, depending on the matrix size, this is a time consuming test. A test is ignored if `spam.options("cholsymmetrycheck")` is set to FALSE.

If a permutation is supplied with `pivot`, `spam.options("cholpivotcheck")` determines if the permutation is tested for validity (defaults to TRUE).

Author(s)

Reinhard Furrer, based on Ng and Peyton (1993) Fortran routines

References

Ng, E. G. and B. W. Peyton (1993) Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Comput.*, **14**, 1034–1056.

George, A. and Liu, J. (1981) *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall.

See Also

[det](#), [solve](#), [forwardsolve](#), [backsolve](#) and [ordering](#).

Examples

```

# generate multivariate normals:
set.seed(13)
n <- 25 # dimension
N <- 1000 # sample size
Sigma <- .25^abs(outer(1:n,1:n,"-"))
Sigma <- as.spam( Sigma, eps=1e-4)

cholS <- chol( Sigma)
# cholS is the upper triangular part of the permuted matrix Sigma
iord <- ordering(cholS, inv=TRUE)

R <- as.spam(cholS)
mvsample <- ( array(rnorm(N*n),c(N,n)) %*% R)[,iord]
# It is often better to order the sample than the matrix
# R itself.

# 'mvsample' is of class 'spam'. We need to transform it to a
# regular matrix, as there is no method 'var' for 'spam' (should there?).
norm( var( as.matrix( mvsample)) - Sigma, type="HS")
norm( t(R) %*% R - Sigma, type="sup")

# To speed up factorizations, memory allocations can be optimized:
opt <- summary(cholS)
# here, some elements of Sigma may be changed...
cholS <- chol( Sigma, memory=list(nnzR=opt$nnzR,nnzcolindices=opt$nnzc))

```

circulant

Create Circulant Matrices

Description

Creates a circulant matrix in spam format.

Usage

```
circulant.spam(x, n = NULL, eps = .Spam$eps)
```

Arguments

x	the first row to form the circulant matrix or a list containing the indices and the nonzero values.
n	if x is a list, the dimension of the matrix.
eps	A tolerance parameter: elements of x such that $\text{abs}(x) \leq \text{eps}$ set to zero. Defaults to $\text{eps} = \text{.Spam\$eps}$

Value

The circulant matrix in spam format.

Author(s)

Reinhard Furrer

See Also

[circulant](#) from package **magic**, [toeplitz.spam](#)

Examples

```
circulant.spam(c(1, .25, 0, 0, 0))
```

cleanup

Cleaning up sparse matrices

Description

Eliminates an zeros in a sparse matrix.

Usage

```
cleanup(x, eps = .Spam$eps)
```

Arguments

x	a sparse matrix of class spam.
eps	numeric scalar > 0. Smaller entries are coerced to zero.

Details

A sparse matrix may still contain zeros. This function (aliased to `as.spam`) filters these values. This often causes confusion when testing such matrices for symmetry or comparing apparently equal matrices with `all.equal` (see ‘Examples’ below).

Author(s)

Reinhard Furrer

See Also

[isSymmetric.spam](#) and [all.equal.spam](#).

Examples

```
A <- diag.spam(2)
A[1,2] <- 0

all.equal(A, t(A))
isSymmetric.spam(A)
all.equal(cleanup(A), diag.spam(2))
```

complexity

Complexity for Sparse Matrices

Description

A few results of computational complexities for selected sparse algorithms in spam

Details

A Cholesky factorization of an n -matrix requires $n^3/3$ flops. In case of banded matrices (bandwidth p , $p \ll n$) a factorization requires about $2np^2$ flops. Forward- and backsolves for banded matrices require essentially $2np$ flops.

George and Liu (1981) proves that any reordering would require at least $O(n^{3/2})$ flops for the factorization and produce at least $O(n \log(n))$ fill-ins for square lattices with a local neighborhood. They also show that algorithms based on nested dissection are optimal in the order of magnitude sense.

More to follow.

References

George, A. and Liu, J. (1981) *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall.

See Also

[det](#), [solve](#), [forwardsolve](#), [backsolve](#) and [ordering](#).

det

Calculate the determinant of a positive definite Sparse Matrix

Description

`det` and `determinant` calculate the determinant of a positive definite sparse matrix. `determinant` returns separately the modulus of the determinant, optionally on the logarithm scale, and the sign of the determinant.

Usage

```
# det(x, ...)
determinant(x, logarithm = TRUE, ...)
```

Arguments

x	sparse matrix of class <code>spam</code> or a Cholesky factor of class <code>spam.chol</code> . NgPeyton.
logarithm	logical; if <code>TRUE</code> (default) return the logarithm of the modulus of the determinant.
...	Optional arguments. Examples include method argument and additional parameters used by the method.

Details

If the matrix is not positive definite, the function issues a warning and returns `NA`.

The determinant is based on the product of the diagonal entries of a Cholesky factor, i.e. internally, a Cholesky decomposition is performed. By default, the NgPeyton algorithm with minimal degree ordering is used. To change the methods or supply additional parameters to the Cholesky factorization function, see the help for [chol](#).

The determinant of a Cholesky factor is also defined.

Value

For `det`, the determinant of `x`. For `determinant`, a list with components

modulus	a numeric value. The modulus (absolute value) of the determinant if <code>logarithm</code> is <code>FALSE</code> ; otherwise the logarithm of the modulus.
sign	integer; either <code>+1</code> or <code>-1</code> according to whether the determinant is positive or negative.

Author(s)

Reinhard Furrer

References

Ng, E. G. and B. W. Peyton (1993) Block sparse Cholesky algorithms on advanced uniprocessor computers, *SIAM J. Sci. Comput.*, **14**, 1034–1056.

See Also

[chol](#)

Examples

```
x <- spam( c(4,3,0,3,5,1,0,1,4),3)
det( x)
determinant( x)

det( chol( x))
```

diag *Sparse Matrix diagonals*

Description

Extract or replace the diagonal of a matrix, or construct a diagonal matrix.

Usage

```
# diag(x)
diag(x=1, nrow, ncol)
diag(x) <- value

diag.spam(x=1, nrow, ncol)
diag.spam(x) <- value
```

Arguments

x a spam matrix, a vector or a scalar.
nrow, ncol Optional dimensions for the result.
value either a single value or a vector of length equal to that of the current diagonal.

Details

Using `diag(x)` can have unexpected effects if `x` is a vector that could be of length one. Use `diag(x, nrow = length(x))` for consistent behaviour.

Value

If `x` is a spam matrix then `diag(x)` returns the diagonal of `x`.

The assignment form sets the diagonal of the sparse matrix `x` to the given value(s).

`diag.spam` works as `diag` for spam matrices: If `x` is a vector (or 1D array) of length two or more, then `diag.spam(x)` returns a diagonal matrix whose diagonal is `x`.

If `x` is a vector of length one then `diag.spam(x)` returns an identity matrix of order the nearest integer to `x`. The dimension of the returned matrix can be specified by `nrow` and `ncol` (the default is square).

The assignment form sets the diagonal of the matrix `x` to the given value(s).

Author(s)

Reinhard Furrer

See Also

[upper.tri](#), [lower.tri](#).

Examples

```
diag.spam(2, 4)          # 2*I4
smat <- diag.spam(1:5)
diag( smat)
diag( smat) <- 5:1

# The last line is equivalent to
diag.spam( smat) <- 5:1

# Note that diag.spam( 1:5) <- 5:1 not work of course.
```

dim

Dimensions of an Object

Description

Retrieve or set the dimension of an spam object.

Usage

```
# dim(x)
# dim(x) <- value
```

Arguments

x	a spam matrix
value	A numeric two-vector, which is coerced to integer (by truncation).

Details

It is important to notice the different behavior of the replacement method for ordinary arrays and spam objects (see ‘Examples’). Here, the elements are not simply rearranged but an entirely new matrix is constructed. If the new column dimension is smaller than the original, the matrix is also cleaned (with `spam.option("eps")` as filter).

For the same operation as with regular arrays, use `spam`

Value

`dim` retrieves the dimension slot of the object. It is a vector of mode `integer`.

The replacement method changes the dimension of the object by truncation or extension (with zeros).

Author(s)

Reinhard Furrer

See Also[dim.](#)**Examples**

```
x <- diag(4)
dim(x) <- c(2,8)
x

s <- diag.spam(4)
dim(s) <- c(7,3) # any positive value can be used

s <- diag.spam(4)
dim(s) <- c(2,8) # result is different than x
```

display*Graphially Represent the Nonzero Entries*

Description

The function represents the nonzero entries in a simple bicolor plot.

Usage

```
display(x, ...)
```

Arguments

`x` matrix of class `spam` or `spam.chol`. NgPeyton.
`...` any other arguments passed to `image.default/plot`.

Details

`spam.getOption("imagesize")` determines if the sparse matrix is coerced into a matrix and the plotted with `image.default` or if the matrix is simply represented as a scatterplot with `pch="."`. The points are scaled according to `cex*spam.getOption("cex")/(nrow + ncol)`. For some devices or for non-square matrices, `cex` needs probably some adjustment.

Author(s)

Reinhard Furrer

See Also

[image](#), [spam.options](#)

Examples

```
set.seed(13)

nz <- 8
ln <- nz
smat <- spam(0, ln, ln)
smat[cbind(sample(ln, nz), sample(ln, nz))] <- 1:nz

par(mfcol=c(1,2), pty='s')
spam.options(imagesize = 1000)
display(smat)
spam.options(imagesize = 10)
display(smat)

# very large but very sparse matrix
nz <- 128
ln <- nz^2
smat <- spam(0, ln, ln)
smat[cbind(sample(ln, nz), sample(ln, nz))] <- 1:nz

par(mfcol=c(1, 1), pty='s')
display(smat, cex = 100)
```

foreign

Transformation to other sparse formats

Description

Transform between the spam sparse format to the matrix.csr format of SparseM and dgRMatrix format of Matrix

Usage

```
as.spam.matrix.csr(x)
# as.matrix.csr.spam(x)
as.dgRMatrix.spam(x)
as.dgCMatrix.spam(x)
as.spam.dgRMatrix(x)
as.spam.dgCMatrix(x)
```

Arguments

x sparse matrix of class spam, matrix.csr, dgRMatrix or dgCMatrix.

Details

We do not provide any S4 methods and because of the existing mechanism a standard S3 does not work.

The functions are based on require.

Notice that `as.matrix.csr.spam` should read as `as."matrix.csr".spam`.

Value

According to the call, a sparse matrix of class `spam`, `matrix.csr`, `dgRMatrix` or `dgCMatrix`.

Author(s)

Reinhard Furrer

See Also

[triplet](#), [Matrix](#) or [matrix.csr](#)

Examples

```
## Not run:
S <- diag.spam(4)
U <- as.matrix.csr.spam( S)
R <- as.dgRMatrix.spam( S)
C <- as.dgCMatrix.spam( S)
as.spam.dgCMatrix(C)
slotNames(U)
slotNames(R)
# For column oriented sparse formats a transpose does not the job,
# as the slot names change.

# as.spam(R) does not work.

## End(Not run)

## Not run:
# a dataset contained in Matrix
data(KNex)
as.spam.dgCMatrix(KNex$mm)

## End(Not run)
```

germany

Administrative districts of Germany

Description

Constructing the adjacency graph and displaying the data over the administrative districts of Germany

Usage

```
adjacency.landkreis( loc)
map.landkreis(data, col=NULL, zlim=range(data), add=FALSE,
              legendpos=c( 0.88,0.9,0.05,0.4))
```

Arguments

loc	location of the graph structure, can be an URL.
data	vector of length 544
col	color scheme to be used. By default uses <code>tim.colors</code> if available or a generic gray scale.
zlim	the minimum and maximum values for which colors should be plotted, defaulting to the range of data.
add	logical, if true adds to current plot.
legendpos	if package fields is loaded, puts a legend at that position.

Details

`adjacency.landkreis` is included as an example on how to construct adjacency matrices from a (common) adjacency structure. For the particular example, note that the nodes are not numbered consecutively and that they start from zero.

The perfect position of the legend is an art per se and depends on various par parameters. See also the source code of the function `image.plot` of **fields**.

Author(s)

Reinhard Furrer

References

The code of `map.landkreis` is very similar to `germany.map` from the package **INLA**.

See Also

[Oral](#).

Examples

```
loc <- system.file("demodata/germany.adjacency", package="spam")
display( adjacency.landkreis( loc))

map.landkreis( Oral$E)
```

image

Display a spam Object as Color Image

Description

The function creates a grid of colored rectangles with colors corresponding to the values in `z`.

Usage

```
image(x, ...)
```

Arguments

`x` matrix of class `spam` or `spam.chol`.NgPeyton.
`...` any other arguments passed to `image.default` and `plot`.

Details

`getOption("imagesize")` determines if the sparse matrix is coerced into a matrix and the plotted similarly to `image.default` or if the matrix is simply represented as a scatterplot with `pch="."`. The points are scaled according to `cex*spam.getOption("cex")/(nrow+ncol)`. For some devices or for non-square matrices, `cex` needs probably some adjustment.

The a zero matrix in `spam` format has as (1,1) entry the value zero and only missing entries are interpreted as NA in the scatter plot.

Author(s)

Reinhard Furrer

See Also

[display](#) and [spam.options](#).
The code is based on [image](#) of `graphics`.

Examples

```
set.seed(13)
nz <- 8
ln <- nz
smat <- spam(0, ln, ln)
smat[ cbind(sample(ln,nz), sample(ln,nz))] <- 1:nz
```

```

par(mfcol=c(1,2),pty='s')
spam.options( imagesize=1000)
image(smat) # better: col=tim.colors(nz))
spam.options( imagesize=10)
image(smat) # better: col=tim.colors(nz))

nz <- 128
ln <- nz^2
smat <- spam(0,ln,ln)
smat[cbind(sample(ln,nz), sample(ln,nz))] <- 1:nz

par(mfcol=c(1,1), pty='s')
image(smat,cex=100) # better:, col=tim.colors(nz))

```

import

Read External Matrix Formats

Description

Read matrices stored in the Harwell-Boeing or MatrixMarket formats.

Usage

```

read.HB(file)
read.MM(file)

```

Arguments

`file` the name of the file to read, as a character scalar.
Alternatively, `read.HB` and `read.MM` accept connection objects.

Details

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".

Currently, only real assembled Harwell-Boeing can be read with `read.HB`. Reading MatrixMarket formats is more flexible. However, as entries of `spam` matrices are of mode `double`, integers matrices are coerced to doubles, patterns lead to matrices containing ones and complex are coerced to the real part thereof. In these aforementioned cases, a warning is issued.

MatrixMarket also defines an array format, in which case a (possibly) dense `spam` object is returned (retaining only elements which are larger than `spam.options('eps')`). A warning is issued.

Value

A sparse matrix of class `spam`.

Note

The functions are based on readHB and readMM from the library Matrix to build the connection and read the raw data. At present, read.MM is more flexible than readMM.

Author(s)

Reinhard Furrer based on Matrix functions by Douglas Bates <bates@stat.wisc.edu> and Martin Maechler <maechler@stat.math.ethz.ch>

References

<http://math.nist.gov/MatrixMarket>

<http://www.cise.ufl.edu/research/sparse/matrices>

Examples

```
## Not run:
image(read.MM(gzcon(url(
  "ftp://math.nist.gov/pub/MatrixMarket2/Harwell-Boeing/bcspwr/bcspwr01.mtx.gz"))))

## End(Not run)

## Not run:
## Datasets supplied within Matrix
str(read.MM(system.file("external/pores_1.mtx", package = "Matrix")))
str(read.HB(system.file("external/utm300.rua", package = "Matrix")))
str(read.MM(system.file("external/lund_a.mtx", package = "Matrix")))
str(read.HB(system.file("external/lund_a.rsa", package = "Matrix")))

## End(Not run)
```

isSymmetric

Test if a spam matrix is Symmetric

Description

Efficient function to test if 'object' is symmetric or not.

Usage

```
# isSymmetric.spam(object, ...)
## S3 method for class 'spam'
isSymmetric(object, tol = 100 * .Machine$double.eps, ...)
```

Arguments

object a spam matrix.
tol numeric scalar ≥ 0 . Smaller differences are not considered, see `all.equal.spam`.
... further arguments passed to `all.equal.spam`.

Details

symmetry is assessed by comparing the sparsity structure of `object` and `t(object)` via the function `all.equal.spam`. If a difference is detected, the matrix is cleaned with `cleanup` and compared again.

Value

logical indicating if `object` is symmetric or not.

Author(s)

Reinhard Furrer

See Also

[all.equal.spam](#), [cleanup](#).

Examples

```
obj <- diag.spam(2)
isSymmetric(obj)

obj[1,2] <- .Machine$double.eps
isSymmetric(obj)
all.equal(obj, t(obj))
```

kronecker

Kronecker Products on Sparse Matrices

Description

Computes the generalised kronecker product of two arrays, X and Y.

Usage

```
kronecker.spam(X, Y, FUN = "*", make.dimnames = FALSE, ...)
```

Arguments

X	sparse matrix of class spam, a vector or a matrix.
Y	sparse matrix of class spam, a vector or a matrix.
FUN	a function; it may be a quoted string. See details
make.dimnames	Provide dimnames that are the product of the dimnames of X and Y.
...	optional arguments to be passed to FUN.

Details

The sparsity structure is determined by the ordinary %x%. Hence, the result of `kroncker(X, Y, FUN = "+")` is different depending on the input.

Value

An array A with dimensions `dim(X) * dim(Y)`.

Author(s)

Reinhard Furrer

Examples

```
# Starting with non-spam objects, we get a spam matrix
kroncker.spam( diag(2), array(1:4, c(2, 2)))

kroncker( diag.spam(2), array(1:4, c(2, 2)))

# Notice the preservation of sparsity structure:
kroncker( diag.spam(2), array(1:4, c(2, 2)), FUN="+")
```

lower.tri

Lower and Upper Triangular Part of a Sparse Matrix

Description

Returns the lower or upper triangular structure or entries of a sparse matrix.

Usage

```
lower.tri(x, diag = FALSE)
upper.tri(x, diag = FALSE)
```

Arguments

x	a sparse matrix of class spam
diag	logical. Should the diagonal be included?

Details

Often not only the structure of the matrix is required but the entries as well. For compatibility, the default is only a structure consisting of ones (representing TRUEs). Setting the flag `spam.getOption("trivalues")` to TRUE, the function returns the actual values.

See Also

[spam.options](#) and [diag](#)

Examples

```
smat <- spam( c( 1,2,0,3,0,0,0,4,5),3)
upper.tri( smat)
upper.tri( smat, diag=TRUE)

spam.options( trivalues=TRUE)
upper.tri( smat)
```

makeprec

Create Precision Matrices

Description

Creates precision matrices for gridded GMRF.

Usage

```
make.prec(par, dims, model = "m1p1", eps = .Spam$eps)
```

Arguments

par	parameters used to construct the matrix.
dims	dimension of the grid.
model	see details and examples.
eps	A tolerance parameter: elements of 'x' such that 'abs(x) <= eps' set to zero. Defaults to 'eps = .Spam\$eps'

Details

The function should be illustrative on how to create precision matrices for gridded GMRF. Hence, no testing (positive definiteness is done). Please see the examples on the meaning of the different models.

Value

A spam matrix of dimension `prod(dims)xprod(dims)`.

Author(s)

Reinhard Furrer

See Also[toeplitz.spam](#), [kronecker.spam](#)**Examples**

```
as.matrix(make.prec(c(.4),          c(4,3), 'm1p1'))
as.matrix(make.prec(c(.4, .3),      c(4,3), 'm1p2'))
as.matrix(make.prec(c(.4, .3, .2),   c(4,3), 'm2p3'))
as.matrix(make.prec(c(.4, .3, .2, .1), c(4,3), 'm2p4'))
```

Math*Mathematical functions*

Description

Applies the Math group functions to 'spam' objects

Usage

```
# ceiling(x)
# floor(x)

# exp(x, base = exp(1))
# log(x, base = exp(1))
# sqrt(x)

# abs(x)
# cumprod(x)
# cumsum(x)

# cos(x)
# sin(x)
# tan(x)
# acosh(x)
...
```

Arguments

x	spam object.
base	positive number. The base with respect to which logarithms are computed. Defaults to e=exp(1).

Details

It is important to note that the zero entries do not enter the evaluation. The operations are performed on the stored non-zero elements. This may lead to differences if compared with the same operation on a full matrix. For example, the cosine of sparse matrix is a (full) matrix with many ones.

Evaluating function resulting in NA/NaN/Inf is possible but the result cannot be used further as NA/NaN/Inf are not meaningful (yet).

Value

All functions operate on the vector `x@entries` and return the result thereof.

Author(s)

Reinhard Furrer

See Also

[Math2](#)

Examples

```
getGroupMembers("Math")

mat <- matrix(c( 1,2,0,3,0,0,0,4,5),3)
smat <- as.spam( mat)
cos( mat)
cos( smat)

sqrt( smat)
```

Math2

Rounding of Numbers

Description

Applies the Math2 group functions to 'spam' objects

Usage

```
# round(x, digits = 0)
# signif(x, digits = 6)
```

Arguments

`x` spam object.
`digits` integer indicating the precision to be used.

Details

For this generic class typical `na.rm` argument has no weight here as NA/NaN/Inf are not meaningful (yet).

Value

All functions operate on the vector `x@entries` and return the result thereof.

Author(s)

Reinhard Furrer

See Also

Math

Examples

```
getGroupMembers("Math2")

smat <- diag.spam( rnorm(15))
round(smat, 3)
```

mle	<i>Maximum likelihood estimates</i>
-----	-------------------------------------

Description

Maximum likelihood estimates of a simple spatial model

Usage

```
neg2loglikelihood.spam(y, X, distmat, Covariance,
                       beta, theta, Rstruct = NULL, ...)

neg2loglikelihood(y, X, distmat, Covariance,
                 beta, theta, ...)

mle.spam(y, X, distmat, Covariance,
         beta0, theta0,
         thetalower, thetaupper, optim.control=NULL,
         Rstruct = NULL, hessian = FALSE,...)

mle(y, X, distmat, Covariance,
    beta0, theta0,
    thetalower, thetaupper, optim.control=NULL,
    hessian = FALSE, ...)
```

```
mle.nomean.spam(y, distmat, Covariance,
               theta0,
               thetalower, thetaupper, optim.control=NULL,
               Rstruct = NULL, hessian = FALSE, ...)

mle.nomean(y, distmat, Covariance,
           theta0,
           thetalower, thetaupper, optim.control=NULL,
           hessian = FALSE, ...)
```

Arguments

y	data vector of length n.
X	the design matrix of dimension n x p.
distmat	a distance matrix. Usually the result of a call to <code>nearest.dist</code> .
Covariance	function defining the covariance. See example.
beta	parameters of the trend (fixed effects).
theta	parameters of the covariance structure.
Rstruct	the Cholesky structure of the covariance matrix.
beta0, theta0	inital values.
thetalower, thetaupper	lower and upper bounds of the parameter theta.
optim.control	arguments passed to <code>optim</code> .
hessian	Logical. Should a numerically differentiated Hessian matrix be returned?
...	additional arguments passed to <code>chol</code> .

Details

We provide functions to calculate the negative-2-log-likelihood and maximum likelihood estimates for the model

$$y \sim N_n(X \beta, \text{Sigma}(h; \theta))$$

in the case of a sparse or ordinary covariance matrices.

In the case of the `*.spam` versions, the covariance function has to return a `spam` object. In the other case, the methods are correctly overloaded and work either way, slightly slower than the `*.spam` counterparts though.

When working on the sphere, the distance matrix has to be transformed by

$$h \rightarrow R / 2 \sin(h/2)$$

where R is the radius of the sphere.

The covariance function requires that the first argument is the distance matrix and the second the parameters. One can imagine cases in which the covariance function does not take the entire distance matrix but only some partial information thereof. (An example is the use of a kronecker type covariance structure.) In case of a sparse covariance construction where the argument `Rstruct` is

not given, the first parameter element needs to be the range parameter. (This results from the fact, that a sparse structure is constructed that is independent of the parameter values to exploit the fast Choleski decomposition.)

In the zero-mean case, the `neg2loglikelihood` is calculated by setting the parameters λ or β to zero.

Value

The negative-2-loglikelihood or the output from the function `optim`.

Author(s)

Reinhard Furrer

See Also

[chol](#)

Examples

```
# True parameter values:
truebeta <- c(1,2,.2) # beta = (intercept, linear in x, linear in y)
truetheta <- c(.5,2,.02) # theta = (range, sill, nugget)

# Auxiliary covariance function, parameterized by
# theta = (range, sill, nugget)
spherical <- function(distmat, theta, eps = 1e-06) {
  Sigma <- distmat
  d <- Sigma@entries/theta[1]

  Sigma@entries <- ifelse(d < eps,
    theta[3]+ theta[2],
    ifelse(d < 1, theta[2]*(1 - 1.5*d + 0.5*d^3), 0))
  return( Sigma)
}

# We now define a grid, distance matrix, and a sample:
x <- seq(0,1,l=5)
locs <- expand.grid( x, x)
X <- as.matrix( cbind(1,locs)) # design matrix
Covariance <- 'spherical' # covariance function

distmat <- nearest.dist( locs, upper=NULL) # distance matrix
Sigma <- spherical( distmat, truetheta) # true covariance matrix

set.seed(15)
y <- c(rmvnorm.spam(1,X %**% truebeta,Sigma)) # construct sample

# Here is the negative 2 log likelihood:
neg2loglikelihood.spam( y, X, distmat, Covariance,
```

```

        truebeta, truetheta)

# We pass now to the mle:
res <- mle.spam(y, X, distmat, Covariance,
               truebeta, truetheta, thetalower=c(0,0,0), thetaupper=c(1,Inf,Inf))

# Similar parameter estimates here, of course:
mle.nomean.spam(y-X%%res$par[1:3], distmat, Covariance,
                truetheta, thetalower=c(0,0,0), thetaupper=c(1,Inf,Inf))

```

nearestdist

Distance Matrix Computation

Description

This function computes and returns specific elements of distance matrix computed by using the specified distance measure.

Usage

```

nearest.dist( x, y=NULL, method = "euclidean",
              delta = 1, upper = if (is.null(y)) FALSE else NULL,
              p=2, miles=TRUE, R=NULL,
              eps = NULL, diag = NULL)

```

Arguments

x	Matrix of first set of locations where each row gives the coordinates of a particular point. See also ‘Details’.
y	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing x is used. See also ‘Details’.
method	the distance measure to be used. This must be one of "euclidean", "maximum", "minkowski" or "greatcircle". Any unambiguous substring can be given.
delta	only distances smaller than delta are recorded, see Details.
upper	Should the entire matrix (NULL) or only the upper-triagonal (TRUE) or lower-triagonal (FALSE) values be calculated.
p	The power of the Minkowski distance.
miles	For great circle distance: If true distances are in statute miles if false distances in kilometers.
R	For great circle distance: Radius to use for sphere to find spherical distances. If NULL the radius is either in miles or kilometers depending on the values of the miles argument. If R=1 then distances are of course in radians.
eps	deprecated. Left for backwards consistency.
diag	deprecated. Left for backwards consistency. See ‘Details’.

Details

For great circle distance, the by 2 matrices `x` and `y` contain the degrees longitudes in the first and the degrees latitudes in the second column. `eps` and `delta` are in degrees. Hence to restrict to distances smaller than `delta.km`, one has to specify `delta=delta.km*360/(6378.388*2*pi)`.

The distance is in single precision (I am still not sure where I lose the double precision in the Fortran code) and if calculating the entire matrix `upper=NULL` (instead of adding its transpose) it may not pass the symmetry checks, for example.

Default value of Earth's radius is 3963.34miles (6378.388km).

The arguments `eps` and `diag` do not have any effect and are left for compatibility reasons.

`x` and `y` can be any object with an existing `as.matrix` method.

A quick scan revealed distance functions in at least 7 packages (around 2008). The argument names should be as general as possible and be coherent with many (but not all) available distance functions.

The Fortran code is based on a idea of Doug Nychka.

Value

A spam object containing the distances spanned between zero and `delta`. The sparse matrix may contain many zeros (e.g., collocated data). However, to calculate covariances, these zeros are essential.

Author(s)

Reinhard Furrer

Examples

```
# Note that upper=T and using t(X)+X is quicker than upper=NULL;
#   upper=T marginally slower than upper=F.

# To compare nearest.dist with dist, use diag=FALSE, upper=TRUE
nx <- 4
x <- expand.grid(as.double(1:nx),as.double(1:nx))
sum( (nearest.dist( x, delta=nx*2, diag=FALSE, upper=TRUE)@entries-
      c(dist(x)))^2)

# Create nearest neighbor structures:
par(mfcol=c(1,2))
x <- expand.grid(1:nx,1:(2*nx))
display( nearest.dist( x, delta=1))
x <- expand.grid(1:(2*nx),1:nx)
display( nearest.dist( x, delta=1))
```

options

*Options Settings***Description**

Allow the user to set and examine a variety of *options* which affect the way in which R computes and displays sparse matrix results.

Usage

```
spam.options(...)
```

```
spam.getOption(x)
```

Arguments

... any options can be defined, using `name = value` or by passing a list of such tagged values. However, only the ones below are used in `spam`. Further, `spam.options('name')` == `spam.options()['name']`, see the example.

x a character string holding an option name.

Details

Invoking `spam.options()` with no arguments returns a list with the current values of the options. To access the value of a single option, one should use `spam.getOption("eps")`, e.g., rather than `spam.options("eps")` which is a *list* of length one.

Internally, the options are kept in the list `.Spam`.

Of course, printing is still subordinate to `getOption("max.print")` or similar options.

Value

For `spam.getOption`, the current value set for option `x`, or `NULL` if the option is unset.

For `spam.options()`, a list of all set options sorted by category. For `spam.options(name)`, a list of length one containing the set value, or `NULL` if it is unset. For uses setting one or more options, a list with the previous values of the options changed (returned invisibly).

Options used

A short description with the default values follows.

`eps=.Machine$double.eps`: values smaller than this are considered as zero. This is only used when creating `spam` objects.

`drop=FALSE`: default parameter for drop when subsetting

`printsize=100`: the max number of elements of a matrix which we display as regular matrix.

`imagesize=10000`: the max number of elements of a matrix we display as regular matrix with `image` or `display`. Larger matrices are represented as dots only.

`trivalues=FALSE`: a flag whether to return the structure (`FALSE`) or the values themselves (`TRUE`) when returning the upper and lower triangular part of a matrix.

`cex=1200`: default dot size for `image` or `display`.

`listmethod="PE"`: algorithm for `spam.list`. Default is suggestion by Paul Eilers (thanks). Any other specification uses a bubble sort algorithm which is only slightly faster for very sparse matrices.

`dopivoting=TRUE`: default parameter for "solve" routines. `FALSE` would solve the system without using the permutation.

`safemode=c(TRUE, TRUE, TRUE)`: The logicals are determine (1) verify double and integer formats when constructing `spam` objects (2) quick sanity check when constructing sparse matrices (3) testing for NAs in Fortran calls. `TRUE`s are safer but slightly slower. The most relevant speedup is the second flag.

`cholsymmetrycheck=TRUE`: for the Cholesky factorization, verify if the matrix is symmetric.

`cholpivotcheck=TRUE`: for the Cholesky factorization, when passing a permutation, should a minimum set of checks be performed?

`cholupdatesingular="warning"`: for a Cholesky update, what happens if the matrix is singular: "warning" only and returning the not updated factor, "error" or return simply "NULL".

`cholincreasefactor=c(1.25, 1.25)`: If not enough memory could be allocated, these are the steps to increase it.

`nnznearestdistnnz=c(400^2, 400)`: Memory allocation parameters for `nearest.dist`.

`nearestdistincreasefactor=1.25`: If not enough memory could be allocated, this is the step to increase it.

Author(s)

`spam.options` is essentially identical to `sm.options`.

See Also

[print](#), [display](#), [image](#), [upper.tri](#), [chol](#), [nearest.dist](#), etc.
[powerboost](#)

Examples

```
op <- spam.options()

# two ways of representing the options nicely.
utils::str(op)
noquote(format(op) )

smat <- diag.spam( 1:8)
```

```

smat
spam.options( printsize=49)
smat

# Reset to default values:
spam.options( eps=.Machine$double.eps, drop=FALSE,
  printsize=100, imagesize=10000, cex=1200,
  trivalues=FALSE, safemode=c(TRUE,TRUE,TRUE),
  dopivoting=TRUE, chol symmetrycheck=TRUE,
  cholpivotcheck=TRUE, cholupdatesingular="warning",
  cholincreasefactor=c(1.25,1.25),
  nearestdistincreasefactor=1.25,
  nearestdistnnz=c(400^2,400) )

```

 Oral

Oral Cavity Cancer

Description

Oral cavity cancer counts in 544 districts in Germany over 1986-1990.

Format

Oral is a dataframe with 3 columns.

Y observed counts

E expected counts

SMR standardized mortality ratios germany is a list of 544 elements, each describing an individual polygon of the district.

Details

The expected counts depend on the number of people in the region and their age distribution. The regions are ordered according the supplied polygon description and adjacency graph.

There is a similar dataset `data(Germany)` from the package **INLA** as well, with an additional smoking covariate. The exact origins and differences between both still needs to be retrieved.

Source

Both datasets are available from the package **INLA** distributed from www.r-inla.org or from <http://www.math.ntnu.no/~hrue/GMRF-book/oral.txt>
<http://www.math.ntnu.no/~hrue/GMRF-book/germany.graph>

References

Knorr-Held, L. and Rasser, G. (2000) Bayesian Detection of Clusters and Discontinuities in Disease Maps, *Biometrics*, 56, 13–21.

See Also

[map.landkreis](#).

ordering

Extract the permutation

Description

Extract the (inverse) permutation used by the Cholesky decomposition

Usage

```
ordering( x, inv=FALSE)
```

Arguments

x	object of class <code>spam.chol.method</code> returned by the function <code>chol</code> .
inv	Return the permutation (default) or inverse thereof.

Details

Recall that calculating a Cholesky factor from a sparse matrix consists of finding a permutation first, then calculating the factors of the permuted matrix. The ordering is important when working with the factors themselves.

The ordering from a full/regular matrix is 1:n.

Note that there exists many different algorithms to find orderings.

See the examples, they speak more than 10 lines.

Author(s)

Reinhard Furrer

See Also

[chol](#), [solve](#).

Examples

```

# Construct a pd matrix S to work with (size n)
n <- 100 # dimension
S <- .25^abs(outer(1:n,1:n,"-"))
S <- as.spam( S, eps=1e-4)
I <- diag(n) # Identity matrix

cholS <- chol( S)
ord <- ordering(cholS)
iord <- ordering(cholS, inv=TRUE)

R <- as.spam( cholS ) # R'R = P S P', with P=I[ord,],
# a permutation matrix (rows permuted).
RtR <- t(R) %**% R

# the following are equivalent:
as.spam( RtR - S[ord,ord] )
as.spam( RtR[iord,iord] - S )
as.spam( t(R[,iord]) %**% R[,iord] - S )

# trivially:
as.spam( t(I[iord,]) - I[ord,]) # (P^-1)' = P
as.spam( t(I[ord,]) - I[,ord]) #
as.spam( I[iord,] - I[,ord])
as.spam( I[ord,]%**%S%**I[,ord] - S[ord,ord] )
# pre and post multiplication with P and P' is ordering

```

permutation

Permute a matrix

Description

Row and/or column permutes a matrix.

Usage

```
permutation.spam(A, P=NULL, Q=NULL, ind=FALSE, check=TRUE)
```

Arguments

A	sparse matrix
P	vector giving the row permutation.
Q	vector giving the column permutation.
ind	are the indices given. See examples.
check	Should rudimentary checks be performed.

Details

If P and Q are permutation matrices, the result is PAQ. However, it is also possible to specify the indices and to perform in a very efficient way A[rowind, colind], see examples.

A row permutation is much faster than a column permutation. For very large matrices, a double transpose might be faster.

The spam option checkpivot determines if the permutation is verified.

Value

A permuted matrix.

Author(s)

Reinhard Furrer

See Also

[ordering](#), [spam.options](#).

Examples

```
A <- spam(1:12, 3)
P <- c(3, 1, 2)
Q <- c(2, 3, 1, 4)

permutation(A, P, Q) - A[order(P), order(Q)]

permutation(A, P, Q, ind=TRUE) - A[P, Q]
```

powerboost

Specific options Setting

Description

Sets several options for speed-up.

Usage

```
powerboost(flag)
```

Arguments

flag on or off

Details

The options turn checking off ("safemode", "cholsymmetrycheck" and "cholpivotcheck") and switch to single precision for "eps".

Value

NULL in any case.

Author(s)

Reinhard Furrer, after receiving too much C.mc.st adds.

See Also

[spam.options](#).

 print

Printing and summarizing sparse matrices

Description

Printing (non-zero elements) of sparse matrices and summarizing the sparsity structure thereof.

Usage

```
print(x, ...)
summary(object, ...)
```

Arguments

x	matrix of class spam or spam.chol. <i>method</i> .
object	matrix of class spam or spam.chol. <i>method</i> .
...	any other arguments passed to print.default.

Details

spam.getOption('printsize') determines if the sparse matrix is coerced into a matrix and the printed as an array or if only the non-zero elements of the matrix are given.

Value

NULL for print, because the information is printed with cat there is no real need to pass any object back.

A list containing the non-zero elements and the density for summary for class spam.

A list containing the non-zero elements of the factor, the density and the fill-in for summary for class spam.chol.NgPeyton.

Author(s)

Reinhard Furrer

See Also[spam.options](#)**Examples**

```

set.seed(13)
nz <- 8
ln <- nz
smat <- spam(0,ln,ln)
smat[cbind(sample(ln,nz),sample(ln,nz))] <- 1:nz

par(mfcol=c(1,2),pty='s')
spam.options( printsize=1000)
print(smat)
spam.options( printsize=10)
print(smat)
summary(smat)
(summary(smat))

```

 rmvnorm

Draw Multivariate Normals

Description

Fast ways to draw multivariate normals when the variance or precision matrix is sparse.

Usage

```

rmvnorm.spam(n,mu=rep(0, nrow(Sigma)), Sigma, Rstruct=NULL, ...)
rmvnorm.prec(n,mu=rep(0, nrow(Q)), Q, Rstruct=NULL, ...)
rmvnorm.canonical(n, b, Q, Rstruct=NULL, ...)

```

Arguments

n	number of observations.
mu	mean vector.
Sigma	covariance matrix of class spam.
Q	precision matrix.
b	vector determining the mean.
Rstruct	the Cholesky structure of Sigma or Q.
...	arguments passed to chol.

Details

The functions `rmvnorm.prec` and `rmvnorm.canonical` do not require sparse precision matrices. For `rmvnorm.spam`, the differences between regular and sparse covariance matrices are too significant to be implemented here.

Often (e.g., in a Gibbs sampler setting), the sparsity structure of the covariance/precision does not change. In such setting, the Cholesky factor can be passed via `Rstruct` in which only updates are performed (i.e., `update.spam.chol.NgPeyton` instead of a full `chol`).

Author(s)

Reinhard Furrer

References

See references in [chol](#).

See Also

[chol](#) and [ordering](#).

Examples

```
# Generate multivariate from a covariance inverse:
# (usefull for GRMF)
set.seed(13)
n <- 25 # dimension
N <- 1000 # sample size
Sigmainv <- .25^abs(outer(1:n,1:n,"-"))
Sigmainv <- as.spam( Sigmainv, eps=1e-4)

Sigma <- solve( Sigmainv) # for verification
iidsample <- array(rnorm(N*n),c(n,N))

mvsample <- backsolve( chol(Sigmainv), iidsample)
norm( var(t(mvsample)) - Sigma, type="HS")

# compare with:
mvsample <- backsolve( chol(as.matrix( Sigmainv)), iidsample)
norm( var(t(mvsample)) - Sigma, type="HS")

# 'solve' step by step:
b <- rnorm( n)
R <- chol(Sigmainv)
norm( backsolve( R, forwardsolve( R, b))-
      solve( Sigmainv, b),type="HS")
norm( backsolve( R, forwardsolve( R, diag(n)))- Sigma,type="HS")
```

spam

Sparse Matrix Class

Description

This group of functions evaluates and coerces changes in class structure.

Usage

```
spam(x, nrow = 1, ncol = 1, eps = .Spam$eps)
```

```
as.spam(x, eps = .Spam$eps)
```

```
is.spam(x)
```

Arguments

x	is a matrix (of either dense or sparse form), a list, vector object or a distance object
nrow	number of rows of matrix
ncol	number of columns of matrix
eps	A tolerance parameter: elements of x such that $\text{abs}(x) < \text{eps}$ set to zero. Defaults to $\text{eps} = \text{.Spam\$eps}$

Details

The functions `spam` and `as.spam` act like `matrix` and `as.matrix` to coerce an object to a sparse matrix object of class `spam`.

If `x` is a list, it should contain either two or three elements. In case of the former, the list should contain a `n` by two matrix of indices (called `ind`) and the values. In case of the latter, the list should contain three vectors containing the row, column indices (called `i` and `j`) and the values. In both cases partial matching is done. In case there are several triplets with the same `i`, `j`, the values are added.

`eps` should be at least as large as `.Machine$double.eps`.

Value

A valid `spam` object.

`is.spam` returns TRUE if `x` is a `spam` object.

Note

The zero matrix has the element zero stored in (1,1).

The functions do not test the presence of NA/NaN/Inf. Virtually all call a Fortran routine with the `NAOK=! .Spam$safemode[3]` argument, which defaults to FALSE resulting in an error. Hence, the NaN do not always properly propagate through (i.e. `spam` is not IEEE-754 compliant).

Author(s)

Reinhard Furrer

References

http://en.wikipedia.org/wiki/Sparse_matrix as a start.

See Also

[SPAM](#) for a general overview of the package; [spam.options](#) for details about the safemode flag; [read.MM](#) and [foreign](#) to create spam matrices from MatrixMarket files and from certain **Matrix** or **SparseM** formats.

Examples

```
# old message, do not loop, when you create a large sparse matrix
set.seed(13)
nz <- 128
ln <- nz^2
smat <- spam(0,ln,ln)
is <- sample(ln,nz)
js <- sample(ln,nz)
system.time(for (i in 1:nz) smat[is[i], js[i]] <- i)
system.time(smat[cbind(is,js)] <- 1:nz)

getClass("spam")

try(as.spam.numeric(NA))
```

spam operations

Basic Linear Algebra for Sparse Matrices

Description

Basic linear algebra operations for sparse matrices of class spam.

Details

Linear algebra operations for matrices of class spam are designed to behave exactly as for regular matrices. In particular, matrix multiplication, transpose, addition, subtraction and various logical operations should work as with the conventional dense form of matrix storage, as does indexing, rbind, cbind, and diagonal assignment and extraction (see for example [diag](#)). Further functions with identical behavior are dim and thus nrow, ncol.

The function norm calculates the (matrix-)norm of the argument. The argument type specifies the l1 norm, sup or max norm (default), or the Frobenius or Hilbert-Schmidt (frobenius/hs) norm.

Partial matching can be used. For example, `norm` is used to check for symmetry in the function `chol` by computing the norm of the difference between the matrix and its transpose

The operator `%d*%` efficiently multiplies a diagonal matrix (in vector form) and a sparse matrix and is used for compatibility with the package fields. More specifically, this method is used in the internal functions of `Krig` to make the code more readable. It avoids having a branch in the source code to handle the diagonal or nondiagonal cases. Note that this operator is not symmetric: a vector in the left argument is interpreted as a diagonal matrix and a vector in the right argument is kept as a column vector.

The operator `%d+%` efficiently adds a diagonal matrix (in vector form) and a sparse matrix, similarly to the operator `%d*%`.

References

Some Fortran functions are based on <http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html>

See Also

[spam](#) for coercion and other class relations involving the sparse matrix classes.

Examples

```
# create a weight matrix and scale it:
## Not run:
wij <- distmat
# with distmat from a nearest.dist(..., upper=TRUE) call

n <- dim(wij)[1]

wij@entries <- kernel( wij@entries, h) # for some function kernel
wij <- wij + t(wij) + diag.spam(n)    # adjust from diag=FALSE, upper=TRUE

sumwij <- wij %*% rep(1,n)
  # row scaling:
  #   wij@entries <- wij@entries/sumwij[ wij@colindices]
  # col scaling:
wij@entries <- wij@entries/sumwij[ rep(1:n, diff(wij@rowpointers))]]

## End(Not run)
```

Description

`backsolve` and `forwardsolve` solve a system of linear equations where the coefficient matrix is upper or lower triangular.

`solve` solves a linear system or computes the inverse of a matrix if the right-hand-side is missing.

Usage

```
# solve(a, b, ...)
# solve.spam(a, b, ...)

## S3 method for class 'spam'
solve(a, b, ...)

# backsolve(r, x, ...)
backsolve.spam(r, x, ...)
# forwardsolve(l, x, ...)
forwardsolve.spam(l, x, ...)
```

Arguments

a	symmetric positive definite matrix of class spam or a Cholesky factor as the result of a chol call.
l,r	object of class spam or spam.chol.method returned by the function chol.
x,b	vector or regular matrix of right-hand-side(s) of a system of linear equations.
...	further arguments passed to or from other methods, see 'Details' below.

Details

We can solve $A \% \% x = b$ by first computing the Cholesky decomposition $A = t(R)\% \% R$, then solving $t(R)\% \% y = b$ for y , and finally solving $R \% \% x = y$ for x . `solve` combines `chol`, a Cholesky decomposition of a symmetric positive definite sparse matrix, with `forwardsolve` and then `backsolve`.

In case `a` is from a `chol`, then `solve` is an efficient way to calculate `backsolve(a, forwardsolve(t(a), b))`.

However, for `a.spam` and `a.mat` from a `chol` call with a sparse and ordinary matrix, note that `forwardsolve(a.mat, b, transpose=T, upper.tri=T)` is equivalent to `forwardsolve(t(a.mat), b)` and `backsolve(a.spam, forwardsolve(a.spam, b, transpose=T, upper.tri=T))` yields the desired result. But `backsolve(a.spam, forwardsolve(t(a.spam), resid))` is wrong because `t(a.spam)` is a `spam` and not a `spam.chol.NgPeyton` object.

`forwardsolve` and `backsolve` solve a system of linear equations where the coefficient matrix is lower (`forwardsolve`) or upper (`backsolve`) triangular. Usually, the triangular matrix is result from a `chol` call and it is not required to transpose it for `forwardsolve`. Note that arguments of the default methods `k`, `upper.tri` and `transpose` do not have any effects here.

Notice that it is more efficient to solve successively the linear equations (both triangular solves) than to implement these in the Fortran code.

If the right-hand-side in `solve` is missing it will compute the inverse of a matrix. For details about the specific Cholesky decomposition, see [chol](#).

Recall that the Cholesky factors are from ordered matrices.

Note

There is intentionally no S3 distinction between the classes `spam` and `spam.chol.method`.

Author(s)

Reinhard Furrer, based on Ng and Peyton (1993) Fortran routines

References

See references in `chol`.

See Also

`det`, `chol` and `ordering`.

Examples

```
# Generate multivariate form a covariance inverse:
# (usefull for GRMF)
set.seed(13)
n <- 25 # dimension
N <- 1000 # sample size
Sigmainv <- .25^abs(outer(1:n,1:n,"-"))
Sigmainv <- as.spam( Sigmainv, eps=1e-4)

Sigma <- solve( Sigmainv) # for verification
iidsample <- array(rnorm(N*n),c(n,N))

mvsample <- backsolve( chol(Sigmainv), iidsample)
norm( var(t(mvsample)) - Sigma, type="HS")

# compare with:
mvsample <- backsolve( chol(as.matrix( Sigmainv)), iidsample)
norm( var(t(mvsample)) - Sigma, type="HS")

# 'solve' step by step:
b <- rnorm( n)
R <- chol(Sigmainv)
norm( backsolve( R, forwardsolve( R, b))-
      solve( Sigmainv, b),type="HS")
norm( backsolve( R, forwardsolve( R, diag(n)))- Sigma,type="HS")
```

spam-class

Class "spam"

Description

The spam class is a representation of sparse matrices.

Objects from the Class

Objects can be created by calls of the form `new("spam", entries, colindices, rowpointers, dimension)`. The standard "old Yale sparse format" is used to store sparse matrices.

The matrix `x` is stored in row form. The first element of row `i` is `x@rowpointers[i]`. The length of row `i` is determined by `x@rowpointers[i+1]-x@rowpointers[i]`. The column indices of `x` are stored in the `x@colindices` vector. The column index for element `x@entries[k]` is `x@colindices[k]`.

Slots

`entries`: Object of class "numeric" contains the nonzero values

`colindices`: Object of class "integer" ordered indices of the nonzero values

`rowpointers`: Object of class "integer" pointer to the beginning of each row in the arrays `entries` and `colindices`

`dimension`: Object of class "integer" ~~

Methods

as.matrix signature(`x = "spam"`): transforming a sparse matrix into a regular matrix.

as.spam signature(`x = "spam"`): cleaning of a sparse matrix.

`[<-` signature(`x = "spam", i, j, value`): assigning a sparse matrix. The negative vectors are not implemented yet.

`[` signature(`x = "spam", i, j`): subsetting a sparse matrix. The negative vectors are not implemented yet.

`%*%` signature(`x, y`): matrix multiplication, all combinations of sparse with full matrices or vectors are implemented.

`c` signature(`x = "spam"`): vectorizes the sparse matrix and takes account of the zeros. Hence the length of the result is `prod(dim(x))`.

cbind signature(`x = "spam"`): binds sparse matrices, see [cbind](#) for details.

chol signature(`x = "spam"`): see [chol](#) for details.

diag signature(`x = "spam"`): see [diag](#) for details.

dim<- signature(`x = "spam"`): truncates or augments the matrix see [dim](#) for details.

dim signature(`x = "spam"`): gives the dimension of the sparse matrix.

image signature(`x = "spam"`): see [image](#) for details.

display signature(x = "spam"): see [display](#) for details.

length<- signature(x = "spam"): Is not implemented and causes an error.

length signature(x = "spam"): gives the number of non-zero elements.

lower.tri signature(x = "spam"): see [lower.tri](#) for details.

Math signature(x = "spam"): see [Math](#) for details.

Math2 signature(x = "spam"): see [Math2](#) for details.

norm signature(x = "spam"): calculates the norm of a matrix.

plot signature(x = "spam", y): same functionality as the ordinary plot.

print signature(x = "spam"): see [print](#) for details.

rbind signature(x = "spam"): binds sparse matrices, see [cbind](#) for details.

solve signature(a = "spam"): see [solve](#) for details.

summary signature(object = "spam"): small summary statement of the sparse matrix.

Summary signature(x = "spam"): All functions of the Summary class (like min, max, range...) operate on the vector x@entries and return the result thereof. See Examples or [Summary](#) for details.

t signature(x = "spam"): transpose of a sparse matrix.

upper.tri signature(x = "spam"): see [lower.tri](#) for details.

Details

The compressed sparse row (CSR) format is often described with the vectors a, ia, ja. To be a bit more comprehensive, we have chosen longer slot names.

Note

The slots colindices and rowpointers are tested for proper integer assignments. This is not true for entries.

Author(s)

Reinhard Furrer, some of the Fortran code is based on A. George, J. Liu, E. S. Ng, B.W Peyton and Y. Saad (alphabetical)

Examples

```
showMethods("as.spam")
```

```
smat <- diag.spam(runif(15))
range(smat)
cos(smat)
```

spam.chol.NgPeyton-class

Class "spam.chol.NgPeyton"

Description

Result of a Cholesky decomposition with the NgPeyton method

Objects from the Class

Objects are created by calls of the form `chol(x,method="NgPeyton", ...)` and should not be created directly with a `new("spam.chol.NgPeyton", ...)` call.

At present, no proper print method is defined. However, the factor can be transformed into a spam object.

Methods

as.matrix signature(`x = "spam.chol.NgPeyton"`): Transform the factor into a regular matrix.

as.spam signature(`x = "spam.chol.NgPeyton"`): Transform the factor into a spam object.

backsolve signature(`r = "spam.chol.NgPeyton"`): solving a triangular system, see [solve](#).

forwardsolve signature(`l = "spam.chol.NgPeyton"`): solving a triangular system, see [solve](#).

c signature(`x = "spam.chol.NgPeyton"`): Coerce the factor into a vector.

determinant signature(`x = "spam.chol.NgPeyton"`): Calculates the determinant from the factor, see also [det](#).

diag signature(`x = "spam.chol.NgPeyton"`): Extracts the diagonal entries.

dim signature(`x = "spam.chol.NgPeyton"`): Retrieve the dimension. Note that "`dim<-`" is not implemented.

display signature(`x = "spam.chol.NgPeyton"`): Transformation to a spam object and display, see also [display](#).

image signature(`x = "spam.chol.NgPeyton"`): Transformation to a spam object and display, see also [image](#).

length signature(`x = "spam.chol.NgPeyton"`): Retrieve the dimension. Note that "`length<-`" is not implemented.

ordering signature(`x = "spam.chol.NgPeyton"`): Retrieves the ordering, in [ordering](#).

print signature(`x = "spam.chol.NgPeyton"`): Short description.

show signature(`object = "spam.chol.NgPeyton"`): Short description.

summary signature(`object = "spam.chol.NgPeyton"`): Description of the factor, returns (as a list) `nnzR`, `nnzcolindices`, the density of the factor `density`, and fill-in ratio `fillin`. For the use of the first two, see 'Examples' in [chol](#).

t signature(`x = "spam.chol.NgPeyton"`): Transformation to a spam object and transposition.

chol signature(`x = "spam.chol.NgPeyton"`): Returns `x` unchanged.

Author(s)

Reinhard Furrer

References

Ng, E. G. and B. W. Peyton (1993), "Block sparse Cholesky algorithms on advanced uniprocessor computers", *SIAM J. Sci. Comput.*, **14**, pp. 1034-1056.

See Also[print.spam ordering](#) and [chol](#)**Examples**

```
x <- spam( c(4,3,0,3,5,1,0,1,4),3)
cf <- chol( x)
cf
as.spam( cf)

# Modify at own risk...
slotNames(cf)
```

Summary

Rounding of Numbers

Description

Applies the Math2 group functions to 'spam' objects

Usage

```
# max(x, ..., na.rm = FALSE)
```

Arguments

x spam object.

Details

For this generic class typical `na.rm` argument has no weight here as `NA/NaN/Inf` are not meaningful (yet).

Value

All functions operate on the vector `x@entries` and return the result thereof.

Author(s)

Reinhard Furrer

See Also

[Math](#) and [Math2](#).

Examples

```
getGroupMembers("Summary")

smat <- diag.spam( runif(15))
range(smat)
```

toeplitz

Create Toeplitz Matrices

Description

Creates symmetric and asymmetric Toeplitz matrices.

Usage

```
toeplitz.spam(x, y = NULL, eps = .Spam$eps)
```

Arguments

x	the first row to form the Toeplitz matrix.
y	for asymmetric Toeplitz matrices, this contains the first column.
eps	A tolerance parameter: elements of x such that $\text{abs}(x) \leq \text{eps}$ set to zero. Defaults to $\text{eps} = \text{.Spam\$eps}$.

Details

The vector y has to be of the same length as x and its first element is discarded.

Value

The Toeplitz matrix in spam format.

Author(s)

Reinhard Furrer

See Also

[toeplitz](#), [circulant.spam](#)

Examples

```
toeplitz.spam(c(1,.25,0,0,0))
```

triplet	<i>Transform a spam format to triplets</i>
---------	--------------------------------------------

Description

Returns a list containing the indices and elements of a spam object.

Usage

```
triplet(x, tri=FALSE)
```

Arguments

x	sparse matrix of class spam or a matrix.
tri	Boolean indicating whether to create individual row and column indices vectors.

Details

The elements are row (column) first if x is a spam object (matrix).

Value

A list with elements

indices	a by two matrix containing the indices if tri=FALSE.
i,j	vectors containing the row and column indices if tri=TRUE.
values	a vector containing the matrix elements.

Author(s)

Reinhard Furrer

See Also

[spam.list](#) for the inverse operation and [foreign](#) for other transformations.

Examples

```
x <- diag.spam(1:4)
x[2,3] <- 5
triplet(x)
all.equal( spam( triplet(x, tri=TRUE)), x)
```

 UScounties

Adjacency structure of the counties in the contiguous United States

Description

First and second order adjacency structure of the counties in the contiguous United States. We consider that two counties are neighbors if they share at least one edge of their polygon description in maps.

Format

Two matrices of class `spam`

UScounties.storder Contains a one in the *i* and *j* element if county *i* is a neighbor of county *j*.

UScounties.ndorder Contains a one in the *i* and *j* element if counties *i* and *j* are a neighbors of county *k* and counties *i* and *j* are not neighbors.

See Also

map from **maps**.

Examples

```
# number of counties:
n <- nrow( UScounties.storder)

## Not run:
# make a precision matrix
Q <- diag.spam( n) + .2 * UScounties.storder + .1 * UScounties.ndorder
display( as.spam( chol( Q)))

## End(Not run)
```

 USprecip

Monthly total precipitation (mm) for April 1948 in the contiguous United States

Description

This is a useful spatial data set of moderate to large size consisting of 11918 locations. See www.image.ucar.edu/GSP/Data/US.monthly.met/ for the source of these data.

Format

This data set is an array containing the following columns:

lon,lat Longitude-latitude position of monitoring stations

raw Monthly total precipitation in millimeters for April 1948

anomaly Precipitation anomaly for April 1948.

infill Indicator, which station values were observed (5906 out of the 11918) compared to which were estimated.

Source

www.image.ucar.edu/GSP/Data/US.monthly.met/

References

Johns, C., Nychka, D., Kittel, T., and Daly, C. (2003) Infilling sparse records of spatial fields. *Journal of the American Statistical Association*, 98, 796–806.

See Also

[RMprecip](#)

Examples

```
# plot
## Not run:
library(fields)

data(USprecip)
par(mfcol=c(2,1))
quilt.plot(USprecip[,1:2],USprecip[,3])
US( add=TRUE, col=2, lty=2)
quilt.plot(USprecip[,1:2],USprecip[,4])
US( add=TRUE, col=2, lty=2)

## End(Not run)
```

version

Spam Version Information

Description

`spam.version` is a variable (`list`) holding detailed information about the version of `spam` loaded.

`spam.Version()` provides detailed information about the version of `spam` running.

Usage

```
spam.version
```

Value

`spam.version` is a list with character-string components

`status` the status of the version (e.g., "beta")

`major` the major version number

`minor` the minor version number

`year` the year the version was released

`month` the month the version was released

`day` the day the version was released

`version.string` a character string concatenating the info above, useful for plotting, etc.

`spam.version` is a list of class "simple.list" which has a print method.

Author(s)

Reinhard Furrer

See Also

See the R counterparts [R.version](#).

Examples

```
spam.version$version.string
```

Index

!=, spam-method (spam operations), 46

*Topic **IO**

import, 24
options, 36

*Topic **algebra**

apply, 6
bandwidth, 7
bdiag, 8
chol, 11
circulant, 13
cleanup, 14
complexity, 15
det, 15
diag, 17
import, 24
kronecker, 26
lower.tri, 27
makeprec, 28
mle, 31
nearestdist, 34
ordering, 39
rmvnorm, 43
spam, 45
spam operations, 46
spam solve, 47
toeplitz, 54

*Topic **array**

allequal, 4
apply, 6
bandwidth, 7
bdiag, 8
cbind, 9
circulant, 13
det, 15
diag, 17
dim, 18
foreign, 20
import, 24
isSymmetric, 25

kronecker, 26
lower.tri, 27
makeprec, 28
nearestdist, 34
permutation, 40
toeplitz, 54
triplet, 55

*Topic **classes**

spam-class, 50
spam.chol.NgPeyton-class, 52

*Topic **datasets**

Oral, 38
UScounties, 56
USprecip, 56

*Topic **documentation**

. SPAM ., 3

*Topic **environment**

options, 36
powerboost, 41
version, 57

*Topic **error**

options, 36

*Topic **hplot**

display, 19
germany, 22
image, 23
print, 42

*Topic **manip**

cbind, 9
foreign, 20
Math, 29
Math2, 30
Summary, 53

*Topic **package**

. SPAM ., 3

*Topic **print**

options, 36

*Topic **programming**

version, 57

***Topic sysdata**

- version, [57](#)
- *, ANY, spam-method (spam operations), [46](#)
- *, spam, ANY-method (spam operations), [46](#)
- *, spam, spam-method (spam operations), [46](#)
- +, ANY, spam-method (spam operations), [46](#)
- +, spam, ANY-method (spam operations), [46](#)
- +, spam, spam-method (spam operations), [46](#)
- , ANY, spam-method (spam operations), [46](#)
- , spam, ANY-method (spam operations), [46](#)
- , spam, spam-method (spam operations), [46](#)
- . SPAM ., [3](#)
- .Spam (options), [36](#)
- /, ANY, spam-method (spam operations), [46](#)
- /, spam, ANY-method (spam operations), [46](#)
- /, spam, spam-method (spam operations), [46](#)
- <, spam-method (spam operations), [46](#)
- <=, spam-method (spam operations), [46](#)
- ==, spam-method (spam operations), [46](#)
- >, spam-method (spam operations), [46](#)
- >=, spam-method (spam operations), [46](#)
- [, spam, ANY, ANY-method (spam operations), [46](#)
- [, spam, ANY-method (spam-class), [50](#)
- [, spam, matrix, matrix-method (spam-class), [50](#)
- [, spam, matrix, missing-method (spam-class), [50](#)
- [, spam, missing, missing-method (spam-class), [50](#)
- [, spam, missing, vector-method (spam-class), [50](#)
- [, spam, spam, missing-method (spam-class), [50](#)
- [, spam, vector, missing-method (spam-class), [50](#)
- [, spam, vector, vector-method (spam-class), [50](#)
- [.spam (spam operations), [46](#)
- [<-, spam, ANY, ANY-method (spam operations), [46](#)
- [<-, spam, ANY-method (spam-class), [50](#)
- [<-, spam, matrix, matrix, numeric-method (spam-class), [50](#)
- [<-, spam, matrix, matrix-method (spam operations), [46](#)
- [<-, spam, matrix, missing, numeric-method (spam-class), [50](#)
- [<-, spam, matrix, missing-method (spam operations), [46](#)
- [<-, spam, missing, missing, numeric-method (spam-class), [50](#)
- [<-, spam, missing, vector, numeric-method (spam-class), [50](#)
- [<-, spam, missing, vector-method (spam operations), [46](#)
- [<-, spam, spam, missing-method (spam operations), [46](#)
- [<-, spam, vector, missing, numeric-method (spam-class), [50](#)
- [<-, spam, vector, missing-method (spam operations), [46](#)
- [<-, spam, vector, vector, numeric-method (spam-class), [50](#)
- [<-, spam, vector, vector-method (spam operations), [46](#)
- [<-.spam (spam operations), [46](#)
- %*%, ANY, ANY-method (spam operations), [46](#)
- %*%, matrix, spam-method (spam operations), [46](#)
- %*%, numeric, spam-method (spam operations), [46](#)
- %*%, spam, matrix-method (spam operations), [46](#)
- %*%, spam, numeric-method (spam operations), [46](#)
- %*%, spam, spam-method (spam operations), [46](#)
- %*%-methods (spam operations), [46](#)
- %/%, spam-method (spam operations), [46](#)
- %%, spam-method (spam operations), [46](#)
- %d*% (spam operations), [46](#)
- %d*%, matrix, ANY-method (spam operations), [46](#)
- %d*%, matrix, spam-method (spam operations), [46](#)
- %d*%, numeric, matrix-method (spam operations), [46](#)
- %d*%, numeric, numeric-method (spam operations), [46](#)
- %d*%, numeric, spam-method (spam operations), [46](#)

- `%d*%`, spam, ANY-method (spam operations), 46
- `%d*%`, spam, numeric-method (spam operations), 46
- `%d*%`, spam, spam-method (spam operations), 46
- `%d+%` (spam operations), 46
- `%d+%`, matrix, ANY-method (spam operations), 46
- `%d+%`, matrix, spam-method (spam operations), 46
- `%d+%`, numeric, matrix-method (spam operations), 46
- `%d+%`, numeric, numeric-method (spam operations), 46
- `%d+%`, numeric, spam-method (spam operations), 46
- `%d+%`, spam, ANY-method (spam operations), 46
- `%d+%`, spam, numeric-method (spam operations), 46
- `%d+%`, spam, spam-method (spam operations), 46
- `&`, ANY, spam-method (spam operations), 46
- `&`, spam, ANY-method (spam operations), 46
- `&`, spam, spam-method (spam operations), 46
- `^`, spam-method (spam operations), 46

- `abs` (Math), 29
- `acos` (Math), 29
- `acosh` (Math), 29
- `adjacency.landkreis` (germany), 22
- `all` (Summary), 53
- `all.equal`, matrix, spam-method (allequal), 4
- `all.equal`, spam, spam-method (allequal), 4
- `all.equal.spam`, 14, 26
- `all.equal.spam` (allequal), 4
- `allequal`, 4
- `any` (Summary), 53
- `apply`, 6
- `Arith`, numeric, spam-method (spam-class), 50
- `Arith`, spam, missing-method (spam-class), 50
- `Arith`, spam, numeric-method (spam-class), 50
- `as.dgCMatrix.spam` (foreign), 20
- `as.dgRMatrix.spam` (foreign), 20
- `as.matrix`, spam-method (spam-class), 50
- `as.matrix.spam.chol.NgPeyton-method` (spam.chol.NgPeyton-class), 52
- `as.matrix.csr.spam` (foreign), 20
- `as.matrix.spam` (spam-class), 50
- `as.spam` (spam), 45
- `as.spam`, dist-method (spam), 45
- `as.spam`, list-method (spam), 45
- `as.spam`, matrix-method (spam), 45
- `as.spam`, numeric-method (spam), 45
- `as.spam`, spam-method (spam), 45
- `as.spam`, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- `as.spam.chol.NgPeyton` (spam), 45
- `as.spam.dgCMatrix` (foreign), 20
- `as.spam.dgRMatrix` (foreign), 20
- `as.spam.dist` (spam), 45
- `as.spam.list` (spam), 45
- `as.spam.matrix` (spam), 45
- `as.spam.matrix.csr` (foreign), 20
- `as.spam.numeric` (spam), 45
- `as.spam.spam` (spam), 45
- `asin` (Math), 29
- `asinh` (Math), 29
- `assign.spam` (spam operations), 46
- `atan` (Math), 29
- `atanh` (Math), 29

- `backsolve`, 12, 15
- `backsolve` (spam solve), 47
- `backsolve`, ANY-method (spam solve), 47
- `backsolve`, matrix-method (spam solve), 47
- `backsolve`, spam-method (spam solve), 47
- `backsolve`, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- `backsolve-methods` (spam solve), 47
- `backsolve.default` (spam solve), 47
- `backsolve.spam` (spam solve), 47
- `bandwidth`, 7
- `bdiag`, 8

- `c`, spam-method (spam-class), 50
- `c`, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- `cbind`, 9, 50, 51
- `cbind`, spam-method (cbind), 9
- `cbind.spam` (cbind), 9
- `ceiling` (Math), 29
- `chol`, 11, 16, 33, 37, 39, 44, 48–50, 52, 53

- chol, ANY-method (chol), 11
- chol, matrix-method (chol), 11
- chol, spam-method (chol), 11
- chol, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- chol.spam (chol), 11
- circulant, 13, 14
- circulant.spam, 54
- cleanup, 5, 14, 26
- Compare, numeric, spam-method
(spam-class), 50
- Compare, spam, numeric-method
(spam-class), 50
- complexities (complexity), 15
- complexity, 15
- cos (Math), 29
- cumprod (Math), 29
- cumsum (Math), 29

- det, 12, 15, 15, 49, 52
- det, spam-method (det), 15
- det, spam.chol.NgPeyton-method (det), 15
- det.spam (det), 15
- determinant (det), 15
- determinant, spam-method (det), 15
- determinant, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- determinant.spam (det), 15
- diag, 17, 28, 46, 50
- diag, ANY-method (diag), 17
- diag, spam-method (diag), 17
- diag, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- diag.assign, spam-method (diag), 17
- diag.of.spam (diag), 17
- diag.spam, 8, 9
- diag.spam (diag), 17
- diag.spam<- (diag), 17
- diag<- (diag), 17
- diag<-, ANY-method (diag), 17
- diag<-, spam-method (diag), 17
- diag<-.spam (diag), 17
- dim, 18, 19, 50
- dim, ANY-method (spam operations), 46
- dim, spam-method (spam operations), 46
- dim, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- dim<-, spam-method (dim), 18

- dim<-, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- dim<-.spam (dim), 18
- display, 19, 23, 37, 51, 52
- display, spam-method (display), 19
- display, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- display.spam (display), 19
- dist.spam (nearestdist), 34
- distance (nearestdist), 34

- exp (Math), 29

- fields, 4
- floor (Math), 29
- foreign, 20, 46
- forwardsolve, 12, 15
- forwardsolve (spam solve), 47
- forwardsolve, ANY-method (spam solve), 47
- forwardsolve, matrix-method (spam
solve), 47
- forwardsolve, spam-method (spam solve),
47
- forwardsolve, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- forwardsolve-methods (spam solve), 47
- forwardsolve.default (spam solve), 47
- forwardsolve.spam (spam solve), 47

- gamma (Math), 29
- germany, 22
- germany (Oral), 38
- germany.graph (germany), 22

- image, 19, 23, 23, 37, 50, 52
- image, spam-method (image), 23
- image, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 52
- image.spam (image), 23
- import, 24
- initialize, spam-method (spam), 45
- is.spam (spam), 45
- isSymmetric, 25
- isSymmetric, spam-method (isSymmetric),
25
- isSymmetric.spam, 5, 14
- isSymmetric.spam (isSymmetric), 25

- kronecker, 26

- kronecker, ANY, spam-method (spam-class), 50
- kronecker, spam, ANY-method (spam-class), 50
- kronecker, spam, spam-method (spam-class), 50
- kronecker.spam, 29
- length, spam-method (spam-class), 50
- length, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- length<-, spam-method (spam-class), 50
- length<-, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- lgamma (Math), 29
- log (Math), 29
- log10 (Math), 29
- lower.tri, 17, 27, 51
- lower.tri, spam-method (spam-class), 50
- make.prec (makeprec), 28
- makeprec, 28
- map.landkreis, 39
- map.landkreis (germany), 22
- Math, 29, 51, 54
- Math, spam-method (Math), 29
- Math2, 30, 30, 51, 54
- Math2, spam, numeric-method (Math2), 30
- Math2, spam-method (Math2), 30
- Matrix, 4, 21
- matrix.csr, 21
- max (Summary), 53
- min (Summary), 53
- mle, 31
- ncol, spam-method (spam operations), 46
- nearest.dist, 37
- nearest.dist (nearestdist), 34
- nearestdist, 34
- neg2loglikelihood (mle), 31
- norm (spam operations), 46
- norm, ANY-method (spam operations), 46
- norm, spam, character-method (spam operations), 46
- norm, spam, missing-method (spam operations), 46
- norm.spam (spam operations), 46
- nrow, spam-method (spam operations), 46
- Ops.spam (spam operations), 46
- options, 36
- Oral, 22, 38
- ordering, 12, 15, 39, 41, 44, 49, 52, 53
- ordering, matrix-method (ordering), 39
- ordering, spam-method (ordering), 39
- ordering, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- ordering-methods (ordering), 39
- ordering.spam.chol (ordering), 39
- overview (. SPAM .), 3
- permutation, 40
- permutation, matrix-method (permutation), 40
- permutation, spam-method (permutation), 40
- permutation.spam (permutation), 40
- plot, spam, missing-method (spam-class), 50
- plot, spam, spam-method (spam-class), 50
- plot.spam (spam operations), 46
- powerboost, 37, 41
- print, 37, 42, 51
- print, spam-method (print), 42
- print, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- print.spam, 53
- print.spam (print), 42
- prod (Summary), 53
- R.version, 58
- range (Summary), 53
- rbind (cbind), 9
- rbind, spam-method (cbind), 9
- rbind.spam (cbind), 9
- read.HB (import), 24
- read.MM, 46
- read.MM (import), 24
- RMprecip, 57
- rmvnorm, 43
- round (Math2), 30
- show, spam-method (spam-class), 50
- show, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- signif (Math2), 30
- sin (Math), 29
- solve, 12, 15, 39, 51, 52
- solve, ANY-method (spam solve), 47

- solve, spam-method (spam solve), 47
- solve.spam (spam solve), 47
- SPAM, 46
- SPAM (. SPAM .), 3
- Spam (. SPAM .), 3
- spam, 4, 45, 47
- spam operations, 46
- spam solve, 47
- spam, list-method (spam), 45
- spam, numeric-method (spam), 45
- spam-class, 50
- spam.chol.NgPeyton-class, 52
- spam.class, 4
- spam.class (spam-class), 50
- spam.creation (spam), 45
- spam.getOption (options), 36
- spam.list, 55
- spam.list (spam), 45
- spam.numeric (spam), 45
- spam.ops, 4
- spam.ops (spam operations), 46
- spam.options, 19, 23, 28, 41–43, 46
- spam.options (options), 36
- spam.Version (version), 57
- spam.version (version), 57
- SparseM.ontology, 4
- sqrt (Math), 29
- subset.spam (spam operations), 46
- sum (Summary), 53
- Summary, 51, 53
- summary (print), 42
- Summary, spam-method (Summary), 53
- summary, spam-method (print), 42
- summary, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- Summary.spam (Summary), 53
- summary.spam (print), 42

- t, spam-method (spam-class), 50
- t, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 52
- t.spam (spam operations), 46
- tan (Math), 29
- toeplitz, 54, 54
- toeplitz.spam, 14, 29
- triplet, 21, 55
- trunc (Math), 29

- update (chol), 11
- update, spam.chol.NgPeyton-method (chol), 11
- update.spam.chol.NgPeyton (chol), 11
- upper.tri, 17, 37
- upper.tri (lower.tri), 27
- upper.tri, spam-method (spam-class), 50
- UScounties, 56
- USprecip, 56

- validspamobject (spam), 45
- version, 57