Package ‘spam’

February 20, 2015

Version 1.0-1
Date 2014-09-09
Author Reinhard Furrer [aut, cre], Florian Gerber [ctb]
Maintainer Reinhard Furrer <reinhard.furrer@math.uzh.ch>
Depends R (>= 2.15), methods, grid
Suggests fields, Matrix
Description Set of functions 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 LGPL-2
Title SPArse Matrix
URL http://www.math.uzh.ch/furrer/software/spam/
NeedsCompilation yes
Repository CRAN
Date/Publication 2014-09-09 15:03:03

R topics documented:

 . SPAM ............................................................. 3
 adjacency ......................................................... 4
 allequal ............................................................ 5
 apply ................................................................. 7
 bandwidth ......................................................... 8
 bdiag ............................................................... 9
 cbind ............................................................. 10
 chol ............................................................... 12
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>circulant: 14</td>
</tr>
<tr>
<td>cleanup: 15</td>
</tr>
<tr>
<td>coerce-methods: 16</td>
</tr>
<tr>
<td>Coercion: 16</td>
</tr>
<tr>
<td>complexity: 17</td>
</tr>
<tr>
<td>constructors: 18</td>
</tr>
<tr>
<td>covmat: 19</td>
</tr>
<tr>
<td>crossprod: 21</td>
</tr>
<tr>
<td>det: 22</td>
</tr>
<tr>
<td>diag: 23</td>
</tr>
<tr>
<td>diff: 24</td>
</tr>
<tr>
<td>dim: 25</td>
</tr>
<tr>
<td>display: 26</td>
</tr>
<tr>
<td>fields wrapper: 27</td>
</tr>
<tr>
<td>foreign: 28</td>
</tr>
<tr>
<td>germany.data: 30</td>
</tr>
<tr>
<td>germany.plot: 31</td>
</tr>
<tr>
<td>grid_trace2: 32</td>
</tr>
<tr>
<td>grid_zoom: 33</td>
</tr>
<tr>
<td>head: 35</td>
</tr>
<tr>
<td>image: 36</td>
</tr>
<tr>
<td>import: 37</td>
</tr>
<tr>
<td>isSymmetric: 39</td>
</tr>
<tr>
<td>kronecker: 40</td>
</tr>
<tr>
<td>lower.tri: 41</td>
</tr>
<tr>
<td>makeprec: 42</td>
</tr>
<tr>
<td>map.landkreis: 43</td>
</tr>
<tr>
<td>Math: 44</td>
</tr>
<tr>
<td>Math2: 45</td>
</tr>
<tr>
<td>mle: 46</td>
</tr>
<tr>
<td>nearestdist: 49</td>
</tr>
<tr>
<td>options: 50</td>
</tr>
<tr>
<td>Oral: 53</td>
</tr>
<tr>
<td>ordering: 54</td>
</tr>
<tr>
<td>pad: 55</td>
</tr>
<tr>
<td>permutation: 56</td>
</tr>
<tr>
<td>powerboost: 57</td>
</tr>
<tr>
<td>precmat: 58</td>
</tr>
<tr>
<td>print: 59</td>
</tr>
<tr>
<td>rmvnorm: 60</td>
</tr>
<tr>
<td>rmvnorm.const: 62</td>
</tr>
<tr>
<td>rowSums: 63</td>
</tr>
<tr>
<td>s3only: 64</td>
</tr>
<tr>
<td>spam: 65</td>
</tr>
<tr>
<td>spam operations: 66</td>
</tr>
<tr>
<td>spam.solve: 67</td>
</tr>
<tr>
<td>spam-class: 70</td>
</tr>
<tr>
<td>spam.chol.NgPeyton-class: 72</td>
</tr>
</tbody>
</table>
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. We deliberately avoided aliases according to analogue helps from the base package.

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, with the help of Florian Gerber, Kaspar Moesinger and many others.
adjacency

References


See Also

See spam.class for a detailed class description, spam and spam.ops for creation, coercion and algebraic operations.

Examples

## Not run:
## Citations:
citation('spam')
citation('spam', auto=TRUE)

## History of changes
file.show(system.file("NEWS", package = "spam"))

## End(Not run)

adjacency

Administrative districts of Germany

Description

Constructing the adjacency graph of the administrative districts of Germany

Usage

adjacency.landkreis(loc)

Arguments

loc location of the graph structure, can be an URL.

Details

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

Value

a sparse matrix in spam format.
Allequal

Author(s)

Reinhard Furrer

References

The adjacency data has been provided by Havard Rue and is also available in INLA.

See Also

germany.plot super-seeding map.landkreis for plotting.

Oral.

Examples

## Not run:
loc <- system.file("demodata/germany.adjacency", package="spam")
display( adjacency.landkreis( loc))

## 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

target a spam object.
current another spam object to be compared with target.
tolerance numeric >= 0. Differences smaller than tolerance are not considered.
scale numeric scalar > 0 (or NULL). See 'Details'.
check.attributes currently not yet implemented.

... 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

```r
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

```r
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 of a Sparse Matrix

**Description**

Returns the lower and upper bandwidth of a sparse matrix

**Usage**

```r
bandwidth(A)
```
**bdiag**

**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**

```r
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.
**cbind**

**Usage**

```r
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**

```r
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, ...)

## S4 method for signature 'spam'
chol(x, pivot = "MMD", method = "NgPeyton", memory =
    list(), eps = .Spam$eps, ...)

# update.spam.chol.NgPeyton(object, x,...)
## S4 method for signature 'spam.chol.NgPeyton'
update(object, x,...)

Arguments

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

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 suppling a Cholesky factor and the updated matrix. Notice that the structure is effectively object <- update(object, x). The update feature without assignment has been disabled.
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 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**


**See Also**

`det.spam`, `solve.spam`, `forwardsolve.spam`, `backsolve.spam` 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 permutated 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='m')
norm( t(R) %*% R - Sigma)

# 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 abs(x) <= eps set to zero. Defaults to eps = .Spam$eps


**Value**

The circulant matrix in spam format.

**Author(s)**

Reinhard Furrer

**See Also**

`circulant` from package `magic`, `toeplitz.spam`

**Examples**

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

---

### Description

Eliminates zeros in a sparse matrix.

### Usage

```r
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

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

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

---

**Coercion**

Coercion to a Class

Description

These functions manage the relations that allow coercing a spam object to a given class.

Methods

- `signature(from = "spam", to = "matrix")` this is essentially equivalent to `as.matrix(object).`
- `signature(from = "spam", to = "list")` this is essentially equivalent to `triplet(object).`
- `signature(from = "spam", to = "vector")` this is essentially equivalent to `object@entries (structurebased=TRUE) or c(object).`
- `signature(from = "spam", to = "logical")` the entries are forced to logicals (nonzeros only in case of `structurebased=TRUE`).
- `signature(from = "spam", to = "integer")` the entries are forced to integers (nonzeros only in case of `structurebased=TRUE`).

Examples

```r
ifelse(diag.spam(2)*c(0,1), TRUE, FALSE)
```

---

**Coercion to a Vector**

Description

Coercion of spam matrices to proper vector objects

Usage

```r
## S4 method for signature 'spam'
as.vector(x, mode = "any")
```
**complexity**  

**Description**  

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

**Arguments**

- **x**: spam object.
- **mode**: character string naming an atomic mode or "any"/"list"/"expression".

**Details**

This coercion allows smooth transitions between different matrix formats, see example below. The Cholesky factors are first transformed to a spam object.

**Value**

If `structurebased=TRUE`, the vector `x@entries`. Conversely, if `structurebased=FALSE`, the result is identical to one with `as.vector(as.matrix(x))`.

**Author(s)**

Reinhard Furrer

**See Also**

`spam.options`

**Examples**

```r
x <- diag(2)  
ifelse(x, x, 1-x)  
ifelse(x, as.vector(x), 1-as.vector(x))

x <- diag.spam(2)  
spam.options( structurebased=FALSE)  
ifelse(x, as.vector(x), 1-as.vector(x))  
spam.options( structurebased=TRUE)  
ifelse(x, as.vector(x), 1-as.vector(x))
```
Details


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


See Also

det, solve, forwardsolve, backsolve and ordering.

Slot modification

Description

Modify slots of spam objects

Usage

```r
rowpointers( x ) <- value
colindices( x ) <- value
entries( x ) <- value
```

Arguments

- `x` a spam matrix
- `value` vector of appropriate length.

Details

Various tests are performed. Thus much slower than direct assignment.

Slot dimension should be changed through pad or dim

Value

Modified spam object.
**Author(s)**

Reinhard Furrer

**Examples**

```r
x <- diagspam(2)
rowpointers(x) <- c(1, 1, 3)

# The last line is equivalent to
x@rowpointers <- as.integer(c(1, 1, 3))
```

---

**covmat**

*Covariance functions*

---

**Description**

Evaluate a covariance function.

**Usage**

```r
covmat(h, theta, ..., type="sph")
```

```r
cov.exp(h, theta, ..., eps=Spam$eps)
cov.sph(h, theta, ..., eps=Spam$eps)
cov.nug(h, theta, ..., eps=Spam$eps)
cov.wu1(h, theta, ..., eps=Spam$eps)
cov.wu2(h, theta, ..., eps=Spam$eps)
cov.wu3(h, theta, ..., eps=Spam$eps)
cov.wend1(h, theta, ..., eps=Spam$eps)
cov.wend2(h, theta, ..., eps=Spam$eps)
cov.mat(h, theta, ..., eps=Spam$eps)
```

**Arguments**

- `h` object containing the lags.
- `theta` parameter of the covariance function, see ‘Details’.
- `type` covariance function specification.
- `...` arguments passed from other methods.
- `eps` tolerance level.
Details

covmat is a wrapper that calls the other functions according to the argument type. The nomenclature is similar to premat.
The parametrization is (range, sill, [smoothness], nugget), where only the range needs to be specified. Default values are (1,1,0). In case of negative parameter values, a warning is issued and the absolute value is retained. Although more cryptic, having all arguments as a single vector simplifies optimization with optim.
Currently, the functions distinguish between a sparse spam object and any other numeric type. In the future, this might change and appropriate methods will be implemented.

Value

Covariance function evaluated on h.

Author(s)

Reinhard Furrer

References

Any classical book about geostatistics.

See Also

precmat.

Examples

locs <- cbind(runif(10),runif(10))
h <- nearest.dist(locs, delta=.3)
Sigma <- cov.sph(h, c(.3, 1, .1))

## Not run:
h <- seq(0, to=1, length.out=100)
plot( h, cov.exp(h, c(1/3,1)), type='l', ylim=c(0,1))
type <- c("sph","wendland1","wendland2","wu1","wu2","wu3")
for (i in 1:6)
  lines( h, covmat(h, 1, type=type[i]), col=i+1)
legend('topright',legend=type, col=2:7, lty=1)

## End(Not run)
**Spam Matrix Crossproduct**

**Description**

Given matrices \( x \) and \( y \) as arguments, return a matrix cross-product. This is formally equivalent to (but usually slightly faster than) the call \( t(x) \%\% y \) (crossprod.spam) or \( x \%\% t(y) \) (tcrossprod.spam).

**Usage**

\[
crossprod.spam(x, y = \text{NULL})
\]

\[
tcrossprod.spam(x, y = \text{NULL})
\]

**Arguments**

\( x, y \) matrices: \( y = \text{NULL} \) is taken to be the same matrix as \( x \). Vectors are promoted to single-column or single-row matrices, depending on the context.

**Value**

A double matrix

**Note**

When \( x \) or \( y \) are not matrices, they are treated as column or row matrices.

**Author(s)**

Reinhard Furrer

**Examples**

\[
crossprod.spam(diag.spam(2),1:2)
\]
**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 `spam` or a Cholesky factor of class `spam.chol.NgPeyton`.
- `logarithm`: logical; if TRUE (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 us 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 logarithm is FALSE; otherwise the logarithm of the modulus.
- `sign`: integer; either +1 or -1 according to whether the determinant is positive or negative.

**Author(s)**

Reinhard Furrer

**References**

### diag

**Description**

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

**Usage**

```r
# diag(x)
diag(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`.

---

**See Also**

- `chol.spam`

**Examples**

```r
x <- spam(c(4,3,0,3,5,1,0,1,4),3)
det(x)
determinant(x)
det(chol(x))
```
If \( x \) is a vector of length one then \( \text{diag.spam}(x) \) returns an identity matrix of order the nearest integer to \( x \). The dimension of the returned matrix can be specified by \( \text{nrow} \) and \( \text{ncol} \) (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**

```r
diag.spam(2, 4) # 2x4
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.
```

---

### diff

**Lagged Differences**

**Description**

Returns suitably lagged and iterated differences.

**Usage**

```r
# diff.spam(x, lag = 1, differences = 1, ...)
## S4 method for signature 'spam'
diff(x, lag = 1, differences = 1, ...)
```

**Arguments**

- **x** a spam matrix containing the values to be differenced.
- **lag** an integer indicating which lag to use.
- **differences** an integer indicating the order of the difference.
- **...** further arguments to be passed to or from methods.
Value

A spam matrix with elements similar to `as.spam(diff(as.matrix(x), ...)).`

Author(s)

Reinhard Furrer

See Also

diff in base, `precmat`.

Examples

```r
# incidence matrix for a RW(3) model
d <- diff.spam(diag.spam(10), lag=1, differences=3)
t(D)
```

Description

Retrieve or set the dimension of an spam object.

Usage

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

Arguments

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

Details

In older version of `spam`, the behavior of the replacement method was different and is now implemented in `pad.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 rearranging.

Author(s)

Reinhard Furrer
See Also

pad.spam.

Examples

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

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

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

Description

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

Usage

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

Arguments

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

Details

`spamgetOption("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*spamgetOption("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

```r
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)
```

Description

These functions are simple wrappers to `nearest.dist` to be used in `fields`.

Usage

```r
spam_rdist( x1, x2, delta = 1)
spam_rdist.earth( x1, x2, delta = 1, miles=TRUE, R=NULL)
```

Arguments

- **x1**: Matrix of first set of locations where each row gives the coordinates of a particular point.
- **x2**: Matrix of second set of locations where each row gives the coordinates of a particular point.
- **delta**: only distances smaller than delta are recorded
- **miles**: For great circle distance: If true distances are in statute miles if false distances in kilometers.
- **R**: 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.
Details

These functions are wrappers to `rdist` and `rdist.earth` in `fields`. They are used to simplify the use of sparse matrices in functions like `mKrig`.

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

See Also

`nearest.dist`

Examples

```r
## Not run:
require(fields)
look <- mKrig(x,Y, Covariance="Wendland", dimension=2, k=1,
             cov.args=list(Distance='spam_rdist'))
## End(Not run)
```

---

**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

```r
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 "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

```r
## 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.data

**Meta-data about administrative districts of Germany**

**Description**

Data for the display of data over the administrative districts of Germany

**Format**

- `germany.info` is a list with elements:
  - `n` 544 (number of districts around 1990).
  - `xrep,yrep` representative coordinates of the districts (vectors of length 544)
  - `xlim,ylim` 2-vectors defining the limits of the districts.
  - `polyid` linking the polygons to the districts (599 vector).
  - `id` linking the districts to Community Identification Number.

- `germany.poly` defines the polygons. It is a 17965 by two matrix, each polygon separated by a row of NAs, each district by two rows.
- `germany` defines the polygons in form of a list (backwards compatibility).

**Details**

The representative coordinates are calculated based on the mean value of the polygon coordinates. This creates sometimes strange values, e.g., district Leer.

**Author(s)**

Reinhard Furrer

**References**

The meta-data has been constructed based on (essentially) files from the package **INLA**, see `demo(Bym)`.

See also [http://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel](http://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel) and [http://de.wikipedia.org/wiki/Liste_der_Landkreise_in_Deutschland](http://de.wikipedia.org/wiki/Liste_der_Landkreise_in_Deutschland)

**See Also**

- `germany.plot Oral`

**Examples**

```r
# Plot the Bundeslaender:
geometry.plot(germany.info$id%%1000,col=rep(2:8,3), legend=FALSE)
```
Germany.plot

Plot administrative districts of Germany

Description

Displaying data over the administrative districts of Germany

Usage

```r
 germany.plot(vect, col=NULL, zlim=range(vect), legend=TRUE,
           main=NULL, cex.axis=1, add=FALSE, ...)
```

Arguments

- `vect` vector of length 544
- `col` color scheme to be used. By default uses `colorRampPalette(brewer.pal(9,"Blues"))(100)``.``
- `zlim` the minimum and maximum values for which colors should be plotted, defaulting to the range of data.
- `legend` Should the legend be added, see ‘Details’.
- `main` an overall title for the plot.
- `cex.axis` label size of legend.
- `add` logical, if true adds to current plot.
- `...` additional arguments passed to `polygon`.

Details

The legend is only added, provided (a) function `image.plot` is available.

The perfect position of the legend is an art per se and depends on various `par` parameters. One possibility for finer control is to not plot it and to manually call the function `image.plot` of `fields`.

Author(s)

Reinhard Furrer

References

See also [http://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel](http://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel) and [http://de.wikipedia.org/wiki/Liste_der_Landkreise_in_Deutschland](http://de.wikipedia.org/wiki/Liste_der_Landkreise_in_Deutschland)

See Also

Oral.
Examples

```r
data( Oral)
germany.plot( Oral$Y/Oral$E)
```

# Plot the Bundeslaender:
germany.plot(germany.info$id%/%1000,col=rep(2:8,3), legend=FALSE)

---

### Description

For two (MCMC) chains of the same length trace plots and a scatter plot are drawn.

### Usage

```r
grid_trace2(chain1, chain2 = NULL,
            xlim = NULL, ylim1 = NULL, ylim2=NULL,
            chain1.lab = "", chain2.lab = "", main = "",
            chain1_yaxis_at = NULL, chain2_yaxis_at = NULL,
            log = FALSE,
            cex_points = unit(0.2, "mm"),
            cex_main = unit(1.2, "mm"),
            lwd_lines = unit(0.1, "mm"),
            lwd_frame = unit(0.8, "mm"),
            draw = TRUE)
```

### Arguments

- **chain1**: Numeric vector or a matrix with two columns.
- **chain2**: Numeric vector of same length as chain1. (Only used if chain1 is specified as vector).
- **xlim**: x axis limits of both chains (numeric vector of length 2).
- **ylim1**: y axis limits of chain 1 (numeric vector of length 2).
- **ylim2**: y axis limits of chain 2 (numeric vector of length 2).
- **chain1.lab**: Label of chain 1 (character of length 1).
- **chain2.lab**: Label of chain 2 (character of length 1).
- **main**: Title (character of length 1).
- **chain1_yaxis_at**: Points at which tick-marks are drawn for chain 1.
- **chain2_yaxis_at**: Points at which tick-marks are drawn for chain 2.
- **log**: Logical, should the date be log transformed?
DESCRIPTION

This function takes a grob object (e.g. created with package grid) and adds a zoom window.

Usage

```r
grid_zoom(inputGro = pointsGro(runif(200), runif(200)),
          inputViewport = viewport(name = 'main',
          x = 'topleft', y, just,
          ratio = c(0.3, 0.4), zoom_xlim, zoom ylim,
          rect = TRUE, rect_lwd = 1, rect_fill = 'gray92',
          draw = TRUE, zoom_fill = 'white',
          zoom_frame_gp = gpar(lwd = 1),
          zoom_gp = NULL, zoom_xaxis = xaxisGro(main = FALSE),
          zoom_yaxis = NULL)
```
Arguments

- **inputGrob**: A grob object, e.g created with package grid.
- **inputViewport**: Viewport related to inputGrob.
- **x**: Specifies the x coordinate of the zoom window. Alternatively it can be set to 'topleft', 'topright', 'botmleft' or 'botmrigh
- **y**: Specifies the y coordinate of the zoom window.
- **just**: Specifies the justification of the zoom window.
- **ratio**: Specifies size of the zoom window relative to the main window.
- **zoom_xlim**: Specifies xlim value of the zoom window.
- **zoom_ylim**: Specifies ylim value of the zoom window.
- **rect**: Logical, if TRUE a rectangle of the zoom region is draw in the main window.
- **rect_lwd**: lwd of the rectangle.
- **rect_fill**: fill of the rectangle.
- **draw**: logical, if TRUE the returned grob object is also drawn.
- **zoom_fill**: fill color of the zoom window.
- **zoom_frame_gp**: gpar() of the frame of the zoom window.
- **zoom_gp**: gpar() of the inputGrob in the zoom viewport.
- **zoom_xaxis**: xaxisGrob() to draw for the zoom window.
- **zoom_yaxis**: yaxisGrob() to draw for the zoom window.

Details

A zoom plot does only make sense if all objects of the inputGrob are specified in native units. Additional caution me be require for certain grobs: e.g. a zoom of a circleGrob() is problematic if the x and y axis are stretched by a different amount.

Value

A grob object.

Author(s)

Florian Gerber <florian.gerber@math.uzh.ch>

See Also

grid_trace2
Examples

```r
## -- Example 1 --
set.seed(133)
grid_zoom(inputGrobs = pointsGrobs(runif(200), runif(200)),
          inputViewport = viewport(name = 'main'),
          zoom_xlim = c(0.2, 0.3), zoom_ylim = c(0.2, 0.3))

## -- Example 2 --
## initial plot
grid.newpage()
vp <- viewport(width = 0.8, height = 0.8, clip = 'on')
gt <- gTree(children = list(polylineGrobs(x = c((0:4)/10, rep(0.5, 5), (10:6)/10, rep(0.5, 5)),
                                    y = c(rep(0.5, 5), (10:6)/10, rep(0.5, 5), (0:4)/10),
                                    id = rep(1:5, 4), default.units = 'native',
                                    gp = gpar(col = 1:5, lwd = 3)),
                       pointsGrobs(runif(1000), runif(1000), pch = '.', gp = gpar(cex = 3)),
                       rectGrobs(gp = gpar(lwd = 3))))
pushViewport(vp)
grid.draw(gt)

## plot with zoom window
grid.newpage()
grid_zoom(inputGrobs = gt,
          inputViewport = vp,
          x = 'topright', zoom_xlim = c(0.6, 0.73), zoom_ylim = c(0.3, 0.43), ratio = 0.4,
          zoom_xaxis = NULL, zoom_gp = gpar(cex = 3))
```

---

**head**

*Return the First or Last Part of an Object*

**Description**

Returns the upper left or lower right part of a `spam` object.

**Usage**

```r
## S4 method for signature 'spam'
head(x, n = 6L, m = n, ...)  
## S4 method for signature 'spam'
tail(x, n = 6L, m = n, addrownums = TRUE, ...)
```

**Arguments**

- `x`  
  A `spam` object

- `n`  
  A single integer. If positive, size for the resulting object: number of elements for a vector (including lists), rows for a matrix or data frame or lines for a function. If negative, all but the `n` last/first number of elements of `x`.
Details

For matrices, 2-dim tables and data frames, head() (tail()) returns the first (last) \( n \) rows and \( m \) columns when \( n > 0 \) or all but the last (first) \( n \) rows when \( n < 0 \) (with similar behavior for \( m \)).

tail() will add row and column names of the form ":[n,]" and ":[,n]" to the result, so that it looks similar to the last lines and columns of \( x \) when printed. Setting `addrownums = FALSE` suppresses this behavior.

A method for `spam.chol.NgPeyton` objects is exported as well.

Value

An regular matrix.

Author(s)

Reinhard Furrer

Examples

```r
head( precmat.RW2( 10))
tail( precmat.season(n=10, season=3), n=4, m=10)
```

Description

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

Usage

```r
# S4 method for signature 'spam'
image(x, cex = NULL, ...)
```

Arguments

- **x**: matrix of class `spam` or `spam.chol.NgPeyton`.
- **cex**: for very large matrices, the dot size may need to be scaled.
- **...**: any other arguments passed to `image.default` and `plot`. 
import

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

```r
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))
```

Description

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

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 return (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)
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.
kroncker

Examples

```r
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

```r
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 `%*%`. Hence, the result of `kronecker(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
kronecker.spam( diag(2), array(1:4, c(2, 2)))

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

# Notice the preservation of sparsity structure:
kronecker( 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 spamgetOption("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)
Create Precision Matrices

Description

Creates precision matrices for gridded GMRF.

Usage

```r
precmat.GMRFreglat(n, m, par, model = "m1p1", eps = .Spam$eps)
```

Arguments

- `n`: first dimension of the grid.
- `m`: second dimension of the grid.
- `par`: parameters used to construct the matrix.
- `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).

The model specification "m" determines the complexity and "p" the number of parameters.

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

`precmat`, `toeplitz.spam`, `kronecker.spam`
Examples

```r
as.matrix(precmat.GMRFreglat(4, 3, c(.4), 'm1p1'))
as.matrix(precmat.GMRFreglat(4, 3, c(.4,.3), 'm1p2'))
as.matrix(precmat.GMRFreglat(4, 3, c(.4,.3,.2), 'm2p3'))
as.matrix(precmat.GMRFreglat(4, 3, c(.4,.3,.2,.1), 'm2p4'))
```

# up to the diagonal, the following are equivalent:
cleanup( precmat.IGMRFreglat(3,4) -
         precmat.GMRFreglat(3,4,1, 'm1p1'))

---

**map.landkreis**

*Administrative districts of Germany*

Description

Displaying data over the administrative districts of Germany

Usage

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

Arguments

- `data` vector of length 544
- `col` color scheme to be used. By default uses `tim.colors` 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

The function `germany.plot` super-seeds `map.landkreis` (it is several factors faster).

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`. 

### 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 when `structurebased=FALSE`. 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.

Value

If `structurebased=TRUE`, all functions operate on the vector `x@entries` and return the result thereof. Conversely, if `structurebased=FALSE`, the result is identical to one with `as.matrix(x)` input and an `as.spam purger`.

Author(s)

Reinhard Furrer

See Also

`Summary.spam`, `Ops.spam` and `Math2.spam`

Examples

groupMembers("Math")

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

spam.options(structurebased=FALSE)
cos(smat)

sqrt(smat)

---

Math2  

**Rounding of Numbers**

Description

Applies the Math2 group functions to 'spam' objects

Usage

```r
## S4 method for signature 'spam'
round(x, digits = 0)
## S4 method for signature 'spam'
signif(x, digits = 6)
```
Arguments

- **x**: spam object.
- **digits**: integer indicating the precision to be used.

Value

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

Author(s)

Reinhard Furrer

See Also

- Ops.spam and Math.spam

Examples

```r
getGroupMembers("Math2")

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

---

**mle**

Maximum likelihood estimates

Description

Maximum likelihood estimates of a simple spatial model

Usage

```r
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.nomega.spam(y, distmat, Covariance, theta0, thetalower, thetauupper, optim.control=NULL, Rstruct = 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 nearest.dist.
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 initial values.
thetalower,thetaupper lower and upper bounds of the parameter theta.
optim.control arguments passed to optim.
hessian Logical. Should a numerically differentiated Hessian matrix be returned?
... additional arguments passed to chol.

Details

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

\[ y \sim \mathcal{N}_n( X \beta, \Sigma(h;\theta) ) \]

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

\[ h \rightarrow \frac{R}{2 \sin(h/2)} \]

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 image 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 \( X \) or \( \beta \) to zero.

**Value**

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

**Author(s)**

Reinhard Furrer

**See Also**

covmat

**Examples**

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

# 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(locs)) # design matrix

distmat <- nearest.dist(locs, upper=NULL) # distance matrix
Sigma <- cov.sph(distmat, trutheta) # true covariance matrix

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

# Here is the negative 2 log likelihood:
neg2loglikelihood.spam(y, X, distmat, cov.sph, 
truebeta, trutheta, thetalower=c(0,0,0), thetupper=c(1,Inf,Inf))

# We pass now to the mle:
res <- mle.spam(y, X, distmat, cov.sph, 
truebeta, trutheta, thetalower=c(0,0,0), thetupper=c(1,Inf,Inf))

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

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.

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 formerly depreciated arguments eps and diag are now eliminated.
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

**See Also**

`spam_rdist`

**Examples**

```r
# 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 as.dist(...)
nx <- 4
x <- expand.grid(as.double(1:nx),as.double(1:nx))
sum( ( as.dist(nearest.dist( x, delta=nx*2)) -
       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))
```

---

<table>
<thead>
<tr>
<th>options</th>
<th>Options Settings</th>
</tr>
</thead>
</table>

**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.
**options**

**Usage**

```r
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.
cex = 1200: default dot size for image or display.
structurebased = FALSE: should operations be carried out on the nonzero entries (the structure) or including the zeros.
inefficiencywarning = 1e6: issue a warning when inefficient operations are performed and the matrix exceeds the specified size. Valid value is a postive integer or a logical. TRUE corresponds to 1 (always), FALSE to Inf.
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.

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.

NAOK=FALSE: logical determines if NA, NaN and Inf are allowed to Fortan. Setting to TRUE allows to work with these but full functionality has not been tested.
safemodevalidity=TRUE: logical determines if sanity check is performed when constructing sparse matrices. Default is safer but somewhat slower.

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.
nonznearestdistnnz=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

Functions influenced by these options include: print.spam, display.spam, image.spam, upper.tri.spam, chol.spam, nearest.dist, etc.

Examples

```r
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
```
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) with larynx cancer cases from the package INLA as well, with an additional smoking covariate.

Source

The data is available from the package INLA distributed from www.r-inla.org or from http://www.math.ntnu.no/~hrue/GMRF-book/oral.txt

References


See Also

germany.plot.
ordering

Description

Extract the (inverse) permutation used by the Cholesky decomposition

Usage

ordering( x, inv=FALSE)

Arguments

x object of class spam.chol.method returned by the function chol.

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.spam, solve.spam.

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)
Description

Resets the dimension of a (spam) matrix by truncation or zero padding.

Usage

```
pad(x) <- value
```

Arguments

- `x` a spam matrix
- `value` A numeric two-vector.

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).

Value

A (spam) matrix of dimension value where truncation or padding has been used.

Author(s)

Reinhard Furrer
permutation

See Also
dim.spam.

Examples

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

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

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

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.
powerboost

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]

<table>
<thead>
<tr>
<th>powerboost</th>
<th>Specific options Setting</th>
</tr>
</thead>
</table>

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.
IGMRF Precision Matrices

Description

Fast ways to create sparse precision matrices for various IGMRF.

Usage

```r
precmat(n, season=12, m=n, A=NULL, order=1, ... , type="RW1")
precmat.RW1(n)
precmat.RW2(n)
precmat.RWn(n, order=3)
precmat.season(n, season=12)
precmat.IGMRFreglat(n, m, order=1, anisotropy=1)
precmat.IGMRFirreglat(A, eps=.Spam$eps)
```

Arguments

- `n`: dimension of the field.
- `type`: the type of the IGMRF.
- `season`: length of season.
- `m`: second dimension (in case of a regular lattice).
- `A`: adjacency matrix (see below).
- `order`: order for higher order RWs.
- `anisotropy`: anisotropy factor, between 0 and 2.
- `eps`: tolerance level.
- `...`: arguments passed to individual functions.

Details

`precmat` is a wrapper that calls the other functions according to the argument `type`. Implements many of the precision matrices discussed in Chapter 3 of Rue and Held (2005). For example, `precmat.RW1`, `precmat.RW2` and `precmat.season` are given in equations (3.22), (3.40) and (3.59); `precmat.IGMRFreglat` on page 107. Note that for the latter we reverse the order of the dimension here!

If adjacency matrix is a regular matrix, it is coerced to a `spam` object. Only the structure is used. Make sure, that the diagonal is empty.

Value

A sparse precision matrix.

Author(s)

Reinhard Furrer
References

Rue and Held (2005).

See Also

precmat.GMRFreglat, rmvnorm.prec, adjacency.landkreis.

Examples

```r
n <- 10
Q <- precmat.RW2(n)
# rmvnorm.prec(1, Q=Q) # does not work, because the matrix is singular.
Q
```

Description

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

Usage

```r
## S4 method for signature 'spam'
print(x, ...)

## S4 method for signature 'spam'
summary(object, ...)
```

Arguments

- `x` matrix of class spam or spam.chol.method.
- `object` matrix of class spam or spam.chol.method.
- `...` 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.
Draw Multivariate Normals

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

**Usage**

```r
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 \texttt{rmvnorm.prec} and \texttt{rmvnorm.canonical} do not require sparse precision matrices. For \texttt{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 \texttt{Rstruct} in which only updates are performed (i.e., update\_spam\_chol\_NgPeyton instead of a full \texttt{chol}).

Author(s)

Reinhard Furrer

References

See references in \texttt{chol}.

See Also

\texttt{chol} and \texttt{ordering}.

Examples

\begin{verbatim}
# 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="m")

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

# 'solve' step by step:
b <- rnorm( n)
R <- chol(Sigmainv)
norm( backsolve( R, forwardsolve( R, b))- solve( Sigmainv, b )
     norm( backsolve( R, forwardsolve( R, diag(n)))- Sigma )
\end{verbatim}
Draw Constrained Multivariate Normals

### Description
Fast ways to draw multivariate normals with linear constrains when the variance or precision matrix is sparse.

### Usage
```r
rmvnorm.const(n, mu = rep(0, nrow(Sigma)), Sigma, Rstruct = NULL,
    A = array(1, c(1,nrow(Sigma))), a=0, U=NULL, ...)
rmvnorm.prec.const(n, mu = rep(0, nrow(Q)), Q, Rstruct = NULL,
    A = array(1, c(1,nrow(Q))), a=0, U=NULL, ...)
rmvnorm.canonical.const(n, b, Q, Rstruct = NULL,
    A = array(1, c(1,nrow(Q))), a=0, U=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`.
- **A**: Constrain matrix.
- **a**: Constrain vector.
- **U**: see below.
- **...**: 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 `chol`).

### Author(s)
Reinhard Furrer
rowSums

References

See references in `chol`.

See Also

`rmvnorm.spam`.

Examples

# to be filled in

---

**rowSums**

*Form Row and Column Sums and Means*

**Description**

Form row and column sums and means for sparse `spam` matrices

**Usage**

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

**Arguments**

- `x`: a `spam` object
- `na.rm`: currently ignored
- `dims`: ignored as we have only two dimensions.
- `...`: potentially further arguments from other methods.

**Details**

Depending on the flag.

**Value**

Vector of appropriate length.

**Author(s)**

Reinhard Furrer

**See Also**

`apply.spam, spam.options`.
s3only

Wrappers for Sparse Matrices

Description

These functions are convenient wrappers for spam objects to classical matrix operations.

Usage

```r
var.spam(x, ...)  
eigen.spam(x, ...)
```

```r
## S3 method for class 'spam'
var(x, ...)  
eigen(x, ...)
```

Arguments

- `x` matrix of class spam.
- `...` further arguments passed to or from other methods.

Details

There is probably no point in fully defining methods here. Typically, these functions do not exploit sparsity structures. Hence, for very large matrices, warnings may be posted.

Value

Depends on function...

Author(s)

Reinhard Furrer

See Also

Option "inefficiencywarning" in spam.options.

Examples

```r
eigen(diag.spam(3))
```


**Description**

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

**Usage**

```r
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 `abs(x) < eps` set to zero. Defaults to `eps = .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 indicies (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=NAOK` 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).
spam operations

Author(s)
Reinhard Furrer

References

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)

class <- getClass("spam")

spam.options(NAOK=TRUE)
as.spam( c(1, 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 $l_1$ 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 `%*%` 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 non-diagonal 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 `%+%` efficiently adds a diagonal matrix (in vector form) and a sparse matrix, similarly to the operator `%*%`.

### References

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

```r
# 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)
```

---

**spam solve**  
Linear Equation Solving for Sparse Matrices
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

```r
## S4 method for signature 'spam'
solve(a, b, Rstruct=NULL, ...)
## S4 method for signature 'spam'
backsolve(r, x, ...)  
## S4 method for signature 'spam'
forwardsolve(l, x, ...)
## S4 method for signature 'spam'
chol2inv(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.
- `Rstruct`: the Cholesky structure of `a`.
- `...`: further arguments passed to or from other methods, see ‘Details’ below.

Details

We can solve \( A \times x = b \) by first computing the Cholesky decomposition \( A = T R T \), then solving \( T^t R^t y = b \) for \( y \), and finally solving \( R^t x = y \) 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 call, 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. `chol2inv(x)` is a faster way to `solve(x)`. 

**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**

`chol.spam` and `ordering`.

**Examples**

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

Sigma <- solve(Sigmaintv)  # for verification
iidsample <- array(rnorm(N*n),c(n,N))
mvsample <- backsolve(chol(Sigmaintv), iidsample)

# compare with:
mvsample <- backsolve(chol(as.matrix(Sigmaintv)), iidsample, n)

# 'solve' step by step:
b <- rnorm(n)
R <- chol(Sigmaintv)
norm(backsolve(R, forwardsolve(R, b))-solve(Sigmaintv, b))
norm(backsolve(R, forwardsolve(R, diag(n)))-Sigma)
```
# 'update':
R1 <- update(r, Sigmainv + diag.spam(n))

---

### 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, rowpointes, 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 \( x@\text{rowpointers}[i] \). The length of row \( i \) is determined by \( x@\text{rowpointers}[i+1] - x@\text{rowpointers}[i] \). The column indices of \( x \) are stored in the \( x@\text{colindices} \) vector. The column index for element \( x@\text{entries}[k] \) is \( x@\text{colindices}[k] \).

**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 \( \text{entries} \) and \( \text{colindices} \).
- **dimension**: Object of class "integer" specifying the dimension of the matrix.

**Methods**

- **as.matrix** signature(\( x = \text{"spam"} \)): transforming a sparse matrix into a regular matrix.
- **as.vector** signature(\( x = \text{"spam"} \)): transforming a sparse matrix into a vector (dependings on structurebased) see `as.vector.spam` for details.
- **as.spam** signature(\( x = \text{"spam"} \)): cleaning of a sparse matrix.
- \( \langle- \) signature(\( x = \text{"spam"} \), i,j, value): assigning a sparse matrix. The negative vectors are not implemented yet.
- \( \langle \) signature(\( x = \text{"spam"} \), 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.

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

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

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.

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

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

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.
na.rm a logical indicating whether missing values should be removed.
Details

The na.rm argument is only meaningful if NAOK=TRUE.

Value

If structurebased=TRUE, all functions operate on the vector x@entries and return the result thereof. Conversely, if structurebased=FALSE, the result is identical to one with as.matrix(x) input.

Author(s)

Reinhard Furrer

See Also

Math.spam and Math2.

Examples

groupMembers("Summary")

smat <- diag.spam( runif(15))
range(smat)
spam.options(structurebased=FALSE)
range(smat)

## Not run:
max( log(spam(c(1,-1))), na.rm=TRUE)

## End(Not run)
# allow 'NA''s first:
spam.options( NAOK=TRUE)
max( log(spam(c(1,-1))), na.rm=TRUE)
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 abs(x) <= eps set to zero. Defaults to eps = .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))

Transform a spam format to triplets

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

```r
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 neighbors of county k and counties i and j are not neighbors.

See Also

- map from maps.
Examples

```r
# 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/](http://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/](http://www.image.ucar.edu/GSP/Data/US.monthly.met/)

References


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-class), 70
!=, spam-method (spam operations), 66
*Topic **IO**
  import, 37
  options, 50
*Topic **algebra**
  apply, 7
  bandwidth, 8
  bdiag, 9
  chol, 12
  circulant, 14
  cleanup, 15
  complexity, 17
  covmat, 19
  crossprod, 21
  det, 22
  diag, 23
  import, 37
  kronecker, 40
  lower.tri, 41
  makeprec, 42
  mle, 46
  nearestdist, 49
  ordering, 54
  precmat, 58
  rmvnorm, 60
  rmvnorm.const, 62
  s3only, 64
  spam, 65
  spam operations, 66
  spam solve, 67
  toeplitz, 74
*Topic **array**
  allequal, 5
  apply, 7
  bandwidth, 8
  bdiag, 9
  cbind, 10
  circulant, 14
  constructors, 18
  crossprod, 21
  det, 22
  diag, 23
  diff, 24
  dim, 25
  fields wrapper, 27
  foreign, 28
  import, 37
  isSymmetric, 39
  kronecker, 40
  lower.tri, 41
  makeprec, 42
  nearestdist, 49
  pad, 55
  permutation, 56
  toeplitz, 74
  triplet, 75
*Topic **classes**
  spam-class, 70
  spam.chol.NgPeyton-class, 72
*Topic **datasets**
  Oral, 53
  UScounties, 76
  USprecip, 77
*Topic **documentation**
  . SPAM .. 3
*Topic **environment**
  options, 50
  powerboost, 57
  version, 78
*Topic **error**
  options, 50
*Topic **grid**
  grid_trace2, 32
  grid_zoom, 33
*Topic **hplot**
  adjacency, 4
  display, 26
(spam-class), 70
[<-, spam, missing, missing-method (spam operations), 66
[<-, spam, missing, vector, ANY-method (spam-class), 70
[<-, spam, missing, vector, numeric-method (spam-class), 70
[<-, spam, missing, vector, spam-method (spam-class), 70
[<-, spam, missing, vector-method (spam operations), 66
[<-, spam, spam, missing, ANY-method (spam-class), 70
[<-, spam, spam, missing, numeric-method (spam-class), 70
[<-, spam, spam, missing-method (spam operations), 66
[<-, spam, vector, missing, ANY-method (spam-class), 70
[<-, spam, vector, missing, numeric-method (spam-class), 70
[<-, spam, vector, missing, spam-method (spam-class), 70
[<-, spam, vector, missing-method (spam operations), 66
[<-, spam, vector, vector, ANY-method (spam-class), 70
[<-, spam, vector, vector, numeric-method (spam-class), 70
[<-, spam, vector, vector, spam-method (spam-class), 70
[<-, spam, vector, vector-method (spam operations), 66
[<-, spam (spam operations), 66
%%*, ANY, ANY-method (spam operations), 66
%%*, matrix, spam-method (spam operations), 66
%%*, numeric, spam-method (spam operations), 66
%%*, spam, matrix-method (spam operations), 66
%%*, spam, numeric-method (spam operations), 66
%%*, spam, spam-method (spam operations), 66
%%*, methods (spam operations), 66
%%%, spam-method (spam operations), 66
%%%, spam-method (spam operations), 66
%%%, matrix, ANY-method (spam operations), 66
%%%, matrix, spam-method (spam operations), 66
%%%, numeric, matrix-method (spam operations), 66
%%%, numeric, numeric-method (spam operations), 66
%%%, numeric, spam-method (spam operations), 66
%%%, spam-method (spam operations), 66
%d%%, (spam methods), 66
%d%%, (spam operations), 66
%d%%, matrix, ANY-method (spam operations), 66
%d%%, matrix, spam-method (spam operations), 66
%d%%, numeric, matrix-method (spam operations), 66
%d%%, numeric, numeric-method (spam operations), 66
%d%%, numeric, spam-method (spam operations), 66
%d%%, spam-method (spam operations), 66
%d%%, spam, ANY-method (spam operations). 66
%d%%, spam, numeric-method (spam operations), 66
%d%%, spam, spam-method (spam operations), 66
%d%%, (spam operations), 66
%d%%, matrix, ANY-method (spam operations), 66
%d%%, matrix, spam-method (spam operations), 66
%d%%, numeric, matrix-method (spam operations), 66
%d%%, numeric, numeric-method (spam operations), 66
%d%%, numeric, spam-method (spam operations), 66
%d%%, spam-method (spam operations), 66
%d%%, spam, ANY-method (spam operations), 66
%d%%, spam, numeric-method (spam operations), 66
%d%%, spam, spam-method (spam operations), 66
%d%%, (spam operations), 66
%d%%, matrix, ANY-method (spam operations), 66
%d%%, matrix, spam-method (spam operations), 66
%d%%, numeric, matrix-method (spam operations), 66
%d%%, numeric, numeric-method (spam operations), 66
%d%%, numeric, spam-method (spam operations), 66
%d%%, spam-method (spam operations), 66
<-, ANY, spam-method (spam operations), 66
<-, spam, ANY-method (spam operations), 66
<-, spam, spam-method (spam operations), 66
^, ANY, spam-method (spam operations), 66
^, spam, ANY-method (spam operations), 66
^, spam, spam-method (spam operations), 66
^, spam-method (spam operations), 66
abs, spam-method (Math), 44
abs.spam (Math), 44
acos, spam-method (Math), 44
acos.spam (Math), 44
acosh, spam-method (Math), 44
acosh.spam (Math), 44
adjacency, 4
adjacency.landkreis, 59
all, spam-method (Summary), 73
all.equal, matrix, spam-method
(alloequal), 5
all.equal, spam, spam-method (allequal), 5
all.equal, spam, 15, 39
all.equal, spam (allequal), 5
all.spam (Summary), 73
allequal, 5
any, spam-method (Summary), 73
any.spam (Summary), 73
apply, 7
apply.spam, 63
Arith, ANY, spam-method (spam-class), 70
Arith, spam, ANY-method (spam-class), 70
Arith, spam, spam-method (spam-class), 70
as.dgCMatrix.spam (foreign), 28
as.dgRMatrix.spam (foreign), 28
as.matrix, spam-method (spam-class), 70
as.matrix, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
as.matrix.csr.spam (foreign), 28
as.matrix.spam (spam-class), 70
as.spam (spam), 65
as.spam, dist-method (spam), 65
as.spam, list-method (spam), 65
as.spam, matrix-method (spam), 65
as.spam, numeric-method (spam), 65
as.spam, spam-method (spam), 65
as.spam, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
as.spam.chol.NgPeyton (spam), 65
as.spam.dgCMatrix (foreign), 28
as.spam.dgRMatrix (foreign), 28
as.spam.dist (spam), 65
as.spam.list (spam), 65
as.spam.matrix (spam), 65
as.spam.matrix.csr (foreign), 28
as.spam.numeric (spam), 65
as.spam.spam (spam), 65
as.vector, spam-method (Coercion), 16
as.vector, spam.chol.NgPeyton-method
(Coercion), 16
as.vector.spam, 70
as.vector.spam (Coercion), 16
asin, spam-method (Math), 44
asin, spam (Math), 44
asinh, spam-method (Math), 44
asinh.spam (Math), 44
assign.spam (spam operations), 66
atan, spam-method (Math), 44
atan, spam (Math), 44
atanh, spam-method (Math), 44
atanh, spam (Math), 44
backsolve, 18
backsolve (spam solve), 67
backsolve, ANY-method (spam solve), 67
backsolve, matrix-method (spam solve), 67
backsolve, spam-method (spam solve), 67
backsolve, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
backsolve-methods (spam solve), 67
backsolve, spam, 13
backsolve, spam (spam solve), 67
bandwidth, 8
bd diag, 9
c, spam-method (spam-class), 70
c, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
cbind, 10, 71
cbind, spam-method (cbind), 10
cbind, spam (cbind), 10
ceiling, spam-method (Math), 44
ceiling, spam (Math), 44
chol, 12, 22, 61, 63, 69, 71, 73
chol, ANY-method (chol), 12
chol, matrix-method (chol), 12
chol, spam-method (chol), 12
chol, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
chol.spam, 23, 52, 54, 69
chol.spam (chol), 12
chol2inv, spam-method (spam solve), 67
chol2inv, spam.chol.NgPeyton-method
(spam solve), 67
chol2inv, spam (spam solve), 67
circulant, 14, 15
circulant.spam, 75
cleanup, 6, 15, 39
coerce, spam, integer-method
(coerce-methods), 16
coerce, spam, list-method
(coerce-methods), 16
coerce, spam, logical-method
(coerce-methods), 16
coerce, spam, matrix-method
(coerce-methods), 16
coerce, spam, vector-method
(coerce-methods), 16
c coerce-methods, 16
coerce.spam (coerce-methods), 16
Coercion, 16
colindices (constructors), 18
colindices<- (constructors), 18
colMeans (rowSums), 63
colMeans, spam-method (rowSums), 63
colMeans (rowSums), 63
colSums (rowSums), 63
colSums, spam-method (rowSums), 63
colSums (rowSums), 63
Compare, ANY, spam-method (spam-class), 70
Compare, spam, ANY-method (spam-class), 70
Compare, spam, spam-method (spam-class), 70
complexities (complexity), 17
complexity, 17
constructors, 18
cos, spam-method (Math), 44
cos.spam (Math), 44
cosh, spam-method (Math), 44
cosh.spam (Math), 44
cov.exp (covmat), 19
cov.mat (covmat), 19
cov.nug (covmat), 19
cov.sph (covmat), 19
cov.wend1 (covmat), 19
cov.wend2 (covmat), 19
cov.wul (covmat), 19
cov.wu2 (covmat), 19
cov.wu3 (covmat), 19
covmat, 19, 48
crossprod, 21
crossprod, ANY, spam-method (spam-class), 70
crossprod, spam, missing-method
(spam-class), 70
cummax, spam-method (Math), 44
cummax.spam (Math), 44
cummin, spam-method (Math), 44
cummin.spam (Math), 44
cumprod, spam-method (Math), 44
cumprod.spam (Math), 44
cumsum, spam-method (Math), 44
cumsum.spam (Math), 44
det, 18, 22, 72
det, spam-method (det), 22
det, spam.chol.NgPeyton-method (det), 22
det.spam, 13
det.spam (det), 22
determinant (det), 22
determinant, spam-method (det), 22
determinant, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
determinant.spam (det), 22
diag, 23, 41, 66, 71
diag, ANY-method (diag), 23
diag, spam-method (diag), 23
diag, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
diag.assign, spam-method (diag), 23
diag.of.spam (diag), 23
diag.spam, 9, 10
diag.spam (diag), 23
diag.spam<- (diag), 23
diag<- (diag), 23
diag<-, ANY-method (diag), 23
diag<-, spam-method (diag), 23
diag<-.spam (diag), 23
diff, 24
diff, spam-method (diff), 24
diff.spam (diff), 24
digamma, spam-method (Math), 44
digamma.spam (Math), 44
dim, 25, 71
dim, ANY-method (spam operations), 66
dim, spam-method (spam operations), 66
dim, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
dim.spam, 56
dim<-, spam-method (dim), 25
dim<-, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
dim<-.spam (dim), 25
dimension<- (constructors), 18
display, 26, 37, 71, 72
display, spam-method (display), 26
display, spam.chol.NgPeyton-method
(spam.chol.NgPeyton-class), 72
display.spam, 52
display.spam (display), 26
dist.spam (nearestdist), 49
INDEX

distance (nearestdist), 49
eigen.spam (s3only), 64
entries (constructors), 18
entries<- (constructors), 18
exp.spam-method (Math), 44
exp.spam (Math), 44
fields wrapper, 27
floor.spam-method (Math), 44
foreign, 28, 66
forwardsolve, 18
forwardsolve (spam solve), 67
forwardsolve, ANY-method (spam solve), 67
forwardsolve, matrix-method (spam solve), 67
forwardsolve, spam-method (spam solve), 67
forwardsolve, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
forwardsolve-methods (spam solve), 67
forwardsolve.spam, 13
forwardsolve.spam (spam solve), 67
gamma, spam-method (Math), 44
gamma.spam (Math), 44
germany (germany.data), 30
germany.data, 30
germany.graph (adjacency), 4
germany.plot, 5, 30, 31, 43, 44, 53
grid_traceR, 32
grid_zoom, 33
head, 35
head, spam-method (head), 35
head, spam.chol.NgPeyton-method (head), 35
head.spam (head), 35
image, 26, 36, 37, 71, 72
image, spam-method (image), 36
image, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
image.spam, 52
image.spam (image), 36
import, 37
initialize, spam-method (spam), 65
is.spam (spam), 65
isSymmetric, 39
isSymmetric, spam-method (isSymmetric), 39
isSymmetric.spam, 6, 15
isSymmetric.spam (isSymmetric), 39
kronecker, 40
kronecker, ANY, spam-method (spam-class), 70
kronecker, spam, ANY-method (spam-class), 70
kronecker, spam, spam-method (spam-class), 70
kronecker.spam, 42
length, spam-method (spam-class), 70
length, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
length<-, spam-method (spam-class), 70
length<-, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
lgamma, spam-method (Math), 44
lgamma.spam (Math), 44
log, spam-method (Math), 44
log.spam (Math), 44
log10, spam-method (Math), 44
log10.spam (Math), 44
log2, spam-method (Math), 44
log2.spam (Math), 44
lower.tri, 24, 41, 71
lower.tri, spam-method (spam-class), 70
makeprec, 42
map.landkreis, 43
Math, 44, 71
Math, spam-method (Math), 44
Math.spam, 46, 74
Math.spam (Math), 44
Math2, 45, 71, 74
Math2, spam, numeric-method (Math2), 45
Math2, spam-method (Math2), 45
Math2.spam, 45
Math2.spam (Math2), 45
Matrix, 29
matrix.csr, 29
max, spam-method (Summary), 73
max.spam (Summary), 73
min, spam-method (Summary), 73
min.spam (Summary), 73
mle, 46
ncol, spam-method (spam operations), 66
nearest.dist, 28, 52
nearest.dist (nearestdist), 49
nearestdist, 49
neg2loglikelihood (mle), 46
norm (spam operations), 66
norm, ANY-method (spam operations), 66
norm, matrix, character-method (spam operations), 66
norm, matrix, missing-method (spam operations), 66
norm, numeric, character-method (spam operations), 66
norm, numeric, missing-method (spam operations), 66
norm, spam, character-method (spam operations), 66
norm, spam, missing-method (spam operations), 66
norm.spam (spam operations), 66
nrow, spam-method (spam operations), 66

Ops.spam, 45, 46
Ops.spam (spam operations), 66
options, 50
options.spam (options), 50
Oral, 5, 30, 31, 53
oral.spam (Oral), 53
ordering, 13, 18, 54, 57, 61, 69, 72, 73
ordering, matrix-method (ordering), 54
ordering, spam-method (ordering), 54
ordering, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
ordering-methods (ordering), 54
ordering.spam (ordering), 54
overview (. SPAM .), 3

pad, 55
pad.spam, 25, 26
pad<- (pad), 55
pad<-, matrix-method (pad), 55
pad<-, spam-method (pad), 55
pad<-.spam (pad), 55
permutation, 56
permutation, matrix-method (permutation), 56
permutation.spam (permutation), 56
plot, spam, missing-method (spam-class), 70
plot, spam, spam-method (spam-class), 70
plot.spam (spam operations), 66
powerboost, 52, 57
precmat, 20, 25, 42, 58
precmat.GMRFreglat, 59
precmat.GMRFreglat (makeprec), 42
print, 59, 71
print, spam-method (print), 59
print, spam.chol.NgPeyton-method (spam.chol.NgPeyton-class), 72
print.spam, 52, 73
print.spam (print), 59
prod, spam-method (Summary), 73
prod.spam (Summary), 73
R.version, 78
range, spam-method (Summary), 73
range.spam (Summary), 73
rbind, spam-method (cbind), 10
rbind.spam (cbind), 10
rdist.dist (fields wrapper), 27
read.HB (import), 37
read.MM, 66
read.MM (import), 37
RMprecip, 77
rmvnorm, 60
rmvnorm.canonical.const (rmvnorm.const), 62
rmvnorm.const, 62
rmvnorm.prec, 59
rmvnorm.prec.const (rmvnorm.const), 62
rmvnorm.prec, 63
round, spam-method (Math2), 45
round.spam (Math2), 45
rowMeans (rowSums), 63
rowMeans, spam-method (rowSums), 63
rowMeans.spam (rowSums), 63
rowpointers (constructors), 18
rowpointers<- (constructors), 18
rowSums, 63
rowSums, spam-method (rowSums), 63
rowSums.spam (rowSums), 63
s3only, 64
show, spam-method (spam-class), 70
show, spam.chol.NgPeyton-method
  (spam.chol.NgPeyton-class), 72
signif, spam-method (Math2), 45
signif, spam (Math2), 45
sin, spam-method (Math), 44
sin, spam (Math), 44
solve, 18, 71, 72
solve, ANY-method (spam.solve), 67
solve, spam-method (spam.solve), 67
solve spam, 13, 54
solve spam (spam.solve), 67
SPAM, 66
SPAM(. . SPAM .), 3
Spam(. . . SPAM . .), 3
spam, 4, 35, 63, 65, 67
spam operations, 66
spam solve, 67
spam, list-method (spam), 65
spam, numeric-method (spam), 65
spam-class, 70
spam.chol.NgPeyton, 36
spam.chol.NgPeyton-class, 72
spam.class, 4
spam.class (spam-class), 70
spam.creation (spam), 65
spam.getOption (options), 50
spam.list, 76
spam.list (spam), 65
spam, numeric (spam), 65
spam.ops, 4
spam.ops (spam operations), 66
spam.options, 17, 26, 37, 41, 57, 60, 63, 64,
  66
spam.options (options), 50
spam.Version (version), 78
spam.version (version), 78
spam_rdlist, 50
spam_rdlist (fields wrapper), 27
sqrt, spam-method (Math), 44
sqrt, spam (Math), 44
subset.rows, spam (spam operations), 66
subset, spam (spam operations), 66
sum, spam-method (Summary), 73
sum, spam (Summary), 73
Summary, 71, 73
Summary, spam-method (Summary), 73
summary, spam-method (print), 59
summary, spam.chol.NgPeyton-method
  (spam.chol.NgPeyton-class), 72
Summary, spam, 45
Summary, spam (Summary), 73
summary, spam (print), 59
t, spam-method (spam-class), 70
t, spam.chol.NgPeyton-method
  (spam.chol.NgPeyton-class), 72
t, spam (spam operations), 66
tail, spam-method (head), 35
tail, spam.chol.NgPeyton-method (head), 35
tail, spam (head), 35
tan, spam-method (Math), 44
tan, spam (Math), 44
tcrossprod, ANY, spam-method
  (spam-class), 70
tcrossprod, spam, missing-method
  (spam-class), 70
tcrossprod, spam, missing-method
  (spam-class), 70
tcrossprod, spam (crossprod), 21
toepilz, 15, 42
toepilz, spam, 15, 42
trigamma, spam-method (Math), 44
trigamma, spam (Math), 44
triplet, 29, 75
trunc, spam-method (Math), 44
trunc, spam (Math), 44
update, spam.chol.NgPeyton-method
  (chol), 12
update, spam (chol), 12
upper.tri, 24
upper.tri (lower.tri), 41
upper.tri, spam-method (spam-class), 70
upper.tri, spam, 52
UScounties, 76
USprecip, 77
validspamobject (spam), 65
var, spam (s3only), 64
version, 78