Package 'spam'

October 23, 2023

Type Package

Title SPArse Matrix

Version 2.10-0

Date 2023-10-23

Depends R (>= 3.5)

Imports dotCall64, grid, methods, Rcpp (>= 1.0.8.3)

LinkingTo Rcpp

Suggests spam64, fields, Matrix, testthat, R.rsp, truncdist, knitr, rmarkdown

VignetteBuilder R.rsp, knitr

Description Set of functions for sparse matrix algebra.
Differences with other sparse matrix packages are:
(1) we only support (essentially) one sparse matrix format,
(2) based on transparent and simple structure(s),
(3) tailored for MCMC calculations within G(M)RF.
(4) and it is fast and scalable (with the extension package spam64).
Documentation about 'spam' is provided by vignettes included in this package, see also Furrer and Sain (2010) <doi:10.18637/jss.v036.i10>; see 'citation(``spam'')' for details.

LazyData true

License LGPL-2 | BSD_3_clause + file LICENSE

URL https://www.math.uzh.ch/pages/spam/

BugReports https://git.math.uzh.ch/reinhard.furrer/spam/-/issues

NeedsCompilation yes

Author Reinhard Furrer [aut, cre] (<https://orcid.org/0000-0002-6319-2332>),
Florian Gerber [aut] (<https://orcid.org/0000-0001-8545-5263>),
Roman Flury [aut] (<https://orcid.org/0000-0002-0349-8698>),
Daniel Gerber [ctb],
Kaspar Moesinger [ctb],
Cincera Annina [ctb],
Youcef Saad [ctb] (SPARSEKIT
http://www-users.cs.umn.edu/~saad/software/SPARSKIT/),

Esmond G. Ng [ctb] (Fortran Cholesky routines),
Barry W. Peyton [ctb] (Fortran Cholesky routines),
Joseph W.H. Liu [ctb] (Fortran Cholesky routines),
Alan D. George [ctb] (Fortran Cholesky routines),
Lehoucq B. Rich [ctb] (ARPACK),
Maschhoff Kristi [ctb] (ARPACK),
Sorensen C. Danny [ctb] (ARPACK),
Yang Chao [ctb] (ARPACK)

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

Repository CRAN

Date/Publication 2023-10-23 09:30:02 UTC

R topics documented:

spam-package *SPArse Matrix Package*

Description

spam is a collection of functions for sparse matrix algebra.

Gereral 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.
Some Fortran routines were written by Youcef Saad, Esmond G. Ng, Barry W. Peyton, Joseph W.H. Liu, Alan D. George.

References

Reinhard Furrer, Stephan R. Sain (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields.", *Journal of Statistical Software*, 36(10), 1-25, doi:10.18637/jss.v036.i10.
Florian Gerber, Reinhard Furrer (2015). "Pitfalls in the Implementation of Bayesian Hierarchical Modeling of Areal Count Data: An Illustration Using BYM and Leroux Models.", *Journal of Statistical Software*, Code Snippets, 63(1), 1-32, doi:10.18637/jss.v063.c01.

F. Gerber, K. Moesinger, R. Furrer (2017), "Extending R packages to support 64-bit compiled code: An illustration with spam64 and GIMMS NDVI3g data.", *Computer & Geoscience* 104, 109-119, doi:10.1016/j.cageo.2016.11.015."

See Also

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

Examples

```
## Citations:
citation('spam')
citation('spam', auto=TRUE)


## History of changes
## Not run:
file.show(system.file("NEWS.md", 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.

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 Sparse Matrices 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 typically on a *relative difference* scale unless the `target` values are close to zero or infinite. Specifically, the scale is computed as the average absolute value of `target`. If this scale is finite and exceeds `tolerance`, differences are expressed relative to it; otherwise, absolute differences are used.

If `scale` is numeric (and positive), absolute comparisons are made after scaling (dividing) by `scale`. Note that if all of scale is sufficiently close to 1 (specifically, within `tolerance`), the difference is still reported as being on an absolute scale.

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

Cholesky decomposition routines use this function to test for symmetry.

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

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

Value

Either `TRUE` or a vector of 'mode' `"character"` describing the differences between `target` and `current`.

Author(s)

Reinhard Furrer

See Also

`isSymmetric.spam` and `cleanup`.

Examples

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

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

all.equal( t(obj), obj)

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

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

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


# A zero matrix contains one element:
str(spam(0))
```

```
# hence
all.equal.spam(spam(0,3,3), diag.spam(0,3) )
norm(spam(0,3,3) - diag.spam(0,3) )
```

---

apply *Apply Functions Over Sparse Matrix Margins*

Description

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

Usage

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

Arguments

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

Details

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

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

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

This function may evolve over the next few releases.

Value

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

Author(s)

Reinhard Furrer

See Also

base:apply for more details on Value.

Examples

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

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


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

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

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

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

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

---

bandwidth *Bandwidth of a Sparse Matrix*

Description

Returns the lower and upper bandwidth of a sparse matrix

Usage

```
bandwidth(A)
```

Arguments

A	spam object

Details

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

Value

Integer vector containing the lower and upper bandwidth

Author(s)

Reinhard Furrer

See Also

diag.spam.

Examples

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

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

---

bdiag *Binds Arrays Corner-to-Corner*

Description

Creates a sparse block-diagonal matrix.

Usage

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

Arguments

...	Arrays to be binded together

Details

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

It is recursively defined.

The arrays are coerced to sparse matrices first.

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

Value

Returns a `spam` matrix as described above.

Author(s)

Reinhard Furrer

See Also

`diag.spam`.

Examples

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

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

norm(AB-ABalt)


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

`cbind` *Combine Sparse Matrices by Rows or Columns*

Description

Take a sequence of vector, matrix or `spam` object arguments and combine by *c*olumns or *r*ows, respectively.

Usage

```
# cbind(\dots, force64 = getOption("spam.force64"), deparse.level = 0)
# rbind(\dots, deparse.level = 0)
```

Arguments

`...`	vectors, matrices or `spam` objects. See 'Details' and 'Value'
`force64`	logical vector of length 1. If `TRUE`, a 64-bit spam matrix is returned in any case. If `FALSE`, a 32-bit matrix is returned when possible.
`deparse.level`	for compatibility reason here. Only `0` is implemented.

Details

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

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

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

Value

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

Author(s)

Reinhard Furrer

Examples

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

y <- cbind(x, 7)

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

chol *Cholesky Factorization for Sparse Matrices*

Description

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

Usage

```
# chol(x, \dots)

## S4 method for signature 'spam'
chol(x, pivot = "MMD", method = "NgPeyton",
       memory = list(), eps = getOption("spam.eps"), Rstruct=NULL,
       ..., verbose=FALSE)


# 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 getOption("spam.eps").
Rstruct	sparsity structure of the factor, see 'Details' below.
...	further arguments passed to or from other methods.
object	an object from a previous call to chol, i.e., sparsity structure of the factor.
verbose	provides more details about the decomposition. Useful when working with huge matrices.

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. For U <- chol(A), update(U, Anew) and chol(Anew, Rstruct=U) are equivalent.

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 options("spam.cholsymmetrycheck") is set to FALSE.

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

Author(s)

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

References

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

Gilbert, J. R., Ng, E. G. and Peyton, B. W. (1994) An efficient algorithm to compute row and column counts for sparse Cholesky factorization, *SIAM J. Matrix Anal. Appl.*, **15**, 1075–1091.

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

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 = getOption("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 = getOption("spam.eps")

Value

The circulant matrix in spam format.

Author(s)

Reinhard Furrer

See Also

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

Examples

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

---

cleanup *Cleaning up sparse matrices*

Description

Eliminates an zeros in a sparse matrix.

Usage

```
cleanup(x, eps = getOption("spam.eps"))
```

Arguments

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

Details

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

Author(s)

Reinhard Furrer

See Also

isSymmetric.spam and all.equal.spam.

Examples

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

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

---

coerce-methods *Force a* spam *Object to Belong 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

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

Coercion *Coercion to a Vector*

Description

Coercion of spam matrices to proper vector objects

Usage

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

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

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

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

---

complexity *Complexity for Sparse Matrices*

Description

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

Details

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

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

More to follow.

References

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

See Also

`det`, `solve`, `forwardsolve`, `backsolve` and `ordering`.

`constructors` *Slot Modification*

Description

Modify slots of `spam` objects

Usage

```
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

```
x <- diag.spam( 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

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

cov.exp(h, theta, ... , eps= getOption("spam.eps"))
cov.sph(h, theta, ... , eps= getOption("spam.eps"))
cov.nug(h, theta, ... , eps= getOption("spam.eps"))
cov.wend1(h, theta, ... , eps= getOption("spam.eps"))
cov.wend2(h, theta, ... , eps= getOption("spam.eps"))
cov.wu1(h, theta, ... , eps= getOption("spam.eps"))
cov.wu2(h, theta, ... , eps= getOption("spam.eps"))
cov.wu3(h, theta, ... , eps= getOption("spam.eps"))
cov.mat(h, theta, ... , eps= getOption("spam.eps"))
cov.finnmat(h, theta, ... , eps= getOption("spam.eps"))
cov.mat12(h, theta, ... , eps= getOption("spam.eps"))
cov.mat32(h, theta, ... , eps= getOption("spam.eps"))
cov.mat52(h, theta, ... , eps= getOption("spam.eps"))

cor.sph(h, range, ... , eps= getOption("spam.eps"))
```

Arguments

h	object containing the lags.
theta	parameter of the covariance function, see 'Details'.
range	parameter defining the compact support.
type	covariance function specification.
...	arguments passed from other methods.
eps	tolerance level, see 'Details'.

Details

covmat is a wrapper that calls the other functions according to the argument type. The nomenclature is similar to precmat.

The parametrization is (range, [partial-sill = 1], [smoothness = 1], [nugget = 0]), where only the range needs to be specified. 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.

The parameters are and locations are up to precision epsilon. That means that all distances smaller

than eps are considered zero; a nugget smaller than eps is ignored; a range smaller than eps represents a nugget model; etc.
`cov.finnmat()` is similar to `cov.mat()` but with the `sqrt(8*smoothness)/range` argument in the Bessel function (instead of `1/range`). `cov.mat12()` is a wrapper to `cov.exp()` `cov.mat32()`, and `cov.mat52()` are fast version of `cov.mat()` with smoothness 3/2 and 5/2, respectively (factor 10).
`cor.sph(,range)` is a fast version of `cov.sph(,c(range,1,0))`.
Currently, the functions distinguish between a sparse spam object h 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

```
set.seed(123)
n <- 200
locs <- cbind(runif(n),runif(n))
h <- nearest.dist(locs, delta=sqrt(2), upper = NULL)
Sigma <- cov.sph(h, c(.3, 1, .1))

iidsample <- rnorm(n)
cholS <- chol.spam(as.spam(Sigma))
iorder <- iord <- ordering(cholS, inv = TRUE)
sample <- (iidsample %*% as.spam(cholS))[iorder]
plot(locs, col = fields::tim.colors(n = 256)[cut(sample, n)], pch = 20)

## 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)
```

`crossprod` *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 = NULL, ...)

tcrossprod.spam(x, y = NULL, ...)
```

Arguments

`x, y`	matrices: `y = NULL` is taken to be the same matrix as `x`. Vectors are promoted to single-column or single-row matrices, depending on the context.
`...`	potentially further arguments from other methods.

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)
```

det *Determinant of a Symmetric Positive Definite Sparse Matrix*

Description

det and determinant calculate the determinant of a symmetric, 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 additonal parameters to the Cholesky factorization function, it is possible to pass via 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	+1, as only symmetric positive definite matrices are considered.

Author(s)

Reinhard Furrer

References

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

See Also

chol.spam

Examples

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

det( chol( x))
```

---

diag *Sparse Matrix diagonals*

Description

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

Usage

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

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

Arguments

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

Details

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

Value

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

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

diag.spam works as diag for spam matrices: If x is a vector (or 1D array) of length two or more, then diag.spam(x) returns a diagonal matrix whose diagonal is x. spam_diag is an alias for diag.spam and in the spirit of the result of diag is a spam object.

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

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

Author(s)

Reinhard Furrer

See Also

upper.tri, lower.tri.

Examples

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

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

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

---

diff *Lagged Differences*

Description

Returns suitably lagged and iterated differences.

Usage

```
# 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

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

---

dim *Dimensions of an Object*

Description

Retrieve or set the dimension of an spam object.

Usage

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

Arguments

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

Details

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 replacemnt method changes the dimension of the object by rearranging.

Author(s)

Reinhard Furrer

See Also

pad.spam.

Examples

```
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
```

---

display *Graphially Represent the Nonzero Entries*

Description

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

Usage

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

Arguments

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

Details

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

Author(s)

Reinhard Furrer

See Also

image, spam.options

Examples

```
set.seed(13)

smat <- spam_random(8)
par(mfcol=c(1,2), pty='s')
options(spam.imagesize = 1000)
display(smat)
options(spam.imagesize = 10)
display(smat, cex=.25)


# very large but very sparse matrix
smat <- spam_random(2^14, distribution=rnorm, density=1e-5, verbose=TRUE)
par(mfcol=c(1, 1), mai=c(.4,.4,.1,.1), pty='s')
display(smat)
```

---

eigen *Eigenvalues for Sparse Matrices*

Description

Functions to calculate eigenvalues and eigenvectors of sparse matrices. It uses the value of `spam.options("inefficiencywarning")` to dispatch between `base::eigen()` or the Implicitly Restarted Arnoldi Process, using 'ARPACK'.

`eigen.spam` is a wrapper function of `eigen_approx` and transforms its output to `base::eigen` like.

Usage

```
eigen.spam(x, nev = 10, symmetric, only.values = FALSE, control = list())
eigen_approx(x, nev, ncv, nitr, mode, only.values = FALSE, verbose = FALSE, f_routine)
```

Arguments

`x`	a matrix of class `spam` whose `nev` eigenvalues and eigenvectors are to be computed.
`nev`	number of eigenvalues to calculate.
`symmetric`	if TRUE, the matrix is assumed to be symmetric.
`only.values`	if TRUE, only `nev` eigenvalues are computed and returned, otherwise `nev` eigenvalues and eigenvectors are returned.
`control`	additional options, see 'Details'.
`ncv`	see 'Details', use the `control` option for `eigen.spam`.
`nitr`	see 'Details', use the `control` option for `eigen.spam`.
`mode`	see 'Details', use the `control` option for `eigen.spam`.
`verbose`	see 'Details', use the `control` option for `eigen.spam`.
`f_routine`	only for `eigen_approx`, to call the Fortran routine for symmetric matrices set this option to "ds_eigen_f" and for non symmetric to "dn_eigen_f".

Details

`mode = " "`: there are different modes available for this function, each mode returns a different range of eigenvalues. Also the available modes are dependent, whether the input matrix is symmetric or not:

- `"LM"`: Eigenvalues with largest magnitude (sym, non sym), that is, largest eigenvalues in the Euclidean norm of complex numbers.
- `"SM"`: Eigenvalues with smallest magnitude (sym, non sym), that is, smallest eigenvalues in the Euclidean norm of complex numbers.
- `"LR"`: Eigenvalues with largest real part (non sym).
- `"SR"`: Eigenvalues with smallest real part (non sym).
- `"LI"`: Eigenvalues with largest imaginary part (non sym).
- `"SI"`: Eigenvalues with smallest imaginary part (non sym).
- `"LA"`: Eigenvalues with largest algebraic value (sym), that is, largest eigenvalues inclusive of any negative sign.
- `"SA"`: Eigenvalues with smallest algebraic value (syn), that is, smallest eigenvalues inclusive of any negative sign.

`ncv`: the largest number of basis vectors that will be used in the Implicitly Restarted Arnoldi Process. Work per major iteration is proportional to x@dimension[1]*ncv*ncv. The default is set if `symmetric` to min(x@dimension[1] + 1, max(2 * nev + 1, 200)) or else to min(x@dimension[1] - 1, max(2 * nev + 1, 100)). Note, this value should not be chosen arbitrary large, but slightly larger than `nev`. Otherwise it could lead to memory allocation problems.

`nitr`: the maximum number of iterations. The default is set to `ncv + 1000`

`spamflag = FALSE`: if TRUE, the Implicitly Restarted Arnoldi Process is used, independent of the dimension of the respective matrix (provided matrix is larger than 10x10).

`verbose = FALSE`: print additional information.

`cmplxeps`: threshold to determine whether a double value is zero, while transforming the ARPACK output to R class complex. The default is set to `.Machine$double.eps`.

Value

A vector of the length corresponding to the dimension of the input matrix. Containing the required `nev` eigenvalues. If requested also the corresponding eigenvectors. In the non symmetric case, the eigenvalues are returned in a matrix with a column containing the real parts and a column containing the imaginary parts of the eigenvalues. The eigenvectors are then returned in two matrices.

Note

The user is advised to choose the `control` options carefully, see 'Details' for more information.

Author(s)

Roman Flury, Reinhard Furrer

References

Lehoucq, R. B. and Sorensen, D. C. and Yang, C. (1997) *ARPACK Users Guide: Solution of Large Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*.

See Also

Option "inefficiencywarning" in spam.options and spam_random.

Examples

```
set.seed(81)
rspam <- spam_random(42^2, density = .0001, spd = TRUE)
SPD <- eigen.spam(rspam, nev = 18, control = list(mode = "SM"),
                  only.values = TRUE)

any(SPD$values <= 0, na.rm = TRUE)
isSymmetric(rspam)
# hence the matrix is symmetric positiv definit

rspam2 <- spam_random(50^2, density = .0001, spd = FALSE, sym = TRUE,
                      distribution = rpois, lambda = 2)
SNPD <- eigen.spam(rspam2, nev = 18, control = list(mode = "SM"),
                   only.values = TRUE)

any(SNPD$values <= 0, na.rm = TRUE)
isSymmetric(rspam2)
# hence the matrix is symmetric but not positiv definit
```

---

fields-wrapper *Wrapper for Distance Matrix Computation*

Description

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

Usage

```
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, see Details.

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.

For great circle distance, the matrices x1 and x2 contain the degrees longitudes in the first and the degrees latitudes in the second column. delta is in degrees. Hence to restrict to distances smaller than delta.km, one has to specify delta=delta.km*360/(6378.388*2*pi).

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

```
## 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

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

Arguments

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

Details

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

The functions are based on require.

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

Value

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

Author(s)

Reinhard Furrer

See Also

triplet, Matrix or matrix.csr from package **SparseM**.

Examples

```
## Not run:
S <- diag.spam(4)
R <- as.dgRMatrix.spam( S)
C <- as.dgCMatrix.spam( S)
as.spam.dgCMatrix(C)
slotNames(C)
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:
# for transformations between SparseM and spam:

as.matrix.csr.spam <- function(x,...) {
  if (new("matrix.csr")) {
    newx <- new("matrix.csr")
    slot(newx,"ra",check=FALSE) <- x@entries
    slot(newx,"ja",check=FALSE) <- x@colindices
    slot(newx,"ia",check=FALSE) <- x@rowpointers
    slot(newx,"dimension",check=FALSE) <- x@dimension
    return(newx)
    }
  stop("function requires 'SparseM' package")
}
# then with `SparseM`:  as.matrix.csr.spam( spamobject )

## End(Not run)

## Not run:
# a dataset contained in Matrix
data(KNex, package='Matrix')
summary( KN <- as.spam.dgCMatrix(KNex$mm) )


## End(Not run)
```

---

germany.data *Meta-data About Administrative Districts of Germany*

Description

Supplementary data used for the display of data from 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 `NA`s, 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 https://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel and https://en.wikipedia.org/wiki/Districts_of_Germany

See Also

`germany.plot` `Oral`.

Examples

```
# Plot the Bundeslaender:
germany.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

```
germany.plot(vect,  col=NULL, zlim=range(vect), legend=TRUE,
             main=NULL, cex.axis=1, cex.main=1.5, 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.
`cex.main`	label size of overall plot title.
`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 possiblity 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 https://de.wikipedia.org/wiki/Amtlicher_Gemeindeschl%C3%BCssel and https://de.wikipedia.org/wiki/Liste_der_Landkreise_in_Deutschland

See Also

`Oral`.

Examples

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


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

---

gmult *Generalized Multiplication*

Description

Multiplying specific submatrices of a spam matrix with different factors.

Usage

```
gmult(x, splits, fact)
```

Arguments

`x`	a spam matrix.
`splits`	vector of how to split the matrix into submatrices. It starts with `1` and ends with `max(dim(X))+1`.
`fact`	matrix of factors to multiply submatrices defined by splits. Dimensions of `fact` must correspond to the `length(splits)-1`.

Value

Spam matrix, where each specified submatrix is multiplied with a factor.

Author(s)

Florian Gerber, Roman Flury

Examples

```
x <- spam(1, 15, 15)
print(x, minimal = FALSE)

splits <- c(1,2,8,ncol(x)+1) # divide matrix into 9 submatrices
fact <- array(1:9, c(3,3))   # multiply each submatrix with a different factor
oF <- gmult(x, splits, fact)
print(oF, minimal = FALSE)
```

---

grid_trace2 *Two trace plots and a scatter plot.*

Description

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

Usage

```
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?
cex_points	Size of points in scatter plot.
cex_main	Size of the title font.
lwd_lines	Line width of trace plots.
lwd_frame	Line width of frames.
draw	Logical, should the returned grob object be drawn?

Details

The figure is optimized for a plot of the format x11(width = 10, height = 3).

Value

A grob object.

Author(s)

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

See Also

grid_zoom

Examples

```
grid_trace2(runif(500), runif(500),
            chain1_yaxis_at = seq(.2, 1, by = .2),
            chain1_lab = "chain1", chain2_lab = "chain2",
            main = "Uniform",
            lwd_lines = grid::unit(.5, "mm"))
```

grid_zoom *grid_zoom*

Description

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

Usage

```
grid_zoom(inputGrob = pointsGrob(runif(200),runif(200)),
          inputViewport = viewport(name='main'),
          x = 'topleft', y, just,
          ratio = c(.3,.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 = xaxisGrob(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', 'bootmleft' or 'bootmright'
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

```
require(grid)
## -- Example 1 --
set.seed(133)
grid_zoom(inputGrob = pointsGrob(runif(200), runif(200)),
          inputViewport = viewport(name = 'main'),
          zoom_xlim = c(.2, .3), zoom_ylim = c(.2, .3))


## -- Example 2 --
## initial plot
grid.newpage()
vp <- viewport(width=.8, height=.8, clip='on')
gt <- gTree(children=gList(polylineGrob(x=c((0:4)/10, rep(.5, 5), (10:6)/10, rep(.5, 5)),
              y=c(rep(.5, 5), (10:6/10), rep(.5, 5), (0:4)/10),
              id=rep(1:5, 4), default.units='native',
              gp=gpar(col=1:5, lwd=3)),
              pointsGrob(runif(1000), runif(1000),pch='.', gp=gpar(cex=3)),
              rectGrob(gp=gpar(lwd=3))))
pushViewport(vp)
grid.draw(gt)

## plot with zoom window
grid.newpage()
grid_zoom(inputGrob = gt,
          inputViewport = vp,
          x='topright', zoom_xlim=c(.6,.73), zoom_ylim=c(.3,.43),ratio=.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

```
## 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.
m	similar to n but for the number of columns.
addrownums	create row and column namves them from the selected elements.
...	arguments to be passed to or from other methods.

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 behaviour.

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

Value

An regular matrix.

Author(s)

Reinhard Furrer

Examples

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

image *Display a Sparse Matrix as Color Image*

Description

The function creates a grid of colored rectangles with colors corresponding to the values of the spam matrix.

Usage

```
## 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.

Details

getOption("spam.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*getOption("spam.cex")/(nrow+ncol). For some devices or for non-square matrices, cex needs probably some adjustment.
The a zero matrix in spam format has as (1,1) entry the value zero and only missing entries are interpreted as NA in the scatter plot.

Author(s)

Reinhard Furrer

See Also

display and spam.options.
The code is based on image of graphics.

Examples

```
set.seed(13)

smat <- spam_random(8)

par(mfcol=c(1,2),pty='s')
options(spam.imagesize=1000)
image(smat) # or use better color schemes
options(spam.imagesize=10)
image(smat, cex=.25)
```

```
smat <- spam_random(2^14, distribution=rnorm, density=1e-5, verbose=TRUE)
par(mfcol=c(1,1), mai=c(.4,.4,.1,.1), pty='s')
image(smat)
```

`import` *Read External Matrix Formats*

Description

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

Usage

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

Arguments

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

Details

The names of files storing matrices in the Harwell-Boeing format usually end in `".rua"` or `".rsa"`. Those storing matrices in the MatrixMarket format usually end in `".mtx"`.

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

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

https://math.nist.gov/MatrixMarket/

https://sparse.tamu.edu/

Examples

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

## End(Not run)

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

## End(Not run)
```

---

isSymmetric *Test if a Sparse Matrix is Symmetric*

Description

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

Usage

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

Arguments

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

Details

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

Value

logical indicating if `object` is symmetric or not.

Author(s)

Reinhard Furrer

See Also

`all.equal.spam`, `cleanup`.

Examples

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

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

---

kronecker *Kronecker Products on Sparse Matrices*

Description

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

Usage

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

Arguments

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

Details

The sparsity structure is determined by the ordinary `%x%`. Hence, the result of `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="+")
```

---

large_matrix *Large 64-bit matrices require the R package* **spam64**

Description

The R package **spam** can handle sparse matrices with up to 2^31-1 non-zero elements. For matrices with more non-zero elements it is necessary to load the **spam64** package in addition.

Details

With the help of the R package **dotCall64** spam interfaces either the compiled code with 32-bit integers provided in **spam** or the compiled code with 64-bit integers provided in **spam64**.
To mimick 64-bit behavior, set `options(spam.force64 = TRUE)`. The subsequent matrix indices are then stored in double format.

Author(s)

Reinhard Furrer, Florian Gerber, Kaspar Moesinger, Daniel Gerber

References

F. Gerber, K. Moesinger, R. Furrer (2017), Extending R packages to support 64-bit compiled code: An illustration with spam64 and GIMMS NDVI3g data, Computer & Geoscience 104, 109-119, https://doi.org/10.1016/j.cageo.2016.11.015.

See Also

spam64-package, dotCall64.

Examples

```
## Not run:
## the following matrices are very large, and hence,
## require much memory and cpu time.
library("spam64")
s1 <- spam(1, ncol=2^30)        # 32-bit matrix
s1

s2 <- cbind(s1, s1)             # 64-bit matrix
s2

s3 <- spam(1, ncol=2^31)        # 64-bit matrix
s3

## End(Not run)
```

---

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 `getOption("spam.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)

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

---

makeprec *Create Precision Matrices*

Description

Creates precision matrices for gridded GMRF.

Usage

```
precmat.GMRFreglat(n,m, par, model = "m1p1",  eps = getOption("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 = getOption("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

```
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

```
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**.

See Also

germany.plot super-seeding map.landkreis.

Examples

```
## Not run:
data( Oral)
par( mfcol=c(1,2))
germany.plot( log( Oral$Y), legend=TRUE)
map.landkreis( log( Oral$Y))

## End(Not run)
```

Math *Mathematical Functions*

Description

Applies the Math group functions to spam objects

Usage

```
# ceiling(x)
# floor(x)
# exp(x, base = exp(1))
# log(x, base = exp(1))
# sqrt(x)

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

# cos(x)
# sin(x)
# tan(x)

# acosh(x)
```

Arguments

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

Details

It is important to note that the zero entries do not enter the evaluation 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 the option `spam.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

```
getGroupMembers("Math")

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

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

sqrt( smat)
```

`Math2` *Rounding of Numbers*

Description

Applies the `Math2` group functions to 'spam' objects

Usage

```
## 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

```
getGroupMembers("Math2")

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

---

mle *Maximum likelihood estimates*

Description

Maximum likelihood estimates of a simple spatial model

Usage

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

neg2loglikelihood(y, X, distmat, Covariance,
                 beta, theta, cov.args = NULL, ...)

neg2loglikelihood.nomean(y, distmat, Covariance,
                 theta, cov.args = NULL, ...)

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

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

```
     hessian = FALSE, cov.args = NULL, ...)

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

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

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	inital 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?
cov.args	additional arguments passed to Covariance.
...	additional arguments passed to chol.

Details

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

y ~ 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 -> 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`, `rmvnorm.spam`

Examples

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



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

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


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

# Here is the negative 2 log likelihood:
neg2loglikelihood.spam( y, X, distmat, cov.sph,
                        truebeta, truetheta)

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

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

nearestdist *Distance Matrix Computation*

Description

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

Usage

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

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.
fortran	Should the C++ (FALSE) or the Fortran code (TRUE) be used. If 64-bit matrices are needed, the argument is set to (TRUE).

Details

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

The distances are calculated based on spherical law of cosines. Care is needed for 'zero' distances due to the final acosin: acos(1-1e-16), especially with an actual radius.

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

There are many other packages providing distance functions. Especially for great circle distances there are considerable differences between the implementations. For high precision results, `sp::spDists` is a good candidate and distances of large amount of locations can be processed in parallel with the `parallelDist` package.

The formerly depreciated arguments `eps` and `diag` are now eliminated.

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

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)

Annina Cincera (C++ code), Reinhard Furrer

See Also

`spam_rdist`

Examples

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

# To compare nearest.dist with dist, use 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))
```

options *Options Settings*

Description

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

Details

Invoking `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 `getOption("spam.eps")`, e.g., rather than `options("spam.eps")` which is a *list* of length one.

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

Value

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

For `options()`, a list of all set options sorted by category. For `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 for the package `spam`

A short description with the default values follows.

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

`spam.drop=FALSE`**:** default parameter for `drop` when subsetting

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

`spam.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.

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

`spam.structurebased=FALSE`**:** should operations be carried out on the nonzero entries (the structure) or including the zeros.

`spam.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`.

`spam.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.

`spam.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.

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

`spam.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.

`spam.safemodevalidity=TRUE`: logical determines if sanity check is peformed when constructing sparse matrices. Default is safer but somewhat slower.

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

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

`spam.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"`.

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

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

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

See Also

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

Examples

```
smat <- diag.spam( 1:8)
smat
options(spam.printsize=49)
smat

# List all spam options:
options()[grep("spam",names(options()))]

# Reset to default values:
options(spam.eps=.Machine$double.eps,
        spam.drop=FALSE,
        spam.printsize=100,
        spam.imagesize=10000,
        spam.cex=1200,
        spam.structurebased=FALSE,
        spam.inefficiencywarning=1e6,
        spam.trivalues=FALSE,
        spam.listmethod="PE",
        spam.NAOK=FALSE,
        spam.safemodevalidity=TRUE,
        spam.dopivoting=TRUE,
```

```
spam.cholsymmetrycheck=TRUE,
spam.cholpivotcheck=TRUE,
spam.cholupdatesingular="warning",
spam.cholincreasefactor=c(1.25,1.25),
spam.nearestdistincreasefactor=1.25,
spam.nearestdistnnz=c(400^2,400))
```

`Oral` *Oral Cavity Cancer*

Description

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

Format

`Oral` is a dataframe with 3 columns.

Y observed counts

E expected counts

SMR standardized mortality ratios

`germany` is a list of 544 elements, each describing an individual polygon of the district.

Details

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

There is a similar dataset `data(Germany)` 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 https://www.r-inla.org.

References

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

See Also

`germany.plot`.

`ordering` *Extract the permutation*

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)
```

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

# the following are equivalent:
as.spam( RtR -            S[ord,ord],    eps=1e-15)
as.spam( RtR[iord,iord] - S,             eps=1e-15)
as.spam( t(R[,iord]) %*% R[,iord] - S, eps=1e-15)
# we use 'eps' to avoid issues close to machine precision

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

---

pad *Padding a Sparse Matrix*

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 option("spam.eps") as filter).

Value

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

Author(s)

Reinhard Furrer

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 (sparse) 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 colum permutation. For very large matrices, a double transpose might be faster.

The spam option `spam.checkpivot` determines if the permutation is verified.

Value

A permuted matrix.

Author(s)

Reinhard Furrer

See Also

ordering, spam.options.

Examples

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

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

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

---

powerboost *Specific Options Setting*

Description

Sets several options for speed-up.

Usage

```
powerboost(flag)
```

Arguments

flag	on or off

Details

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

Value

NULL in any case.

Author(s)

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

See Also

spam.options.

precmat *IGMRF Precision Matrices*

Description

Fast ways to create sparse precision matrices for various IGMRF.

Usage

```
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=getOption("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

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

Q%*%cbind(1,1:n)
```

---

print *Printing and Summarizing Sparse Matrices*

Description

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

Usage

```
  ## 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. If the non-standard argument minimal is set to FALSE, an extended spam print is available with logical argument rowpointer to print rowpointers, and zerosymbol defining the character to display the zero element.

Details

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

Value

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

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

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

Author(s)

Reinhard Furrer

See Also

display or image for a graphical visualization; spam.options

Examples

```
set.seed(13)
smat <- spam_random(8)
par(mfcol=c(1,2),pty='s')
options(spam.printsize=1000)
print(smat)
options(spam.printsize=10)
print(smat)
summary(smat)
summary(smat)$nnz

smat@entries[1:5] <- 0
print(smat, minimal = FALSE)
print(smat, minimal = FALSE, rowpointer = TRUE)
smat@rowpointers
print_nnzpos(smat)
```

---

random *Create Random Sparse Matrices*

Description

Creates random spam matrix given the dimension and other parameters.

Usage

```
spam_random(nrow = 1L, ncol = nrow, density = 0.5, distribution = NULL, digits = NULL,
            sym = FALSE, spd = FALSE, verbose = FALSE, ...)
```

Arguments

nrow	integer value for the number of rows for the spam matrix to create.
ncol	integer value for the number of columns. The default value is the same as nrow.
density	A numeric value between 0 and 1 specifying the approximate density of matrix. If equal to zero the spam matrix contains only zeros and if equal to 1 the spam matrix is full.
distribution	a random number generating distribution function to sample the entries of the spam matrix. The function must have an argument with the name n, possible candidates are rnorm, rexp, rpois, rweibull, etc..
...	possible additional arguments for the distribution function if specified with distribution.
digits	an integer value for the number of digits the entries should be rounded.
sym	logical value to specify symmetry of the spam matrix.
spd	logical value to specify positive definitness of the spam matrix, via diagonal dominace criteria. Note, if spd TRUE, then sym is overwritten to TRUE in any case.
verbose	logical value to specify verbose statments of the function.

Details

To create a random spam64 matrix, set `options(spam.force64 = TRUE)`.

Value

A random matrix in spam format.

Author(s)

Florian Gerber, Roman Flury, Reinhard Furrer

See Also

spam-class and display.spam

Examples

```
set.seed(42)
rspam <- spam_random(500, digits = 2, distribution = rnorm, sd = 2, mean = 10, density = .01)
display.spam(rspam, cex = 2)
```

rgrf *Draw From a Gaussian Random Field*

Description

Fast and intuitive ways to draw from a Gaussian random field.

Usage

```
rgrf( n,
   locs, nx, ny=nx, xlim=c(0,1), ylim=c(0,1), tau=0,
   Covariance, theta, beta=0, X,
   method=c('chol'),  method.args=list(sparse=FALSE),
   eps = getOption("spam.eps"), drop=TRUE, attributes=TRUE, ...)
```

Arguments

n	number of observations.
locs	locations, the result of as.matrix(locs) will be used.
nx,ny	if no locations are specified, at least one of these to specify the grid dimension.
xlim,ylim	Domain, see 'Details'.
tau	perturbation degree, see 'Details'.
Covariance	covariance function name.
theta	covariance parameter.
beta	mean or vector for regression-type mean.
X	design matrix for regression-type mean.
method	based on Choleski factorization.
method.args	list of arguments that can be passed to the corresponding approach. For "chol" it can be, e.g., RStruct, chol.args, cov.args.
eps	small value, anything smaller is considered a collocation.
drop	logical, if a single realization should be returned as a vector.
attributes	logical, if should attributes be passed back.
...	currently not used.

Details

If no locations are given, the function constructs these according a regular or a regular perturbed grid. The perturbation is determined by tau, which has to be greater than zero (no perturbation) and strictly smaller than 1/2 (max perturbation).

The regular grid has spacing (here for x) dx=diff(xlim)/nx and runs from xlim[1]+dx/2 to xlim[2]-dx/2. The locations are at least (1/nx-2*tau*dx) separated.

Currently, the only method implemented is a Cholesky factorization routine, (much as in rmvnorm).

The rdist() from the fields package is awefully fast. Unless one has very sparse covariance matrices, a sparse approach is not bringing a lot of improvements.

The methods may use different covariance construction approaches and thus the nesting of cov.args in method.args.

Author(s)

Reinhard Furrer

See Also

rgrf, chol and ordering.

Examples

```
require(fields)
# Regular grid with constant mean:
nx <- 10
field <- rgrf(1, nx=nx,  Covariance="cov.wend2", theta=c(.5, 1), beta=5)
quilt.plot(cbind(attr(field,"locs"),z=field), nx=nx, ny=nx)
points(attr(field,"locs"))

# Irregluar grid:
field <- rgrf(1, nx=10, tau=0.3, Covariance="cov.mat", theta=c(.2, 1, 1.5))
fields::quilt.plot(attr(field,"locs"), field)
```

---

rmvnorm *Draw Multivariate Normals*

Description

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

Usage

```
rmvnorm(n, mu=rep.int(0, dim(Sigma)[1]), Sigma, ..., mean, sigma)
rmvnorm.spam(n, mu=rep.int(0, dim(Sigma)[1]), Sigma, Rstruct=NULL, ..., mean, sigma)
rmvnorm.prec(n, mu=rep.int(0, dim(Q)[1]), 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.
mean,sigma	similar to mu and Sigma. Here for portability with mvtnorm::rmvnorm()

Details

All functions rely on a Cholesky factorization of the covariance or precision matrix.
The functions rmvnorm.prec and rmvnorm.canonical do not require sparse precision matrices Depending on the the covariance matrix Sigma, rmvnorm or rmvnorm.spam is used. If wrongly specified, dispatching to the other function is done.
Default mean is zero. Side note: mean is added via sweep() and no gain is accieved by distinguishing this case.
Often (e.g., in a Gibbs sampler setting), the sparsity structure of the covariance/precision does not change. In such setting, the Cholesky factor can be passed via Rstruct in which only updates are performed (i.e., update.spam.chol.NgPeyton instead of a full chol).

Author(s)

Reinhard Furrer

References

See references in chol.

See Also

rgrf, chol and ordering.

Examples

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


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

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

`rmvnorm.conditional` *Draw Conditional Multivariate Normals*

Description

Fast way to draw conditional multivariate normals when the covariance matrix is sparse.

Usage

```
rmvnorm.conditional(n, y, mu = rep.int(0, dim(SigmaXX)[1]+dim(SigmaYY)[1]),
                    SigmaXX, SigmaYY, SigmaXY, noise, RstructYY = NULL, ...)
```

Arguments

`n`	number of observations.
`y`	observed vector.
`mu`	mean vector.
`SigmaXX`	covariance of X, required (of class `spam`).
`SigmaXY`	cross-covariance of X-Y, optional (of class `spam`).
`SigmaYY`	covariance of Y, required (of class `spam`).
`noise`	observational noice of Y, optional. See 'Details'.
`RstructYY`	the Cholesky structure of `SigmaYY`.
`...`	arguments passed to `chol`.

Details

Quite often, we want to draw condional observations $X|y$ from the model $Y = X + e$, where X has covariance matrix `SigmaXX` and e has white noise.

Covariance of Y can be specified by `SigmaYY` or `SigmaXX+diag(noise,)`. If Y and X do not have the same dimensions, `SigmaXY` needs to be specified.

The function also implmements a general multivariate model, where the we only observe part of the vector. The components are first X then Y.

The function `rmvnorm.cond()` is a wrapper to `rmvnorm.conditional()` and included to increase similarities with other packages.

Author(s)

Reinhard Furrer

See Also

rmvnorm.spam.

Examples

```
set.seed(12)
N <- 300
y <- c(5, -5, -5, 5)
SigmaXX <- as.spam(.95^abs(outer(1:N, 1:N, "-")), eps=1e-4)
sel <- c(10, 100, 120, 300)        # where we observe y
SigmaXY <- SigmaXX[, sel]
SigmaYY <- SigmaXX[sel,sel] + diag.spam(.01, length(y)) # some noise
x <- rmvnorm.conditional(3, y, SigmaXX=SigmaXX, SigmaXY=SigmaXY,
                         SigmaYY=SigmaYY)
# unconditional sample:
ux <- rmvnorm(1, Sigma=SigmaXX)
matplot(t(rbind(x, ux)), type='l', lty=1)
points(sel, y, pch=19)
```

---

rmvnorm.const *Draw Constrainted Multivariate Normals*

Description

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

Usage

```
rmvnorm.const(n, mu = rep.int(0, dim(Sigma)[1]), Sigma, Rstruct = NULL,
              A = array(1, c(1,dim(Sigma)[1])), a=0, U=NULL,  ...)
rmvnorm.prec.const(n, mu = rep.int(0, dim(Q)[1]), Q, Rstruct = NULL,
              A = array(1, c(1,dim(Q)[1])), a=0, U=NULL,  ...)
rmvnorm.canonical.const(n, b, Q, Rstruct = NULL,
              A = array(1, c(1,dim(Q)[1])), 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 requrie sparse precision matrices. For rmvnorm.spam, the differences between regular and sparse covariance matrices are too significant to be implemented here.
Often (e.g., in a Gibbs sampler setting), the sparsity structure of the covariance/precision does not change. In such setting, the Cholesky factor can be passed via Rstruct in which only updates are performed (i.e., update.spam.chol.NgPeyton instead of a full chol).

Author(s)

Reinhard Furrer

References

See references in chol.

See Also

rmvnorm.spam.

Examples

```
# to be filled in
```

---

rmvt *Draw From a Multivariate t-Distribution*

Description

Fast ways to draw from a multivariate t-distribution the scale (covariance) matrix is sparse.

Usage

```
rmvt(n, Sigma, df = 1, delta = rep(0, nrow(Sigma)),
    type = c("shifted", "Kshirsagar"), ..., sigma)
rmvt.spam(n, Sigma, df = 1, delta = rep(0, nrow(Sigma)),
    type = c("shifted", "Kshirsagar"), ..., sigma)
```

Arguments

n	number of observations.
Sigma	scale matrix (of class spam).
df	degrees of freedom.
delta	vector of noncentrality parameters.
type	type of the noncentral multivariate t distribution.
...	arguments passed to rmvnorm.spam.
sigma	similar to Sigma. Here for portability with mvtnorm::rmvt()

Details

This function is very much like rmvt() from the package **mvtnorm**. We refer to the help of the afore mentioned.

Author(s)

Reinhard Furrer

References

See references in mvtnorm::rmvt().

See Also

rmvnorm.

---

rowSums *Form Row and Column Sums and Means*

Description

Form row and column sums and means for sparse spam matrices

Usage

```
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.

Examples

```
x <- spam( rnorm(20), 5, 4)
rowSums( x)
c( x %*% rep(1,4))
```

---

s3only *Wappers for Sparse Matrices*

Description

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

Usage

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

## S3 method for class 'spam'
var(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.

---

spam *Sparse Matrix Class*

Description

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

Usage

```
spam(x, nrow = 1, ncol = 1, eps = getOption("spam.eps"))


as.spam(x, eps = getOption("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 = getOption("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.

If getOption("spam.force64") is TRUE, a 64-bit spam matrix is returned in any case. If FALSE, a 32-bit matrix is returned when possible.

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).

Author(s)

Reinhard Furrer

References

Reinhard Furrer, Stephan R. Sain (2010). "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields.", *Journal of Statistical Software*, 36(10), 1-25, doi:10.18637/jss.v036.i10.

See Also

SPAM for a general overview of the package; spam_random to create matrices with a random sparsity pattern; cleanup to purge a sparse matrix; 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)
## IGNORE_RDIFF_BEGIN
system.time(for (i in 1:nz) smat[is[i], js[i]] <- i)
system.time(smat[cbind(is,js)] <- 1:nz)
## IGNORE_RDIFF_END

getClass("spam")


options(spam.NAOK=TRUE)
as.spam(c(1, NA))
```

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

Slots

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

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

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

`dimension`**:** Object of class `"integer"` specifying the dimension of the matrix.

Methods

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

as.vector `signature(x = "spam")`: transforming a sparse matrix into a vector (dependings on `structurebased`) see `as.vector.spam` for details.

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

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

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

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

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

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

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

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

dim<- `signature(x = "spam")`: rearranges the matrix to reflect a new dimension.

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

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

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

display `signature(x = "spam")`: see `display` for details.

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

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

lower.tri `signature(x = "spam")`: see `lower.tri` for details.

Math `signature(x = "spam")`: see `Math` for details.

Math2 `signature(x = "spam")`: see `Math2` for details.

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

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

print `signature(x = "spam")`: see `print` for details.

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

solve `signature(a = "spam")`: see `solve` for details.

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

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

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

upper.tri `signature(x = "spam")`: see `lower.tri` for details.

Details

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

Note

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

Author(s)

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

Examples

```
showMethods("as.spam")


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

spam-defunct *Defunct Objects in Package* **spam**

Description

The functions or variables listed here are defunct, i.e. thorw an error when used.

Usage

```
validspamobject(...)
```

Arguments

...	some arguments

See Also

Deprecated, Defunct

spam-operations *Basic Linear Algebra for Sparse Matrices*

Description

Basic linear algebra operations for sparse matrices of class spam.

Details

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

The function norm calculates the (matrix-)norm of the argument. The argument type specifies the l1 norm, sup or max norm (default), or the Frobenius or Hilbert-Schmidt (frobenius/hs) norm. Partial matching can be used. For example, norm is used to check for symmetry in the function chol by computing the norm of the difference between the matrix and its transpose

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

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

References

Some Fortran functions are based on https://github.com/johannesgerer/jburkardt-f/blob/master/sparsekit/sparsekit.html

See Also

spam for coercion and other class relations involving the sparse matrix classes.

Examples

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

n <- dim(wij)[1]

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

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

## End(Not run)
```

---

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

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

In case a is from a chol 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 Cholsesky 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

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


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

mvsample <- backsolve( chol(Sigmainv), iidsample)
norm( var(t(mvsample)) - Sigma)

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



# '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 )


# 'update':
R1 <- update( R, Sigmainv + diag.spam( n))
```

---

spam.chol.NgPeyton-class

Class "spam.chol.NgPeyton"

Description

Result of a Cholesky decomposition with the NgPeyton method

Details

It is not possible to directly change the length, dimension and the diagonal entries of a "spam.chol.NgPeyton" object.

Objects from the Class

Objects are created by calls of the form chol(x,method="NgPeyton", ...) and should not be created directly with a new("spam.chol.NgPeyton", ...) call.
At present, no proper print method is defined. However, the factor can be transformed into a spam object.

Methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Author(s)

Reinhard Furrer

References

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

See Also

`print.spam` `ordering` and `chol`

Examples

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


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

`Summary` *Rounding of Numbers*

Description

Applies the `Math2` group functions to `spam` objects

Usage

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

Arguments

`x`	spam object.
`na.rm`	a logical indicating whether missing values should be removed.

Details

The `na.rm` argument is only meaninful 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

```
getGroupMembers("Summary")

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

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

## End(Not run)
# allow 'NA's first:
# TODO
# options(spam.NAOK=TRUE)
# max( log(spam(c(1,-1))), na.rm=TRUE)
```

---

toeplitz *Create Toeplitz Matrices*

Description

Creates symmetric and asymmetric Toeplitz matrices.

Usage

```
toeplitz.spam(x, y = NULL, eps = getOption("spam.eps"))
```

Arguments

x	the first row to form the Toeplitz matrix.
y	for asymmetric Toeplitz matrices, this contains the first column.
eps	A tolerance parameter: elements of x such that abs(x) <= eps set to zero. Defaults to eps = getOption("spam.eps").

Details

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

Value

The Toeplitz matrix in spam format.

Author(s)

Reinhard Furrer

See Also

toeplitz, circulant.spam

Examples

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

---

triplet *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.creation for the inverse operation and foreign for other transformations.

Examples

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

UScounties *Adjacency Structure of the Counties in the Contiguous United States*

Description

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

Format

Two matrices of class spam

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

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

See Also

map, from **maps**.

Examples

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

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

## End(Not run)
```

USprecip *Monthly Total Precipitation (mm) for April 1948 in the Contiguous United States*

Description

This is a useful spatial data set of moderate to large size consisting of 11918 locations. See https://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 Preipitation anomaly for April 1948.

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

Source

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

References

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

See Also

RMprecip

Examples

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

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

## End(Not run)
```

validate_spam *Validate a Sparse Matrix*

Description

Checks if the sparse matrix has the correct structure.

Usage

```
validate_spam(object)
```

Arguments

object	a spam matrix.

Value

Returns TRUE if object is a valid spam objects.

See Also

spam.creation

Examples

```
validate_spam(spam(1, 20))
```

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