The spam Package

February 21, 2009

Version 0.15-3

Date 2009-02-20

Author Reinhard Furrer

Maintainer Reinhard Furrer <rfurrer@mines.edu>

Depends R (>= 2.4), methods

Suggests SparseM (>= 0.72), Matrix

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

LazyLoad Yes

LazyData Yes

License GPL | file LICENSE

Title SPArse Matrix

URL http://www.mines.edu/~rfurrer/software/spam/

R topics documented:

`SPAM` *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`).

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 `spdep` use `spam` as a required package.

Author(s)

Reinhard Furrer

References

`www.mines.edu/~rfurrer/software/spam/`

See Also

See `spam.class` for a detailed class description, `spam` and `spam.ops` for creation, coercion and algebraic operations.
`demo(package='spam')` lists available demos.
Related packages are `fields`, `Matrix` and `SparseM.ontology`.

Examples

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

`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`

Examples

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

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

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

Description

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

Format

This data set is an array containing the following columns:

lon,lat Longitude-latitude position of monitoring stations

raw Monthly total precipitation in millimeters for April 1948

anomaly Preipitation anomaly for April 1948.

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

Source

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

References

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

See Also

`RMprecip`

Examples

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

data(USprecip)
par(mfcol=c(2,1))
quilt.plot(USprecip[,1:2],USprecip[,3])
US( add=TRUE, col=2, lty=2)
quilt.plot(USprecip[,1:2],USprecip[,4])
US( add=TRUE, col=2, lty=2)
## End(Not run)
```

`allequal` *Test if Two 'spam' Objects are (Nearly) Equal*

Description

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

Usage

```
all.equal.spam(target, current, tolerance = .Machine$double.eps^0.5,
    scale = NULL, check.attributes = FALSE,...)
```

Arguments

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

Details

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

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

Don't 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.

Value

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

Author(s)

Reinhard Furrer

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))
```

---

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

update.spam.chol.NgPeyton(object, x,...)
```

Arguments

x	symmetric positive definite matrix of class spam.
pivot	should the matrix be permuted, and if, with what algorithm, see Details below.
method	Currently, only NgPeyton is implemented.

`memory`	Parameters specific to the method, see Details below.
`eps`	threshold to test symmetry. Defaults to `.Spam$eps`.
`...`	further arguments passed to or from other methods.
`object`	an object from a previous call to `chol`.

Details

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

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

Often the sparseness 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.

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 sparseness structure can be obtained from a `summary` call.

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

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

Value

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

Note

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

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

Author(s)

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

References

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

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

See Also

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

Examples

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

cholS <- chol( Sigma)
# cholS is the upper triangular part of the 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="HS")
norm( t(R) %*% R - Sigma, type="sup")
```

`det` *Calculate the determinant of a positive definite Sparse Matrix*

Description

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

Usage

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

Arguments

`x`	sparse matrix of class `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, see the help for `chol`.

The determinant of a Cholesky factor is also defined.

Value

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

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

Author(s)

Reinhard Furrer

References

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

See Also

`chol`

Examples

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

det( chol( x))
```

`diag` *Sparse Matrix diagonals*

Description

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

Usage

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

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

Arguments

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

Details

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

Value

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

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

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

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

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

Author(s)

Reinhard Furrer

See Also

`upper.tri`, `lower.tri`.

Examples

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

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

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

`dim` *Dimensions of an Object*

Description

Retrieve or set the dimension of an `spam` object.

Usage

```
# dim(x)
# dim(x) <- value
"dim<-.spam"(x,value)
```

Arguments

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

Details

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

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

Value

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

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

Author(s)

Reinhard Furrer

See Also

`dim`.

Examples

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

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

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

`display` *Graphially represent the nonzero entries*

Description

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

Usage

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

Arguments

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

Details

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

Author(s)

Reinhard Furrer

See Also

`image`, `spam.options`

Examples

```
set.seed(13)

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

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

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

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

`foreign` *Transformation to other sparse formats*

Description

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

Usage

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

Arguments

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

Details

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

The functions are based on `require`.

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

Value

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

Author(s)

Reinhard Furrer

See Also

`triplet`, `Matrix` or `matrix.csr`

Examples

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

# as.spam(R) does not work.
## End(Not run)

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

## End(Not run)
```

`history` *Brief overview of the history*

Description

Brief overview of `spams` history

Usage

```
spam.history()
```

Details

This list is maintained for releases 0.05 and higher.

0.15 Induced by need to catch up the devel versions of 0.14.

0.14 Complete rewrite of `chol`, implementation of `update.spam.chol.NgPeyton` and related routines. Rewrite of `spam.chol.NgPeyton` class. Proper `spam.options` handling. Kronecker product implementation. Rewrite of `cbind`. Importing and handling of foreign formats. Many internal changes for efficieny increases. Major file renaming. Some backwards compatibility loss.

0.13 Minor changes to avoid some warning/errors when compiling on different platforms.

0.12 Many internal changes. Start maintaining a proper CHANGELOG file.

0.11 Considerable help file improvements. Changes in `c/rbind`, `chol`, print/summary methods and other minor improvements, first push to CRAN.

0.10 Minor bug fixes and help file improvements.

0.09 Proper NAMESPACE handling.

0.08 `fields` uses `spam` now as a required package. The required linking functions are implemented in fields.

0.07 Change of slot names.

0.06 Implements `spam` and the function linking `fields` version 3.6. Last stable version before the "merging" with fields. Update to R2.4 and documentation fill-in.

0.05 Implements `spam` and the function linking `fields` version 3.2. It should represent a fair working version, with large gaps in the documentation.

References

See also a more detailed description at `http://www.mines.edu/~rfurrer/software/spam/history.shtml`.

See Also

`todo`.

Examples

```
## Not run:
## A complete history of changes
file.show(system.file("NEWS", package = "spam"))
## End(Not run)
```

---

image *Display a spam Object as Color Image*

Description

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

Usage

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

Arguments

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

Details

`getOption('imagesize')` determines if the sparse matrix is coerced into a matrix and the plotted similarly to `image.default` or if the matrix is simply represented as a scatterplot with `pch="."`. The points are scaled according to `cex*spam.getOption('cex')/(nrow+ncol)`. For some devices or for non-square matrices, `cex` needs probably some adjustment.
The a zero matrix in `spam` format has as (1,1) entry the value zero and only missing entries are interpreted as `NA` in the scatter plot.

Author(s)

Reinhard Furrer

See Also

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

Examples

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

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

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

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

`import` *Read external matrix formats*

Description

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

Usage

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

Arguments

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

Details

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

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

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

Value

A sparse matrix of class `spam`.

Note

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

Author(s)

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

References

http://math.nist.gov/MatrixMarket

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

Examples

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

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

`isSymmetric` *Test if a spam matrix is Symmetric*

Description

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

Usage

```
isSymmetric.spam(object, 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`.

Value

logical indicating if `object` is symmetric or not.

Author(s)

Reinhard Furrer

See Also

`all.equal.spam`.

Examples

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

obj[1,2] <- .Machine$double.eps
isSymmetric(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 sparseness 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 sparseness structure:
kronecker( diag.spam(2), array(1:4,c(2,2)),FUN="+")
```

---

lower.tri *Lower and Upper Triangular Part of a Sparse Matrix*

Description

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

Usage

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

Arguments

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

Details

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

See Also

spam.options and diag

Examples

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

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

`Math` *Mathematical functions*

Description

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

Usage

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

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

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

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

Arguments

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

Details

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

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

Value

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

Author(s)

Reinhard Furrer

See Also

`Math2`

Examples

```
getGroupMembers("Math")

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

sqrt( smat)
```

`Math2` *Rounding of Numbers*

Description

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

Usage

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

Arguments

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

Details

This set of functions The for this generic class typical `na.rm` argument has no weight here as NA/NaN/Inf are not meaningful (yet).

Value

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

Author(s)

Reinhard Furrer

See Also

`Math`

Examples

```
getGroupMembers("Math2")

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

`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",
              eps = .Spam$eps, delta = 1,
              diag = FALSE, upper = FALSE,
              p=2, miles=TRUE, R=NULL)
```

Arguments

`x`	Matrix of first set of locations where each row gives the coordinates of a particular point. See also Details.
`y`	Matrix of second set of locations where each row gives the coordinates of a particular point. If this is missing `x` is used. See also Details.
`method`	the distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"minkowski"` or `"greatcircle"`. Any unambiguous substring can be given.
`eps`	distances smaller than this number are considered zero.
`delta`	only distances smaller than `delta` are recorded.
`diag`	Should the diagonal be included? See details.

`upper`	Should the entire matrix (`NULL`) or only the upper-triagonal (`TRUE`) or lower-triagonal (`FALSE`) values be calculated.
`p`	The power of the Minkowski distance.
`miles`	For great circle distance: If true distances are in statute miles if false distances in kilometers.
`R`	For great circle distance: Radius to use for sphere to find spherical distances. If `NULL` the radius is either in miles or kilometers depending on the values of the miles argument. If `R=1` then distances are of course in radians.

Details

For great circle distance, the by 2 matrices `x` and `y` contain the degrees longitudes in the first and the degrees latitudes in the second column. `eps` and `delta` are in degrees. The distance is in single precision (I am still not sure where I loose the double in the Fortran code) and if calculating the entire matrix `upper=NULL` (instead of adding its transpose) it may not pass the symmetry checks, for example.

The argument `dist=TRUE` determines if diagonal elements will also be included if smaller than `eps`. This is useful when calculating covariance matrices based on a distance matrix. The default values of `dist=FALSE` and `upper=FALSE` are borrowed from `dist`.

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

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

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

Value

A `spam` object containing the distances spanned by `eps` and `delta`.

Author(s)

Reinhard Furrer

Examples

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

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

# Create nearest neighbor structures:
```

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

`spam operations` *Basic Linear Algebra for Sparse Matrices*

Description

Basic linear algebra operations for sparse matrices of class `spam`.

Details

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

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

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

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

References

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

See Also

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

Examples

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

`options` *Options Settings*

Description

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

Usage

```
spam.options(...)

spam.getOption(x)
```

Arguments

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

Details

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

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

Value

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

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

Options used

A short description with the default values follows.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Author(s)

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

See Also

`print`, `display`, `image`, `upper.tri`, `chol`, `nearest.dist`, etc.

Examples

```
op <- spam.options()

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

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

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

`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, solve.

Examples

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

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

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

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

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

---

print *Printing and summarizing sparse matrices*

Description

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

Usage

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

Arguments

`x`	matrix of class `spam` or `spam.chol.`*method*.
`object`	matrix of class `spam` or `spam.chol.`*method*.
`...`	any other arguments passed to `print.default`.

Details

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

Value

`NULL` for `print`, because the information is printed with `cat` there is no real need to pass any object back.
A list containing the non-zero elements and the density for `summary` for class `spam`.
A list containing the non-zero elements of the factor, the density and the fill-in for `summary` for class `spam.chol.NgPeyton`.

Author(s)

Reinhard Furrer

See Also

`spam.options`

Examples

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

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

`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

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

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

Arguments

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

Details

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

`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.

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.

Note

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

Author(s)

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

References

See references in chol.

See Also

det, chol and ordering.

Examples

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

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

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

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


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

---

spam-class *Class "spam"*

Description

The spam class is a representation of sparse matrices.

Objects from the Class

Objects can be created by calls of the form new("spam", entries, colindices, 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"`

Methods

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

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

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

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

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

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

cbind `signature(x = "spam")`: binds sparse matrices.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

rbind `signature(x = "spam")`: binds sparse matrices.

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.

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

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

Details

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

Note

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

Author(s)

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

Examples

```
showMethods("as.spam")

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

spam.chol.NgPeyton-class *Class "spam.chol.NgPeyton"*

Description

Result of a Cholesky decomposition with the `NgPeyton` method

Objects from the Class

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

Methods

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

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

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

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

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

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

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

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)
```

`spam` *Sparse Matrix Class*

Description

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

Usage

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

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

is.spam(x)
```

Arguments

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

Details

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

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

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

Value

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

Note

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

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

Author(s)

Reinhard Furrer

References

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

See Also

`SPAM` general overview of the package. `spam.options` for details about the `safemode` flag. `read.MM` and `foreign` to create `spam` matrices from MatrixMarket files and from certain `Matrix/SparseM` formats.

Examples

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

getClass("spam")


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

`spam internal` *Spam internal and auxiliary functions*

Description

The functions listed below are auxiliary functions but are exported by the NAMESPACE. The user should not require to call these directly.

Details

The functions are listed here for a better understanding of the code (to fulfill the tutorial style paradigm).

`validspamobject(object)` A few sanity checks if `object` is a proper `spam` object.

`dcheck(x)`, `icheck(x)` testing and forcing of `x` to doubles and integers. These functions are used when calling Fortran routines.

Note

The integers `int0`, `int1` and `int2` (0-2) are not exported.

`todo` *Small "ToDo" list*

Description

List of what needs to be done within `spam`

Usage

```
todo()
```

Details

This is a non exhaustive list of where we need to work on spam (of course the list is in random order):

- extend demo(s)

- write vignette

- complete help files

- extend basic matrix operatation, comparisons, etc:
`unique`, `duplicated`, ...
- improve subsetting via row extraction, incorporate matrix permutation

- implement other Cholesky routines (one eye glances to the LDL library).

- what about an LU/SVD decomposition?

Any other items are welcome (⟨rfurrer@mines.edu⟩).

`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.list` for the inverse operation and `foreign` for other transformations.

Examples

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

`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
```

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

Arguments

`...`	vectors, matrices or `spam` objects. See Details and Value
`deparse.level`	for compatibility reason here. Only `0` is implemented.

Details

rbind and cbind are not exactly symmetric in how the objects are processed. The former is essentially an concatenation of the slots due to the sparse storage format. Different types of inputs are handled differently. The former 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

See Also

cbind,spam-method.

Examples

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

y <- cbind(x, 7)

# for some large matrices the following might be slightly faster:

t( cbind( t(x), t(x)))

## Not run:
# Method is only defined for the first argument:
cbind(diag(2),diag.spam(2))
## End(Not run)
```

Index