Package ‘SparseM’

July 17, 2024

Version 1.84-2

Maintainer Roger Koenker <rkoenker@uiuc.edu>

Depends R (>= 2.15), methods

Imports graphics, stats, utils

VignetteBuilder knitr

Suggests knitr

Description Some basic linear algebra functionality for sparse matrices is provided: including Cholesky decomposition and backsolving as well as standard R subsetting and Kronecker products.

License GPL (>= 2)

Title Sparse Linear Algebra

URL http://www.econ.uiuc.edu/~roger/research/sparse/sparse.html

NeedsCompilation yes

Author Roger Koenker [cre, aut],
Pin Tian Ng [ctb] (Contributions to Sparse QR code),
Yousef Saad [ctb] (author of sparskit2),
Ben Shaby [ctb] (author of chol2csr),
Martin Maechler [ctb] (chol() tweaks; S4,
<https://orcid.org/0000-0002-8685-9910>)

Repository CRAN

Date/Publication 2024-07-17 16:10:06 UTC

Contents

`character or NULL-class` *Class "character or NULL"*

Description

A virtual class needed by the "matrix.csc.hb" class

Objects from the Class

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

Methods

No methods defined with class "character or NULL" in the signature.

`lsq` *Least Squares Problems in Surveying*

Description

One of the four matrices from the least-squares solution of problems in surveying that were used by Michael Saunders and Chris Paige in the testing of LSQR

Usage

```
data(lsq)
```

Format

A list of class `matrix.csc.hb` or `matrix.ssc.hb` depending on how the coefficient matrix is stored with the following components:

ra ra component of the csc or ssc format of the coefficient matrix, X.

ja ja component of the csc or ssc format of the coefficient matrix, X.

ia ia component of the csc or ssc format of the coefficient matrix, X.

rhs.ra ra component of the right-hand-side, y, if stored in csc or ssc format; right-hand-side stored in dense vector or matrix otherwise.

rhs.ja ja component of the right-hand-side, y, if stored in csc or ssc format; a null vector otherwise.

rhs.ia ia component of the right-hand-side, y, if stored in csc or ssc format; a null vector otherwise.

xexact vector of the exact solutions, b, if they exist; a null vector o therwise.

guess vector of the initial guess of the solutions if they exist; a null vector otherwise.

dim dimenson of the coefficient matrix, X.

rhs.dim dimenson of the right-hand-side, y.

rhs.mode storage mode of the right-hand-side; can be full storage or same format as the coefficient matrix.

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
http://www.econ.uiuc.edu/~roger/research/home.html

Matrix Market, https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/lsq/lsq.html

See Also

`read.matrix.hb`

Examples

```
data(lsq)
class(lsq) # -> [1] "matrix.csc.hb"
model.matrix(lsq)->X
class(X) # -> "matrix.csr"
dim(X) # -> [1] 1850  712
y <- model.response(lsq) # extract the rhs
length(y) # [1] 1850
```

matrix.coo-class *Class "matrix.coo" – Sparse Matrices in [Coo]rdinate Format*

Description

Class for sparse matrices stored in coordinate aka “triplet” format, storing for each non-zero entry `x[i,j]` the triplet `(i,j, x[i,j])`, in slots `(ia, ja, ra)`.

Objects from the Class

Objects can be created by calls of the form `new("matrix.coo", ...)`, but typically rather by `as.matrix.coo()`.

Slots

`ra`: Object of class `numeric`, a real array of nnz elements containing the non-zero elements of A.

`ja`: Object of class `integer`, an integer array of nnz elements containing the column indices of the elements stored in ‘ra’.

`ia`: Object of class `integer`, an integer array of nnz elements containing the row indices of the elements stored in ‘ra’.

`dimension`: Object of class `integer`, dimension of the matrix

Methods

as.matrix.coo `signature(x = "matrix.coo")`: ...

as.matrix.csr `signature(x = "matrix.coo")`: ...

as.matrix `signature(x = "matrix.coo")`: ...

dim `signature(x = "matrix.coo")`: ...

See Also

`matrix.csr-class`

Examples

```
try(  new("matrix.coo") ) # fails currently {FIXME!}     # the 1x1 matrix [0]

## correponds to base  matrix()
mcoo <- new("matrix.coo", ra=NA_real_, ia = 1L, ja = 1L, dimension = c(1L, 1L))
mcoo # currently *does* print but wrongly:  as.matrix.csr(<matrix.coo>) fails to keep NA !!
co2 <- new("matrix.coo", ra = c(-Inf, -2, 3, Inf),
           ia = c(1L,1:3), ja = 2L*(1:4), dimension = c(7L, 12L))
co2 # works fine (as has no NA)

## Sparse Diagonal (from "numeric"):
as(1:5, "matrix.diag.csr") # a sparse version of  diag(1:5)
```

matrix.csc-class *Class "matrix.csc" - Sparse Matrices in [C]ompressed [S]parse [C]olumn Format*

Description

A class for sparse matrices stored in compressed sparse column ('csc') format.

Objects from the Class

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

Slots

`ra`**:** Object of class `numeric`, a real array of nnz elements containing the non-zero elements of A, stored in column order. Thus, if i<j, all elements of column i precede elements from column j. The order of elements within the column is immaterial.

`ja`**:** Object of class `integer`, an integer array of nnz elements containing the row indices of the elements stored in 'ra'.

`ia`**:** Object of class `integer`, an integer array of n+1 elements containing pointers to the beginning of each column in the arrays 'ra' and 'ja'. Thus 'ia[i]' indicates the position in the arrays 'ra' and 'ja' where the ith column begins. The last, (n+1)st, element of 'ia' indicates where the n+1 column would start, if it existed.

`dimension`**:** Object of class `integer`, dimension of the matrix

Methods

as.matrix.csr `signature(x = "matrix.csc")`: ...

as.matrix.ssc `signature(x = "matrix.csc")`: ...

as.matrix.ssr `signature(x = "matrix.csc")`: ...

as.matrix `signature(x = "matrix.csc")`: ...

chol `signature(x = "matrix.csc")`: ...

dim `signature(x = "matrix.csc")`: ...

t `signature(x = "matrix.csc")`: ...

See Also

`matrix.csr-class`

Examples

```
cscM <- as.matrix.csc(as(diag(4:1), "matrix.csr"))
cscM
str(cscM)
stopifnot(identical(dim(cscM), c(4L, 4L)))
```

matrix.csc.hb-class *Class "matrix.csc.hb" - Column Compressed Sparse Matrices stored in Harwell-Boeing Format*

Description

A class consisting of the coefficient matrix and the right-hand-side of a linear system of equations, initial guess of the solution and the exact solutions if they exist stored in external files using the Harwell-Boeing format.

Objects from the Class

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

Slots

`ra`**:** Object of class `numeric`, ra component of the csc or ssc format of the coefficient matrix, X.

`ja`**:** Object of class `integer`, ja component of the csc or ssc format of the coefficient matrix, X.

`ia`**:** Object of class `numeric`, ia component of the csc or ssc format of the coefficient matrix, X.

`rhs.ra`**:** Object of class `numeric`, ra component of the right-hand-side, y, if stored in csc or ssc format; right-hand-side stored in dense vector or matrix otherwise.

`guess`**:** Object of class `numeric` or `NULL` vector of the initial guess of the solutions if they exist; a null vector otherwise.

`xexact`**:** Object of class `numeric` or `NULL` vector of the exact solutions, b, if they exist; a null vector otherwise.

`dimension`**:** Object of class `integer`, dimenson of the coefficient matrix, X.

`rhs.dim`**:** Object of class `integer`, dimenson of the right-hand-side, y.

`rhs.mode`**:** Object of class `character` or `NULL` storage mode of the right-hand-side; can be full storage or same format as the coefficient matrix.

Methods

model.matrix `signature(object = "matrix.csc.hb")`: ...

show `signature(object = "matrix.csc.hb")`: `show()` the object, notably also when auto-printing.

See Also

`model.matrix`, `model.response`, `read.matrix.hb`, `matrix.ssc.hb-class`

matrix.csr-class	*Class "matrix.csr" - Sparse Matrices in Compressed Sparse Row Format*

Description

A class for sparse matrices stored in compressed sparse row ('csr') format.

Objects from the Class

Objects can be created by calls of the form `new("matrix.csr", ...)` and coerced from various other formats. Coercion of integer scalars and vectors into identity matrices and diagonal matrices respectively is accomplished by `as(x,"matrix.diag.csr")` which generates an object that has all the rights and responsibilties of the `"matrix.csr"` class.

The default `"matrix.csr"` object, i.e., `new("matrix.csr")`, is a scalar (1 by 1) matrix with element 0.

Slots

`ra`**:** Object of class `numeric`, a real array of nnz elements containing the non-zero elements of A, stored in row order. Thus, if $i < j$, all elements of row i precede elements from row j. The order of elements within the rows is immaterial.

`ja`**:** Object of class `integer`, an integer array of nnz elements containing the column indices of the elements stored in `ra`.

`ia`**:** A class `integer` array of n+1 elements containing pointers to the beginning of each row in the arrays `ra` and `ja`. Thus 'ia[i]' indicates the position in the arrays `ra` and `ja` where the ith row begins. The last, (n+1)st, element of `ia` indicates where the n+1 row would start, if it existed.

`dimension`**:** An `integer`, dimension of the matrix

Methods

%*% `signature(x = "matrix.csr", y = "matrix.csr")`: ...

%*% `signature(x = "matrix.csr", y = "matrix")`: ...

%*% `signature(x = "matrix.csr", y = "numeric")`: ...

%*% `signature(x = "matrix", y = "matrix.csr")`: ...

%*% `signature(x = "numeric", y = "matrix.csr")`: ...

as.matrix.csc `signature(x = "matrix.csr")`: ...

as.matrix.ssc `signature(x = "matrix.csr")`: ...

as.matrix.ssr `signature(x = "matrix.csr")`: ...

as.matrix.coo `signature(x = "matrix.csr")`: ...

as.matrix `signature(x = "matrix.csr")`: ...

chol `signature(x = "matrix.csr")`: ...

diag `signature(x = "matrix.csr")`: ...

diag<- `signature(x = "matrix.csr")`: ...

dim `signature(x = "matrix.csr")`: ...

image `signature(x = "matrix.csr")`: ...

solve `signature(a = "matrix.csr")`: ...

t `signature(x = "matrix.csr")`: ...

diff `signature(x = "matrix.csr")`: ...

diag<- `signature(x = "matrix.diag.csr")`: ...

See Also

`matrix.csc-class`

Examples

```
new("matrix.csr")      # the 1x1 matrix [0]
new("matrix.diag.csr") # the 'same'

as(1:5, "matrix.diag.csr") # a sparse version of  diag(1:5)
```

---

matrix.csr.chol-class *Class "matrix.csr.chol" (Block Sparse Cholesky Decomposition)*

Description

A class of objects returned from Ng and Peyton's (1993) block sparse Cholesky algorithm.

Details

Note that the `perm` and notably `invp` maybe important to back permute rows and columns of the decompositions, see the Examples, and our `chol` help page.

Objects from the Class

Objects may be created by calls of the form `new("matrix.csr.chol", ...)`, but typically result from `chol(<matrix.csr>)`.

Slots

`nrow`**:** an `integer`, the number of rows of the original matrix, or in the linear system of equations.

`nnzlindx`**:** Object of class `numeric`, number of non-zero elements in `lindx`

`nsuper`**:** an `integer`, the number of supernodes of the decomposition

`lindx`**:** Object of class `integer`, vector of integer containing, in column major order, the row subscripts of the non-zero entries in the Cholesky factor in a compressed storage format

`xlindx`**:** Object of class `integer`, vector of integer of pointers for `lindx` `nnzl`: of class `"numeric"`, an integer, the number of non-zero entries, including the diagonal entries, of the Cholesky factor stored in `lnz`

`lnz`: a `numeric` vector of the entries of the Cholesky factor

`xlnz`: an `integer` vector, the column pointers for the Cholesky factor stored in `lnz`

`invp`: inverse permutation vector, `integer`

`perm`: permutation vector, `integer`

`xsuper`: Object of class `integer`, array containing the supernode partioning

`det`: `numeric`, the determinant of the Cholesky factor

`log.det`: `numeric`, the log determinant of the Cholesky factor

`ierr`: an `integer`, the error flag (from Fortran's '`src/chol.f`')

`time`: `numeric`, unused (always `0.`) currently.

Methods

as.matrix.csr `signature(x = "matrix.csr.chol",upper.tri=TRUE)`: to get the sparse (`"matrix.csr"`) upper triangular matrix corresponding to the Cholesky decomposition.

backsolve `signature(r = "matrix.csr.chol")`: for computing $R^{-1}b$ when the Cholesky decomposition is $A = R'R$.

See Also

Base R's `chol` and **SparseM**'s `chol`, notably for examples; `backsolve`

Examples

```
x5g <- new("matrix.csr",
          ra = c(300, 130, 5, 130, 330,
                 125, 10, 5, 125, 200, 70,
                 10, 70, 121.5, 1e30),
          ja = c(1:3, 1:4, 1:4, 2:5),
          ia = c(1L, 4L, 8L, 12L, 15L, 16L),
          dimension = c(5L, 5L))
(m5g <- as.matrix(x5g)) # yes, is symmetric, and positive definite:
eigen(m5g, only.values=TRUE)$values  # all positive (but close to singular)
ch5g <- chol(x5g)
str(ch5g) # --> the slots of the "matrix.csr.chol" class
mch5g <- as.matrix.csr(ch5g)
print.table(as.matrix(mch5g), zero.print=".") # indeed upper triagonal w/ positive diagonal

## x5 has even more extreme entry at [5,5]:
x5 <- x5g; x5[5,5] <- 2.9e32
m5 <- as.matrix(x5)
(c5 <- chol(m5))# still fine, w/ [5,5] entry = 1.7e16 and other diag.entries in (9.56, 17.32)
ch5 <- chol(x5) # --> warning  "Replaced 3 tiny diagonal entries by 'Large'"
                # gave error for a while
(mmc5 <- as.matrix(as.matrix.csr(ch5)))
        # yes, these replacements were extreme, and the result is "strange'
## Solve the problem (here) specifying non-default  singularity-tuning par 'tiny':
```

```
ch5. <- chol(x5, tiny = 1e-33)
(mmc5. <- as.matrix(as.matrix.csr(ch5.))) # looks much better.
## Indeed: R'R  back-permuted *is* the original matrix x5, here m5:
(RtR <- crossprod(mmc5.)[ch5.@invp, ch5.@invp])
          all.equal(m5, RtR, tolerance = 2^-52)
stopifnot(all.equal(m5, RtR, tolerance = 1e-14)) # on F38 Linux, only need tol = 1.25e-16
```

---

matrix.ssc-class *Class "matrix.ssc" - Sparse Matrices in [S]ymmetric [S]parse [C]olumn Format*

Description

A class for sparse matrices stored in symmetric sparse column ('ssc') format.

Objects from the Class

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

Slots

`ra`**:** Object of class `numeric`, a real array of nnz elements containing the non-zero elements of the lower triangular part of A, stored in column order. Thus, if i<j, all elements of column i precede elements from column j. The order of elements within the column is immaterial.

`ja`**:** Object of class `integer`, an integer array of nnz elements containing the row indices of the elements stored in 'ra'.

`ia`**:** Object of class `integer`, an integer array of n+1 elements containing pointers to the beginning of each column in the arrays 'ra' and 'ja'. Thus 'ia[i]' indicates the position in the arrays 'ra' and 'ja' where the ith column begins. The last, (n+1)st, element of 'ia' indicates where the n+1 column would start, if it existed.

`dimension`**:** Object of class `integer`, dimension of the matrix

Methods

as.matrix.csc `signature(x = "matrix.ssc")`: ...

as.matrix.csr `signature(x = "matrix.ssc")`: ...

as.matrix.ssr `signature(x = "matrix.ssc")`: ...

as.matrix `signature(x = "matrix.ssc")`: ...

dim `signature(x = "matrix.ssc")`: ...

See Also

`matrix.csr-class`

`matrix.ssc.hb-class` *Class "matrix.ssc.hb"*

Description

A new class consists of the coefficient matrix and the right-hand-side of a linear system of equations, initial guess of the solution and the exact solutions if they exist stored in external files using the Harwell-Boeing format.

Objects from the Class

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

Slots

`ra`: Object of class `numeric`, ra component of the csc or ssc format of the coefficient matrix, X.

`ja`: Object of class `integer`, ja component of the csc or s sc format of the coefficient matrix, X.

`ia`: Object of class `integer`, ia component of the csc or ssc format of the coefficient matrix, X.

`rhs.ra`: Object of class `numeric`, ra component of the right-hand-side, y, if stored in csc or ssc format; right-hand-side stored in dense vector or matrix otherwise.

`guess`: Object of class `numeric or NULL` vector of the initial guess of the solutions if they exist; a null vector otherwise.

`xexact`: Object of class `numeric or NULL` vector of the exact solutions, b, if they exist; a null vector otherwise.

`dimension`: Object of class `integer`, dimenson of the coefficient matrix, X.

`rhs.dim`: Object of class `integer`, dimenson of the right-hand-side, y.

`rhs.mode`: Object of class `character or NULL` storage mode of the right-hand-side; can be full storage or same format as the coefficient matrix.

Extends

Class `"matrix.csc.hb"`, directly.

Methods

model.matrix `signature(object = "matrix.ssc.hb")`: ...

See Also

`model.matrix`, `model.response`, `read.matrix.hb`, `matrix.csc.hb-class` `matrix.ssr-class` *Class "matrix.ssr" - Sparse Matrices in [S]ymmetric [S]parse [R]ow Format*

Description

A class for sparse matrices stored in symmetric sparse row ('ssr') format.

Objects from the Class

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

Slots

`ra`**:** Object of class `numeric`, a real array of nnz elements containing the non-zero elements of the lower triangular part of A, stored in row order. Thus, if i<j, all elements of row i precede elements from row j. The order of elements within the rows is immaterial.

`ja`**:** Object of class `integer`, an integer array of nnz elements containing the column indices of the elements stored in 'ra'.

`ia`**:** Object of class `integer`, an integer array of n+1 elements containing pointers to the beginning of each row in the arrays 'ra' and 'ja'. Thus 'ia[i]' indicates the position in the arrays 'ra' and 'ja' where the ith row begins. The last, (n+1)st, element of 'ia' indicates where the n+1 row would start, if it existed.

`dimension`**:** Object of class `integer`, dimension of the matrix

Methods

as.matrix.csc `signature(x = "matrix.ssr")`: ...

as.matrix.csr `signature(x = "matrix.ssr")`: ...

as.matrix.ssr `signature(x = "matrix.ssr")`: ...

as.matrix `signature(x = "matrix.ssr")`: ...

dim `signature(x = "matrix.ssr")`: ...

See Also

`matrix.csr-class`

Examples

```
ssr <- as.matrix.ssr(diag(c(2,3,5,7)))
ssr
```

mslm-class *Class "mslm"*

Description

A sparse extension of `lm`

Objects from the Class

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

Slots

`coefficients`**:** Object of class `numeric` estimated coefficients

`chol`**:** Object of class `matrix.csr.chol` generated by the function `chol`

`residuals`**:** Object of class `"numeric"` residuals

`fitted`**:** Object of class `"numeric"` fitted values

Extends

Class `"lm"`, directly. Class `"slm"`, directly. Class `"oldClass"`, by class "lm".

Methods

coef `signature(object = "mslm")`: ...

fitted `signature(object = "mslm")`: ...

residuals `signature(object = "mslm")`: ...

summary `signature(object = "mslm")`: ...

See Also

`slm`

numeric or NULL-class *Class "numeric or NULL"*

Description

A virtual class needed by the "matrix.csc.hb" class

Objects from the Class

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

Methods

No methods defined with class "numeric or NULL" in the signature.

slm *Fit a linear regression model using sparse matrix algebra*

Description

This is a function to illustrate the use of sparse linear algebra to solve a linear least squares problem using Cholesky decomposition. The syntax and output attempt to emulate `lm()` but may fail to do so fully satisfactorily. Ideally, this would eventually become a method for `lm`. The main obstacle to this step is that it would be necessary to have a model.matrix function that returned an object in sparse csr form. For the present, the objects represented in the formula must be in dense form. If the user wishes to specify fitting with a design matrix that is already in sparse form, then the lower level function `slm.fit()` should be used.

Usage

```
slm(formula, data, weights, na.action, method = "csr", contrasts = NULL, ...)
```

Arguments

`formula`	a formula object, with the response on the left of a ~ operator, and the terms, separated by + operators, on the right. As in `lm()`, the response variable in the formula can be matrix valued.
`data`	a data.frame in which to interpret the variables named in the formula, or in the subset and the weights argument. If this is missing, then the variables in the formula should be on the search list. This may also be a single number to handle some special cases – see below for details.
`weights`	vector of observation weights; if supplied, the algorithm fits to minimize the sum of the weights multiplied into the absolute residuals. The length of weights must be the same as the number of observations. The weights must be nonnegative and it is strongly recommended that they be strictly positive, since zero weights are ambiguous.
`na.action`	a function to filter missing data. This is applied to the model.frame after any subset argument has been used. The default (with `na.fail`) is to create an error if any missing values are found. A possible alternative is `na.omit`, which deletes observations that contain one or more missing values.
`method`	there is only one method based on Cholesky factorization
`contrasts`	a list giving contrasts for some or all of the factors default = `NULL` appearing in the model formula. The elements of the list should have the same name as the variable and should be either a contrast matrix (specifically, any full-rank matrix with as many rows as there are levels in the factor), or else a function to compute such a matrix given the number of levels.
`...`	additional arguments for the fitting routines

Value

A list of class `slm` consisting of:

`coefficients`	estimated coefficients
`chol`	cholesky object from fitting
`residuals`	residuals
`fitted`	fitted values
`terms`	terms
`call`	call

...

Author(s)

Roger Koenker

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
http://www.econ.uiuc.edu/~roger/research/home.html

See Also

`slm.methods` for methods `summary`, `print`, `fitted`, `residuals` and `coef` associated with class `slm`, and `slm.fit` for lower level fitting functions. The latter functions are of special interest if you would like to pass a sparse form of the design matrix directly to the fitting process.

Examples

```
data(lsq)
X <- model.matrix(lsq) #extract the design matrix
y <- model.response(lsq) # extract the rhs
X1 <- as.matrix(X)
slm.time <- system.time(slm(y~X1-1) -> slm.o) # pretty fast
lm.time <- system.time(lm(y~X1-1) -> lm.o) # very slow
cat("slm time =",slm.time,"\n")
cat("slm Results: Reported Coefficients Truncated to 5  ","\n")
sum.slm <- summary(slm.o)
sum.slm$coef <- sum.slm$coef[1:5,]
sum.slm
cat("lm time =",lm.time,"\n")
cat("lm Results: Reported Coefficients Truncated to 5  ","\n")
sum.lm <- summary(lm.o)
sum.lm$coef <- sum.lm$coef[1:5,]
sum.lm
```

slm-class *Class "slm"*

Description

A sparse extension of `lm`

Objects from the Class

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

Slots

`coefficients`**:** Object of class `numeric` estimated coefficients

`chol`**:** Object of class `matrix.csr.chol` generated by function `chol`

`residuals`**:** Object of class `"numeric"` residuals

`fitted`**:** Object of class `"numeric"` fitted values

Extends

Class `"lm"`, directly. Class `"oldClass"`, by class "lm".

Methods

coef `signature(object = "slm")`: ...

fitted `signature(object = "slm")`: ...

residuals `signature(object = "slm")`: ...

summary `signature(object = "slm")`: ...

See Also

`slm`

slm.fit *Internal slm fitting functions*

Description

Fitting functions for sparse linear model fitting.

Usage

```
slm.fit(x,y,method, ...)
slm.wfit(x,y,weights,...)
slm.fit.csr(x, y, ...)
```

Arguments

`x`	design matrix.
`y`	vector of response observations.
`method`	only `csr` is supported currently
`weights`	an optional vector of weights to be used in the fitting process. If specified, weighted least squares is used with weights 'weights' (that is, minimizing $$\sum w_i * e_i^2$$ The length of weights must be the same as the number of observations. The weights must be nonnegative and it is strongly recommended that they be strictly positive, since zero weights are ambiguous.
`...`	additional arguments.

Details

`slm.fit` and `slm.wfit` call `slm.fit.csr` to do Cholesky decomposition and then backsolve to obtain the least squares estimated coefficients. These functions can be called directly if the user is willing to specify the design matrix in `matrix.csr` form. This is often advantageous in large problems to reduce memory requirements.

Value

A list of class `slm` consisting of:

`coef`	estimated coefficients
`chol`	cholesky object from fitting
`residuals`	residuals
`fitted`	fitted values
`df.residual`	degrees of freedom
`terms`	terms
`call`	call

...

Author(s)

Roger Koenker

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
`http://www.econ.uiuc.edu/~roger/research/home.html`

See Also

`slm`

Examples

```
data(lsq)
X <- model.matrix(lsq) #extract the design matrix
y <- model.response(lsq) # extract the rhs
class(X) # -> "matrix.csr"
class(y) # -> NULL
slm.fit(X,y)->slm.fit.o # this is much more efficient in memory usage than slm()
slm(y~as.matrix(X)-1) -> slm.o # this requires X to be transformed into dense mode
cat("Difference between `slm.fit' and `slm' estimated coefficients =",
sum(abs(slm.fit.o$coef-slm.o$coef)),"\n")
```

---

slm.methods *Methods for slm objects*

Description

Summarize, print, and extract objects from `slm` objects.

Usage

```
## S3 method for class 'slm'
summary(object, correlation, ...)
## S3 method for class 'mslm'
summary(object, ...)
## S3 method for class 'slm'
print(x, digits, ...)
## S3 method for class 'summary.slm'
print(x, digits, symbolic.cor, signif.stars, ...)
## S3 method for class 'slm'
fitted(object, ...)
## S3 method for class 'slm'
residuals(object, ...)
## S3 method for class 'slm'
coef(object, ...)
## S3 method for class 'slm'
extractAIC(fit, scale = 0, k = 2, ...)
## S3 method for class 'slm'
deviance(object, ...)
```

Arguments

`object, x, fit`	object of class `slm`.
`digits`	minimum number of significant digits to be used for most numbers.
`scale`	optional numeric specifying the scale parameter of the model, see 'scale' in 'step'. Currently only used in the '"lm"' method, where 'scale' specifies the estimate of the error variance, and 'scale = 0' indicates that it is to be estimated by maximum likelihood.

`k` numeric specifying the "weight" of the equivalent degrees of freedom ('edf') part in the AIC formula.

`symbolic.cor` logical; if `TRUE`, the correlation of coefficients will be printed. The default is `FALSE`

`signif.stars` logical; if `TRUE`, P-values are additionally encoded visually as "significance stars" in order to help scanning of long coefficient tables. It defaults to the 'show.signif.stars' slot of 'options'.

`correlation` logical; if `TRUE`, the correlation matrix of the estimated parameters is returned and printed.

`...` additional arguments passed to methods.

Value

`print.slm` and `print.summary.slm` return invisibly. `fitted.slm`, `residuals.slm`, and `coef.slm` return the corresponding components of the `slm` object. `extractAIC.slm` and `deviance.slm` return the AIC and deviance values of the fitted object.

Author(s)

Roger Koenker

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
http://www.econ.uiuc.edu/~roger/research/home.html

See Also

`slm`

Examples

```
data(lsq)
X <- model.matrix(lsq) #extract the design matrix
y <- model.response(lsq) # extract the rhs
X1 <- as.matrix(X)
slm.time <- system.time(slm(y~X1-1) -> slm.o) # pretty fast
cat("slm time =",slm.time,"\n")
cat("slm Results: Reported Coefficients Truncated to 5  ","\n")
sum.slm <- summary(slm.o)
sum.slm$coef <- sum.slm$coef[1:5,]
sum.slm
fitted(slm.o)[1:10]
residuals(slm.o)[1:10]
coef(slm.o)[1:10]
```

`SparseM.hb` *Harwell-Boeing Format Sparse Matrices*

Description

Read, and extract components of data in Harwell-Boeing sparse matrix format.

Usage

```
read.matrix.hb(file)
model.matrix(object, ...)
model.response(data,type)
```

Arguments

`file`	file name to read from or
`data, object`	an object of either 'matrix.csc.hb' or 'matrix.ssc.hb' class
`type`	One of '"any"', '"numeric"', '"double"'. Using the either of latter two coerces the result to have storage mode '"double"'
`...`	additional arguments to model.matrix

Details

Sparse coefficient matrices in the Harwell-Boeing format are stored in 80-column records. Each file begins with a multiple line header block followed by two, three or four data blocks. The header block contains summary information on the storage formats and storage requirements. The data blocks contain information of the sparse coefficient matrix and data for the right-hand-side of the linear system of equations, initial guess of the solution and the exact solutions if they exist. The function `model.matrix` extracts the X matrix component. The function `model.response` extracts the y vector (or matrix). The function `model.guess` extracts the guess vector. The function `model.xexact` extracts the xexact vector. This function is written in R replacing a prior implementation based on iohb.c which had memory fault difficulties. The function write.matrix.hb has been purged; users wishing to write matrices in Harwell-Boeing format are advised to convert SparseM matrices to Matrix classes and use writeHB from the Matrix package. Contributions of code to facilitate this conversion would be appreciated!

Value

The function `read.matrix.hb` returns a list of class `matrix.csc.hb` or `matrix.ssc.hb` depending on how the coefficient matrix is stored in the `file`.

`ra`	ra component of the csc or ssc format of the coefficient matrix, X.
`ja`	ja component of the csc or ssc format of the coefficient matrix, X.
`ia`	ia component of the csc or ssc format of the coefficient matrix, X.
`rhs.ra`	ra component of the right-hand-side, y, if stored in csc or ssc format; right-hand-side stored in dense vector or matrix otherwise.

`rhs.ja`	ja component of the right-hand-side, y, if stored in csc or ssc format; a null vector otherwise.
`rhs.ia`	ia component of the right-hand-side, y, if stored in csc or ssc format; a null vector otherwise.
`xexact`	vector of the exact solutions, b, if they exist; a null vector otherwise.
`guess`	vector of the initial guess of the solutions if they exist; a null vector otherwise.
`dimension`	dimenson of the coefficient matrix, X.
`rhs.dim`	dimenson of the right-hand-side, y.
`rhs.mode`	storage mode of the right-hand-side; can be full storage or same format as the coefficient matrix, for the moment the only allowed mode is "F" for full, or dense mode.

The function `model.matrix` returns the X matrix of class `matrix.csr`. The function `model.response` returns the y vector (or matrix). The function `model.guess` returns the guess vector (or matrix). The function `model.xexact` returns the xexact vector (or matrix).

Author(s)

Pin Ng

References

Duff, I.S., Grimes, R.G. and Lewis, J.G. (1992) User's Guide for Harwell-Boeing Sparse Matrix Collection at https://math.nist.gov/MatrixMarket/collections/hb.html

See Also

`slm` for sparse version of `lm`
`SparseM.ops` for operators on class `matrix.csr`
`SparseM.solve` for linear equation solving for class `matrix.csr`
`SparseM.image` for image plotting of class `matrix.csr`
`SparseM.ontology` for coercion of class `matrix.csr`

Examples

```
Xy <- read.matrix.hb(system.file("extdata","lsq.rra",package = "SparseM"))
class(Xy) # -> [1] "matrix.csc.hb"
X <- model.matrix(Xy)->X
class(X) # -> "matrix.csr"
dim(X) # -> [1] 1850  712
y <- model.response(Xy) # extract the rhs
length(y) # [1] 1850
Xy <- read.matrix.hb(system.file("extdata","rua_32_ax.rua",package = "SparseM"))
X <- model.matrix(Xy)
y <- model.response(Xy) # extract the rhs
g <- model.guess(Xy) # extract the guess
a <- model.xexact(Xy) # extract the xexact
fit <- solve(t(X) %*% X, t(X) %*% y) # compare solution with xexact solution
```

SparseM.image *Image Plot for Sparse Matrices*

Description

Display the pattern of non-zero entries of a matrix of class `matrix.csr`.

Usage

```
## S4 method for signature 'matrix.csr'
image(x, col=c("white","gray"),
      xlab="column", ylab="row", ...)
```

Arguments

`x`	a matrix of class `matrix.csr`.
`col`	a list of colors such as that generated by `rainbow`. Defaults to c("white","gray")
`xlab, ylab`	each a character string giving the labels for the x and y axis.
`...`	additional arguments.

Details

The pattern of the non-zero entries of a sparse matrix is displayed. By default nonzero entries of the matrix appear as gray blocks and zero entries as white background.

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
`http://www.econ.uiuc.edu/~roger/research/home.html`

See Also

`SparseM.ops`, `SparseM.solve`, `SparseM.ontology`

Examples

```
a <- rnorm(20*5)
A <- matrix(a,20,5)
A[row(A)>col(A)+4|row(A)<col(A)+3] <- 0
b <- rnorm(20*5)
B <- matrix(b,20,5)
B[row(A)>col(A)+2|row(A)<col(A)+2] <- 0
image(as.matrix.csr(A)%*%as.matrix.csr(t(B)))
```

SparseM.ontology *Sparse Matrix Class Ontology*

Description

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

Usage

```
as.matrix.csr(x, nrow, ncol, eps = .Machine$double.eps, ...)
## S4 method for signature 'matrix.csr.chol'
as.matrix.csr(x, nrow, ncol, eps, upper.tri=TRUE, ...)
## S4 method for signature 'matrix.csr'
as.matrix.csc(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
## S4 method for signature 'matrix.coo'
as.matrix.ssr(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
## S4 method for signature 'matrix.csc'
as.matrix.ssc(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
## S4 method for signature 'matrix.csr'
as.matrix.coo(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)

is.matrix.csr(x)
is.matrix.csc(x)
is.matrix.ssr(x)
is.matrix.ssc(x)
is.matrix.coo(x)
```

Arguments

`x`	is a matrix, or vector object, of either dense or sparse form
`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. This argument is only relevant when coercing matrices from dense to sparse form. Defaults to `eps = .Machine$double.eps`
`upper.tri`	`logical`, to choose upper or lower triangular matrix result.
`...`	other arguments

Details

The function `matrix.csc` acts like `matrix` to coerce a vector object to a sparse matrix object of class `matrix.csr`. This aspect of the code is in the process of conversion from S3 to S4 classes. For the most part the S3 syntax prevails. An exception is the code to coerce vectors to diagonal matrix form which uses `as(v,"matrix.diag.csr"`. The generic functions `as.matrix.xxx` coerce a matrix `x` into a matrix of storage class `matrix.xxx`. The argument matrix `x` may be of conventional dense form, or of any of the four supported classes: `matrix.csr`, `matrix.csc`, `matrix.ssr`, matrix.ssc. The generic functions is.matrix.xxx evaluate whether the argument is of class matrix.xxx. The function as.matrix transforms a matrix of any sparse class into conventional dense form. The primary storage class for sparse matrices is the compressed sparse row matrix.csr class. An *n* by *m* matrix *A* with real elements a_{ij}, stored in matrix.csr format consists of three arrays:

- ra: a real array of *nnz* elements containing the non-zero elements of *A*, stored in row order. Thus, if *i<j*, all elements of row *i* precede elements from row *j*. The order of elements within the rows is immaterial.
- ja: an integer array of *nnz* elements containing the column indices of the elements stored in ra.
- ia: an integer array of *n+1* elements containing pointers to the beginning of each row in the arrays ra and ja. Thus ia[i] indicates the position in the arrays ra and ja where the *i*th row begins. The last, *(n+1)*st, element of ia indicates where the *n+1* row would start, if it existed.

The compressed sparse column class matrix.csc is defined in an analogous way, as are the matrix.ssr, symmetric sparse row, and matrix.ssc, symmetric sparse column classes.

Note

as.matrix.ssr and as.matrix.ssc should ONLY be used with symmetric matrices.

as.matrix.csr(x), when x is an object of class matrix.csr.chol (that is, an object returned by a call to chol(a) when a is an object of class matrix.csr or matric.csc), by default returns an upper triangular matrix, which is *not* consistent with the result of chol in the **base** package. To get an lower triangular matric.csr matrix, use either as.matrix.csr(x, upper.tri = FALSE) or t(as.matrix.csr(x)).

References

Koenker, R and Ng, P. (2002) *SparseM: A Sparse Matrix Package for* R. http://www.econ.uiuc.edu/~roger/research/home.html

See Also

SparseM.hb for handling Harwell-Boeing sparse matrices.

Examples

```
t(m5 <- as.matrix.csr(c(-1:1,0,0)))
t(M4 <- as.matrix.csc(c(0:2,0), 4))
 (S3 <- as.matrix.ssr(diag(x = 0:2))) # *symmetric*
stopifnot(identical(dim(m5), c(5L, 1L)),
          identical(dim(M4), c(4L, 1L)),
          identical(dim(S3), c(3L, 3L)))

n1 <- 10
p <- 5
a <- round(rnorm(n1*p), 2)
a[abs(a) < 0.7] <- 0
A <- matrix(a,n1,p)
```

```
B <- t(A) %*% A
A.csr <- as.matrix.csr(A)
A.csc <- as.matrix.csc(A)
B.ssr <- as.matrix.ssr(B)
B.ssc <- as.matrix.ssc(B)
stopifnot(exprs = {
  is.matrix.csr(A.csr) # -> TRUE
  is.matrix.csc(A.csc) # -> TRUE
  is.matrix.ssr(B.ssr) # -> TRUE
  is.matrix.ssc(B.ssc) # -> TRUE
})
as.matrix(A.csr)
as.matrix(A.csc)
as.matrix(B.ssr)
as.matrix(B.ssc)
as.matrix.csr(0, 2,3)    # sparse matrix of all zeros
## Diagonal (sparse) :
as(4,   "matrix.diag.csr") # identity matrix of dimension 4
as(2:0, "matrix.diag.csr") # diagonal 3x3 matrix
```

`SparseM.ops` *Basic Linear Algebra for Sparse Matrices*

Description

Basic linear algebra operations for sparse matrices, mostly of class `matrix.csr`.

Arguments

`x`	matrix of class `matrix.csr`.
`y`	matrix of class `matrix.csr` or a dense matrix or vector.
`value`	replacement values.
`i, j`	vectors of elements to extract or replace.
`nrow`	optional number of rows for the result.
`lag`	an integer indicating which lag to use.
`differences`	an integer indicating the order of the difference.

Details

Linear algebra operations for matrices of class `matrix.csr` are designed to behave exactly as for regular matrices. In particular, matrix multiplication, kronecker product, 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. The method diag may be used to extract the diagonal of a `matrix.csr` object, to create a sparse diagonal see `SparseM.ontology`.

The function `determinant` computes the (log) determinant, of the argument, returning a `"det"` object as the base function. This is typically *preferred* over using the function `det()` which is a simple wrapper for `determinant()`, in a way it will work for our sparse matrices, as well. `determinant()` computes the determinant of the argument matrix. If the matrix is of class `matrix.csr` then it must be symmetric, or an error will be returned. If the matrix is of class `matrix.csr.chol` then the (pre-computed) determinant of the Cholesky factor is returned, i.e., the product of the diagonal elements.

The function `norm`, i.e. `norm(x, type)`, by default computes the "sup" (or `"M"`aximum norm, i.e., the maximum of the matrix elements. Optionally, this `type = "sup"` (equivalently, `type = "M"`) norm can be replaced by the Hilbert-Schmidt, `type = "HS"` or equivalently, `type = "F"` norm, or the `type = "l1"`, norm. Note that for historical reasons, the default `type` differs from R's own `norm()`, see the examples using `B`, below. The `"sup" === "M"` and `"HS" === "F"` equivalences have been introduced in **SparseM** version 1.84.

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
http://www.econ.uiuc.edu/~roger/research/home.html

See Also

`slm` for sparse linear model fitting. `SparseM.ontology` for coercion and other class relations involving our sparse matrix classes.

Examples

```
n1 <- 10
n2 <- 10
p <- 6
y <- rnorm(n1)
a <- round(rnorm(n1*p), 2)
a[abs(a) < 0.5] <- 0
A <- matrix(a, n1,p)
A.csr <- as.matrix.csr(A)

B <- matrix(c(1.5, 0,  0,  0, -1.4,    0,    0,    0,   0, -1.4,
              2,   0, -1,  0,  0,      2.1, -1.9, 1.4, 0,  0,
              0,-2.3,  0,  0, -1.9,    0,    0,    0,   0, -1.4,
              0,   0,  0,  0,  0,     -3,    0,    1.3, 0,  1.1,
              0,   0,  0,  0,  2,      0,    0,    0,  -1,  0,
              0,   0, -1.6,0,  0,      0,    0,    0,  -1.7,0),
            10L, 6L)
rcond(B) # 0.21 .. i.e., quite well conditioned
B.csr <- as.matrix.csr(B)
B.csr

## norm() : different 'type' for base R and  SparseM:
(nR <- vapply(c("1", "I", "F", "M", "2"), norm, 0, x = B))
##        1        I        F        M        2
## 8.400000 5.300000 7.372923 3.000000 4.464801
(nSpM <- vapply(c("sup","M",  "HS","F",  "l1"), norm, 0, x = B.csr))
##      sup         M        HS         F        l1
## 3.000000  3.000000  7.372923  7.372923 30.000000
```

```
stopifnot(all.equal(unname(nSpM[c("M", "F")]),
                    unname(nR  [c("M", "F" )]), tolerance = 1e-14))

# matrix transposition and multiplication
BtB <- crossprod(B) # == t(B) %*% B  {more efficiently}

A.csr %*% t(B.csr)
BtBs <- t(B.csr) %*% B.csr
BtBs
stopifnot(all.equal(    BtB,  as.matrix(BtBs),  tolerance = 1e-14),
          all.equal(det(BtB), print(det(BtBs)), tolerance = 1e-14))

# matrix  o  vector
stopifnot(all.equal(y %*%  A , y   %*% A.csr) ,
          all.equal(A %*% 1:6, A.csr %*% 1:6)
)

# kronecker product - via kronecker() methods:
A.csr %x% matrix(1:4,2,2)
```

`SparseM.solve` *Linear Equation Solving via Cholesky Decomposition for Sparse Matrices*

Description

`chol()` performs a Cholesky decomposition of a symmetric positive definite sparse matrix `x` of class `matrix.csr`.

`backsolve()` performs a triangular back-fitting to compute the solutions of a system of linear equations in one step.

`backsolve()` **and** `forwardsolve()` can also split the functionality of `backsolve` into two steps.

`solve()` combines `chol()` and `backsolve()` to compute the inverse of a matrix if the right-hand-side is missing.

Usage

```
chol(x, ...)
## S4 method for signature 'matrix.csr'
chol(x, pivot = FALSE,
    nsubmax, nnzlmax, tmpmax,
    eps = .Machine$double.eps, tiny = 1e-30, Large = 1e128, warnOnly = FALSE,
    cacheKb = 1024L, level = 8L, ...)

## S4 method for signature 'matrix.csr.chol'
backsolve(r, x, k, upper.tri, transpose,
          twice = TRUE, drop = TRUE, ...)
## S4 method for signature 'matrix.csr.chol'
forwardsolve(l, x, k, upper.tri, transpose)
```

```
## S4 method for signature 'matrix.csr'
solve(a, b, ...)
```

Arguments

- `a` — symmetric positive definite matrix of class "`matrix.csr`".
- `r, l` — object of class "`matrix.csr.chol`" as returned by the `chol()` method.
- `x` — **For `chol()`:** One of the sparse matrix classes, "`matrix.csr`" or "`matrix.csc`";
  For `{back,forward,}solve()`: vector or regular matrix of right-hand-side(s) of a system of linear equations.
- `b` — vector or matrix right-hand-side(s) to solve for.
- `k` — inherited from the generic; not used here.
- `pivot` — inherited from the generic; not used here.
- `nsubmax, nnzlmax, tmpmax` — positive integer numbers with smart defaults; do *not* set unless you know what you are doing!
- `eps` — positive tolerance for checking symmetry; change with caution.
- `tiny` — positive tolerance for checking diagonal entries to be "essentially zero" and hence to be replaced by `Large`, during Cholesky decomposition. Chaning this value may help in close to singular cases, see 'Examples'.
- `Large` — large positive number, "essentially infinite", to replace tiny diagonal entries during Cholesky.
- `warnOnly` — `logical`; when set to true, a result is returned with a `warning` instead of an error (via `stop()`); notably in close to singular cases.
- `cacheKb` — a positive integer, specifying an approximate size of the machine's cache memory in kilo (1024) bytes ('Kb'); used to be hard wired to 64.
- `level` — level of loop unrolling while performing numerical factorization; an integer in `c(1, 2, 4, 8)`; used to be hard wired to 8.
- `upper.tri, transpose` — inherited from the generic; not used here.
- `twice` — `logical` flag: If true, `backsolve()` solves twice, see below.
- `drop` — `logical` flag: If true, `backsolve()` returns `drop(.)`, i.e., a vector instead of a column-1 matrix.
- `...` — further arguments passed to or from other methods.

Details

`chol` performs a Cholesky decomposition of a symmetric positive definite sparse matrix `a` of class `matrix.csr` using the block sparse Cholesky algorithm of Ng and Peyton (1993). The structure of the resulting `matrix.csr.chol` object is relatively complicated. If necessary it can be coerced back to a `matrix.csr` object as usual with `as.matrix.csr`. `backsolve` does triangular back-fitting to compute the solutions of a system of linear equations. For systems of linear equations that only vary on the right-hand-side, the result from `chol` can be reused. Contrary to the behavior of `backsolve` in base R, the default behavior of `backsolve(C,b)` when C is a `matrix.csr.chol` object is to produce a solution to the system $Ax = b$ where `C <- chol(A)`, see the example section. When the flag `twice` is `FALSE` then backsolve solves the system $Cx = b$, up to a permutation – see the comments below. The command `solve` combines `chol` and `backsolve`, and will compute the inverse of a matrix if the right-hand-side is missing. The determinant of the Cholesky factor is returned providing a means to efficiently compute the determinant of sparse positive definite symmetric matrices.

There are several integer storage parameters that are set by default in the call to the Cholesky factorization, these can be overridden in any of the above functions and will be passed by the usual "dots" mechanism. The necessity to do this is usually apparent from error messages like: Error in local(X...) increase tmpmax. For example, one can use, `solve(A,b, tmpmax = 100*nrow(A))`. The current default for tmpmax is `50*nrow(A)`. Some experimentation may be needed to select appropriate values, since they are highly problem dependent. See the code of chol() for further details on the current defaults.

Note

There is no explicit checking for positive definiteness of the matrix so users are advised to ensure that this condition is satisfied. Messages such as "insufficient space" may indicate that one is trying to factor a singular matrix. Because the sparse Cholesky algorithm re-orders the positive definite sparse matrix `A`, the value of `x <- backsolve(C, b)` does *not* equal the solution to the triangular system $Cx = b$, but is instead the solution to the system $CPx = Pb$ for some permutation matrix P (and analogously for `x <- forwardsolve(C, b)`). However, a little algebra easily shows that `backsolve(C, forwardsolve(C, b), twice = FALSE)` *is* the solution to the equation $Ax = b$. Finally, if `C <- chol(A)` for some sparse covariance matrix `A`, and z is a conformable standard normal vector, then the product `y <- as.matrix.csr(C) %*% z` is normal with covariance matrix `A` irrespective of the permutation of the Cholesky factor.

References

Koenker, R and Ng, P. (2002) SparseM: A Sparse Matrix Package for R. http://www.econ.uiuc.edu/~roger/research/home.html

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

See Also

`slm()` for a sparse version of **stats** package's `lm()`.

Examples

```
data(lsq)
class(lsq) # -> [1] "matrix.csc.hb"
model.matrix(lsq)->design.o
class(design.o) # -> "matrix.csr"
dim(design.o) # -> [1] 1850  712
y <- model.response(lsq) # extract the rhs
length(y) # [1] 1850

X <- as.matrix(design.o)
c(object.size(X) / object.size(design.o)) ## X is 92.7 times larger
```

```
t(design.o) %*% design.o -> XpX
t(design.o) %*% y -> Xpy
chol(XpX) -> chol.o

determinant(chol.o)

b1 <- backsolve(chol.o,Xpy) # least squares solutions in two steps
b2 <- solve(XpX,Xpy)        # least squares estimates in one step
b3 <- backsolve(chol.o, forwardsolve(chol.o, Xpy),
                twice = FALSE) # in three steps
## checking that these three are indeed equal :
stopifnot(all.equal(b1, b2), all.equal(b2, b3))
```

---

summary.mslm-class *Class "summary.mslm"*

Description

Sparse version of `summary.lm`

Objects from the Class

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

Methods

print `signature(x = "summary.mslm")`: ...

---

summary.slm-class *Class "summary.slm"*

Description

Sparse version of `summary.lm`

Objects from the Class

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

Methods

print `signature(x = "summary.slm")`: ...

triogramX *A Design Matrix for a Triogram Problem*

Description

This is a design matrix arising from a bivariate smoothing problem using penalized triogram fitting. It is used in the SparseM vignette to illustrate the use of the sparse matrix image function.

Usage

```
data(triogramX)
```

Format

A 375 by 100 matrix stored in compressed sparse row format

References

Koenker, R and Ng, P. (2002). SparseM: A Sparse Matrix Package for R,
http://www.econ.uiuc.edu/~roger/research/home.html

Index