SparseM.hb *Harwell-Boeing Format Sparse Matrices*

Description

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

Usage

```
read.matrix.hb(filename)
write.matrix.hb(filename = "hb.out", X, title, key, mxtype, rhs = NULL,
guess = FALSE,  xsol = FALSE, ptrfmt = "(16I5)", indfmt = "(16I5)",
valfmt = "(5D16.9)", rhsfmt = "(5D16.9)")
model.matrix(object, ...)
model.response(data)
```

Arguments

`filename`	file name to read from or write to
`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"'.
`X`	coefficient matrix stored in csc (for unsymmetric and rectangular matrix) or ssc (for symmetric matrix) format
`title`	72-character title for the matrix
`key`	8-character identifier for the matrix
`mxtype`	3-character identifier for type of the coefficient matrix; First character: currently only "R" for real matrix is supported; second character: "S" for symmetric, "U" for unsymmetric" and "R" for rectangular matrix; third character: currently only "A" for assembled matrix is supported
`rhs`	vector or matrix of right-hand-side(s) including starting guesses and solution vectors if present stored in full storage mode
`guess`	logical flag for the presence of initial guess of the solutions; if TRUE, the values of initial guess are appended to the end of `rhs`
`xsol`	logical flag for the presence of exact solutions; if TRUE, the values of the exact solutions are appended to the end of `rhs`
`ptrfmt`	printing format for the column pointers
`indfmt`	printing format for the row indices
`valfmt`	printing format for the values
`rhsfmt`	printing format for the right-hand-sides
`...`	additional arguments

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).

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.

The function `model.matrix` returns the X matrix of class `matrix.csr`. The function `model.response` returns the y 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 http://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

```
read.matrix.hb(system.file("HBdata","lsq.rra",package = "SparseM"))-> hb.o
class(hb.o) # -> [1] "matrix.csc.hb"
model.matrix(hb.o)->X
class(X) # -> "matrix.csr"
dim(X) # -> [1] 1850  712
y <- model.response(hb.o) # extract the rhs
length(y) # [1] 1850
```

`SparseM.image` *Image Plot for Sparse Matrices*

Description

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

Usage

```
image(x, col=c("white","gray"),xlab="column",ylab="row", ...)
```

Arguments

`x`	a matrix of class `matrix.csr` or `matrix.csc`.
`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

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*

Description

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

Usage

```
as.matrix.csr(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
as.matrix.csc(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
as.matrix.ssr(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
as.matrix.ssc(x, nrow = 1, ncol = 1, eps = .Machine$double.eps)
is.matrix.csr(x, ...)
is.matrix.csc(x, ...)
is.matrix.ssr(x, ...)
is.matrix.ssc(x, ...)
```

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
...	other arguments

Details

The function `matrix.csc` acts like `matrix` to coerce a vector object to a sparse matrix object of class `matrix.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 ith 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.

References

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

See Also

`SparseM.hb` for handling Harwell-Boeing sparse matrices.

Examples

```
n1 <- 10
p <- 5
a <- rnorm(n1*p)
a[abs(a)<0.5] <- 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)
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(rep(0,9),3,3) #sparse matrix of all zeros
```

SparseM.ops *Basic Linear Algebra for Sparse Matrices*

Description

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

Usage

```
t(x); diag(x, nrow); diag(x) <- value; ncol(x); nrow(x); dim(x);rbind(...);
cbind(...);x[i,j]; x %*% y; x %% y; x %/% y; x + y; x - y; x * y;
x / y; x ^ y; x > y; x >= y; x < y; x <= y; x == y; x != y; x & y; x | y
```

Arguments

`x`	matrix of class `matrix.csr`.
`y`	matrix of class `matrix.csr` or a dense vector.
`value`	replacement values.
`i,j`	vectors of elements to extract or replace.
`nrow`	optional number of rows for the result.

Details

Linear algebra operations for matrices of class `matrix.csr` are designed to behave exactly as for regular matrices. In particular, matrix multiplication, addition, subtraction and various logical operations work as with the conventional dense form of matrix storage, as does indexing, rbind, cbind, and diagonal assignment and extraction.

References

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

See Also

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

Examples

```
n1 <- 10
n2 <- 10
p <- 6
y <- rnorm(n1)
a <- rnorm(n1*p)
a[abs(a)<0.5] <- 0
A <- matrix(a,n1,p)
A.csr <- as.matrix.csr(A)
b <- rnorm(n2*p)
b[abs(b)<1.0] <- 0
B <- matrix(b,n2,p)
B.csr <- as.matrix.csr(B)

# matrix transposition and multiplication
A.csr%*%t(B.csr)
```

`SparseM.solve` *Linear Equation Solving 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.
`solve` combines `chol` and `backsolve` and will compute the inverse of a matrix if the right-hand-side is missing.

Usage

```
chol(x, pivot = FALSE, nsubmax, nnzlmax, tmpmax, ...)
backsolve(r, x, k, upper.tri, transpose)
solve(a, b, ...)
```

Arguments

`a`	symmetric positive definite matrix of class `matrix.csr`.
`r`	object of class `matrix.csr.chol` returned by the function `chol`.
`x,b`	vector(regular matrix) of right-hand-side(s) of a system of linear equations.
`k`	inherited from the generic; not used here.
`pivot`	inherited from the generic; not used here.
`nsubmax,nnzlmax,tmpmax`	dimensions of work arrays, in normal operation these are determined inside the algorithm.

`upper.tri`	inherited from the generic; not used here.
`transpose`	inherited from the generic; not used here.
`...`	further arguments passed to or from other methods.

Details

`chol` performs a Cholesky decomposition of a symmetric positive definite sparse matrix `x` of class `matrix.csr` using the block sparse Cholesky algorithm of Ng and Peyton (1993). `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. `solve` combines `chol` and `backsolve`, and will compute the inverse of a matrix if the right-hand-side is missing.

References

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

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

`slm` for sparse version of `lm`

Examples

```
# lsq.rra is real rectangular stored in csc (compressed sparse column) format
read.matrix.hb(system.file("HBdata","lsq.rra",package = "SparseM"))-> hb.o
class(hb.o) # -> [1] "matrix.csc.hb"
model.matrix(hb.o)->design.o
class(design.o) # -> "matrix.csr"
dim(design.o) # -> [1] 1850  712
y <- model.response(hb.o) # extract the rhs
length(y) # [1] 1850
t(design.o)%*%design.o -> XpX
t(design.o)%*%y -> Xpy
chol(XpX)->chol.o
backsolve(chol.o,Xpy)-> b1 # least squares solutions in two steps
solve(XpX,Xpy) -> b2 # least squares estimates in one step
```

`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`

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

See Also

`read.matrix.hb`, `write.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.csc-class` *Class "matrix.csc"*

Description

A new class for sparse matrices stored in compressed sparse column format

Objects from the Class

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

Slots

- `ra`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` an integer array of nnz elements containing the column indices of the elements stored in 'ra'.
- `ia`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` dimension of the matrix

Extends

Class `"matrix.csr"`, directly.

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`

`matrix.csc.hb-class` *Class "matrix.csc.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.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 `"numeric"` ja component of the csc or s sc 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 `"numeric"` dimenson of the coefficient matrix, X.

`rhs.dim`: Object of class `"numeric"` 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")`: ...

See Also

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

---

`matrix.csr-class` *Class "matrix.csr"*

Description

A new class for sparse matrices stored in compressed sparse row format

Objects from the Class

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

Slots

`ra`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` an integer array of nnz elements containing the column indices of the elements stored in 'ra'.

`ia`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` dimension of the matrix

Methods

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

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

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

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

as.matrix.ssr `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")`: ...

See Also

`matrix.csc-class`

`matrix.csr.chol-class` *Class "matrix.csr.chol"*

Description

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

Objects from the Class

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

Slots

`nrow`: Object of class `"numeric"` number of rows in the linear system of equations

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

`nsuper`: Object of class `"numeric"` number of supernodes

`lindx`: Object of class `"numeric"` 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 `"numeric"` vector of integer of pointers for lindx

`nnzl`: Object of class `"numeric"` number of non-zero entries, including the diagonal entries, of the Cholesky factor stored in lnz

`lnz`: Object of class `"numeric"` contains the entries of the Cholesky factor

`xlnz`: Object of class `"numeric"` column pointer for the Cholesky factor stored in lnz

`invp`: Object of class `"numeric"` vector of integer of inverse permutation vector

`perm`: Object of class `"numeric"` vector of integer of permutation vector

`xsuper`: Object of class `"numeric"` array containing the supernode partioning

`ierr`: Object of class `"numeric"` error flag

`time`: Object of class `"numeric"` execution time

Methods

backsolve `signature(r = "matrix.csr.chol")`: ...

See Also

`chol`, `backsolve`

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

Description

A new class for sparse matrices stored in symmetric sparse column format

Objects from the Class

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

Slots

`ra`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` an integer array of nnz elements containing the column indices of the elements stored in 'ra'.

`ia`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` dimension of the matrix

Extends

Class `"matrix.csr"`, directly.

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 `"numeric"` ja component of the csc or s sc 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 `"numeric"` dimenson of the coefficient matrix, X.

`rhs.dim`: Object of class `"numeric"` 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`, `write.matrix.hb`, `matrix.csc.hb-class`

`matrix.ssr-class`	*Class "matrix.ssr"*

Description

A new class for sparse matrices stored in symmetric sparse row format

Objects from the Class

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

Slots

`ra`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` an integer array of nnz elements containing the column indices of the elements stored in 'ra'.

`ia`: Object of class `"numeric", from class "matrix.csr"` 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 `"numeric", from class "matrix.csr"` dimension of the matrix

Extends

Class `"matrix.csr"`, directly.

Methods

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

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

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

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

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

See Also

`matrix.csr-class`

`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-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"` Cholesky object from fitting
`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`

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 fails to do so fully satisfactorily. Ideally, this would eventually become a method for `lm`.

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.
`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

See Also

`slm.methods` for methods `summary`, `print`, `fitted`, `residuals` and `coef` associated with class `slm`, and `slm.fit` for lower level fitting functions

Examples

```
# lsq.rra is real rectangular matrix stored in compressed sparse column format
read.matrix.hb(system.file("HBdata","lsq.rra",package = "SparseM"))-> hb.o
X <- model.matrix(hb.o) #extract the design matrix
y <- model.response(hb.o) # extract the rhs
X1 <- as.matrix(X)
slm.time <- unix.time(slm(y~X1-1) -> slm.o) # pretty fast
lm.time <- unix.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.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 $$\sum w_i * e_i^2$$
`...`	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
`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

See Also

`slm`

`slm.methods` *Methods for slm objects*

Description

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

Usage

```
summary(object, correlation = FALSE, ...)
print(x, digits = max(3, getOption("digits") - 3),
        symbolic.cor = p > 4, signif.stars = getOption("show.signif.stars"),
        ...)
fitted(object, ...)
residuals(object, ...)
coef(object, ...)
```

Arguments

`object,x`	object of class `slm`.
`digits`	minimum number of significant digits to be used for most numbers.
`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.

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`

See Also

`slm`

Examples

```
# lsq.rra is real rectangular matrix stored in compressed sparse column format
read.matrix.hb(system.file("HBdata","lsq.rra",package = "SparseM"))-> hb.o
X <- model.matrix(hb.o) #extract the design matrix
y <- model.response(hb.o) # extract the rhs
X1 <- as.matrix(X)
slm.time <- unix.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]
```

`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