

# Package ‘bdsmatrix’

February 14, 2012

**Title** Routines for Block Diagonal Symmetric matrices

**Maintainer** Terry Therneau <therneau@mayo.edu>

**Version** 1.3

**Date** 2011-12-18

**Depends** methods, R (>= 2.0.0)

**LazyData** Yes

**LazyLoad** Yes

**Author** Terry Therneau

**Description** This is a special case of sparse matrices, used by coxme

**License** GPL-2

**URL** <http://r-forge.r-project.org>

**Collate** bdsmatrix.R gchol.R gchol.bdsmatrix.R as.matrix.bdsmatrix.R  
bdsBlock.R bdsI.R bdsmatrix.ibd.R bdsmatrix.reconcile.R  
diag.bdsmatrix.R listbdsmatrix.R multiply.bdsmatrix.R  
solve.bdsmatrix.R solve.gchol.R solve.gchol.bdsmatrix.R backsolve.R

**Repository** CRAN

**Date/Publication** 2011-12-19 17:44:42

## R topics documented:

as.matrix.bdsmatrix . . . . .	2
backsolve . . . . .	3
bdsBlock . . . . .	4
bdsI . . . . .	5
bdsmatrix . . . . .	6
bdsmatrix-class . . . . .	7
bdsmatrix.ibd . . . . .	9

bdsmatrix.reconcile . . . . .	10
gchol . . . . .	11
gchol-class . . . . .	12
gchol.bdsmatrix-class . . . . .	13
listbdsmatrix . . . . .	14
solve.bdsmatrix . . . . .	15
solve.gchol . . . . .	16

<b>Index</b>	<b>18</b>
--------------	-----------

---

as.matrix.bdsmatrix	<i>Convert a bdsmatrix to a ordinary (dense) matrix</i>
---------------------	---

---

## Description

Method to convert from a Block Diagonal Sparse (bdsmatrix) matrix representation to an ordinary one

## Usage

```
## S3 method for class 'bdsmatrix'
as.matrix(x, ...)
```

## Arguments

x	a bdsmatrix object
...	other arguments are ignored (necessary to match the as.matrix template)

## Details

Note that the conversion of a large bdsmatrix can easily exceed memory.

## Value

a matrix

## See Also

bdsmatrix

---

backsolve                      *Solve an Upper or Lower Triangular System*

---

### Description

Solves a system of linear equations where the coefficient matrix is upper (or ‘right’, ‘R’) or lower (‘left’, ‘L’) triangular.

`x <- backsolve(R, b)` solves  $Rx = b$ .

### Usage

```
backsolve(r, ...)
## S4 method for signature 'gchol'
backsolve(r, x, k=ncol(r), upper.tri=TRUE, ...)
## S4 method for signature 'gchol.bdsmatrix'
backsolve(r, x, k=ncol(r), upper.tri=TRUE, ...)
```

### Arguments

<code>r</code>	a matrix or matrix-like object
<code>x</code>	a vector or a matrix whose columns give the right-hand sides for the equations.
<code>k</code>	The number of columns of <code>r</code> and rows of <code>x</code> to use.
<code>upper.tri</code>	logical; if TRUE (default), the <i>upper triangular</i> part of <code>r</code> is used. Otherwise, the lower one.
<code>...</code>	further arguments passed to other methods

### Details

The generalized Cholesky decomposition of a symmetric matrix  $A$  is  $A = LDL'$  where  $D$  is diagonal,  $L$  is lower triangular, and  $L'$  is the transpose of  $L$ . These functions solve either  $L\sqrt{D}x = b$  (when `upper.tri=FALSE`) or  $\sqrt{D}L'x = b$ .

### Value

The solution of the triangular system. The result will be a vector if `x` is a vector and a matrix if `x` is a matrix.

Note that `forwardsolve(L, b)` is just a wrapper for `backsolve(L, b, upper.tri=FALSE)`.

### Methods

Use `showMethods(backsolve)` to see all the defined methods; the two created by the `bdsmatrix` library are described here:

**bdsmatrix** signature=(`r= "gchol"`) for a generalized cholesky decomposition

**bdsmatrix** signature=(`r= "gchol.bdsmatrix"`) for the generalize cholesky decomposition of a `bdsmatrix` object

**Note**

The `bdsmatrix` package promotes the base R `backsolve` function to a generic. To see the full documentation for the default method, view `backsolve` from the base package.

**See Also**

[forwardsolve](#), [gchol](#)

---

bdsBlock

*Block diagonal matrices.*

---

**Description**

Create a block-diagonal matrix of ones.

**Usage**

```
bdsBlock(id, group)
```

**Arguments**

<code>id</code>	the identifier list. This will become the dimnames of the final matrix, and must be a set of unique values. It's length determines the dimension of the final matrix
<code>group</code>	a vector giving the grouping structure. All rows/cols belonging to a given group will form a block of 1's in the final matrix.

**Value**

a block-diagonal matrix of class `bdsmatrix`

**See Also**

`bdsmatrix`, `bdsI`

**Examples**

```
id <- letters[1:10]
group <- c(1,1,3,2,3,3,3,2,3,2,4)
bdsBlock(id, group)
## Not run:
  a b d g i c e f h j
a 1 1 0 0 0 0 0 0 0 0
b 1 1 0 0 0 0 0 0 0 0
d 0 0 1 1 1 0 0 0 0 0
g 0 0 1 1 1 0 0 0 0 0
i 0 0 1 1 1 0 0 0 0 0
c 0 0 0 0 0 1 1 1 1 0
e 0 0 0 0 0 1 1 1 1 0
```

```

f 0 0 0 0 0 1 1 1 1 0
h 0 0 0 0 0 1 1 1 1 0
j 0 0 0 0 0 0 0 0 0 1

# Create the matrices for a sparse nested fit of family within city
group <- paste(mydata$city, mydata$family, sep='/')
mat1 <- bdsI(group)
mat2 <- bdsBlock(group, mydata$city)

fit <- coxme(Surv(time, status) ~ age + sex + (1|group), data=mydata,
             varlist=list(mat1, mat2))

## End(Not run)

```

---

bdsI	<i>Sparse identity matrices</i>
------	---------------------------------

---

### Description

This function will create an identity matrix, in the sparse `bdsmatrix` format.

### Usage

```
bdsI(id, blocksize)
```

### Arguments

id	the identifier list. This will become the dimnames of the final matrix, and must be a set of unique values. It's length determines the dimension of the final matrix
blocksize	the blocksize vector of the final matrix. If supplied, the sum of blocksizes must equal the dimension of the matrix. By default, the created matrix is as sparse as possible.

### Value

an identity matrix.

### Examples

```
imat <- bdsI(1:10)
```

bdsmatrix

*Create a sparse symmetric block diagonal matrix object***Description**

Sparse block diagonal matrices are used in the the large parameter matrices that can arise in random-effects coxph and survReg models. This routine creates such a matrix. Methods for these matrices allow them to be manipulated much like an ordinary matrix, but the total memory use can be much smaller.

**Usage**

```
bdsmatrix(blocksize, blocks, rmat, dimnames)
```

**Arguments**

blocksize	vector of sizes for the matrices on the diagonal
blocks	contents of the diagonal blocks, strung out as a vector
rmat	the dense portion of the matrix, forming a right and lower border
dimnames	a list of dimension names for the matrix

**Details**

Consider the following matrix, which has been divided into 4 parts.

```
1 2 0 0 0 | 4 5 2 1 0 0 0 | 6 7 0 0 3 1 2 | 8 8 0 0 1 4 3 | 1 1 0 0 2 3 5 | 2 2 —————+—— 4 6 8 1 2 |
7 6 5 7 8 1 2 | 6 9
```

The upper left is block diagonal, and can be stored in a compressed form without the zeros. With a large number of blocks, the zeros can actually account for over 99% of a matrix; this commonly happens with the kinship matrix for a large collection of families (one block/family). The arguments to this routine would be block sizes of 2 and 3, along with a 2 by 7 "right hand" matrix. Since the matrix is symmetrical, the bottom slice is not needed.

**Value**

an object of type bdsmatrix

**Examples**

```
# The matrix shown above is created by
tmat <- bdsmatrix(c(2,3), c(1,2,1, 3,1,2, 4,3, 5),
                 rmat=matrix(c(4,6,8,1,2,7,6, 5,7,8,1,2,6,9), ncol=2))

# Note that only the lower part of the blocks is needed, however, the
# entire block set is also allowed, i.e., c(1,2,2,1, 3,1,2,1,4,3,2,3,5)
```

---

bdsmatrix-class	Class "bdsmatrix"
-----------------	-------------------

---

## Description

Representation for a Block Diagonal Sparse matrix

## Objects from the Class

Objects of this class are usually created using the `bdsmatrix`, `bdsI` or `bdsBlock` functions. The result is a symmetric matrix whose upper left portion is block-diagonal, with an optional border on the right and bottom that is dense. The matrices were originally created to represent familial correlation structures, which have a block for each family but no connection between families.

## Slots

**blocksize:** An integer vector containing the sizes of the diagonal blocks

**blocks:** A numeric vector containing the contents of the block portion. Only the lower triangle of each block is stored.

**rmat:** An optional numeric matrix containing the dense portion

**offdiag:** A single numeric element, default zero, which is the value for elements off the block-diagonal

**Dim:** The dimension of the matrix, an integer vector of length 2

**Dimnames:** The dimnames of the matrix, a list with 2 elements

## Methods

**%\*%** signature(x = "matrix", y = "bdsmatrix"): the result will be an ordinary matrix

**%\*%** signature(x = "numeric", y = "bdsmatrix"): the result will be a vector

**%\*%** signature(x = "bdsmatrix", y = "matrix"): the result will be an ordinary matrix

**%\*%** signature(x = "bdsmatrix", y = "numeric"): the result will be a vector

**Math2** signature(x = "bdsmatrix"):

**Math** signature(x = "bdsmatrix"):

**Ops** signature(e1 = "bdsmatrix", e2 = "numeric"):

**Ops** signature(e1 = "bdsmatrix", e2 = "bdsmatrix"):

**Ops** signature(e1 = "bdsmatrix", e2 = "matrix"):

**Ops** signature(e1 = "numeric", e2 = "bdsmatrix"):

**Ops** signature(e1 = "matrix", e2 = "bdsmatrix"):

[ signature(x = "bdsmatrix"): if the subscripts are a set of increasing integers, and the row and column subscripts are identical, then the result is also a `bdsmatrix`. This is useful for example to create the kinship matrix for all females from an overall kinship matrix. If the subscripts do not match, then an ordinary matrix is created

**all** signature(x = "bdsmatrix"): ...  
**any** signature(x = "bdsmatrix"): ...  
**coerce** signature(from = "bdsmatrix", to = "matrix"): ...  
**coerce** signature(from = "bdsmatrix", to = "vector"): ...  
**diag** signature(x = "bdsmatrix"): retrieve the diagonal of the matrix  
**diag<-** signature(x = "bdsmatrix"): set the diagonal of the matrix to a given value  
**dim** signature(x = "bdsmatrix"): dimension of the matrix  
**dimnames** signature(x = "bdsmatrix"): dimnames of the matrix  
**dimnames<-** signature(x = "bdsmatrix"): set the dimnames of the matrix  
**gchol** signature(x = "bdsmatrix"): generalized cholesky decomposition of the matrix  
**max** signature(x = "bdsmatrix"): maximum of the matrix  
**min** signature(x = "bdsmatrix"): minimum of the matrix  
**prod** signature(x = "bdsmatrix"): ...  
**range** signature(x = "bdsmatrix"): ...  
**show** signature(object = "bdsmatrix"): print out the matrix  
**sum** signature(x = "bdsmatrix"): ...

### Note

Many of the actions above will result in conversion to an ordinary matrix object, including print, addition to an ordinary matrix, etc. This can easily create objects that are too large for system memory. By default the value of options('bdsmatrixsize') is consulted first, and if the resulting object would be have a length greater than this option the conversion an error is generated and conversion is not attempted. The default value for the option is 1000.

### Author(s)

Terry Therneau

### See Also

[gchol](#)

### Examples

```
showClass("bdsmatrix")
```

---

bdsmatrix.ibd	<i>Create a bdsmatrix from a list</i>
---------------	---------------------------------------

---

### Description

Routines that create identity-by-descent (ibd) coefficients often output their results as a list of values (i, j, x[i,j]), with unlisted values of the x matrix assumed to be zero. This routine recasts such a list into `bdsmatrix` form.

### Usage

```
bdsmatrix.ibd(id1, id2, x, idmap, diagonal)
```

### Arguments

<code>id1</code>	row identifier for the value, in the final matrix. Optionally, <code>id1</code> can be a 3 column matrix or <code>data.frame</code> , in which case it is assumed to contain the first 3 arguments, in order.
<code>id2</code>	column identifier for the value, in the final matrix.
<code>x</code>	the value to place in the matrix
<code>idmap</code>	a two column matrix or data frame. Sometimes routines create output with integer values for <code>id1</code> and <code>id2</code> , and then this argument is the mapping from this internal label to the “real” name)
<code>diagonal</code>	If diagonal elements are not preserved in the list, this value will be used for the diagonal of the result. If the argument appears, then the output matrix will contain an entry for each value in <code>id1</code> list. Otherwise only those with an explicit entry appear.

### Details

The routine first checks for non-symmetric or otherwise inconsistent input. It then groups observations together into ‘families’ of related subjects, which determines the structure of the final matrix. As with the `makekinship` function, singletons with no relationships are first in the output matrix, and then families appear one by one.

### Value

a `bdsmatrix` object representing a block-diagonal sparse matrix.

### See Also

`bdsmatrix`, `kinship`, `coxme`, `lmekin`

## Examples

```
## Not run:  
ibdmat <- bdsmatrix.ibd(i,j, ibdval, idlist=subject)  
  
## End(Not run)
```

---

bdsmatrix.reconcile    *Ensure alignment of two bdsmatrix objects*

---

## Description

This function is used by `coxme`. When a random effect is expressed as a sum of variance terms (matrices), it is important that all of them have the same row/column order and the same block structure. This does so, while retaining as much sparsity in the result as possible.

## Usage

```
bdsmatrix.reconcile(varlist, group)
```

## Arguments

`varlist`            a list, each element of which is a matrix or bdsmatrix object  
`group`              a vector of dimnames, the target match for matrix's dimnames

## Value

a varlist, whose individual elements may have had row/column rearrangement.

## Author(s)

Terry Therneau

## See Also

[bdsmatrix](#)

---

gchol	<i>Generalized Cholesky decomposition</i>
-------	---

---

### Description

Perform the generalized Cholesky decomposition of a real symmetric matrix.

### Usage

```
gchol(x, tolerance=1e-10)
```

### Arguments

x	the symmetric matrix to be factored
tolerance	the numeric tolerance for detection of singular columns in x.

### Details

A symmetric matrix  $A$  can be decomposed as  $LDL'$ , where  $L$  is a lower triangular matrix with 1's on the diagonal,  $L'$  is the transpose of  $L$ , and  $D$  is diagonal. The inverse of  $L$  is also lower-triangular, with 1's on the diagonal. If all elements of  $D$  are positive, then  $A$  must be symmetric positive definite (SPD), and the solution can be reduced to the usual Cholesky decomposition  $U'U$  where  $U$  is upper triangular and  $U = \text{sqrt}(D) L'$ .

The main advantage of the generalized form is that it admits of matrices that are not of full rank:  $D$  will contain zeros marking the redundant columns, and the rank of  $A$  is the number of non-zero columns. If all elements of  $D$  are zero or positive, then  $A$  is a non-negative definite (NND) matrix. The generalized form also has the (quite minor) numerical advantage of not requiring square roots during its calculation. To extract the components of the decomposition, use the `diag` and `as.matrix` functions.

The `solve` has a method for `gchol` decompositions, and there are `gchol` methods for block diagonal symmetric (`bdsmatrix`) matrices as well.

### Value

an object of class `gchol` containing the generalized Cholesky decomposition. It has the appearance of a lower triangular matrix.

### See Also

`bdsmatrix`, `solve.gchol`

## Examples

```
# Create a matrix that is symmetric, but not positive definite
# The matrix temp has column 6 redundant with cols 1-5
smat <- matrix(1:64, ncol=8)
smat <- smat + t(smat) + diag(rep(20,8)) #smat is 8 by 8 symmetric
temp <- smat[c(1:5, 5:8), c(1:5, 5:8)]
ch1 <- gchol(temp)

print(as.matrix(ch1), digits=4) # print out L
print(diag(ch1)) # Note the zero at position 6

ginv <- solve(ch1) # generalized inverse
diag(ginv) # also has column 6 marked as singular
```

---

gchol-class

*Class "gchol"*


---

## Description

The result of a generalized Cholesky decomposition  $A=LDL'$  where  $A$  is a symmetric matrix,  $L$  is lower triangular with 1s on the diagonal, and  $D$  is a diagonal matrix.

## Objects from the Class

These objects are created by the `gchol` function.

## Slots

**.Data:** A numeric vector containing the results of the decomposition

**Dim:** An integer vector of length 2, the dimension of the matrix

**Dimnames:** A list of length 2 containing the dimnames. These default to the dimnames of the matrix  $A$

**rank:** The rank of the matrix

## Methods

**%\*%** signature( $x = "gchol"$ ,  $y = "matrix"$ ): multiply the cholesky decomposition by a matrix. That is, if  $A=LDL'$  is the decomposition, then `gchol(A) %*% B` will return  $L D^{.5} B$ .

**%\*%** signature( $x = "matrix"$ ,  $y = "gchol"$ ): multiply by a matrix on the left

**[** signature( $x = "gchol"$ ): if a square portion from the upper left corner is selected, then the result will be a `gchol` object, otherwise an ordinary matrix is returned. The latter most often occurs when printing part of the matrix at the command line.

**coerce** signature( $from = "gchol"$ ,  $to = "matrix"$ ): Use of the `as.matrix` function will return  $L$

**diag** signature( $x = "gchol"$ ): Use of the `diag` function will return  $D$

**dim** signature(x = "gchol"): returns the dimension of the matrix  
**dimnames** signature(x = "gchol"): returns the dimnames  
**show** signature(object = "gchol"): By default a triangular matrix is printed showing D on the diagonal and L off the diagonal  
**gchol** signature(x= "matrix"): create a generalized Cholesky decomposition of the matrix

**Note**

The primary advantages of the gearalized decomposition, as compared to the standard chol function, has to do with redundant columns and generalized inverses (g-inverse). The lower triangular matrix L is always of full rank. The diagonal matrix D has a 0 element at position j if and only if the jth column of A is linearly dependent on columns 1 to j-1 preceding it. The g-inverse of A involves the inverse of L and a g-inverse of D. The g-inverse of D retains the zeros and inverts non-zero elements of D. This is very useful inside modeling functions such as coxph, since the X matrix can often contain a redundant column.

**Author(s)**

Terry Therneau

**See Also**

[gchol](#)

**Examples**

```
showClass("gchol")
```

---

```
gchol.bdsmatrix-class  Class "gchol.bdsmatrix"
```

---

**Description**

Generalized cholesky decomposition of a bdsmatrix object,  $A = LDL'$  where A is symmetric, L is lower triangular with 1 on the diagonal, and D is diagonal.

**Objects from the Class**

These are created by the gchol function.

**Slots**

**blocksize:** Integer vector of block sizes  
**blocks:** Numeric vector containing the blocks  
**rmat:** Dense portion of the decomposition  
**rank:** The rank of A  
**Dim:** Integer vector of length 2 containing the dimension  
**Dimnames:** List of length 2 containing the dimnames

**Methods**

```

%% signature(x = "gchol.bdsmatrix", y = "matrix"): ...
%% signature(x = "gchol.bdsmatrix", y = "numeric"): ...
%% signature(x = "matrix", y = "gchol.bdsmatrix"): ...
%% signature(x = "numeric", y = "gchol.bdsmatrix"): ...
[ signature(x = "gchol.bdsmatrix"): ...
coerce signature(from = "gchol.bdsmatrix", to = "matrix"): ...
diag signature(x = "gchol.bdsmatrix"): ...
dim signature(x = "gchol.bdsmatrix"): ...
show signature(object = "gchol.bdsmatrix"): ...

```

**Note**

The Cholesky decomposition of a block diagonal symmetric matrix is also block diagonal symmetric, so is stored in the same manner as a `bdsmatrix` object

**Author(s)**

Terry Therneau

**See Also**

[bdsmatrix](#), [gchol](#)

**Examples**

```
showClass("gchol.bdsmatrix")
```

---

listbdsmatrix	<i>List out a bdsmatrix as row/col/value triplets</i>
---------------	---

---

**Description**

This routine is the inverse of the `bdsmatrix.ibd` function found in the `kinship` library.

**Usage**

```
listbdsmatrix(x, id = TRUE, diag = FALSE)
```

**Arguments**

<code>x</code>	a <code>bdsmatrix</code> object
<code>id</code>	if true, the <code>dimnames</code> of the object are used as the row and column identifiers in the output, if false integer row and column numbers are used
<code>diag</code>	include the diagonal elements in the output

**Details**

The non-zero elements of the matrix are listed out as row-col-value triplets, one per line, in a data frame. Since the matrix is known to be symmetric, only elements with row  $\geq$  col are listed. When familial correlation data is represented in a bdsmatrix, e.g. kinship or identity-by-descent information, the diagonal is a known value and can be omitted from the listing. Genetic software often produces matrices in the list form; this routine is the inverse of the bdsmatrix.ibd routine, found in the kinship library, which converts list form to bdsmatrix form.

**Value**

a data frame with variables row, col, and value.

**Author(s)**

Terry Therneau

**See Also**

[bdsmatrix](#)

---

solve.bdsmatrix

*Solve a matrix equation using the generalized Cholesky decomposition*

---

**Description**

This function solves the equation  $Ax=b$  for  $x$ , when  $A$  is a block diagonal sparse matrix (an object of class bdsmatrix).

**Usage**

```
## S3 method for class 'bdsmatrix'
solve(a, b, full=TRUE, tolerance=1e-10, ...)
```

**Arguments**

a	a block diagonal sparse matrix object
b	a numeric vector or matrix, that forms the right-hand side of the equation.
full	if true, return the full inverse matrix; if false return only that portion corresponding to the blocks. This argument is ignored if b is present. If the bdsmatrix a has a non-sparse portion, i.e., if the rmat component is present, then the inverse of a will not be block-diagonal sparse. In this case setting full=F returns only a portion of the inverse. The elements that are returned are those of the full inverse, but the off-diagonal elements that are not returned would not have been zero.
tolerance	the tolerance for detecting singularity in the a matrix
...	other arguments are ignored

**Details**

The matrix `a` consists of a block diagonal sparse portion with an optional dense border. The inverse of `a`, which is to be computed if `y` is not provided, will have the same block diagonal structure as `a` only if there is no dense border, otherwise the resulting matrix will not be sparse.

However, these matrices may often be very large, and a non sparse version of one of them will require gigabytes of even terabytes of space. For one of the common computations (degrees of freedom in a penalized model) only those elements of the inverse that correspond to the non-zero part of `a` are required; the `full=F` option returns only that portion of the (block diagonal portion of) the inverse matrix.

**Value**

if argument `b` is not present, the inverse of `a` is returned, otherwise the solution to matrix equation. The equation is solved using a generalized Cholesky decomposition.

**See Also**

`bdsmatrix`, `gchol`

**Examples**

```
tmat <- bdsmatrix(c(3,2,2,4),
                 c(22,1,2,21,3,20,19,4,18,17,5,16,15,6,7, 8,14,9,10,13,11,12),
                 matrix(c(1,0,1,1,0,0,1,1,0,1,0,10,0,
                          0,1,1,0,1,1,0,1,1,0,1,0,10), ncol=2))
dim(tmat)
solve(tmat, cbind(1:13, rep(1,13)))
```

---

`solve.gchol`

*Solve a matrix equation using the generalized Cholesky decomposition*

---

**Description**

This function solves the equation  $Ax=b$  for  $x$ , given  $b$  and the generalized Cholesky decomposition of  $A$ . If only the first argument is given, then a G-inverse of  $A$  is returned.

**Usage**

```
## S3 method for class 'gchol'
solve(a, b, full=TRUE, ...)
```

**Arguments**

<code>a</code>	a generalized cholesky decomposition of a matrix, as returned by the <code>gchol</code> function.
<code>b</code>	a numeric vector or matrix, that forms the right-hand side of the equation.
<code>full</code>	solve the problem for the full (original) matrix, or for the cholesky matrix.
<code>...</code>	other arguments are ignored

**Details**

A symmetric matrix  $A$  can be decomposed as  $LDL'$ , where  $L$  is a lower triangular matrix with 1's on the diagonal,  $L'$  is the transpose of  $L$ , and  $D$  is diagonal. This routine solves either the original problem  $Ay=b$  (full argument) or the subproblem  $\text{sqrt}(D)L'y=b$ . If  $b$  is missing it returns the inverse of  $A$  or  $L$ , respectively.

**Value**

if argument  $b$  is not present, the inverse of  $a$  is returned, otherwise the solution to matrix equation.

**See Also**

gchol

**Examples**

```
# Create a matrix that is symmetric, but not positive definite
# The matrix temp has column 6 redundant with cols 1-5
smat <- matrix(1:64, ncol=8)
smat <- smat + t(smat) + diag(rep(20,8)) #smat is 8 by 8 symmetric
temp <- smat[c(1:5, 5:8), c(1:5, 5:8)]
ch1 <- gchol(temp)

ginv <- solve(ch1, full=FALSE) # generalized inverse of ch1
tinv <- solve(ch1, full=TRUE) # generalized inverse of temp
all.equal(temp %*% tinv %*% temp, temp)
```

# Index

\*Topic **algebra**  
  backsolve, 3

\*Topic **array**  
  as.matrix.bdsmatrix, 2  
  backsolve, 3  
  bdsBlock, 4  
  bdsmatrix, 6  
  bdsmatrix.ibd, 9  
  bdsmatrix.reconcile, 10  
  gchol, 11  
  solve.bdsmatrix, 15  
  solve.gchol, 16

\*Topic **classes**  
  bdsmatrix-class, 7  
  gchol-class, 12  
  gchol.bdsmatrix-class, 13

\*Topic **survival**  
  bdsI, 5

[,bdsmatrix-method (bdsmatrix-class), 7  
[,gchol-method (gchol-class), 12  
[,gchol.bdsmatrix-method  
  (gchol.bdsmatrix-class), 13  
%\*%,bdsmatrix,matrix-method  
  (bdsmatrix-class), 7  
%\*%,bdsmatrix,numeric-method  
  (bdsmatrix-class), 7  
%\*%,gchol,matrix-method (gchol-class),  
  12  
%\*%,gchol.bdsmatrix,matrix-method  
  (gchol.bdsmatrix-class), 13  
%\*%,gchol.bdsmatrix,numeric-method  
  (gchol.bdsmatrix-class), 13  
%\*%,matrix,bdsmatrix-method  
  (bdsmatrix-class), 7  
%\*%,matrix,gchol-method (gchol-class),  
  12  
%\*%,matrix,gchol.bdsmatrix-method  
  (gchol.bdsmatrix-class), 13  
%\*%,numeric,bdsmatrix-method  
  (bdsmatrix-class), 7  
%\*%,numeric,gchol.bdsmatrix-method  
  (gchol.bdsmatrix-class), 13

all,bdsmatrix-method (bdsmatrix-class),  
  7  
any,bdsmatrix-method (bdsmatrix-class),  
  7  
as.matrix.bdsmatrix, 2  
backsolve, 3  
backsolve,gchol-method (backsolve), 3  
backsolve,gchol.bdsmatrix-method  
  (backsolve), 3  
backsolve-methods (backsolve), 3  
bdsBlock, 4  
bdsI, 5  
bdsmatrix, 6, 10, 14, 15  
bdsmatrix-class, 7  
bdsmatrix.ibd, 9  
bdsmatrix.reconcile, 10  
coerce,bdsmatrix,matrix-method  
  (bdsmatrix-class), 7  
coerce,bdsmatrix,vector-method  
  (bdsmatrix-class), 7  
coerce,gchol,matrix-method  
  (gchol-class), 12  
coerce,gchol.bdsmatrix,matrix-method  
  (gchol.bdsmatrix-class), 13  
diag,bdsmatrix-method  
  (bdsmatrix-class), 7  
diag,gchol-method (gchol-class), 12  
diag,gchol.bdsmatrix-method  
  (gchol.bdsmatrix-class), 13  
diag<- ,bdsmatrix-method  
  (bdsmatrix-class), 7  
dim,bdsmatrix-method (bdsmatrix-class),  
  7

- dim,gchol-method (gchol-class), 12
- dim,gchol.bdsmatrix-method  
(gchol.bdsmatrix-class), 13
- dimnames,bdsmatrix-method  
(bdsmatrix-class), 7
- dimnames,gchol-method (gchol-class), 12
- dimnames<- ,bdsmatrix-method  
(bdsmatrix-class), 7
  
- forwardsolve, 4
  
- gchol, 4, 8, 11, 13, 14
- gchol,bdsmatrix-method  
(bdsmatrix-class), 7
- gchol,matrix-method (gchol-class), 12
- gchol-class, 12
- gchol.bdsmatrix-class, 13
  
- listbdsmatrix, 14
  
- Math,bdsmatrix-method  
(bdsmatrix-class), 7
- Math2,bdsmatrix-method  
(bdsmatrix-class), 7
- max,bdsmatrix-method (bdsmatrix-class),  
7
- min,bdsmatrix-method (bdsmatrix-class),  
7
  
- Ops,bdsmatrix,bdsmatrix-method  
(bdsmatrix-class), 7
- Ops,bdsmatrix,matrix-method  
(bdsmatrix-class), 7
- Ops,bdsmatrix,numeric-method  
(bdsmatrix-class), 7
- Ops,matrix,bdsmatrix-method  
(bdsmatrix-class), 7
- Ops,numeric,bdsmatrix-method  
(bdsmatrix-class), 7
  
- prod,bdsmatrix-method  
(bdsmatrix-class), 7
  
- range,bdsmatrix-method  
(bdsmatrix-class), 7
  
- show,bdsmatrix-method  
(bdsmatrix-class), 7
- show,gchol-method (gchol-class), 12
- show,gchol.bdsmatrix-method  
(gchol.bdsmatrix-class), 13
- showMethods, 3
- solve.bdsmatrix, 15
- solve.gchol, 16
- sum,bdsmatrix-method (bdsmatrix-class),  
7