

Package ‘kinship’

January 16, 2012

Version 1.1.3

Date 2012-1-16

Title mixed-effects Cox models, sparse matrices, and modeling data from large pedigrees

Author Beth Atkinson (atkinson@mayo.edu) for pedigree functions. Terry Therneau (therneau@mayo.edu) for all other functions.

Maintainer Jing hua Zhao <jinghua.zhao@mrc-epid.cam.ac.uk>

Depends methods, survival, nlme, lattice, R (>= 2.8.0)

LazyLoad Yes

LazyData Yes

Description coxme: general mixed-effects Cox models; kinship: routines to create and manipulate n by n matrices that describe the genetic relationships between n persons; pedigree: create and plot pedigrees; bdsmatrix: a class of objects for sparse block-diagonal matrices (which is how kinship matrices are stored); gchol: generalized cholesky decompositions

License GPL (>= 2)

Repository CRAN

Date/Publication 2012-01-16 17:11:10

R topics documented:

align.pedigree	2
as.matrix.bdsmatrix	3
autohint	4
bdsBlock	5
bdsI	6
bdsmatrix	7
bdsmatrix-class	8

bdsmatrix.ibd	9
besthint	10
coxme	11
coxme.control	13
familycheck	15
gchol	16
gchol-class	18
gchol-methods	18
gchol.bdsmatrix-class	19
kinship	19
kinship.ops	21
list or NULL-class	21
lmekin	21
makefamid	23
makekinship	25
pedigree	26
plot.pedigree	27
solve.bdsmatrix	30
solve.gchol	31
solve.gchol.bdsmatrix	32

Index **34**

align.pedigree	<i>Generate plotting information for a pedigree</i>
----------------	-----------------------------------------------------

Description

Given a pedigree, this function creates helper matrices that describe the layout of a plot of the pedigree.

Usage

```
align.pedigree(ped, packed=T, hints=ped$hints, width=6, align=T)
```

Arguments

ped	a pedigree object
packed	should the pedigree be compressed, i.e., to allow diagonal lines connecting parents to children in order to have a smaller overall width for the plot.
hints	two column hints matrix. The first column determines the relative order of subjects within a sibship, as well as the relative order of processing for the founder couples. (For this latter, the female founders are ordered as though they were sisters). The second column contains spouse information, e.g., if hints[2,6] = 17, then subject number 17 of the pedigree is a spouse of number 2, and is preferentially plotted to the right of number 2. Negative numbers plot the spouse preferentially to the left.

width for a packed output, the minimum width
 align should iterations of the ‘springs’ algorithm be used to improve the plotted output. If True, a default number of iterations is used. If numeric, this specifies the number of iterations.

Details

This is an internal routine, used almost exclusively by `plot.pedigree`. The subservient functions `alignedped1`, `alignedped2`, `alignedped3`, and `alignedped4` contain the bulk of the computation.

Value

a structure with components

`n` a vector giving the number of subjects on each horizontal level of the plot
`nid` a matrix with one row for each level, giving the numeric id of each subject plotted. (An value of 17 means the 17th subject in the pedigree).
`pos` a matrix giving the horizontal position of each plot point
`fam` a matrix giving the family id of each plot point. A value of "3" would mean that the two subjects in positions 3 and 4, in the row above, are this subject’s parents.
`spouse` a matrix with values 1= subject plotted to the immediate right is a spouse, 2= subject plotted to the immediate right is an inbred spouse, 0 = not a spouse
`twins` optional matrix which will only be present if the pedigree contains twins. It has values 1= sibling to the right is a monozygotic twin, 2= sibling to the right is a dizygotic twin, 3= sibling to the right is a twin of unknown zygosity, 0 = not a twin

See Also

[plot.pedigree](#)

as.matrix.bdsmatrix *a function for bdsmatrix*

Description

A function for `bdsmatrix`

Usage

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

Arguments

`x` a `bdsmatrix`
`...` additional arguments to be passed

Examples

```

## Not run:
# The function is currently defined as
function(x)
{
  if(class(x) != "bdsmatrix")
    stop("argument must be a bdsmatrix object")
  dd <- dim(x)
  d3 <- sum(x@blocksize)
  # dim of square portion
  d4 <- sum(x@blocksize^2)
  # size of x@blocks
  newmat <- matrix(0., dd[1], dd[2], dimnames = x@.Dimnames)
  temp <- .C("bdsmatrix_index1",
    as.integer(length(x@blocksize)),
    as.integer(x@blocksize),
    as.integer(c(1, 0, 0)),
    as.integer(d3),
    as.integer(1:d3 - 1),
    indexa = integer(d3 * d3),
    indexb = 0,
    indexc = 0)$indexa
  newmat[x@permute, x@permute] <- c(x@offdiag, x@blocks)[1 + temp]
  if(length(x@rmat) > 0) {
    newmat[, - (1:d3)] <- x@rmat
    newmat[- (1:d3), ] <- t(x@rmat)
  }
  newmat
}

## End(Not run)

```

 autohint

Align a pedigree to print well

Description

A pedigree structure can contain a hints matrix which helps to reorder the pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine is called by the pedigree function to create an initial hints matrix.

Usage

```
autohint(ped)
```

Arguments

ped a pedigree structure

Details

This routine would not normally be called by a user. It moves children within families, so that marriages are on the "edge" of a set children, closest to the spouse. For pedigrees that have only a single connection between two families this simple-minded approach works surprisingly well. For more complex structures either hand-tuning of the hints matrix, or use of the `besthint` routine will usually be required.

Value

a two column hints matrix

See Also

[pedigree](#), [besthint](#)

bdsBlock

Block diagonal matrices.

Description

Create a block-diagonal matrix of ones.

Usage

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

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
group	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

```
## Not run:
id   <- letters[1:10]
group <- c(1,1,3,2,3,3,3,2,3,2,4)
bdsBlock(id, group)
  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, data=mydata,
             random= ~1|group, varlist=list(mat1, mat2))

## End(Not run)
```

bdsI

Sparse identity matrices

Description

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

Usage

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

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>blocksize</code>	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	<i>Class "bdsmatrix"</i>
-----------------	--------------------------

Description

A bdsmatrix class

Objects from the Class

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

Slots

blocksize: Object of class "integer" vector of sizes for the matrices on the diagonal
blocks: Object of class "numeric" contents of the diagonal blocks, strung out as a vector
rmat: Object of class "matrix" the dense portion of the matrix, forming a right and lower border
offdiag: Object of class "numeric" 0s
.Dim: Object of class "integer" dimensions
.Dimnames: Object of class "list" or NULL" a list of dimension names for the matrix

Extends

Class "matrix", directly.

Methods

```
%*% signature(x = "matrix", y = "bdsmatrix"):
%*% signature(x = "numeric", y = "bdsmatrix"):
Math2 signature(x = "bdsmatrix"):
Math signature(x = "bdsmatrix"):
Ops signature(e1 = "bdsmatrix", e2 = "numeric"):
Ops signature(e1 = "bdsmatrix", e2 = "bdsmatrix"):
Ops signature(e1 = "numeric", e2 = "bdsmatrix"):
[ signature(x = "bdsmatrix"):
all signature(x = "bdsmatrix"):
```

```

any signature(x = "bdsmatrix"):
coerce signature(from = "bdsmatrix", to = "matrix"):
coerce signature(from = "bdsmatrix", to = "vector"):
diag signature(x = "bdsmatrix"):
diag<- signature(x = "bdsmatrix"):
dim signature(x = "bdsmatrix"):
dimnames signature(x = "bdsmatrix"):
dimnames<- signature(x = "bdsmatrix"):
max signature(x = "bdsmatrix"):
min signature(x = "bdsmatrix"):
prod signature(x = "bdsmatrix"):
range signature(x = "bdsmatrix"):
show signature(object = "bdsmatrix"):
sum signature(x = "bdsmatrix"):
unique signature(x = "bdsmatrix", incomparables = "missing"):

```

bdsmatrix.ibd

Create a bdsmatrix from a list

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>idlist</code> . 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)
```

besthint

Create a hints matrix for a pedigree.

Description

A pedigree structure can contain a hints matrix which helps to reorder the pedigree (e.g. left-to-right order of children within family) so as to plot with minimal distortion. This routine tries out a large number of configurations, finding the best by brute force.

Usage

```
besthint(ped, wt=c(1000, 10, 1), tolerance=0)
```

Arguments

<code>ped</code>	a pedigree object
<code>wt</code>	relative weights for three types of "distortion" in a plotted pedigree. The final score for a pedigree is the weighted sum of these; the lowest score is considered the best. The three components are 1: the number of dotted lines, connecting two instances of the same person; 2: the lengths of those dotted lines; and 3: the horizontal offsets between parent/child pairs.
<code>tolerance</code>	the threshold for acceptance. If any of the orderings that are attempted have a score that is less than or equal to this value, the routine ceases searching for a better one.

Details

Assume that a pedigree has k founding couples, i.e., husband-wife pairs for which neither has a parent in the pedigree. The routine tries all $k!/2$ possible left to right orderings of the founders (in random order), uses the `autohint` function to optimize the order of children within each family, and computes a score. The hints matrix for the first pedigree to match the tolerance level is returned, or that for the best score found if none match the tolerance.

Value

a hints matrix

See Also

[pedigree](#), [plot.pedigree](#), [autohint](#)

Examples

```
## Not run:
# Find a good plot, only trying to avoid dotted connectors
myped$hints <- besthint(myped, wt=c(1000,100,0))

## End(Not run)
```

 coxme

Fit a mixed-effects Cox model

Description

Returns an object of class `coxme` representing the fitted model.

Usage

```
coxme(fixed, data=parent.frame(), random,
      weights, subset, na.action, init, control,
      ties=c("efron", "breslow", "exact"), singular.ok=T,
      varlist, variance, vinit=.2, sparse=c(50, .02), rescale=T, pdcheck=T, x=F, y=T, shortlabel=T, ...)
```

Arguments

<code>fixed</code>	formula describing the fixed effects part of the model.
<code>data</code>	a data frame containing the variables.
<code>random</code>	a one-sided formula describing the random effects part of the model.
<code>weights</code>	case weights for each observation
<code>subset</code>	an expression describing the subset of the data that should be used in the fit.
<code>na.action</code>	a function giving the default action on encountering missing values. It is more usual to use the global <code>na.action</code> system option to control this.

<code>init</code>	initial values for the coefficients for the fixed portion of the model, or the frailties followed by the fixed effect coefficients.
<code>control</code>	the result of a call to <code>coxme.control</code>
<code>ties</code>	the approximation to be used for tied death times: either "efron" or "breslow"
<code>singular.ok</code>	if TRUE, then redundant coefficients among the fixed effects are set to NA, if FALSE the program will fail with an error message if there are redundant variables.
<code>varlist</code>	variance specifications, often of class <code>bdsmatrix</code> , describing the variance/covariance structure of one or more of the random effects.
<code>variance</code>	fixed values for the variances of selected random effects. Values of 0 indicate that the final value should be solved for.
<code>vinit</code>	vector of initial values for variance terms. It is necessary that the initial variance matrix be symmetric positive definite. Normally, a simple sum of the <code>varlist</code> matrices will suffice, i.e., a vector of 1s; but not always.
<code>sparse</code>	determines which levels of random effects factor variables, if any, for which the program will use sparse matrix techniques. If a grouping variable has less than <code>sparse[1]</code> levels, then sparse methods are not used for that variable. If it has greater than or equal to <code>sparse[1]</code> unique levels, sparse methods will be used for those values which represent less than <code>sparse[2]</code> as a proportion of the data. For instance, if a grouping variable has 4000 levels, but 40% of the subjects are in group 1 then 3999 of the levels will be represented sparsely in the variance matrix. A single logical value of F is equivalent to setting <code>sparse[1]</code> to infinity.
<code>rescale</code>	scale any user supplied variance matrices so as to have a diagonal of 1.0.
<code>pdcheck</code>	verify that any user-supplied variance matrix is positive definite (SPD). It has been observed that IBD matrices produced by some software are not strictly SPD. Sometimes models with these matrices still work (throughout the iteration path, the weighted sum of variance matrices was always SPD) and sometimes they don't. In the latter case, messages about taking the log of negative numbers will occur, and the results of the fit are not necessarily trustworthy.
<code>x</code>	retain the X matrix in the output.
<code>y</code>	retain the dependent variable (a <code>Surv</code> object) in the output.
<code>shortlabel</code>	no comment(s)
<code>...</code>	any other arguments

Value

an object of class `coxme`

See Also

`coxph`

Examples

```
## Not run:
coxme(Surv(time, status) ~ rx, data=rats, random= ~1|litter)

Cox mixed-effects kinship model fit by maximum likelihood
Data: rats
n= 150
              NULL Integrated Penalized
Log-likelihood -185.6556  -180.849  -173.774

Penalized loglik: chisq= 23.76 on 13.17 degrees of freedom, p= 0.036
Integrated loglik: chisq= 9.61 on 2 degrees of freedom, p= 0.0082

Fixed effects: Surv(time, status) ~ rx
              coef exp(coef) se(coef)      z      p
rx 0.9132825  2.492491 0.3226856  2.830255 0.0046511

Random effects: ~ 1 | litter
                litter
Variance: 0.4255484

## End(Not run)
```

coxme.control	<i>Control parameters for coxme</i>
---------------	-------------------------------------

Description

Set various control parameters for the coxme function.

Usage

```
coxme.control(eps=0.00001, toler.chol=.Machine$double.eps^0.75,
toler.ms=.01, inner.iter=4,
iter.max=10, simplex=0, lower=0, upper=Inf, sparse.calc=NULL )
```

Arguments

eps	convergence criteria for the inner Cox model computations. Iteration ceases when the relative change in the log-likelihood is less than eps.
toler.chol	tolerance that is used to detect singularity, i.e., redundant predictor variables in the model, in the underlying Cholesky decomposition routines.
toler.ms	convergence criteria for the minimization of the integrated loglikelihood over the variance parameters. Since this “outer” iteration uses the Cox iteration as an inner loop, and the Cox iteration in turn uses the cholesky decomposition as an inner look, each of these treating the computations below it as if they were exact, the cholesky tolerance should be tighter than the Cox tolerance, which in turn should be tighter than that for the variance estimates.

<code>inner.iter</code>	the number of iterations for the inner iteration loop.
<code>iter.max</code>	maximum number of iterations for solution of a Cox partial likelihood, given the values of the random effect variances. Calls with <code>iter=0</code> are useful to evaluate the likelihood for a prespecified parameter vector, such as in the computation of a profile likelihood.
<code>simplex</code>	number of iterations for the Nelder-Mead simplex algorithm. The simplex method is very good at finding the general neighborhood of a minimum without getting lost, but can take a very large number of iterations to narrow in on the final answer; opposite strengths to the standard minimizer <code>optim</code> . For hard problems, adding 50-100 iterations of the simplex as a starting estimate for the usual method can be very helpful.
<code>lower, upper</code>	limits for the variance parameters, used by <code>optim</code> .
<code>sparse.calc</code>	style of computation for the inner likelihood code. The results of the two computations are identical, but can differ in total compute time. The optional calculation (<code>calc=1</code>) uses somewhat more memory, but can be substantially faster when the total number of random effects is of order n , the total sample size. The standard calculation (<code>calc=0</code>) is faster when the number of random effects is small. By default, the <code>coxme.fit</code> function chooses the method dynamically. It may not always do so optimally.

Details

The central computation consists of an outer maximization to determine the variances of the random effects, performed by the `optim` function. Each evaluation for `optim`, however, itself requires the solution of a minimization problem; this is the inner loop. It is important that the inner loop use a fixed number of iterations, but it is not yet clear what is the minimal sufficient number for that inner loop. Making this number smaller will make the routine faster, but perhaps at the expense of accuracy.

Value

a list containing values for each option.

References

Therneau and Pankratz

See Also

[coxme](#)

 familycheck

Error check for a family classification

Description

Given a family id vector, also compute the familial grouping from first principles using the parenting data, and compare the results.

Usage

```
familycheck(famid, id, father.id, mother.id, newfam)
```

Arguments

famid	a vector of family identifiers
id	a vector of unique subject identifiers
father.id	vector containing the id of the biological father
mother.id	vector containing the id of the biological mother
newfam	the result of a call to makefamid. If this has already been computed by the user, adding it as an argument shortens the running time somewhat.

Details

The makefamid function is used to create a de novo family id from the parentage data, and this is compared to the family id given in the data.

Value

a data frame with one row for each unique family id in the famid argument. Components of the output are

famid	the family id, as entered into the data set
n	number of subjects in the family
unrelated	number of them that appear to be unrelated to anyone else in the entire pedigree set. This is usually marry-ins with no children (in the pedigree), and if so are not a problem.
split	number of unique "new" family ids. If this is 0, it means that no one in this "family" is related to anyone else (not good); 1 = everything is fine; 2+= the family appears to be a set of disjoint trees. Are you missing some of the people?
join	number of other families that had a unique famid, but are actually joined to this one. 0 is the hope. If there are any joins, then an attribute "join" is attached. It will be a matrix with famid as row labels, new-family-id as the columns, and the number of subjects as entries.

See Also

[makefamid](#), [makekinship](#)

Examples

```
## Not run:
#
# This is from a pedigree that had some identifier errors
#
> checkit<- familycheck(ids2$famid, ids2$gid, ids2$fatherid, ids2$motherid)
> table(checkit$split) # should be all 1's
  0  1  2
112 424 4
# Shows 112 of the "families" were actually isolated individuals,
# and that 4 of the families actually split into 2.
# In one case, a mistyped father id caused one child, along with his spouse
# and children, to be "set adrift" from the connected pedigree.

> table(checkit$join)
  0  1  2
531 6  3
#
# There are 6 families with 1 other joined to them (3 pairs), and 3 with
# 2 others added to them (one triplet).
# For instance, a single mistyped father id of someone in family 319,
# which was by bad luck the id of someone else in family 339,
# was sufficient to join two groups.
> attr(checkit, 'join')
  [,1] [,2] [,3] [,4] [,5] [,6] [,7]
31  78   0   0   0   0   0   0
32   3  15   0   0   0   0   0
33   6   0  12   0   0   0   0
63   0   0   0  63   0   0   0
65   0   0   0  17  16   0   0
122  0   0   0   0   0  16   0
127  0   0   0   0   0   0  30   0
319  0   0   0   0   0   0   0  20
339  0   0   0   0   0   0   0  37

## End(Not run)
```

gchol

Generalized Cholesky decomposition

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 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

```
## Not run:
# 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)) # print out L
print(diag(ch1))      # print out D
aeq <- function(x,y) all.equal(as.vector(x), as.vector(y))
aeq(diag(ch1)[6], 0)  # Check that it has a zero in the proper place

ginv <- solve(ch1)   # see if I get a generalized inverse
```

```

aeq(temp %**% ginv %**% temp, temp)
aeq(ginv %**% temp %**% ginv, ginv)

## End(Not run)

```

gchol-class

Class "gchol"

Description

a class of gchol

Objects from the Class

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

Slots

```

.Data: Object of class "numeric" ~~
.Dim: Object of class "integer" ~~
.Dimnames: Object of class "list or NULL" ~~
rank: Object of class "integer" ~~

```

Methods

```

coerce signature(from = "gchol", to = "matrix"): ...
diag signature(x = "gchol"): ...
show signature(object = "gchol"): ...

```

gchol-methods

Methods for Function gchol in Package 'kinship'

Description

gchol methods

Methods

`x = "matrix"` an ordinary matrix or bdsmatrix object

gchol.bdsmatrix-class *Class "gchol.bdsmatrix"*

Description

an class generated from gchol(bdsmatrix object)

Objects from the Class

Objects can be created by calls of the form `new("gchol.bdsmatrix", ...)`. or `gchol(bdsmatrix object)`

Slots

`blocksize`: Object of class "integer" ~~
`blocks`: Object of class "numeric" ~~
`rmat`: Object of class "matrix" ~~
`rank`: Object of class "integer" ~~
`.Dim`: Object of class "integer" ~~
`.Dimnames`: Object of class "list or NULL" ~~

Methods

`%*%` 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"): ...

kinship	<i>Compute a kinship matrix</i>
---------	---------------------------------

Description

Computes the n by n kinship matrix for a set of n related subjects

Usage

`kinship(id, father.id, mother.id)`

Arguments

`id` a vector of subject identifiers. It may be either numeric or character.
`father.id` for each subject, the identifier of the biological father.
`mother.id` for each subject, the identifier of the biological mother.

Details

Two genes G1 and G2 are identical by descent (ibd) if they are both physical copies of the same ancestral gene; two genes are identical by state if they represent the same allele. So the brown eye gene that I inherited from my mother is ibd with hers; the same gene in an unrelated individual is not.

The kinship coefficient between two subjects is the probability that a randomly selected allele will be ibd between them. It is obviously 0 between unrelated individuals. If there is no inbreeding in the pedigree, it will be .5 for an individual with themselves (we could choose the same allele twice), .25 between mother and child, etc.

The computation is based on a recursive algorithm described in Lange. It is unfortunately not vectorizable, so the S code is slow. For studies with multiple disjoint families see the `makekinship` routine.

Value

a matrix of kinship coefficients.

References

K Lange, *Mathematical and Statistical Methods for Genetic Analysis*, Springer-Verlag, New York, 1997.

See Also

[makekinship](#), [makefamid](#)

Examples

```
## Not run:
test1 <- data.frame(id =c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14),
                    mom =c(0, 0, 0, 0, 2, 2, 4, 4, 6, 2, 0, 0, 12, 13),
                    dad =c(0, 0, 0, 0, 1, 1, 3, 3, 3, 7, 0, 0, 11, 10),
                    sex =c(0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1))
round(8*kinship(test1$id, test1$dad, test1$mom))

  1 2 3 4 5 6 7 8 9 10 11 12 13 14
1 4 0 0 0 2 2 0 0 1 0 0 0 0 0
2 0 4 0 0 2 2 0 0 1 2 0 0 0 1
3 0 0 4 0 0 0 2 2 2 1 0 0 0 0
4 0 0 0 4 0 0 2 2 0 1 0 0 0 0
5 2 2 0 0 4 2 0 0 1 1 0 0 0 0
6 2 2 0 0 2 4 0 0 2 1 0 0 0 0
7 0 0 2 2 0 0 4 2 1 2 0 0 0 1
```

```

 8 0 0 2 2 0 0 2 4 1 1 0 0 0 0
 9 1 1 2 0 1 2 1 1 4 1 0 0 0 0
10 0 2 1 1 1 1 2 1 1 4 0 0 0 2
11 0 0 0 0 0 0 0 0 0 0 4 0 2 1
12 0 0 0 0 0 0 0 0 0 0 0 4 2 1
13 0 0 0 0 0 0 0 0 0 0 2 2 4 2
14 0 1 0 0 0 0 1 0 0 2 1 1 2 4

```

```
## End(Not run)
```

 kinship.ops

Basic Linear Algebra for classes bdsmatrix and gchol.bdsmatrix

Description

Basic linear algebra operators for class `bdsmatrix` and `gchol.bdsmatrix`

 list or NULL-class

Class "list or NULL"

Description

a class used in `bdsmatrix`

Objects from the Class

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

Methods

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

 lmekin

Linear Mixed Effects model using a kinship matrix.

Description

A similar function to `lme`, but allowing for a complete specification of the covariance matrix for the random effects.

Usage

```
lmekin(fixed, data=parent.frame(), random,
       varlist=NULL, variance, sparse=c(20, .05),
       rescale=T, pdcheck=T,
       subset, weight, na.action)
```

Arguments

<code>fixed</code>	model statement for the fixed effects
<code>random</code>	model statement for the random effects
<code>data</code>	data frame containing the variables
<code>varlist</code>	variance specifications, often of class <code>bdsmatrix</code> , describing the variance/covariance structure of one or more of the random effects.
<code>variance</code>	fixed values for the variances of selected random effects. Values of 0 indicate that the final value should be solved for.
<code>sparse</code>	determines which levels of random effects factor variables, if any, for which the program will use sparse matrix techniques. If a grouping variable has less than <code>sparse[1]</code> levels, then sparse methods are not used for that variable. If it has greater than or equal to <code>sparse[1]</code> unique levels, sparse methods will be used for those values which represent less than <code>sparse[2]</code> as a proportion of the data. For instance, if a grouping variable has 4000 levels, but 40% of the subjects are in group 1 then 3999 of the levels will be represented sparsely in the variance matrix. A single logical value of F is equivalent to setting <code>sparse[1]</code> to infinity.
<code>rescale</code>	scale any user supplied variance matrices so as to have a diagonal of 1.0.
<code>pdcheck</code>	verify that any user-supplied variance matrix is positive definite (SPD). It has been observed that IBD matrices produced by some software are not strictly SPD. Sometimes models with these matrices still work (throughout the iteration path, the weighted sum of variance matrices was always SPD) and sometimes they don't. In the latter case, messages about taking the log of negative numbers will occur, and the results of the fit are not necessarily trustworthy.
<code>subset</code>	selection of a subset of data
<code>weight</code>	optional case weights
<code>na.action</code>	the action for missing data values

Details

The `lme` function is designed to accept a prototype for the variance matrix of the random effects, with the same prototype applying to all of the groups in the data. For familial genetic random effects, however, each family has a different covariance pattern, necessitating the input of the entire set of covariance matrices. In return, at present `lmekin` does not have the prototype abilities of `lme`.

Value

an object of class `'lmekin'`, sharing similarities with both `lm` and `lme` objects.

References

Pinheiro and Bates, Mixed Effect Models in S and Splus

See Also

[print.lmekin](#), `lme`

Examples

```
## Not run:
#
# Make a kinship matrix for the entire study
# These two functions are NOT fast, the makekinship one in particular
#
cfam <- makefamid(main$gid, main$momid, main$dadid)
kmat <- makekinship(cfam, main$gid, main$momid, main$dadid)

# The kinship matrix for the females only: quite a bit smaller
#
kid <- dimnames(kmat)[[1]]
temp <- main$sex[match(kid, main$gid)] == 'F'
fkmat <- kmat[temp,temp]

# The dimnames on kmat are the gid value, which are necessary to match
# the appropriate row/col of kmat to the analysis data set
# A look at %dense tissue on a mammogram, with age at mammogram and
# weight as covariates, and a familial random effect
#
fit <- lmeKin(percdens ~ mamage + weight, data=anal1,
             random = ~1|gid, varlist=list(fkmat))

Linear mixed-effects kinship model fit by maximum likelihood
Data: anal1
Log-likelihood = -6093.917
n= 1535

Fixed effects: percdens ~ mamage + weight
(Intercept)  mamage  weight
      87.1593 -0.5333198 -0.1948871

Random effects: ~ 1 | gid
               Kinship Residual
StdDev: 7.801603 10.26612

## End(Not run)
```

makefamid

Identify family groups

Description

Given a set of parentage relationships, this subdivides a set of subjects into families.

Usage

```
makefamid(id, father.id, mother.id)
```

Arguments

id a vector of unique subject identifiers
father.id for each subject, the identifier of their biological father
mother.id for each subject, the identifier of their biological mother

Details

This function may be useful to create a family identifier if none exists in the data (rare), to check for anomalies in a given family identifier (see the `familycheck` function), or to create a more space and time efficient kinship matrix by separating out marry-ins without children as 'unrelated'.

Value

a vector of family identifiers. Individuals who are not blood relatives of anyone else in the data set as assigned a family id of 0.

See Also

[makefamid](#), [kinship](#), [makekinship](#)

Examples

```
## Not run:
> newid <- makefamid(cdata$gid, cdata$dadid, cdata$momid)
> table(newid==0)
  FALSE TRUE
  17859 8191
# So nearly 1/3 of the individuals are not blood relatives.

> kin1 <- makekinship(cdata$famid, cdata$gid, cdata$dadid, cdata$momid)
> kin2 <- makekinship(newid, cdata$gid, cdata$dadid, cdata$momid, unique=0)
> dim(kin2)
[1] 26050 26050
> dim(kin1)
[1] 26050 26050

> length(kin2@blocks)/length(kin1@blocks)
[1] 0.542462
# Basing kin1 on newid rather than cdata$famid (where marry-ins were each
# labeled as members of one of the 426 families) reduced its size by just
# less than half.

## End(Not run)
```

makekinship *Create a sparse kinship matrix*

Description

Compute the overall kinship matrix for a collection of families, and store it efficiently.

Usage

```
makekinship(famid, id, father.id, mother.id, unrelated=0)
```

Arguments

famid	a vector of family identifiers
id	a vector of unique subject identifiers
father.id	for each subject, the identifier of their biological father
mother.id	for each subject, the identifier of their biological mother
unrelated	subjects with this family id are considered to be unrelated singletons, i.e., not related to each other or to anyone else.

Details

For each family of more than one member, the kinship function is called to calculate a per-family kinship matrix. These are stored in an efficient way into a single block-diagonal sparse matrix object, taking advantage of the fact that between family entries in the full matrix are all 0. Unrelated individuals are considered to be families of size 0, and are placed first in the matrix.

The final order of the rows within this matrix will not necessarily be the same as in the original data, since each family must be contiguous. The dimnames of the matrix contain the id variable for each row/column. Also note that to create the kinship matrix for a subset of the data it is necessary to create the full kinship matrix first and then subset it. One cannot first subset the data and then call the function. For instance, a call using only the female data would not detect that a particular man's sister and his daughter are related.

Value

a sparse kinship matrix of class `bdsmatrix`

See Also

kinship, makefamid

Examples

```
## Not run:
# Data set from a large family study of breast cancer
# there are 26050 subjects in the file, from 426 families
> table(cdata$sex)
  F    M
12699 13351
> length(unique(cdata$famid))
[1] 426

> kin1 <- makekinship(cdata$famid, cdata$gid, cdata$dadid, cdata$momid)
> dim(kin1)
[1] 26050 26050
> class(kin1)
[1] "bdsmatrix"
# The next line shows that few of the elements of the full matrix are >0
> length(kin1@blocks)/ prod(dim(kin1))
[1] 0.00164925

# kinship matrix for the females only
> femid <- cdata$gid[cdata$sex=='F']
> femindex <- !is.na(match(dimnames(kin1)[[1]], femid))
> kin2 <- kin1[femindex, femindex]
#
# Note that "femindex <- match(femid, dimnames(kin1)[[1]])" is wrong, since
# then kin1[femindex, femindex] might improperly reorder the rows/cols
# (if families were not contiguous in cdata).
# However sort(match(femid, dimnames(kin1)[[1]])) would be okay.

## End(Not run)
```

pedigree

Create pedigree structure

Description

Create pedigree structure in format needed for plotting function.

Usage

```
pedigree(id, dadid, momid, sex, affected, status, relations)
```

Arguments

id	Identification variable for individual
dadid	Identification variable for father
momid	Identification variable for mother

sex	Gender of individual noted in 'id'. Character("male","female","unknown", "terminated") or numeric (1="male", 2="female", 3="unknown", 4="terminated") allowed.
affected	One variable, or a matrix, indicating affection status. Assumed that 1="unaffected", 2="affected", NA or 0 = "unknown".
status	Status (0="censored", 1="dead")
relations	A matrix with 3 columns (id1, id2, code) specifying special relationship between pairs of individuals. Codes: 1=Monozygotic twin, 2=Dizygotic twin, 3=Twin of unknown zygosity, 4=Spouse and no children in pedigree

Value

An object of class pedigree.

See Also

[plot.pedigree](#)

Examples

```
## Not run:
ptemp <- pedigree(id=d10$upn, dadid=d10$dadid,momid=d10$momid,
                  sex=d10$sex, affected=d10$affect)
plot(ptemp)

## End(Not run)
```

plot.pedigree

plot pedigrees

Description

plot objects created with the pedigree function

Usage

```
## S3 method for class 'pedigree'
plot(x, id = x$id, status = x$status, affected =
      x$affected, cex = 1, col = 1, symbolsize = 1, branch =
      0.6, packed = TRUE, align = c(1.5,2), width = 8,
      density = c(-1, 35, 55, 25), mar = c(4.1, 1, 4.1, 1),
      angle = c(90, 65, 40, 0), keep.par = FALSE, subregion,
      ...)
```

Arguments

x	object created by the function pedigree.
id	id variable - used for labeling.
status	can be missing. If it exists, 0=alive/missing and 1=death.
affected	vector, or matrix with up to 4 columns for affected indicators. Subject's symbol is divided into sections for each status, shaded if indicator is 1, not-shaded for 0, and symbol "?" if missing (NA)
cex	controls text size. Default=1.
col	color for each id. Default assigns the same color to everyone.
symbolsize	controls symbolsize. Default=1.
branch	defines how much angle is used to connect various levels of nuclear families.
packed	default=TRUE. If TRUE, uniform distance between all individuals at a given level.
align	these parameters control the extra effort spent trying to align children underneath parents, but without making the pedigree too wide. Set to FALSE to speed up plotting.
width	default=8. For a packed pedigree, the minimum width allowed in the realignment of pedigrees.
density	defines density used in the symbols. Takes up to 4 different values.
mar	margin parameters, as in the par function
angle	defines angle used in the symbols. Takes up to 4 different values.
keep.par	Default = FALSE, allows user to keep the parameter settings the same as they were for plotting (useful for adding extras to the plot)
subregion	4-element vector for (min x, max x, min depth, max depth), used to edit away portions of the plot coordinates returned by align.pedigree
...	Extra options that feed into the plot function.

Details

Two important parameters control the looks of the result. One is the user specified maximum width. The smallest possible width is the maximum number of subjects on a line, if the user's suggestion is too low it is increased to 1+ that amount (to give just a little wiggle room). To make a pedigree where all children are centered under parents simply make the width large enough, however, the symbols may get very small.

The second is align, a vector of 2 alignment parameters a and b . For each set of siblings at a set of locations x and with parents at $p=c(p_1, p_2)$ the alignment penalty is

$$(1/k^a) \sum_i i = 1k[(x_i - (p_1 + p_2)/2)]^2$$

$\sum(x - \text{mean}(p))^2/(k^a)$ where k is the number of siblings in the set. when $a=1$ moving a sibship with k sibs one unit to the left or right of optimal will incur the same cost as moving one with only 1 or two sibs out of place. If $a=0$ then large sibships are harder to move than small ones, with the default value $a=1.5$ they are slightly easier to move than small ones. The rationale for

the default is as long as the parents are somewhere between the first and last siblings the result looks fairly good, so we are more flexible with the spacing of a large family. By tethering all the sibs to a single spot they are kept close to each other. The alignment penalty for spouses is $b(x_1 - x_2)^2$, which tends to keep them together. The size of `$b` controls the relative importance of sib-parent and spouse-spouse closeness.

Value

an invisible list containing

<code>plist</code>	a list that contains all the position information for plotting the pedigree. This will be useful for further functions (yet unwritten) for manipulating the plot, but likely not to an ordinary user.
<code>x,y</code>	the x and y plot coordinates of each subject in the plot. The coordinate is for the top of the plotted symbol. These will be in the same order as the input pedigree. If someone in the pedigree does not appear in the plot their coordinates will be NA. If they appear multiple times one of the instances is chosen. (Which one is a function of the order in which the pedigree was constructed.)
<code>boxh</code>	the height of the symbol, in user coordinates
<code>boxw</code>	the width of the symbol
<code>call</code>	a copy of the call that generated the plot

Side Effects

creates plot on current plotting device.

See Also

[pedigree](#)

Examples

```
## Not run:
# The example to R News and Bioinformatics
# pedigree 10081 in GAW14
p1 <- scan(nlines=16,what=list(0,0,0,0,0,"",""))
1  2  3  2  2  7/7  7/10
2  0  0  1  1  -/-  -/-
3  0  0  2  2  7/9  3/10
4  2  3  2  2  7/9  3/7
5  2  3  2  1  7/7  7/10
6  2  3  1  1  7/7  7/10
7  2  3  2  1  7/7  7/10
8  0  0  1  1  -/-  -/-
9  8  4  1  1  7/9  3/10
10 0  0  2  1  -/-  -/-
11 2 10 2  1  7/7  7/7
12 2 10 2  2  6/7  7/7
13 0  0  1  1  -/-  -/-
14 13 11 1  1  7/8  7/8
```

```

15  0  0  1  1  -/-  -/-
16 15 12  2  1  6/6  7/7

p2 <- as.data.frame(p1)
names(p2) <- c("id", "fid", "mid", "sex", "aff", "GABRB1", "D4S1645")
attach(p2)
sex <- sex-1
p3 <- pedigree(id, fid, mid, sex, aff)
pdf("10081.pdf")
par(xpd=TRUE)
plot(p3)
dev.off()

## End(Not run)

```

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, tolerance=1e-10, full=T, ...)

```

Arguments

<code>a</code>	a block diagonal sparse matrix object
<code>b</code>	a numeric vector or matrix, that forms the right-hand side of the equation.
<code>tolerance</code>	the tolerance for detecting singularity in the <code>a</code> matrix
<code>full</code>	if true, return the full inverse matrix; if false return only that portion corresponding to the blocks. This argument is ignored if <code>b</code> is present. If the <code>bdsmatrix</code> <code>a</code> has a non-sparse portion, i.e., if the <code>rmat</code> component is present, then the inverse of <code>a</code> will not be block-diagonal sparse. In this case setting <code>full=F</code> 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.
<code>...</code>	an argument to achieve compatibility with <code>solve</code> from R base

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

```
## Not run:
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(1,0,1,1,0,0,1,1,0,1,0,1,0,10,0,
                           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)))

## End(Not run)
```

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=T, ...)
```

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>	an argument to achieve compatibility with <code>solve</code> from R base

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

```
## Not run:
# 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)) # print out L
print(diag(ch1))      # print out D
aeq <- function(x,y) all.equal(as.vector(x), as.vector(y))
aeq(diag(ch1)[6], 0)  # Check that it has a zero in the proper place

ginv <- solve(ch1)    # see if I get a generalized inverse
aeq(temp %*% ginv %*% temp, temp)
aeq(ginv %*% temp %*% ginv, ginv)

## End(Not run)
```

solve.gchol.bdsmatrix *function solve for gchol.bdsmatrix*

Description

This is a generalized solve function for gchol.bdsmatrix object

Usage

```
## S3 method for class 'gchol.bdsmatrix'
solve(a, b, full=T, ...)
```

Arguments

a	a
b	b
full	An argument full
...	an argument to achieve compatibility with solve from R base

Index

- *Topic **algebra**
 - kinship.ops, 21
 - solve.bdsmatrix, 30
 - solve.gchol, 31
 - solve.gchol.bdsmatrix, 32
- *Topic **array**
 - align.pedigree, 2
 - as.matrix.bdsmatrix, 3
 - autohint, 4
 - bdsBlock, 5
 - bdsI, 6
 - bdsmatrix.ibd, 9
 - besthint, 10
 - gchol, 16
 - kinship, 19
- *Topic **classes**
 - bdsmatrix-class, 8
 - gchol-class, 18
 - gchol.bdsmatrix-class, 19
 - list or NULL-class, 21
- *Topic **dplot**
 - pedigree, 26
- *Topic **hplot, genetics**
 - plot.pedigree, 27
- *Topic **manip**
 - bdsmatrix, 7
 - familycheck, 15
 - makefamid, 23
 - makekinship, 25
- *Topic **methods**
 - gchol-methods, 18
- *Topic **regression**
 - lmekin, 21
- *Topic **survival**
 - coxme, 11
 - coxme.control, 13
- [, bdsmatrix-method (bdsmatrix-class), 8
- [, gchol.bdsmatrix-method (gchol.bdsmatrix-class), 19
- [.pedigree (kinship.ops), 21
- %% (kinship.ops), 21
- %%, ANY, ANY-method (kinship.ops), 21
- %%, bdsmatrix, numeric-method (kinship.ops), 21
- %%, bdsmatrix-method (kinship.ops), 21
- %%, gchol.bdsmatrix, numeric-method (kinship.ops), 21
- %%, gchol.bdsmatrix-method (kinship.ops), 21
- %%, matrix, bdsmatrix-method (bdsmatrix-class), 8
- %%, matrix, gchol.bdsmatrix-method (gchol.bdsmatrix-class), 19
- %%, numeric, bdsmatrix-method (bdsmatrix-class), 8
- %%, numeric, gchol.bdsmatrix-method (gchol.bdsmatrix-class), 19
- %%-methods (kinship.ops), 21
- %%.bdsmatrix (kinship.ops), 21
- %%.default (kinship.ops), 21
- %%.gchol.bdsmatrix (kinship.ops), 21
- align.pedigree, 2
- alignped1 (align.pedigree), 2
- alignped2 (align.pedigree), 2
- alignped3 (align.pedigree), 2
- alignped4 (align.pedigree), 2
- all, bdsmatrix-method (bdsmatrix-class), 8
- any, bdsmatrix-method (bdsmatrix-class), 8
- as.matrix.bdsmatrix, 3
- as.matrix.gchol (kinship.ops), 21
- as.vector.bdsmatrix (kinship.ops), 21
- autohint, 4, 11
- bdsBlock, 5
- bdsI, 5, 6
- bdsmatrix, 5, 7, 10, 17, 31

- bdsmatrix-class, 8
- bdsmatrix.ibd, 9
- bdsmatrix.reconcile (kinship.ops), 21
- besthint, 5, 10

- coerce,bdsmatrix,matrix-method
(bdsmatrix-class), 8
- coerce,bdsmatrix,vector-method
(bdsmatrix-class), 8
- coerce,gchol,matrix-method
(gchol-class), 18
- coerce,gchol.bdsmatrix,matrix-method
(gchol.bdsmatrix-class), 19
- coxme, 10, 11, 14
- coxme.control, 13

- diag,bdsmatrix-method
(bdsmatrix-class), 8
- diag,gchol-method (gchol-class), 18
- diag,gchol.bdsmatrix-method
(gchol.bdsmatrix-class), 19
- diag<- ,bdsmatrix-method
(bdsmatrix-class), 8
- dim,bdsmatrix-method (bdsmatrix-class),
8
- dim,gchol.bdsmatrix-method
(gchol.bdsmatrix-class), 19
- dimnames,bdsmatrix-method
(bdsmatrix-class), 8
- dimnames<- ,bdsmatrix-method
(bdsmatrix-class), 8

- familycheck, 15

- gchol, 16, 31, 32
- gchol,matrix-method (gchol-methods), 18
- gchol-class, 18
- gchol-methods, 18
- gchol.bdsmatrix-class, 19
- getCovariateFormula2 (lmekin), 21
- getCrossedTerms (lmekin), 21
- getGroupsFormula2 (lmekin), 21

- is.bdsmatrix (kinship.ops), 21
- is.gchol.bdsmatrix (kinship.ops), 21
- is.list.bdsmatrix (kinship.ops), 21
- is.list.gchol.bdsmatrix (kinship.ops),
21
- is.matrix.bdsmatrix (kinship.ops), 21

- is.matrix.gchol.bdsmatrix
(kinship.ops), 21

- kindepth (kinship), 19
- kinship, 10, 19, 24
- kinship.ops, 21

- list or NULL-class, 21
- lmekin, 10, 21

- makefamid, 16, 20, 23, 24
- makekinship, 16, 20, 24, 25
- Math,bdsmatrix-method
(bdsmatrix-class), 8
- Math2,bdsmatrix-method
(bdsmatrix-class), 8
- max,bdsmatrix-method (bdsmatrix-class),
8
- min,bdsmatrix-method (bdsmatrix-class),
8

- Ops,bdsmatrix,bdsmatrix-method
(bdsmatrix-class), 8
- Ops,bdsmatrix,numeric-method
(bdsmatrix-class), 8
- Ops,numeric,bdsmatrix-method
(bdsmatrix-class), 8

- pedigree, 5, 11, 26, 29
- plot.pedigree, 3, 11, 27, 27
- print.bdsmatrix (bdsmatrix), 7
- print.coxme (coxme), 11
- print.lmekin, 22
- print.lmekin (lmekin), 21
- prod,bdsmatrix-method
(bdsmatrix-class), 8

- range,bdsmatrix-method
(bdsmatrix-class), 8

- show,bdsmatrix-method
(bdsmatrix-class), 8
- show,gchol-method (gchol-class), 18
- show,gchol.bdsmatrix-method
(gchol.bdsmatrix-class), 19
- solve.bdsmatrix, 30
- solve.gchol, 17, 31
- solve.gchol.bdsmatrix, 32
- strata2 (coxme.control), 13

sum,bdsmatrix-method (bdsmatrix-class),
8

unique,bdsmatrix,missing-method
(bdsmatrix-class), 8