Package ‘nadiv’

February 26, 2013

Type  Package
Title  Functions to construct non-additive genetic relatedness matrices
Version 2.11
Date  2012-12-14
Author  Matthew Wolak
Maintainer  Matthew Wolak <matthewwolak@gmail.com>
Depends  Matrix
Suggests  parallel, asreml
Description  Constructs non-additive genetic relationship matrices from a pedigree. These and their inverses can be passed to a linear mixed effect model, known as the ‘animal model’.
License  GPL (>= 2)
LazyLoad  yes
LazyData  yes
NeedsCompilation  yes
Repository  CRAN
Date/Publication  2013-02-26 07:59:12

R topics documented:

<table>
<thead>
<tr>
<th>Package</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>nadiv-package</td>
<td>2</td>
</tr>
<tr>
<td>aiCI</td>
<td>3</td>
</tr>
<tr>
<td>aiFun</td>
<td>4</td>
</tr>
<tr>
<td>constrainFun</td>
<td>6</td>
</tr>
<tr>
<td>DFC</td>
<td>7</td>
</tr>
<tr>
<td>FG90</td>
<td>7</td>
</tr>
</tbody>
</table>
Functions to create non-additive relationship matrices and their inverses

Description

From a supplied pedigree, creates the matrices of dominance and epistatic relatedness. Also, the inverses of these matrices are produced, which is what is needed to estimate these non-additive variance components in a linear mixed effects model (i.e., the 'animal model').

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>nadjiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>2.11</td>
</tr>
<tr>
<td>Date:</td>
<td>2012-12-14</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;=2)</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Author(s)

Matthew Wolak <matthewwolak@gmail.com>
aiCI

Confidence Intervals for Variance Components

Description

Produces the 1-alpha Upper and Lower Confidence Limits for the variance components in an ASReml-R model.

Usage

aiCI(asr.model, Dimnames = NULL, alpha = 0.05)

Arguments

asr.model Object from a call to asreml
Dimnames A vector of characters if names are desired for the output. If not specified, the default labels from the asreml object will be used.
alpha A numeric value indicating the level of Type I error for constructing the Confidence Intervals.

Details

Variances from the inverse of the Average Information matrix of an ASReml model are translated according to the varTrans function and used in constructing the 1-alpha Confidence Interval.

Value

A matrix is returned with a row for each variance component. The three columns correspond to the Lower Confidence Limit, estimate from the asreml model, and Upper Confidence Limit for each variance component.

Note

The vector of Dimnames should match the same order of variance components specified in the model.

Author(s)

<matthewwolak@gmail.com>

See Also

aifun proLik
Examples

```r
## Not run:
library(asreml)
ginvA <- asreml.Ainverse(warcolak)$ginv
ginvD <- makeD(warcolak[,1:3])$listDinv
warcolak$IDD <- warcolak$ID
warcolak.mod <- asreml(trait1 ~ 1, random = ~ped(ID) + giv(IDD),
  ginverse = list(ID = ginvA, IDD = ginvD), data = warcolak)
summary(warcolak.mod)$varcomp
aiCI(warcolak.mod)
## End(Not run)
```

aiFun

### aiFun

#### Sampling (co)variances

This function returns the sampling (co)variances of the random effects fitted in an mixed model solved using the Average Information algorithm.

#### Usage

```r
aiFun(model = NULL, AI.vec = NULL, inverse = TRUE, Dimnames = NULL)
```

#### Arguments

- **model**: A model object returned by a call to the `asreml` function.
- **AI.vec**: A numeric vector of the Average Information matrix. The order must be the row-wise lower triangle of the matrix (including the diagonal).
- **inverse**: A logical indicating whether the elements of the INVERSE Average Information matrix are being provided. If FALSE, the Average Information matrix (and not its inverse) is being supplied.
- **Dimnames**: A vector of characters if names are desired for the output (co)variance matrix. If not specified, either the default labels from the `asreml` object will be used or the rows and columns will be un-labeled.

#### Details

The inverse of the Average Information matrix provides the sampling (co)variance of each random term in the mixed model. If a model from the ASReml-R function is supplied (model is not NULL), this function extracts the inverse of the AI matrix from an ASReml-R model and organizes it so that the sampling covariances between random terms are the off-diagonals and the sampling variances of random terms are located along the diagonal. The order of the variances along the diagonal is the same as the order entered in the random section of the `asreml` function. This is also the same order as the rows of a call to the summary function, `summary(model)$varcomp`. 
aiFun

If model is NULL then AI.vec should contain the vector of values from an Average Information matrix. The function will then reconstruct this matrix, invert it, and supply the sampling (co)variances for the random terms in the model as described above. Note, either model or AI.vec must be supplied, but not both.

Value

A matrix of k x k dimensions is returned, if k is the number of random effects estimated in the model. Sampling covariances are above and below the diagonal while variances are located along the diagonal. If Dimnames is specified, the row and column names are assigned according the vector of names in this argument.

Note

The vector of Dimnames should match the same order of variance components specified in the model.

Author(s)

<matthewwolak@gmail.com>

References


Examples

```r
## Not run:
library(asreml)
ginvA <- asreml.Ainverse(warcolak)$ginv
ginvD <- makeD(warcolak[,1:3])$listDinv
warcolak$IDD <- warcolak$ID
warcolak.mod <- asreml(trait1 ~ 1, random = ~ped(ID) + giv(IDD),
ginverse = list(ID = ginvA, IDD = ginvD), data = warcolak)
summary(warcolak.mod)$varcomp
aiFun(model = warcolak.mod, Dimnames = c("Va", "Vd", "Ve"), inverse = TRUE)

## End(Not run)

output <- c(7.3/zero.noslash75921, 7./zero.noslash635161, 12.342338/zero.noslash, 1.9539486, 2.758634/zero.noslash, /zero.noslash.6626111)
aiFun(AI.vec = output, inverse = FALSE, Dimnames = c("Va", "Vd", "Ve"))
```
constrainFun  

Function used in conjunction with others to produce a profile likelihood for a variance component

Description

Given a model object from asreml and a range of estimates of the parameter, the function will supply the likelihood ratio test statistic for the comparison of the full model to one where the parameter of interest is constrained.

Usage

constrainFun(parameter.val, full, fm2, comp, G, mit = 600)

Arguments

- **parameter.val**: a value for which the log Likelihood of a model is to be calculated
- **full**: the full model asreml object
- **fm2**: starting values for the full model
- **comp**: which variance component to constrain
- **G**: logical, indicating if the component is part of the G structure
- **mit**: numeric, indicating maximum number of iterations for the constrained asreml model

Details

Used internally in the proLik function

Author(s)

<matthewwolak@gmail.com>

See Also

See Also proLik
DFC

**Indicates Double First Cousins**

**Description**

Given all eight grandparents of two individuals, the function will indicate whether they are double first cousins. Mainly to be used inside of `findDFC`

**Usage**

`DFC(ij.grandparents)`

**Arguments**

- `ij.grandparents`  
  a vector indicating the dam and sire of both i’s and j’s dam and sire

**Value**

returns either a 0 if individual’s i and j are not double first cousins, or a 1 if they are

**Author(s)**

<matthewwolak@gmail.com>

**See Also**

`findDFC`

---

**FG90 Pedigree, adapted from Table 1 in Fernando & Grossman (1990)**

**Description**

An example pedigree

**Usage**

`data(FG90)`

**Format**

A data frame with 8 observations on the following 4 variables.

- `id` a factor with levels `1 2 3 4 5 6 7 8`
- `dam` a factor with levels `2 4 6`
- `sire` a factor with levels `1 3 5`
- `sex` a factor with levels `0 1`
findDFC

Finds the double first cousins in a pedigree

Description
Given a pedigree, all pairs of individuals that are double first cousins are returned.

Usage
findDFC(pedigree, parallel = FALSE, ncores = getOption("mc.cores", 2L))

Arguments
pedigree          A pedigree with columns organized: ID, Dam, Sire
parallel          A logical statement indicating if parallelization should be attempted. Note, only
                   reliable for Mac and Linux operating systems.
ncores            Number of cpus to use, default is maximum available

Details
parallel = TRUE should only be used on Linux or Mac OSes (i.e., not Windows).

Value
PedPositionList   gives the list of row numbers for all the pairs of individuals that are related as
double first cousins
DFC               gives the list of IDs, as characters, for all the pairs of individuals that are related
double first cousins
FamilyCnt         If two individuals, i and j, are double first cousins, then i’s siblings will also
                   be double first cousins with j’s siblings. Therefore, this is the total number of
                   family pairs where offspring are related as double first cousins.

Author(s)
<matthewwolak@gmail.com>
**genAssign**

*Generation assignment*

**Description**
Given a pedigree, the function assigns the generation number each individual belongs in.

**Usage**
```r
genAssign(pedigree)
```

**Arguments**
- `pedigree` A pedigree with columns organized: ID, Dam, Sire

**Details**
- Zero is the base population.
- Adapted from kindepth function in kinship

**Value**
A vector of values is returned. This vector is in the same order as the ID column of the pedigree.

**See Also**
- kinship

**makeA**

*Creates the additive genetic relationship matrix*

**Description**
This returns the additive relationship matrix in sparse matrix format.

**Usage**
```r
makeA(pedigree)
```

**Arguments**
- `pedigree` A pedigree where the columns are ordered ID, Dam, Sire

**Details**
- Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.
- Used as a support function to makeD.
Value

Returns A, or the numerator relationship matrix, in sparse matrix form.

Author(s)

<matthewwolak@gmail.com>

See Also

makeD

Examples

makeA(Mrode2)

makeAA

Creates the additive by additive epistatic genetic relationship matrix

Description

Given a pedigree, the matrix of additive by additive genetic relatedness (AA) among all individuals in the pedigree is returned.

Usage

makeAA(pedigree)

Arguments

pedigree A pedigree where the columns are ordered ID, Dam, Sire

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.

The function first estimates the A matrix using makeA, then it calculates the Hadamard (element-wise) product of the A matrix with itself ($A \# A$).

Value

AA the AA matrix in sparse matrix form
logDet the log determinant of the AA matrix
AAinv the inverse of the AA matrix in sparse matrix form
listAAinv the three column form of the non-zero elements for the inverse of the AA matrix

Author(s)

<matthewwolak@gmail.com>
makeD

See Also

makeA

Examples

makeAA(Mrode2)

makeD

Creates the dominance genetic realtionship matrix

Description

Given a pedigree, the matrix of coefficients of fraternity are returned - the D matrix. Note, no
inbreeding must be assumed. Will return the inverse of the D matrix by default, otherwise this
operation can be skipped if desired.

Usage

makeD(pedigree, parallel = FALSE, ncores = getOption("mc.cores", 2L),
invertD = TRUE, returnA = FALSE, det = FALSE)

Arguments

pedigree A pedigree with columns organized: ID, Dam, Sire
parallel Logical, indicating whether computation should be run on multiple processors
at once. See details for considerations.
ncores Number of cores to use when parallel = TRUE. Default is maximum available.
Otherwise, set with an integer. See details for considerations.
invertD A logical indicating whether or not to invert the D matrix
returnA Logical, indicating if the numerator relationship matrix (A) should be stored and
returned.
det Logical, indicating if the determinant of the D matrix should be returned.

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.
There exists no convenient method of obtaining the inverse of the dominance genetic relatedness
matrix (or the D matrix itself) directly from a pedigree (such as for the inverse of A, i.e., Quaas
(1995)). Therefore, this function computes the coefficient of fraternity (Lynch and Walsh, 1998) for
every individual in the pedigree with a non-zero additive genetic relatedness. Note, the construction
of the D matrix is more computationally demanding (in computing time and space requirements)
than is the construction of A.

To overcome the computational difficulties, this function can enable parallel processing (see pack-
age parallel included in the R distribution) to speed up the execution. Note this may not be
possible on Windows (See parallel documentation for further information), therefore parallel
= TRUE should only be used on Linux or Mac operating systems (i.e., not Windows). The default is to use the maximum available cpus to the machine, but this can be restricted by indicating the number desired in the argument ncores. Setting up the multi-processing takes some overhead, so no real advantage is gained for small pedigrees. Also, since all processes are sharing a fixed amount of RAM, for very large pedigrees using many processes in parallel may not be feasible due to RAM restrictions (i.e., if each process needs “n” amount of RAM to run, then ncores should be set to = total RAM/n). Otherwise the machine can become overworked.

Note, for very large pedigrees returnA should be set to FALSE to avoid drastically increasing the memory requirements while making D. When this occurs, ’NULL’ is returned for the element of ‘A’ in the output of makeD.

Value

A      the A matrix in sparse matrix form
D      the D matrix in sparse matrix form
logDet the log determinant of the D matrix
Dinv   the inverse of the D matrix in sparse matrix form
listDinv the three column form of the non-zero elements for the inverse of the D matrix

Author(s)

<matthewwolak@gmail.com>

References


See Also

makeDsim

Examples

DinvMat <- makeD(Mrode9, parallel = FALSE)$Dinv

makeDomEpi

Creates the additive by dominance and dominance by dominance epistatic genetic relationship matrices

Description

Given a pedigree, the matrix of additive by dominance (AD) genetic relatedness, dominance by dominance (DD) genetic relatedness, or both are returned.
Usage

makeDomEpi(pedigree, output = c("AD", "DD", "both"), parallel = FALSE, Dinverse=FALSE, det = FALSE)

Arguments

  pedigree       A pedigree where the columns are ordered ID, Dam, Sire
  output         Character(s) denoting which matrix and its inverse is to be constructed.
  parallel       A logical indicating whether or not to use parallel processing. Note, this may
                  only be available on Mac and Linux operating systems.
  Dinverse       A logical indicating whether or not to invert the D matrix
  det            A logical indicating whether or not to return the determinants for the epistatic
                  relationship matrices

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.
Because of the computational demands of constructing the D matrix (see makeD), this function
allows for the inverses that are derived from the D matrix (i.e., D-inverse, AD-inverse, and DD-
inverse)to be constructed at the same time. This way, the D matrix will only have to be constructed
once for use in the three separate genetic relatedness inverse matrices that depend upon it. However,
using the output and Dinverse options in different combinations will ensure that only the desired
matrix inverses are constructed.

parallel = TRUE should only be used on Linux or Mac OSes (i.e., not Windows).
Both the AD and DD matrix are computed from the Hadamard product of the respective matrices
(see also, makeAA).

Value

All of the following will be returned. However, the values of the output and Dinverse options
passed to the function will determine which of the following are not NULL objects within the list:

  D             the D matrix in sparse matrix form
  logDetD       the log determinant of the D matrix
  AD            the AD matrix in sparse matrix form
  logDetAD      the log determinant of the AD matrix
  DD            the DD matrix in sparse matrix form
  logDetDD      the log determinant of the DD matrix
  Dinv          the inverse of the D matrix in sparse matrix form
  ADinv         the inverse of the AD matrix in sparse matrix form
  DDinv         the inverse of the DD matrix in sparse matrix form
  listDinv      the three column form of the non-zero elements for the inverse of the D matrix
  listADinv     the three column form of the non-zero elements for the inverse of the AD matrix
  listDDinv     the three column form of the non-zero elements for the inverse of the DD matrix
makeDsim

Creates the dominance genetic relationship matrix through iteration

Description

Alleles are explicitly traced through a pedigree to obtain coefficients of fraternity between pairs of individuals (the probability of sharing both alleles identical by descent). This is accomplished in an iterative process to account for the various routes by which an allele will progress through a pedigree due to Mendelian sampling. This is an implementation of the simulation approach of Ovaskainen et al. (2008).

Usage

makeDsim(pedigree, N, parallel = FALSE, ncores = getOption("mc.cores", 2L), invertD = TRUE, calcSE = FALSE, returnA = FALSE)

Arguments

- `pedigree`: A pedigree with columns organized: ID, Dam, Sire
- `N`: The number of times to iteratively trace alleles through the pedigree
- `parallel`: A logical indicating whether or not to use parallel processing. Note, this may only be available for Mac and Linux operating systems.
- `ncores`: The number of cpus to use when constructing the dominance relatedness matrix. Default is all available.
- `invertD`: A logical indicating whether or not to invert the D matrix
- `calcSE`: A logical indicating whether or not the standard errors for each coefficient of fraternity should be calculated
- `returnA`: Logical, indicating if the numerator relationship matrix (A) should be stored and returned.

Examples

Boutput <- makeDomEpi(Mrode9, output = "b", parallel = FALSE, Dinverse = FALSE)
str(Boutput)

DADoutput <- makeDomEpi(Mrode9, output = "AD", parallel = FALSE, Dinverse = TRUE)
str(DADoutput)
makeDsim

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.
parallel = TRUE should only be used on Linux or Mac operating systems (i.e., not Windows).
Ovaskainen et al. (2008) indicated that the method of calculating the D matrix (see makeD) is only
an approximation. They proposed a simulation method that is implemented here. This should be
more appropriate, especially when inbreeding occurs in the pedigree.
The value, listDsim will list both the approximate values (returned from makeD) as well as the
simulated values. If calcSE is TRUE, these values will be listed in listDsim.

Value

A     | the A matrix in sparse matrix form
D     | the approximate D matrix in sparse matrix form
logDetD | the log determinant of the approximate D matrix
Dinv  | the inverse of the approximate D matrix in sparse matrix form
listDinv | the three column form of the non-zero elements for the inverse of the approxi-
          | mate D matrix
Dsim  | the simulated D matrix in sparse matrix form
logDetDsim | the log determinant of the simulated D matrix
Dsiminv | the inverse of the simulated D matrix in sparse matrix form
listDsim | the three column form of the non-zero and non-self elements for the simulated
          | D matrix
listDsiminv | the three column form of the non-zero elements for the inverse of the simulated
          | D matrix

Note

This simulation can take a long time for large values of N. If unsure, it is advisable to start with a
lower N and gradually increase to gain a sense of the time required to execute a desired N.

Author(s)

<matthewwolak@gmail.com>

References


See Also

makeD

Examples

```R
simDinv <- makeDsim(Mrode9, N = 1000, parallel = FALSE, invertD = TRUE, calcSE = TRUE)$listDsim
```
**Description**

This is a preliminary function and may still contain some errors. The function returns the inverse of the additive relationship matrix in sparse matrix format for the sex chromosomes (i.e., either X or Z).

**Usage**

```
makeS(pedigree, heterogametic = "0", DosageComp = c(NULL, "ndc", "hoin"), returnS = FALSE)
```

**Arguments**

- `pedigree` A pedigree where the columns are ordered ID, Dam, Sire, Sex
- `heterogametic` Character indicating the label corresponding to the heterogametic sex used in the "Sex" column
- `DosageComp` A character indicating whether a model of no dosage compensation ("ndc") or homozygous inactivation ("hoin") should be used. The former ("ndc") follows the algorithm presented in Fernando and Grossman (1990) and has been validated. However, the latter ("hoin") is still being validated and results should be used cautiously. If "NULL" then the "ndc" model is assumed.
- `returnS` Logical statement, indicating if the relationship matrix should be constructed in addition to the inverse

**Details**

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.

The inverse of the sex-chromosome additive genetic relationship matrix (S-matrix) is constructed according to the algorithm presented in Fernando & Grossman (1990). Additionally, the S-matrix itself can be constructed (although this takes much longer than computing S-inverse directly).

The algorithm in Fernando & Grossman (1990) does not account for any form of sex chromosome dosage compensation mechanism. I have implemented one model, random X-inactivation (i.e., lyonization), in an attempt for more congruence with the biology of many organisms (e.g., eutherian mammals). This second model of random inactivation in the homozygous sex is still preliminary and has yet to be properly peer reviewed.

**Value**

- `model` the model of sex-chromosome dosage compensation assumed.
- `S` the sex-chromosome relationship matrix in sparse matrix form or NULL if `returnS = FALSE`
- `Sinv` the inverse of the S matrix in sparse matrix form
- `listSinv` the three column form of the non-zero elements for the inverse of the S matrix
Author(s)
<matthewwolak@gmail.com>

References

Examples

makeS(FG9/zero.noslash, heterogametic = "0", returnS = TRUE)

Description
An example pedigree

Usage
data(Mrode2)

Format
A data frame with 6 observations on the following 3 variables.

id  a numeric vector
dam a numeric vector
sire a numeric vector

Source
Mrode9 Pedigree, adapted from example 9.1 of Mrode (2005)

Description
An example pedigree

Usage
data(Mrode9)

Format
A data frame with 12 observations on the following 3 variables.
- pig a numeric vector
- dam a numeric vector
- sire a numeric vector

Source

numPed Creates a numeric form of a pedigree

Description
This function takes a pedigree and converts it into a standard integer form used by many functions in the ‘nadiv’ package.

Usage
numPed(pedigree, check = TRUE)

Arguments
- pedigree A three column pedigree object, where the columns correspond to: ID, Dam, & Sire
- check A logical argument indicating if checks on the validity of the pedigree structure should be made
parConstrainFun

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', or '*'.

Individuals must appear in the ID column in rows preceeding where they appear in either the Dam or Sire column. If check = FALSE any pedigree will be transformed into a pedigree consisting of integers and missing values denoted by '-998'. CAUTION: this should not be set to TRUE without careful thought.

Based on code from the MCMCglmm package

Value

The pedigree, where individuals are now numbered from 1 to n and unknown parents are assigned a value of '-998'.

See Also

MCMCglmm

Description

Given a model object from asreml and a range of estimates of the parameter, the function will supply the likelihood ratio test statistic for the comparison of the full model to one where the parameter of interest is constrained.

Usage

parConstrainFun(x, parameters, full, fm2, comp, G)

Arguments

x section of all parameter values to analyze
parameters a value for which the log Likelihood of a model is to be calculated
full the full model asreml object
fm2 starting values for the full model
comp which variance component to constrain
G logical indicating if the component is part of the G structure

Details

Used internally in the proLik function to call constrainFun
plot.proLik

Author(s)

<matthewwolak@gmail.com>

See Also

See Also proLik, constrainFun

plot.proLik       Plot profile Likelihood

Description

This function graphically depicts the output from a proLik object.

Usage

## S3 method for class 'proLik'
plot(x, CL = TRUE, alpha = 0.05, type = "l", ...)

Arguments

x         the output from proLik
CL        a logical indicating whether a line representing the Confidence Limit is to be drawn
alpha     if CL is TRUE, the confidence level at which CL is to be drawn
type      the type of plot to be generated, see arguments to plot
...       other arguments to plot

Author(s)

<matthewwolak@gmail.com>

See Also

proLik

Examples

##See examples in \code{proLik}
**proLik** Estimates the profile likelihood of a random effect

**Description**

When a mixed model is run in ASReml-R, this function can estimate the profile likelihood of variance components from the output.

**Usage**

```r
proLik(full.model, component, G = TRUE, negative = FALSE,
       nsample.units = 3, nse = 3, alpha = 0.05, tolerance = 0.001,
       parallel = FALSE, ncores = getOption("mc.cores", 2L))
```

**Arguments**

- `full.model` An `asreml` model object
- `component` A character indicating for which variance component the profile likelihood will be constructed. Must be an object in `full.model$gammas`.
- `G` Logical indicating whether component is part of the G structure. If the component is part of the R structure, `G = FALSE`.
- `negative` Logical indicating whether or not the component can take on a negative value (i.e., a covariance)
- `nsample.units` Number of sample units to be used in constructing the area between the confidence limits for the profile likelihood
- `nse` Number of standard errors on either side of the estimate, over which the confidence limits should be evaluated.
- `alpha` The critical value for determining the Confidence Interval
- `tolerance` Acceptable distance, between actual sample values and interpolated values, for determining the upper and lower limits of the Confidence Interval. Actual sample points will be no more than this distance from the true value of the estimate.
- `parallel` A logical indicating whether or not parallel processing will be used. Note, may only be available for Mac and Linux operating systems.
- `ncores` Argument indicating number of cpu units to use. Default is all available.

**Details**

For the `negative` argument, this should be used if the profile likelihood of a covariance component is to be constructed.

`parallel = TRUE` should only be used on Linux or Mac OSes (i.e., not Windows).

The function uses the `optimize` function to obtain the approximate confidence limits. Therefore, `nse` should be carefully thought about beforehand when running the function. Increasing this value will ensure the confidence limits are contained by the search space, but at a cost to time.
If `negative` is FALSE, and the lower bound of the sampling interval extends beyond zero, this will instead be set to effectively zero.

Obtaining the profile likelihood for residual variances may necessitate explicitly specifying the R structure of the ASReml model. See example below.

**Value**

- **lambdas**: negative log Likelihood ratio test statistic. Estimated from the log Likelihood of the `full.model` and the log Likelihood of the model with the component constrained to a value in the sampling interval
- **var.estimates**: value along the sampling interval for which the component was constrained
- **UCL**: approximate Upper Confidence Limit
- **LCL**: approximate Lower Confidence Limit
- **component**: the component for which the profile likelihood surface has been constructed

**Warning**

May be unfeasible to estimate profile likelihoods for complex models with many variance components

**Author(s)**

`<matthewwolak@gmail.com>`

**See Also**

`aiFun`

**Examples**

```r
## Not run:
library(asreml)
ginvA <- asreml.Ainverse(warcolak[, c(1,3,2)])$ginv
ginvD <- makeD(warcolak[,1:3])$listDinv
warcolak$IDD <- warcolak$ID
warcolak.mod <- asreml(trait1 ~ 1, random = ~ped(ID) + giv(IDD),
ginverse = list(ID = ginvA, IDD = ginvD), rcov = ~ idv(units), data = warcolak)
summary(warcolak.mod)$varcomp

profileA <- proLik(full.model = warcolak.mod, component = "ped(ID)!ped",
G = TRUE, negative = FALSE, nsample.units = 3, nse = 3)
profileA

profileD <- proLik(warcolak.mod, component = "giv(IDD).giv",
G = TRUE, negative = FALSE, nsample.units = 3, nse = 3)
profileD

profileE <- proLik(warcolak.mod, component = "R!units.var", G = FALSE, negative = FALSE)

x11(w = 6, h = 8)
par(mfrow = c(3,1))
plot.proLik(profileA)
plot.proLik(profileD)
plot.proLik(profileE)
```
sm2list

Converts a sparse matrix into a three column format.

Description

From a sparse matrix object, the three column, row ordered lower triangle of non-zero elements is created. Mostly used within other functions (i.e., makeD)

Usage

sm2list(A, rownames = NULL, colnames = c("row", "column", "A"))

Arguments

A: a sparse matrix

rownames: a list of rownames from the 'A' matrix.

colnames: the columns will be labelled however they are entered in this character vector

Details

The sparse matrix and three column format must fit CERTAIN assumptions about row/column sorting and lower/upper triangle matrix.

Adapted from a function in the MCMCglmm package

Value

returns the list form of the sparse matrix as a data.frame

See Also

MCMCglmm
varTrans
Transforms ASReml-R gamma variances to component scale

Description
The inverse of the Average Information matrix in an ASReml-R object produces the sampling variances of the random effects on the gamma scale. This function scales these variances to the original component scale. This allows for Confidence Intervals to be constructed about the variance component estimates.

Usage
varTrans(asr.object)

Arguments
asr.object Object from a call to asreml

Value
Returns a numeric vector of variances for each variance component in an ASReml-R model.

Author(s)
<matthewwolak@gmail.com>

Examples
## Not run:
library(asreml)
ginvA <- asreml.Ainverse(warcolak)$ginv
ginvD <- makeD(warcolak[,1:3])$listDinv
warcolak$IDD <- warcolak$ID
warcolak.mod <- asreml(trait1 ~ 1, random = ~ped(ID) + giv(IDD),
ginverse = list(ID = ginvA, IDD = ginvD), data = warcolak)
summary(warcolak.mod)$varcomp
varTrans(warcolak.mod)

## End(Not run)
Description

A two trait example pedigree from the three generation breeding design of Fairbairn & Roff (2006) with two un-correlated traits.

Usage

data(warcolak)

Format

A data frame with 5400 observations on the following 5 variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>a factor specifying 5400 unique individual IDs</td>
</tr>
<tr>
<td>Dam</td>
<td>a factor specifying the unique ID for each dam</td>
</tr>
<tr>
<td>Sire</td>
<td>a factor specifying the unique ID for each sire</td>
</tr>
<tr>
<td>sex</td>
<td>a vector specifying '0' if the individual is a male and '1' if it is a female</td>
</tr>
<tr>
<td>trait1</td>
<td>a numeric vector of phenotypic values</td>
</tr>
<tr>
<td>trait2</td>
<td>a numeric vector of phenotypic values</td>
</tr>
</tbody>
</table>

Details

Unique sets of relatives are specified (Fairbairn & Roff, 2006) for a three generation breeding design. Each set contains 72 individuals. This pedigree reflects an experiment which produces 75 of these basic sets from Fairbairn & Roff’s design.

The dataset was simulated to have a two un-correlated traits with different genetic architectures. The additive genetic, dominance genetic, and environmental (or residual) variances for both trait1 and trait2 are specified to be 0.4, 0.3, & 0.3, respectively. However, the additive genetic variance for trait2 can be further decomposed to autosomal additive genetic variance (0.3) and X-linked additive genetic variance (0.1). These variances were drawn from multivariate random normal distributions [e.g., additive effects: $N ~ (0, A \cdot V_a)$] with means of zero and variances equal to the product of the desired and the relatedness (or incidence) matrix. Because of this, the actual variance in random effects will vary slightly from the amount specified in the simulation.

References

Index

*Topic datasets
   FG90, 7
   Mrode2, 17
   Mrode9, 18
   warcolak, 25

aiCI, 3
aiFun, 3, 4, 22

constrainFun, 6

DFC, 7

FG90, 7
findDFC, 7, 8

genAssign, 9

makeA, 9, 10, 11, 14
makeAA, 10, 13, 14
makeD, 9, 10, 11, 13–15
makeDomEpi, 12
makeDsim, 12, 14
makeS, 16
MCMCglmm, 19, 23
Mrode2, 17
Mrode9, 18

nadiv (nadiv-package), 2
nadiv-package, 2
numPed, 18

parConstrainFun, 19
plot (plot.proLik), 20
plot.proLik, 20
proLik, 3, 21

sm2list, 23

varTrans, 3, 24

warcolak, 25

26