Package 'nadiv'

September 22, 2016

Type Package

Title (Non)Additive Genetic Relatedness Matrices

Version 2.14.3.1

Description Constructs (non)additive genetic relationship matrices, and their inverses, from a pedigree to be used in linear mixed effect models (A.K.A. the 'animal model'). Also includes other functions to facilitate the use of animal models. Some functions have been created to be used in conjunction with the R package 'asreml' for the 'ASReml' software, which can be obtained upon purchase from 'VSN' international (http://www.vsni.co.uk/software/asreml).

Date 2016-09-22

URL http://github.com/matthewwolak/nadiv

BugReports http://github.com/matthewwolak/nadiv/issues

Depends Matrix

Suggests parallel

Enhances MCMCglmm, asreml

Imports graphics, methods, stats

License GPL (>= 2)

LazyData yes

NeedsCompilation yes

Author Matthew Wolak [cre, aut]

Maintainer Matthew Wolak <matthewwolak@gmail.com>

Repository CRAN

Date/Publication 2016-09-22 22:49:49

R topics documented:

`nadiv-package` *(Non)Additive Genetic Relatedness Matrices in Animal Model Analyses*

Description

Constructs (non)additive genetic relationship matrices, and their inverses, from a pedigree to be used in linear mixed effect models (A.K.A. the 'animal model'). Also includes other functions to facilitate the use of animal models. Some functions have been created to be used in conjunction with the R package for 'ASReml' software, which can be obtained upon purchase from 'VSN' international (http://www.vsni.co.uk/software/asreml).

Details

Package:	nadiv
Type:	Package
Version:	2.14.3.1
Date:	2016-09-22
URL:	http://github.com/matthewwolak/nadiv
BugReports:	http://github.com/matthewwolak/nadiv/issues
License:	GPL (>=2)
LazyLoad:	yes

Author(s)

Matthew Wolak <matthewwolak@gmail.com>

`aic` *Akaike Information Criterion*

Description

Calculates AIC/AICc values, AIC differences, Likelihood of models, and model probabilities.

Usage

```
aic(logLik, fp, n = NULL)
```

Arguments

`logLik`	A vector of model log-Likelihoods
`fp`	A vector containing the numbers of free parameters of each model included in the logLik vector
`n`	An optional vector of sample sizes for each model. Used to calculate AICc (small sample un-biased AIC).

Details

Calculations and notation follows chapter 2 of Burnham and Anderson (2002).

Value

AIC	vector containing AIC/AICc (depending on value of n
delta_AIC	vector containing AIC differences from the minimum AIC(c)
AIClik	vector containing likelihoods for each model, given the data. Represents the relative strength of evidence for each model.
w	Akaike weights.

Author(s)

<matthewwolak@gmail.com>

References

Burnham, K.P. and D.R. Anderson. 2002. Model Selection and Multimodel Inference. A Practical Information-Theoretic Approach, 2nd edn. Springer, New York.

Examples

```
aic(c(-3139.076, -3136.784, -3140.879, -3152.432), c(8, 7, 8, 5))
```

---

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

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

## End(Not run)
```

---

aiFun *Sampling (co)variances*

Description

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

Usage

```
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 (co)variance component in the random portion of 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.

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 (co)variance components 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

Gilmour, A.R., Gogel, B.J., Cullis, B.R., & Thompson, R. 2009. ASReml User Guide Release 3.0. VSN International Ltd., Hemel Hempstead, UK.

Examples

```
  ## Not run:
    library(asreml)
    ginvA <- asreml.Ainverse(warcolak)$ginv
    ginvD <- makeD(warcolak[,1:3])$listDinv
    warcolak$IDD <- warcolak$ID
    warcolak.mod <- asreml(trait1 ~ sex, 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(7.3075921, 7.0635161, 12.3423380, 1.9539486, 2.7586340, 0.6626111)
  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

---

drfx *Simulated design random effects*

Description

This function simulates effects for random terms in a linear mixed model based on design matrices. The intended purpose is for simulating environmental effects from a pedigree.

Usage

```
drfx(G, fac, dataf, ...)
```

Arguments

G	The variance-covariance matrix to model the effects after
fac	A character indicating the factor in dataf with which to construct the design matrix
dataf	A dataframe with fac in it
...	Arguments to be passed to the internal use of grfx

Details

If G = x, where 'x' is a single number, then 'x' should still be specified as a 1-by-1 matrix (e.g., matrix(x)). Note, the G-matrix should never have a structure which produces a correlation exactly equal to 1 or -1. Instead, covariances should be specified so as to create a correlation of slightly less than (greater than) 1 (-1). For example: 0.9999 or -0.9999.

Value

fx	A matrix with 'd' columns of random effects
Z	A design matrix (of the format 'Matrix') from which the random effects in fx were assigned

Author(s)

<matthewwolak@gmail.com>

See Also

grfx

Examples

```
# Create maternal common environment effects for 2 traits
# with perfectly correlated effects
  Gmat <- matrix(c(10, 7.071, 7.071, 5), 2, 2)
  cfx <- drfx(G = Gmat, fac = "Dam", dataf = warcolak[1:200, ])
```

F2009 *Pedigree, adapted from Fikse (2009)*

Description

An example pedigree with genetic groups and fuzzy classification of genetic groups.

Usage

```
data("F2009")
```

Format

A data frame with 16 observations on the following 11 variables.

`id` a factor with levels indicating the unique individuals (including phantom parents) and genetic groups

`dam` a factor of observed maternal identities

`sire` a factor vector of observed paternal identities

`damGG` a factor of maternal identities with genetic groups inserted instead of `NA`

`sireGG` a factor of paternal identities with genetic groups inserted instead of `NA`

`phantomDam` a factor of maternal identities with phantom parents inserted instead of `NA`

`phantomSire` a factor of paternal identities with phantom parents inserted instead of `NA`

`group` a factor of genetic groups to which each phantom parent belongs

`g1` a numeric vector with probabilities of group `g1` membership for each phantom parent

`g2` a numeric vector with probabilities of group `g2` membership for each phantom parent

`g3` a numeric vector with probabilities of group `g3` membership for each phantom parent

Source

Fikse, F. 2009. Fuzzy classification of phantom parent groups in an animal model. Genetics Selection Evolution 41:42.

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`

Source

Fernando, R.L. & M. Grossman. 1990. Genetic evaluation with autosomal and X-chromosomal inheritance. Theoretical and Applied Genetics 80:75-80.

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, exact = FALSE, parallel = FALSE,
ncores = getOption("mc.cores", 2L))
```

Arguments

`pedigree`	A pedigree with columns organized: ID, Dam, Sire
`exact`	A logical statement indicating if individuals who are exactly double first cousins are to be identified
`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

When exact = TRUE, only those individuals whose grandparents are completely unrelated will be identified as double first cousins. When exact = FALSE, as long as the parents of individuals i and j are two sets of siblings (i.e., either sires full brothers/dams full sisters or two pairs of opposite sex full sibs) then i and j will be considered double first cousins. In the event where the grandparents of i and j are also related, exact = FALSE will still consider i and j full sibs, even though genetically they will be more related than exact = TRUE double first cousins.

`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 indidivuals that are related as double first cousins

`DFC`
: gives the list of IDs, as characters, for all the pairs of individuals that are related as 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>

`founderLine` *Identifies the matriline or patriline to which each individual in a pedigree belongs*

Description

For every individual in a pedigree, the function identifies either the one female or male ancestor that is a founder (defined here as an individual identity in the pedigree for which both dam and sire information are missing).

Usage

```
founderLine(pedigree, sex)
```

Arguments

`pedigree`
: A pedigree where the columns are ordered ID, Dam, Sire, Sex

`sex`
: Character indicating the column name in pedigree identifying either the dam (for matriline) or sire (for patriline) identities

Details

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

Individuals with a missing parent for the column identified by the 'sex' argument are assigned themselves as their founder line. Thus, the definition of the founder population from a given pedigree is simply all individuals with missing parents (and in this case just a single missing parent classifies an individual as a founder).

Value

A vector of length equal to the number of rows in the pedigree

Author(s)

<matthewwolak@gmail.com>

Examples

```
founderLine(FG90, sex = "dam")  # matriline from this example pedigree

#Create random pedigree, tracking the matrilines
## Then compare with founderLine() output
K <- 8  # No. individuals per generation (KEEP and even number)
gen <- 10 # No. of generations
datArr <- array(NA, dim = c(K, 5, gen))
dimnames(datArr) <- list(NULL, c("id", "dam", "sire", "sex", "matriline"), NULL)
# initialize the data array
datArr[, "id", ] <- seq(K*gen)
datArr[, "sex", ] <- c(1, 2)
femRow <- which(datArr[, "sex", 1] == 2) # assume this is same each generation
# (Why K should always be an even number)
datArr[femRow, "matriline", 1] <- femRow
# males have overlapping generations, BUT females DO NOT
for(g in 2:gen){
 datArr[, "sire", g] <- sample(c(datArr[femRow-1, "id", 1:(g-1)]), size = K, replace = TRUE)
  gdams <- sample(femRow, size = K, replace = TRUE)
  datArr[, c("dam", "matriline"), g] <- datArr[gdams, c("id", "matriline"), g-1]
}
ped <- data.frame(apply(datArr, MARGIN = 2, FUN = function(x){x}))
nrow(ped)
#Now run founderLine() and compare
ped$line <- founderLine(ped, sex = "dam")
stopifnot(identical(ped$matriline, ped$line),
sum(ped$matriline-ped$line, na.rm = TRUE) == 0,
range(ped$matriline-ped$line, na.rm = TRUE) == 0)
```

genAssign *Generation assignment*

Description

Given a pedigree, the function assigns the generation number to which each individual belongs.

Usage

```
genAssign(pedigree)
```

Arguments

pedigree	A pedigree with columns organized: ID, Dam, Sire

Details

0 is the base population.

Migrants, or any individuals where both parens are unknown, are assigned to generation zero. If parents of an individual are from two different generations (e.g., mom = 0 and dad = 1), the individual is assigned to the generation following the greater of the two parents (e.g., 2 in this example).

Value

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

ggcontrib *Genetic group contribution*

Description

Calculates the genomic contribution each genetic group makes to every individual in a pedigree

Usage

```
ggcontrib(pedigree, ggroups = NULL, fuzz = NULL, output = "matrix")
```

Arguments

pedigree	A pedigree where the columns are ordered ID, Dam, Sire
ggroups	An optional vector of either: genetic group assignment for every individual or just the unique genetic groups. fuzz must be NULL if an object is supplied to the ggroups argument.
fuzz	A matrix containing the fuzzy classification of phantom parents into genetic groups. ggroups must be NULL if an object is supplied to the fuzz argument.
output	Format for the output

Details

The specification of genetic groups is done in one of two approaches, either using fuzzy classification or not.

Fuzzy classification enables phantom parents to be assigned to (potentially) more than one genetic group (Fikse 2009). This method requires unique phantom parent identities to be included in the pedigree for all observed individuals with unknown parents. For 'p' phantom parents, 'p' identities should be listed as individuals in the first 'p' rows of the pedigree and these should be the only individuals in the pedigree with missing values in their Dam and Sire columns (denoted by either 'NA', '0', or '*'). The matrix supplied to the `fuzz` argument should have 'p' rows (one for each phantom parent) and 'r' columns, where 'r' is the number of genetic groups.

Non-fuzzy classification can handle the specification of genetic groups in three formats:

(1) similar to ASReml's format for specifying genetic groups, the first 'r' rows of the pedigree (given to the `pedigree` argument) contain the label for each genetic group in the ID column and indicate missing values for the Dam and Sire columns (denoted by either 'NA', '0', or '*'). No object is supplied to the `ggroups` argument. All individuals in the pedigree must then have one of the 'r' genetic groups as parent(s) for each unknown parent. Note, a warning message indicating `In numPed(pedigree): Dams appearing as Sires` is expected, since the dam and sire can be the same for all individuals in the pedigree composing the base population of a genetic group.

(2) similar to Jarrod Hadfield's `rbv` function arguments in the `MCMCglmm` package, for a pedigree of dimension i x 3 (given to the `pedigree` argument), where 'i' is the total number of individuals in the pedigree, a similar vector of length 'i' is given to the `ggroups` argument. This vector lists either the genetic group to which each individual's phantom parents belong or NA if the individual is not to be considered part of one of the base populations (genetic groups). NOTE, this approach does not allow phantom dams and phantom sires of a given individual to be from different genetic groups.

(3) similar to DMU's format for specifying genetic groups. For a pedigree of dimension i x 3 (given to the `pedigree` argument), where 'i' is the total number of individuals in the pedigree, instead of missing values for a parent, one of the 'r' genetic groups is specified. A character vector of length 'r' with unique genetic group labels is given to the `ggroups` argument. Note, that all individuals with a missing parent should have a genetic group substituted instead of the missing value symbol (i.e., either 'NA', '0', or '*').

Value

Returns i x r genetic group contributions to all 'i' individuals from each of the 'r' genetic groups. Default output is a matrix (dense), but this format can be changed (e.g., "dgCMatrix" for a sparse matrix).

Author(s)

<matthewwolak@gmail.com>

References

Fikse, F. 2009. Fuzzy classification of phantom parent groups in an animal model. Genetics, Selection, Evolution. 41:42.

Quaas, R.L. 1988. Additive genetic model with groups and relationships. Journal of Dairy Science. 71:1338-1345.

Examples

```
# Use the pedigree from Quaas 1988 (See `data(Q1988)`)
##########################
# Fuzzy classification
  ## Fuzzy classification with complete assignment to one group
    Q1988fuzz <- Q1988[-c(1:2), c("id", "phantomDam", "phantomSire")]
    Qfnull <- matrix(c(1,0,0,1,0, 0,1,1,0,1), nrow = 5, ncol = 2,
dimnames = list(letters[1:5], c("g1", "g2")))
    (Qfuzznull <- ggcontrib(Q1988fuzz, fuzz = Qfnull))

    ## Should be identical to the non-fuzzy classification output
    # format (1) from above
      (Q <- ggcontrib(Q1988[-c(3:7), c(1,4,5)]))
    stopifnot(Qfuzznull == Q)

  ## Fuzzy classification with arbitrary assignments
    Qf <- matrix(c(1,0,0.5,0.5,0.5, 0,1,0.5,0.5,0.5), nrow = 5, ncol = 2,
dimnames = list(letters[1:5], c("g1", "g2")))
    (Qfuzz <- ggcontrib(Q1988fuzz, fuzz = Qf))

  ## Using the pedigree and fuzzy classification in Fikse (2009)
    F2009fuzz <- data.frame(id = c(letters[1:7], LETTERS[1:6]),
dam = c(rep(NA, 7), "a", "c", "e", "A", "C", "D"),
sire = c(rep(NA, 7), "b", "d", "f", "B", "g", "E"))
    Ff <- matrix(c(1,0,1,0,0,0,0.2,
0,1,0,0.6,0,0.3,0.4,
0,0,0,0.4,1,0.7,0.4),
nrow = 7, ncol = 3,
dimnames = list(letters[1:7], paste0("g", 1:3)))
    # Actual Q matrix printed in Fikse (2009)
      Fikse2009Q <- matrix(c(0.5,0.5,0,0.5,0.1,0.3,
0.5,0.3,0.15,0.4,0.275,0.3375,
0,0.2,0.85,0.1,0.625,0.3625),
nrow = 6, ncol = 3,
dimnames = list(LETTERS[1:6], paste0("g", seq(3))))

    Ffuzz <- ggcontrib(F2009fuzz, fuzz = Ff)
      (diffFfuzz <- Ffuzz - Fikse2009Q)
      # Encountering some rounding error
      stopifnot(length((drop0(diffFfuzz, tol = 1e-12))@x) == 0)


##########################
# Non-fuzzy classification
  # format (1) from above
    Q1 <- Q1988[-c(3:7), c(1,4,5)]
    (gg1 <- ggcontrib(Q1, ggroups = NULL)) # note the warning message which is typical

  # format (2) from above
    Q2 <- Q1988[-c(1:7), 1:3]
    # arbitrarily assign individuals genetic groups for unknown parents
    ## Means gg2 is NOT comparable to gg1 or gg3!
```

```
  ggvec.in <- c("g1", "g2", "g1", NA)
  (gg2 <- ggcontrib(Q2, ggroups = ggvec.in))

# format (3) from above
  Q3 <- Q1988[-c(1:7), c(1,4,5)]
  gg3 <- ggcontrib(Q3, ggroups = c("g1", "g2"))

stopifnot(gg1 == gg3)
```

`ggTutorial` *Simulated dataset used to analyze data with genetic group animal models*

Description

The dataset was simulated using the `simGG` function so that the pedigree contains a base population comprised of founders and non-founder immigrants. These data are then used in the main manuscript and tutorials accompanying Wolak & Reid (unpublished manuscript).

Usage

```
data("ggTutorial")
```

Format

A data frame with 6000 observations on the following 10 variables.

`id` an integer vector specifying the 6000 unique individual identities

`dam` an integer vector specifying the unique dam for each individual

`sire` an integer vector specifying the unique sire for each individual

`parAvgU` a numeric vector of the average autosomal total additive genetic effects (`u`) of each individual's parents

`mendel` a numeric vector of the Mendelian sampling deviations from `parAvgU` autosomal total additive genetic effects that is unique to each individual

`u` a numeric vector of the total autosomal additive genetic effects underlying `p`

`r` a numeric vector of the residual (environmental) effects underlying `p`

`p` a numeric vector of phenotypic values

`is` an integer vector with `0` for individuals born in the focal population and `1` for individuals born outside of the focal population, but immigrated

`gen` an integer vector specifying the generation in which each individual was born

Details

The dataset was simulated as described in the 'examples' section using the simGG function. Full details of the function and dataset can be found in Wolak & Reid (unpublished manuscript).

The data.frame contains 6000 individuals across 15 generations. In each generation, the carrying capacity is limited to 400 individuals, the number of mating pairs limited to 200 pairs, and 40 immigrants per generation arrive starting in the second generation.

The breeding values of the founders are drawn from a normal distribution with an expected mean of 0 and a variance of 1. The breeding values of all immigrants are drawn from a normal distribution with an expected mean of 3 and variance of 1. Consequently, the expected difference between mean breeding values in the founders and immigrants is 3. All individuals are assigned a residual (environmental) deviation that is drawn from a normal distribution with an expected mean of 0 and variance of 1.

Source

Wolak, M.E. & J.M. Reid. Unpublished manuscript. Accounting for genetic differences among unknown parents in microevolutionary studies: How to include genetic groups in quantitative genetic animal models.

Examples

```
## Not run:
 rm(list = ls())
 set.seed(102)     #<-- seed value used originally
 library(nadiv)
 # create data using `simGG()`
 ggTutorial <- simGG(K = 400, pairs = 200, noff = 4, g = 15,
   nimm = 40, nimmG = seq(2, g-1, 1),    # nimmG default value
   VAf = 1, VAi = 1, VRf = 1, VRi = 1,    # all default values
   mup = 20, muf = 0, mui = 3, murf = 0, muri = 0, # mup and mui non-default values
   d_bvf = 0, d_bvi = 0, d_rf = 0, d_ri = 0)     # all default values

## End(Not run)
```

---

grfx *Simulated genetic random effects*

Description

This function simulates effects for random terms in a linear mixed model based on relatedness matrices. The intended purpose is for simulating genetic and environmental effects from a pedigree.

Usage

```
grfx(n, G, incidence = NULL, saveIncidence = FALSE,
output = "matrix", stdnorms = NULL, warn = TRUE)
```

Arguments

n	The number of individuals for which to simulate effects
G	The variance-covariance matrix to model the effects after
incidence	The covariance structure of the 'n' individuals
saveIncidence	A logical or NULL, indicating if the cholesky decomposition of the incidence matrix should be saved/retrieved to the global environment. SEE details below!!
output	Format for the output
stdnorms	Standard normal deviates to use
warn	Should a warning be produced when `incidence = NULL` indicating that a previous incidence matrix is being used if one exists, otherwise an Identity matrix

Details

The total number of effects simulated will be n*d, where d is the number of columns in the 'G' matrix. The standard normal deviates can be supplied instead of generated within the function when `stdnorms != NULL`. The length of this vector must be `n*nrow(G)`.

Supplied incidence matrices should be n-by-n symmetric matrices. For simulated random effects using design matrices, see `drfx`. If no incidence matrix is supplied, `incidence = NULL`, the function first checks the environment to see if anything with the name 'nadiv_prev_Mincidence' exists when checking `ls()`. If so, this saved version is used with a warning. Otherwise the Identity matrix is used, which assumes that all 'n' random effects are independently and identically distributed (default to Identity matrix).

BE CAREFUL with `saveIncidence = TRUE` as this will save the incidence matrix outside of the function environment so as to be accessed within the function at a later call. This can be useful for Monte Carlo simulation, to avoid performing the cholesky decomposition on a large matrix at each iteration. Setting `warn = FALSE` will suppress the warnings that this is occuring. DO NOT turn this warning off unless you are sure which incidence matrix will be used by `grfx`.

If G = x, where 'x' is a single number, then 'x' should still be specified as a 1-by-1 matrix (e.g., `matrix(x)`). Note, the G-matrix should never have a structure which produces a correlation exactly equal to 1 or -1. Instead, covariances should be specified so as to create a correlation of slightly less than (greater than) 1 (-1). For example: 0.9999 or -0.9999.

Value

The random effects coerced to be in the format specified by output. The default is a "matrix".

Author(s)

<matthewwolak@gmail.com>

See Also

`MCMCglmm`, `drfx`, `makeA`, `makeAA`, `makeD`, `makeDomEpi`, `makeDsim`, `makeS`

Examples

```
# Create additive genetic breeding values for 2 uncorrelated traits
# with different additive genetic variances
  A <- makeA(warcolak[1:200, 1:3])
  Gmat <- matrix(c(20, 0, 0, 10), 2, 2)
  breedingValues <- grfx(n = 200, G = Gmat, incidence = A, saveIncidence = FALSE)

 # Now with a user supplied set of standard normal deviates
  snorms <- rnorm(nrow(warcolak[1:200,]) * ncol(Gmat))
  breedingValues2a <- grfx(n = 200, G = Gmat, incidence = A, stdnorms = snorms)
  breedingValues2b <- grfx(n = 200, G = Gmat, incidence = A, stdnorms = snorms)
  identical(breedingValues2a, breedingValues2b)  # TRUE
  var(breedingValues2a)
  var(breedingValues2b)
```

LRTest *log-Likelihood Ratio Test*

Description

Test the null hypothesis that the two models fit the data equally well.

Usage

```
LRTest(full, reduced, df = 1, boundaryCorrection = FALSE)
```

Arguments

full	A numeric variable indicating the log-likelihood of the full model
reduced	A numeric variable indicating the log-likelihood of the reduced model
df	The number of degrees of freedom to use, representing the difference between the full and reduced model in the number of parameters estimated
boundaryCorrection	A logical argument indicating whether a boundary correction under one degree of freedom should be included. If the parameter that is dropped from the reduced model is estimated at the boundary of its parameter space in the full model, the boundary correction is often required. See Details for more.

Details

Boundary correction should be applied if the parameter that is dropped from the full model was on the boundary of its parameter space. In this instance, the distribution of the log-likelihood ratio test statistic is approximated by a mix of chi-square distributions (Self and Liang 1987). A TRUE value will implement the boundary correction for a one degree of freedom test. This is equivalent to halving the p-value from a test using a chi-square distribution with one degree of freedom (Dominicus et al. 2006).

Currently, the test assumes that both log-likelihoods are negative or both are positive and will stop if they are of opposite sign. The interpretation is that the model with a greater negative log-likelihood (closer to zero) or greater positive log-likelihood provides a better fit to the data.

Value

`lambda`	a numeric log-likelihood ratio test statistic
`Pval`	a numeric p-value given the `lambda` tested against a chi-squared distribution with the number of degrees of freedom as specified. May have had a boundary correction applied.
`corrected.Pval`	a logical indicating if the p-value was derived using a boundary correction. See `Details`

Author(s)

<matthewwolak@gmail.com>

References

Self, S. G., and K. Y. Liang. 1987. Asymptotic properties of maximum likelihood estimators and likelihood ratio tests under nonstandard conditions. Journal of the American Statistical Association 82:605-610.

Dominicus, A., A. Skrondal, H. K. Gjessing, N. L. Pedersen, and J. Palmgren. 2006. Likelihood ratio tests in behavioral genetics: problems and solutions. Behavior Genetics 36:331-340.

See Also

`constrainFun`

Examples

```
# No boundary correction
(noBC <- LRTest(full = -2254.148, reduced = -2258.210,
df = 1, boundaryCorrection = FALSE))
# No boundary correction
(withBC <- LRTest(full = -2254.148, reduced = -2258.210,
df = 1, boundaryCorrection = TRUE))
stopifnot(noBC$Pval == 2*withBC$Pval)
```

`makeA` *Creates the additive genetic relationship matrix*

Description

This returns the additive relationship matrix in sparse matrix format.

Usage

```
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.

See function makeAinv for directly obtaining the inverse of the additive genetic relationship matrix.

Value

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

Author(s)

<matthewwolak@gmail.com>

See Also

makeD, makeS

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>

See Also

makeA

Examples

```
makeAA(Mrode2)
```

---

makeAinv *Creates the inverse additive genetic relationship matrix*

Description

This returns the inverse of the numerator relationship matrix (inverse additive genetic relatedness matrix). It can also be used to obtain coefficients of inbreeding for the pedigreed population.

Usage

```
makeAinv(pedigree, f = NULL, ggroups = NULL, fuzz = NULL,
      gOnTop = FALSE, det = FALSE, ...)
## Default S3 method:
makeAinv(pedigree, f = NULL, ggroups = NULL, fuzz = NULL,
      gOnTop = FALSE, det = FALSE, ...)
## S3 method for class 'fuzzy'
makeAinv(pedigree, f = NULL, ggroups = NULL, fuzz,
      gOnTop = FALSE, det = FALSE, ...)
```

Arguments

pedigree	A pedigree where the columns are ordered ID, Dam, Sire
f	A numeric indicating the level of inbreeding. See Details
ggroups	Either a vector with the unique name of each genetic group, or a numeric indicating the number of unique genetic groups. See Details for different ways to specify. Note, if NULL then the regular A-inverse will be constructed. Also, must be NULL if fuzz is non-NULL.
fuzz	A matrix containing the fuzzy classification of phantom parents into genetic groups. See Details.
gOnTop	A logical indicating if (when including genetic groups) the A-inverse should be constructed with the 'g' genetic groups located in the first 'g' rows and columns if TRUE, else the 'g' genetic groups are located in the last 'g' rows and columns of A-inverse
det	Logical, indicating if the (log) determinant of the A matrix should be returned
...	Arguments to be passed to methods

Details

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

The function implements an adaptation of the Meuwissen and Luo (1992) algorithm (particularly, forllowing the description of the algorithm in Mrode 2005) with some code borrowed from the `inverseA` function by Jarrod Hadfield in the `MCMCglmm` package. Further, providing a non-NULL argument to `ggroups` incorporates the Quaas (1988) algorithm for directly obtaining the augmented A-inverse matrix for genetic groups into Meuwissen and Luo's (1992) algorithm, thereby, considering inbreeding during the construction of the A-inverse. Further calculations needed for the algorithm to incorporate inbreeding and genetic groups follow the theory presented in VanRaden (1992).

At the moment, providing the inbreeding level of individuals or the base population has not been implemented. However, this argument is a placeholder for now.

Genetic groups can be incorporated into the A-inverse by providing a value to the `ggroups` argument. The value supplied to `ggroups` can either be (1) a single integer indicating the number of unique genetic groups or (2) a character vector containing the name for each genetic group. These are referred to as pedigree types "A" and "D", respectively, and further details follow below. (Type="A") the pedigree contains unique IDs for the 'g' genetic groups in the first 'g' lines of the pedigree. The dam and sire of the genetic group rows should contain missing values (e.g., NA, "0", or "*"). All individuals in the pedigree should then have one of the 'g' genetic groups instead of an unknown parent. (Type="D") the pedigree contains only individuals in the ID column (no genetic groups have an ID) and there should be no missing values for any dams or sires. Instead, individuals for whom the dam and/or sire is unknown should have one of the genetic groups identified in the vector supplied to `ggroups` as the dam or sire.

'Fuzzy classification' of genetic groups (Fikse 2009) can be implemented if a 'matrix' (of class `matrix` or `Matrix`) is supplied to the `fuzzy` argument. The fuzzy classification matrix must have row names matching all of the phantom parents in the pedigree and the column names must be present and specify the genetic groups. The fuzzy classification matrix essentially contains probability of group membership for each phantom parent. Therefore, each row should sum to 1. The

pedigree must have an identity in a unique row for every phantom parent and cannot have genetic groups as either identities (in the first column) or as dam or sire (second and third columns). Further, if fuzzy classification is desired, the function must specify `ggroups = NULL`.

When genetic groups (including the case of fuzzy classification of genetic groups) are included in the A-inverse matrix, the argument to `gOnTop` specifies if the genetic group elements in the A-inverse should occupy the top-left (`gOnTop = TRUE`) or bottom-right (`gOnTop = FALSE`) of the matrix. Depending on how the software implementing an animal model solves the mixed model equations, the equations for the genetic groups (and thus the elements in the augmented A-inverse) should be the first or last set of equations.

Value

`Ainv`	the inverse of the additive genetic relationship matrix in sparse matrix form
`listAinv`	the three column list of the non-zero elements for the inverse of the additive genetic relationship matrix with attributes `rowNames` and `geneticGroups`. `attr(*, "rowNames")` links the integer for rows/columns to the ID column from the pedigree. `attr(*, "geneticGroups")` is a two element vector with the first integer indicating how many genetic groups are included in the pedigree. This last attribute is necessary for some software programs to correctly specify the residual degrees of freedom when calculating the log-likelihood in a model that implicitly fits fixed genetic group effects.
`f`	the individual coefficients of inbreeding for each individual in the pedigree (matches the order of the first/ID column of the pedigree). If the pedigree contains 'g' genetic groups in the first 'g' rows, then the first 'g' elements of `f` are assigned 0. If the pedigree contains 'p' phantom parents in the first 'p' rows, then the first 'p' elements of `f` are assigned 0.
`logDet`	the log determinant of the A matrix

Author(s)

<matthewwolak@gmail.com>

References

Fikse, F. 2009. Fuzzy classification of phantom parent groups in an animal model. Genetics Selection Evolution 41:42.

Meuwissen, T.H.E & Luo, Z. 1992. Computing inbreeding coefficients in large populations. Genetics, Selection, Evolution. 24:305-313.

Mrode, R.A. 2005. Linear Models for the Prediction of Animal Breeding Values, 2nd ed. Cambridge, MA: CABI Publishing.

Quaas, R.L. 1988. Additive genetic model with groups and relationships. Journal of Dairy Science. 71:1338-1345.

VanRaden, P.M. 1992. Accounting for inbreeding and crossbreeding in genetic evaluation of large populations. Journal of Dairy Science. 75:3136-3144.

See Also

`makeAstarMult`, `makeA`

Examples

```
 ##  Without genetic groups  ##
 makeAinv(Mrode2)

 ##  With genetic groups  ##
  ## Type A
   typeAped <- Q1988[-c(3:7), c("id", "damGG", "sireGG")]
   AstarA <- makeAinv(typeAped, ggroups = 2, gOnTop = FALSE)$Ainv
  ## Type D
   typeDped <- Q1988[-c(1:7), c("id", "damGG", "sireGG")]
   AstarD <- makeAinv(typeDped, ggroups = c("g1", "g2"), gOnTop = FALSE)$Ainv
  stopifnot(identical(AstarA, AstarD))

  # Show that the augmented A-inverse with genetic groups
  # contains the normal A-inverse (i.e., without genetic groups)
   ## Augmented A-inverse with genetic groups
    ggAinv <- makeAinv(Mrode3[-c(1,2), c("calf", "damGG", "sireGG")],
ggroups = c("g1", "g2"), gOnTop = FALSE)$Ainv
    noggAinv <- makeAinv(Mrode3[-c(1,2), c("calf", "dam", "sire")],
ggroups = NULL)$Ainv
    # First 8 rows & columns of ggAinv are same as the A-inverse without genetic groups
    ggAinv[1:8, 1:8]
    noggAinv
   stopifnot(all.equal(ggAinv[1:8, 1:8], noggAinv))

 ##  With fuzzy classification of genetic groups  ##
  ## example in Fikse (2009)
  Fped <- F2009[-c(1:3), c("id", "phantomDam", "phantomSire")]
    Fped$id <- factor(Fped$id, levels = as.character(unique(Fped$id)))
  Ffuzz <- as.matrix(F2009[4:10, c("g1", "g2", "g3")])
    dimnames(Ffuzz)[[1]] <- as.character(F2009[4:10, 1])
  AstarF <- makeAinv(Fped, fuzz = Ffuzz, gOnTop = FALSE)$Ainv

  ## Show that A-inverse with fuzzy classification of genetic groups
  ### can be the same as genetic group A-inverse without fuzzy classification
  ### Create a 'null' fuzzy classification matrix for Q1988 pedigree
  QfuzzNull <- matrix(c(1,0,0,1,0, 0,1,1,0,1), nrow = 5, ncol = 2,
dimnames = list(letters[1:5], c("g1", "g2")))
  typeFped <- Q1988[-c(1:2), c("id", "phantomDam", "phantomSire")]
  AstarNullFuzzy <- makeAinv(typeFped, fuzz = QfuzzNull, gOnTop = FALSE)$Ainv
  # Same as above using either pedigree type 'A' or 'D'
  stopifnot(identical(AstarNullFuzzy, AstarA),
identical(AstarNullFuzzy, AstarD))
```

`makeAstarMult` *Creates the inverse additive genetic relationship matrix with genetic groups*

Description

This returns the invers of the additive genetic relationship matrix with genetic groups (A*). The matrix is set up through matrix multiplication of two sub-matrices instead of directly (as `makeAinv` does).

Usage

```
makeAstarMult(pedigree, ggroups, fuzz = NULL, gOnTop = FALSE)
```

Arguments

`pedigree`	A pedigree where the columns are ordered ID, Dam, Sire
`ggroups`	Either a vector with the unique name of each genetic group, or a numeric indicating the number of unique genetic groups. See Details for different ways to specify. Note, cannot be NULL.
`fuzz`	A matrix containing the fuzzy classification of individuals into genetic groups.
`gOnTop`	A logical indicating if the A-inverse should be constructed with the 'g' genetic groups located in the first 'g' rows and columns if TRUE, else the 'g' genetic groups are located in the last 'g' rows and columns of A-inverse

Details

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

The function implements the matrix multiplication, using sub-matrices `Q` and `A^-1`, as detailed in Quaas (1988, pp. 1342-1343).

Genetic groups can be incorporated into the A-inverse by providing a value to the `ggroups` argument. The value supplied to `ggroups` can either be (1) a single integer indicating the number of unique genetic groups or (2) a character vector containing the name for each genetic group. These are referred to as pedigree types "A" and "D", respectively, and further details follow below. (Type="A") the pedigree contains unique IDs for the 'g' genetic groups in the first 'g' lines of the pedigree. The dam and sire of the genetic group rows should contain missing values (e.g., NA, "0", or "*"). All individuals in the pedigree should then have one of the 'g' genetic groups instead of an unknown parent. (Type="D") the pedigree contains only individuals in the ID column (no genetic groups have an ID) and there should be no missing values for any dams or sires. Instead, individuals for whom the dam and/or sire is unknown should have one of the genetic groups identified in the vector supplied to `ggroups` as the dam or sire.

Fuzzy classification of genetic groups is implemented when `fuzz` is non-NULL.

The argument to `gOnTop` specifies if the elements in the A-inverse should come at the beggining (`gOnTop = TRUE`) or end (`gOnTop = FALSE`) of the matrix. Depending on how the software implementing an animal model solves the mixed model equations, the equations for the genetic groups (and thus the elements in the augmented A-inverse) should be the first or last set of equations.

See function `makeAinv` for directly obtaining the inverse of the additive genetic relationship matrix with genetic groups.

Value

Returns A*, or the inverse of the numerator relationship with groups, in sparse matrix form.

Author(s)

<matthewwolak@gmail.com>

References

Quaas, R.L. 1988. Additive genetic model with groups and relationships. Journal of Dairy Science. 71:1338-1345.

See Also

makeAinv, ggcontrib

Examples

```
 # Using the Q1988 dataset in nadiv
 ## assign a null fuzzy classification matrix
 QfuzzNull <- matrix(c(1,0,0,1,0, 0,1,1,0,1), nrow = 5, ncol = 2,
dimnames = list(letters[1:5], c("g1", "g2")))

 # Type A
 ## no fuzzy classification
  Astar_A <- makeAstarMult(Q1988[-c(3:7), c(1,4,5)], ggroups = 2)
 ## with fuzzy classification
  Astar_Afuzzy <- makeAstarMult(Q1988[, c(1, 6, 7)],
ggroups = 2, fuzz = QfuzzNull)

 # Type D
 ## no fuzzy classification
  Astar_D <- makeAstarMult(Q1988[-c(1:7), c(1, 4, 5)], ggroups = c("g1", "g2"))
 ## with fuzzy classification
  Astar_Dfuzzy <- makeAstarMult(Q1988[-c(1:2), c(1, 6, 7)],
ggroups = c("g1", "g2"), fuzz = QfuzzNull)


 # Obtain the matrix directly
 ## no fuzzy classification
 Astar_direct <- makeAinv(Q1988[-c(3:7), c(1,4,5)], ggroups = 2)$Ainv
 stopifnot(length(drop0(round(Astar_direct
- (Astar_A - Astar_Afuzzy)
- (Astar_D - Astar_Dfuzzy)
- Astar_direct, 10))@x) == 0)

 ## with fuzzy classification
 Astar_directF <- makeAinv(Q1988[-c(1:2), c(1, 6, 7)], fuzz = QfuzzNull)$Ainv
 stopifnot(length(drop0(round(Astar_directF
- (Astar_A - Astar_Afuzzy)
- (Astar_D - Astar_Dfuzzy)
- Astar_direct, 10))@x) == 0)
```

makeD *Creates the dominance genetic realationship matrix*

Description

Given a pedigree, the matrix of coefficients of fraternity are returned - the D matrix. Note, inbreeding is not directly incorporated into the calculation of the coefficients (see Details). 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. These coefficients are only approximations that assume no inbreeding. The algorithm used here, however, incorporates inbreeding into the calculation of coefficients of coancestry (using 'makeA()') that are used to calculate coefficients of fraternity. Similarly, the diagonals of the D matrix are corrected for inbreeding. Meaning, the diagonals of D are (1-f) so that the overall dominance genetic variance is equal to (1-f)V_D, where f is the coefficient of inbreeding and V_D is dominance genetic variance. This is interpreted as the amount of dominance genetic variance that would be expected if the allele frequencies in the inbred population were representative of a non-inbred, randomly mating population (Shaw et al. 1998; Wolak and Keller 2014). 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 package `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 number of cpus available 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, 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

Quaas, R.L. 1995. Fx algorithms. An unpublished note.

Lynch M., & Walsh, B. 1998. Genetics and Analysis of Quantitative Traits. Sinauer, Sunderland, Massachusetts.

Shaw, R.G., D.L. Byers, and F.H. Shaw. 1998. Genetic components of variation in Nemophila menziesii undergoing inbreeding: Morphology and flowering time. Genetics. 150:1649-1661.

Wolak, M.E. and L.F. Keller. 2014. Dominance genetic variance and inbreeding in natural populations. In Quantitative Genetics in the Wild, A. Charmantier, L.E.B. Kruuk, and D. Garant eds. Oxford University Press, pp. 104-127.

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,
invertD = 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.
`invertD`	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 seperate genetic relatedness inverse matrices that depend upon it. However, using the `output` and `invertD` 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 `invertD` 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

Author(s)

<matthewwolak@gmail.com>

See Also

makeA, makeD, makeAA

Examples

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

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

---

makeDsim — *Creates the dominance genetic relationship matrix through an iterative (simulation) process*

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.

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 approximate 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

Ovaskainen, O., Cano, J.M., & Merila, J. 2008. A Bayesian framework for comparative quantitative genetics. Proceedings of the Royal Society B 275, 669-678.

See Also

makeD

Examples

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

---

makeS *Creates the additive genetic relationship matrix for the shared sex chromosomes*

Description

The function returns the inverse of the additive relationship matrix in sparse matrix format for the sex chromosomes (e.g., either X or Z).

Usage

```
makeS(pedigree, heterogametic, DosageComp = c(NULL, "ngdc",
"hori", "hedo", "hoha", "hopi"), 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 of the pedigree
DosageComp	A character indicating which model of dosage compensation. If "NULL" then the "ngdc" 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 implementing the Meuwissen and Luo (1992) algorithm to directly construct inverse additive relationship matrices (borrowing code from Jarrod Hadfield's MCMCglmm function, `inverseA`) and using equations presented in Fernando & Grossman (1990; see Wolak et al. 2013). Additionally, the S-matrix itself can be constructed (although this takes much longer than computing S-inverse directly).

The choices of dosage compensation models are: no global dosage compensation ("ngdc"), random inactivation in the homogametic sex ("hori"), doubling of the single shared sex chromosome in the heterogametic sex ("hedo"), halving expression of both sex chromosomes in the homogametic sex ("hoha"), or inactivation of the paternal sex chromosome in the homogametic sex ("hopi").

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
`inbreeding`	the sex-linked inbreeding coefficients for all individuals in the pedigree
`v`	a vector of the Mendelian sampling variances for a sex-linked locus

Author(s)

<matthewwolak@gmail.com>

References

Wolak, M.E., D.A. Roff, and D.J. Fairbairn. in prep. The contribution of sex chromosomal additive genetic (co)variation to the phenotypic resemblance between relatives under alternative models of dosage compensation.

Fernando, R.L. & Grossman, M. 1990. Genetic evaluation with autosomal and X-chromosomal inheritance. Theoretical and Applied Genetics, 80:75-80.

Meuwissen, T.H.E. and Z. Luo. 1992. Computing inbreeding coefficients in large populations. Genetics, Seleciton, Evolution, 24:305-313.

Examples

```
makeS(FG90, heterogametic = "0", returnS = TRUE)
```

Mrode2 *Pedigree from Table 2.1 of Mrode (2005)*

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

Mrode, R.A. 2005. Linear Models for the Prediction of Animal Breeding Values, 2nd ed. Cambridge, MA: CABI Publishing.

Mrode3 *Pedigree, from chapter 3 of Mrode (2005)*

Description

An example pedigree with genetic groups and a data column

Usage

```
data(Mrode3)
```

Format

A data frame with 10 observations on the following 8 variables.

`calf` a factor with levels indicating the unique genetic groups and individuals

`dam` a numeric vector of maternal identities

`sire` a numeric vector of paternal identities

`damGG` a factor of maternal identities with genetic groups inserted instead of `NA`

`sireGG` a factor of paternal identities with genetic groups inserted instead of `NA`

`sex` a factor with levels `female` `male`

`WWG` a numeric vector of pre-weaning weight gain (kg) for five beef calves

Source

Mrode, R.A. 2005. Linear Models for the Prediction of Animal Breeding Values, 2nd ed. Cambridge, MA: CABI Publishing.

`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

Mrode, R.A. 2005. Linear Models for the Prediction of Animal Breeding Values, 2nd ed. Cambridge, MA: CABI Publishing.

`numPed` *Integer Format Pedigree*

Description

Conversion, checking, and row re-ordering of a pedigree in integer form of class 'numPed'.

Usage

```
  numPed(pedigree, check = TRUE)

  ## S3 method for class 'numPed'
is(x)

  ronPed(x, i, ...)
```

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, but see Details
x	A pedigree of class 'numPed'
i, ...	Index specifying elements to extract or replace: see [

Details

Missing parents (e.g., base population) should be denoted by either 'NA', '0', '-998', or '*'.

Individuals must appear in the ID column in rows preceeding where they appear in either the Dam or Sire column. See the prepPed function if this is not the case.

If pedigree inherits the class "numPed" (from a previous call to numPed()) and check = TRUE, the checks are skipped. If check = FALSE any pedigree will be transformed into a pedigree consisting of integers and missing values denoted by '-998'.

Based on code from the MCMCglmm package

Value

An S3 object of class "numPed" representing the pedigree, where individuals are now numbered from 1 to n and unknown parents are assigned a value of '-998'.

See Also

prepPed, MCMCglmm, [

Examples

```
(nPed <- numPed(Mrode2))
is(nPed)

# re-order and retain class 'numPed'
ronPed(nPed, order(nPed[, 2], nPed[, 3]))
is(nPed)
```

---

parConstrainFun *Function used in the* proLik *function 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

```
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

Author(s)

<matthewwolak@gmail.com>

See Also

See Also proLik, constrainFun

---

pcc *REML convergence checks*

Description

Mainly checks to ensure the variance components in a REML mixed model do not change between the last two iterations more than what is allowed by the tolerance value. See details for extra check on asreml-R models.

Usage

```
pcc(object, traces = NULL, tol = 0.01, silent = FALSE)
```

Arguments

object	A list with at least one element named: monitor (see Details)
traces	Optionally, a matrix to substitute instead of the monitor element to object. Each row corresponds to a different variance component in the model and each column is a different iteration of the likelihood calculation (column 1 is the first iterate).
tol	The tolerance level for which to check against all of the changes in variance component parameter estimates
silent	Optional argument to silence the output of helpful (indicating default underlying behavior) messages

Details

Object is intended to be an asreml-R model output. NOTE, The first 3 rows are ignored and thus should not be variance components from the model (e.g., they should be the loglikelihood or degrees of freedom, etc.). Also, the last column is ignored and should not be an iteration of the model (e.g., it indicates the constraint).

The function also checks `object` to ensure that the output from the asreml-R model does not contain a log-likelihood value of exactly 0.00. An ASReml model can sometimes fail while still returning a `monitor` object and `TRUE` value in the `converge` element of the output. This function will return `FALSE` if this is the case.

Value

Returns `TRUE` if all variance parameters change less than the value specified by `tol`, otherwise returns `FALSE`. Also see the `details` section for other circumstances when `FALSE` might be returned.

Author(s)

<matthewwolak@gmail.com>

Examples

```
# Below is the last 3 iterations from the trace from an animal model of
# tait1 of the warcolak dataset.
# Re-create the output from a basic, univariate animal model in asreml-R
   tracein <- matrix(c(0.6387006, 1, 0.6383099, 1, 0.6383294, 1, 0.6383285, 1),
nrow = 2, ncol = 4, byrow = FALSE)
   dimnames(tracein) <- list(c("ped(ID)!ped", "R!variance"), c(6, 7, 8, 9))

   pcc(object = NULL, trace = tracein)
```

---

pin *Approximate standard errors for linear functions of variance components*

Description

This function is similar to the pin calculations performed by the standalone ASReml. This function, written by Ian White, applies the delta method for the estimation of approximate standard errors on linear functions of variance components from a REML mixed model

Usage

```
pin(object, transform)
```

Arguments

`object`	A list with at least the following elements: `gammas`, `gammas.type`, and `ai` from a REML mixed model
`transform`	A formula specifying the linear transformation of variance components to conduct

Details

Object is intended to be an asreml-R model output.

The formula can use `V1, ..., Vn` to specify any one of the `n` variance components. These should be in the same order as they are in the object (e.g., see the row order of `summary(object)$varcomp` for asreml-R models.

Value

A dataframe with row names corresponding to the operator on the left hand side of the `transform` formula and the entries corresponding to the `Estimate` and approximate `SE` of the linear transformation.

Author(s)

Ian White

See Also

See Also as `aiCI`, `aiFun`

Examples

```
# Below is the heritability calculation for tait1 of the warcolak dataset
# Re-create the output from a basic, univariate animal model in asreml-R
   asrMod <- list(gammas = c(0.6383285, 1.00),
gammas.type = c(2, 1),
ai = c(0.0044461106, -0.0011754947, 0.0004424668))
   namevec <- c("ped(ID)!ped", "R!variance")
   names(asrMod[[1]]) <- names(asrMod[[2]]) <- namevec

   nadiv:::pin(asrMod, h2 ~ V1 / (V1 + V2))
```

---

prepPed *Prepares a pedigree by sorting and adding 'founders'*

Description

This function takes a pedigree, adds missing founders, and then sorts the pedigree.

Usage

```
prepPed(pedigree, gender = NULL, check = TRUE)
```

Arguments

pedigree	An object, where the first 3 columns correspond to: ID, Dam, & Sire. See details.
gender	An optional character for the name of the column in pedigree that corresponds to the gender/sex of individuals. If specified, prepPed will assign a gender to any founders it adds to the pedigree.
check	A logical argument indicating if checks on the validity of the pedigree structure should be made

Details

Many functions (both in nadiv and from other programs) dealing with pedigrees must first sort a pedigree such that individuals appear in the ID column in rows preceeding where they appear in either the Dam or Sire column. Further, these functions and programs require that all individuals in the dam and sire columns of a pedigree also have an entry in the ID column. This function easily prepares data sets to accommodate these requirements using a very fast topological sorting algorithm.

NOTE: more columns than just a pedigree can be passed in the pedigree argument. In the case of missing founders, these columns are given NA values for all rows where founders have been added to the pedigree. The entire object supplied to pedigree is ordered, ensuring that all information remains connected to the individual

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

When a non-null argument is given to gender, dams without an entry in the ID column (that are subsequently added to the pedigree) are given the gender designated for other dams (and similarly for sires).

The check argument performs checks on the format of the pedigree supplied to try and identify any issues regarding the notation of missing values and validity of the basic pedigree for further processing.

Value

The pedigree object (can have more columns than just ID, Dam, and Sire), where: (1) the ID column cotains an ID for all individuals from the original pedigree object's ID, Dam, and Sire columns (i.e., founders are added) and (2) the pedigree is now sorted so that individuals are not in rows preceding either their Dam or Sire.

See Also

genAssign, prunePed

Examples

```
# First create an unordered pedigree with (4) missing founders
  warcolak_unsuitable <- warcolak[sample(seq(5, nrow(warcolak), 1),
size = (nrow(warcolak) - 4), replace = FALSE), ]
  nrow(warcolak)
  nrow(warcolak_unsuitable)
# Fix and sort the pedigree
## Automatically assign the correct gender to the added founders
### Also sort the data accompanying each individual
  warcolak_fixed_ordered <- prepPed(warcolak_unsuitable, gender = "sex")
  head(warcolak_fixed_ordered)
```

`proLik` *Profile Likelihoods*

Description

Estimation, checking, and plotting of profile likelihoods and objects of class `proLik` from a mixed model in ASReml-R.

Usage

```
  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))

  ## S3 method for class 'proLik'
is(x)

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

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.
x	Object of class proLik.
CL	A logical indicating whether a line representing the Confidence Limit is to be drawn when plotting.
type	The type of plot to be generated, see arguments to plot.
...	other arguments to plot.

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

An S3 object of class proLik

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

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

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

## End(Not run)
```

---

prunePed *Prunes a pedigree based on individuals with phenotypes*

Description

This function removes individuals who are either not themselves or not ancestors to phenotyped individuals

Usage

```
prunePed(pedigree, phenotyped)
```

Arguments

pedigree	An object, where the first 3 columns correspond to: ID, Dam, & Sire. See details.
phenotyped	A vector indicating which individuals in the pedigree have phenotypic information available.

Details

Often mixed effect models run much faster when extraneous information is removed before running the model. This is particularly so when reducing the number of random effects associated with a relationship matrix constructed from a pedigree.

NOTE: more columns than just a pedigree can be passed in the `pedigree` argument.

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

This function is very similar to (and the code is heavily borrowed from) a function of the same name in the `MCMCglmm` package by Jarrod Hadfield.

Value

The pedigree object (can have more columns than just ID, Dam, and Sire), where the ID column cotains an ID for all individuals who are actually phenotyped or are an ancestor to an individual with a phenotype (and are thus informative for estimating parameters in the base population).

See Also

`prepPed`

Examples

```
# Make a pedigree (with sex) from the warcolak dataset
  warcolak_ped <- warcolak[, 1:4]

# Reduce the number of individuals that have a phenotype for "trait1" in
  #the warcolak dataset
  t1phenotyped <- warcolak
  t1phenotyped[sample(seq.int(nrow(warcolak)), 1500, replace = FALSE), "trait1"] <- NA
  t1phenotyped <- t1phenotyped[which(!is.na(t1phenotyped$trait1)), ]

# The following will give a pedigree with only individuals that have a
# phenotype for "trait1" OR are an ancestor to a phenotyped individual.
  pruned_warcolak_ped <- prunePed(warcolak_ped, phenotyped = t1phenotyped$ID)

# Now compare the sizes (note, pruned_warcolak_ped retained its column indicating sex.
# We could have kept all of the data associated with individuals who had phenotypic
# information on "trait1" by instead specifying
#  prunePed(warcolak, phenotyped = t1phenotyped$ID)
  dim(warcolak_ped)
  dim(pruned_warcolak_ped)
```

Q1988 *Pedigree, adapted from Quaas (1988) equation [5]*

Description

An example pedigree with genetic groups

Usage

```
data("Q1988")
```

Format

A data frame with 11 observations on the following 8 variables.

`id` a factor with levels indicating the unique individuals (including phantom parents) and genetic groups

`dam` a factor of observed maternal identities

`sire` a factor vector of observed paternal identities

`damGG` a factor of maternal identities with genetic groups inserted instead of `NA`

`sireGG` a factor of paternal identities with genetic groups inserted instead of `NA`

`phantomDam` a factor of maternal identities with phantom parents inserted instead of `NA`

`phantomSire` a factor of paternal identities with phantom parents inserted instead of `NA`

`group` a factor of genetic groups to which each phantom parent belongs

Source

Quaas, R.L. 1988. Additive genetic model with groups and relationships. Journal of Dairy Science 71:1338-1345.

simGG *Genetic group pedigree and data simulation*

Description

Simulates a pedigree and phenotype for a focal population receiving immigrants. Genetic and environmental differences can be specified between the focal and immigrant populations. Further, these differences can have temporal trends.

Usage

```
simGG(K, pairs, noff, g,
nimm = 2, nimmG = seq(2, g - 1, 1),
VAf = 1, VAi = 1, VRf = 1, VRi = 1,
mup = 20, muf = 0, mui = 0, murf = 0, muri = 0,
d_bvf = 0, d_bvi = 0, d_rf = 0, d_ri = 0)
```

Arguments

K	Integer number of individuals per generation, or the focal population carrying capacity
pairs	Integer number of mating pairs created by sampling with replacement from adults of a given generation
noff	Integer number of offspring each pair contributes to the next generation
g	Integer number of (non-overlapping) generations to simulate
nimm	Integer number of immigrants added to the population each generation of migration
nimmG	Sequence of integers for the generations in which immigrants arrive in the focal population
VAf	Numeric value for the expected additive genetic variance in the first generation of the focal population - the founders
VAi	Numeric value for the expected additive genetic variance in each generation of immigrants
VRf	Numeric value for the expected residual variance in the focal population
VRi	Numeric value for the expected residual variance in each generation of the immigrants
mup	Numeric value for the expected mean phenotypic value in the first generation of the focal population - the founders
muf	Numeric value for the expected mean breeding value in the first generation of the focal population - the founders
mui	Numeric value for the expected mean breeding value for the immigrants
murf	Numeric value for the expected mean residual (environmental) deviation in the first generation of the focal population - the founders
muri	Numeric value for the expected mean residual (environmental) deviation for the immigrants
d_bvf	Numeric value for the expected change between generations in the mean breeding value of the focal population. Sets the rate of genetic selection occurring across generations
d_bvi	Numeric value for the expected change between generations in the mean breeding value of the immigrant population each generation
d_rf	Numeric value for the expected change between generations in the mean residual (environmental) deviation of the focal population each generation
d_ri	Numeric value for the expected change between generations in the mean residual (environmental) deviation of the immigrant population each generation

Details

Offspring total additive genetic values u are the average of their parents u plus a Mendelian sampling deviation drawn from a normal distribution with mean of 0 and variance equal to $0.5V_A(1 - f_{sd})$ where V_A is VAf and f_{sd} is the average of the parents' coefficient of inbreeding f (p. 447 Verrier et al. 1993). Each 'immigrant' (individual with unknown parents in generations >1) is given a

total additive genetic effect that is drawn from a normal distribution with mean of mui and variance equal to VAi. Residual deviations are sampled for 'focal' and 'immigrant' populations separately, using normal distributions with means of murf and muri, respectively, and variances of VRf and VRi, respectively. Phenotypes are the sum of total additive genetic effects and residual deviations plus an overall mean mup.

Trends in total additive genetic effects and/or residual deviations can be specified for both the focal and immigrant populations. Trends in total additive genetic effects occurring in the immigrants, in the residual deviations occurring in the focal population, and in the residual deviations occurring in the immigrants are produced by altering the mean each generation for the separate distribution from which these effects are each drawn. The change in mean over a generation is specified in units of standard deviations of the respective distributions (e.g., square roots of VAi, VRf, and VRi) and is set with d_bvi, d_rf, or d_ri, respectively.

Trends in total additive genetic effects for the focal population are produced by selecting individuals to be parents of the next generation according to their *predicted* total additive genetic effects. Individuals are assigned probabilities of being selected as a parent of the next generation depending on how closely their predicted total additive genetic effect matches an optimum value. Probabilities are assigned:

$$exp((\frac{-1}{2\sigma_x})(x-\theta)^2)$$

where x is the vector of predicted total additive genetic effects (u), σ_x is the standard deviation of x, and θ is the optimum value. Sampling is conducted with replacement based on these probabilities.

The parameter d_bvf specifies how much the optimal total additive genetic effect changes per generation. The optimal total additive genetic effect in a given generation is calculated as: muf + d_bvf *sqrt(VAf) * (i-2). Individuals with predicted total additive genetic effects closest to this optimum have a higher probability of being randomly sampled to be parents of the next generation. This represents selection directly on predicted total additive genetic effects.

Total additive genetic effects are predicted for the first generation of focal individuals and all immigrants using equation 1.3 in Mrode (2005, p.3): $h^2 * (phenotype_i - meanpopulationphenotype)$. The heritability is either VAf / (VAf + VRf) or VAi / (VAi + VRi). Total additive genetic effects are predicted for all other individuals using equation 1.9 in Mrode (2005, p. 10) - or as the average of each individual's parents' predicted total additive genetic effects.

Value

A data.frame with columns corresponding to:

id	Integer for each individual's unique identifying code
dam	Integer indicating each individual's dam
sire	Integer indicating each individual's sire
parAvgU	Numeric value for the average of each individual's dam and sire additive genetic effects
mendel	Numeric value for each individual's Mendelian sampling deviate from the mid-parental total additive genetic value
u	Numeric value of each individual's total additive genetic effect
r	Numeric value of each individual's residual (environmental) deviation
p	Numeric value of each individual's phenotypic value

pred.u	Numeric value of each individual's predited total additive genetic effect
is	Integer of either 0 if an individual was born in the focal population or 1 if they were born in an immigrant population
gen	Integer value of the generation in which each individual was born

Author(s)

<matthewwolak@gmail.com>

References

Verrier, V., J.J. Colleau, and J.L. Foulley. 1993. Long-term effects of selection based on the animal model BLUP in a finite population. Theoretical and Applied Genetics. 87:446-454.

Mrode, R.A. 2005. Linear Models for the Prediction of Animal Breeding Values, 2nd ed. Cambridge, MA: CABI Publishing.

See Also

ggTutorial

Examples

```
  ## Not run:
  # The dataset 'ggTutorial' was simulated as:
  set.seed(102)       # seed used to simulate ggTutorial
  ggTutorial <- simGG(K = 400, pairs = 200, noff = 4, g = 15,
nimm = 40,
muf = 0, mui = 3)

## End(Not run)

# Use genetic group methods to approximate the breeding values for ggTutorial
  ## First, construct a pedigree with genetic groups
  ggPed <- ggTutorial[, c("id", "dam", "sire", "is", "gen")]
  naPar <- which(is.na(ggPed[, 2]))
  ggPed$GG <- rep("NA", nrow(ggPed))
    # 'focal' population genetic group = "foc0" and 'immigrant' = "g1"
    # obtained by pasting "foc" & "g" with immigrant status "0" or "1", respectively
    ggPed$GG[naPar] <- as.character(ggPed$is[naPar])
    ggPed$GG[ggPed$GG == "0"] <- paste0("foc", ggPed$GG[ggPed$GG == "0"])
    ggPed$GG[ggPed$GG == "1"] <- paste0("g", ggPed$GG[ggPed$GG == "1"])
  ggPed[naPar, 2:3] <- ggPed[naPar, "GG"]

  ## Now create the Q matrix
  Q <- ggcontrib(ggPed[, 1:3], ggroups = c("foc0", "g1"))

  ## obtain the true values of the genetic group means
  foc0_mean <- mean(ggTutorial$u[which(ggTutorial$gen == 1 & ggTutorial$is == 0)])
  g1_mean <- mean(ggTutorial$u[which(ggTutorial$is == 1)])
  g_exp <- matrix(c(foc0_mean, g1_mean), ncol = 1)
```

```
## breeding values (a) are:
### tot. add. gen. effects (u) minus genetic group effects for each individual (Qg):
a <- ggTutorial$u - Q
```

---

simPedDFC *Double first cousin pedigree construction*

Description

Simulates a pedigree for the "double first cousin" mating design (Fairbairn and Roff 2006).

Usage

```
simPedDFC(F, gpn = 4, fsn = 4, s = 2, prefix = NULL)
```

Arguments

F	Number of blocks for the design
gpn	Number of grandparents in the first/GP generation (must be >= 2)
fsn	Number of offspring in the full-sib families of the second/P generation (must be an even number >= 4)
s	Number of sires per full-sib family in the second/P generation (must be >=2)
prefix	Optional prefix to add to every identity

Details

This is an adaption to a half-sib breeding design which also produces first cousins and double first cousins. Double first cousins are produced by mating two brothers to two sisters (the offspring of the resulting two families are double first cousins with one another). This is described in Fairbairn and Roff (2006) as being particularly effective for separating autosomal additive genetic variance from sex chromosomal additive genetic variance. It is also amenable to estimating dominance variance, however, it still has difficulty separating dominance variance from common maternal environmental variance (Meyer 2008).

Value

A `data.frame` with columns corresponding to: id, dam, sire, and sex. Sex is `M` for males and `F` for females.

Author(s)

<matthewwolak@gmail.com>

References

Fairbairn, D.J. and D.A. Roff. 2006. The quantitative genetics of sexual dimorphism: assessing the importance of sex-linkage. Heredity 97:319-328.

Meyer, K. 2008. Likelihood calculations to evaluate experimental designs to estimate genetic variances. Heredity 101:212-221.

See Also

simPedHS, warcolak

Examples

```
DFC1 <- simPedDFC(F = 1, gpn = 2, fsn = 4, s = 2)
```

---

simPedHS *Half-sib pedigree construction*

Description

Simulates a pedigree for a half-sib mating design (sometimes also called the North Carolina Design 1).

Usage

```
simPedHS(s, d, n, uniqueDname = TRUE, prefix = NULL)
```

Arguments

s	Number of sires
d	Number of dams per sire
n	Number of offspring per mating (must be > or = 2)
uniqueDname	Logical indicating if dams should have unique names within sire families or throughout the entire pedigree
prefix	Optional prefix to every identity

Details

n must be greater than or equal to 2, because one male and one female offspring are produced from each mating

Some functions/calculations get bogged down if no two dams have the same ID in the entire pedigree (e.g., aov). However, other functions must have unique identifiers for every individual.

Value

A data.frame with columns corresponding to: id, dam, sire, and sex. Sex is "M" for males and "F" for females.

Author(s)

<matthewwolak@gmail.com>

See Also

simPedDFC

Examples

```
simPedHS(s = 1, d = 3, n = 2)
```

---

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 sampling variances to component scale*

Description

The inverse of the Average Information matrix in an ASReml-R object produces the sampling variances of the (co)variance components 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 ~ sex, random = ~ ped(ID) + giv(IDD),
ginverse = list(ID = ginvA, IDD = ginvD), data = warcolak)
    summary(warcolak.mod)$varcomp
   sqrt(varTrans(warcolak.mod))  # sqrt() so can compare with standard errors from summary

## End(Not run)
```

`warcolak` *Pedigree and phenotypic values for a mythical population of Warcolaks*

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 13 variables.

`ID` a factor specifying 5400 unique individual identities

`Dam` a factor specifying the unique Dam for each individual

`Sire` a factor specifying the unique Sire for each individual

`sex` a factor specifying "M" if the individual is a male and "F" if it is a female

`trait1` a numeric vector of phenotypic values: see 'Details'

`trait2` a numeric vector of phenotypic values: see 'Details'

`t1_a` a numeric vector of the autosomal additive genetic effects underlying 'trait1'

`t2_a` a numeric vector of the autosomal additive genetic effects underlying 'trait2'

`t2_s` a numeric vector of the sex-chromosomal additive genetic effects underlying 'trait2'

`t1_d` a numeric vector of the autosomal dominance genetic effects underlying 'trait1'

`t2_d` a numeric vector of the autosomal dominance genetic effects underlying 'trait2'

`t1_r` a numeric vector of the residual (environmental) effects underlying 'trait1'

`t2_r` a numeric vector of the residual (environmental) effects underlying 'trait2'

Details

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

The dataset was simulated to have two un-correlated traits with different genetic architectures (see `examples` below). The trait means are both equal to 1 for males and 2 for females. The additive genetic, dominance genetic, and environmental (or residual) variances for both `trait1` and `trait2` are 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; assuming the 'no global dosage compensation' mechanism).

Females and males have equal variances (except for sex-chromosomal additive genetic variance, where by definition, males have half of the additive genetic variance as females; Wolak 2013) and

a between-sex correlation of one for all genetic and residual effects (except the cross-sex residual covariance=0). All random effects were drawn from multivariate random normal distributions [e.g., autosomal additive genetic effects: N ~ (0, kronecker(A, G))] with means of zero and (co)variances equal to the product of the expected sex-specific (co)variances (e.g., G) and the relatedness (or incidence) matrix (e.g., A).

The actual variance in random effects will vary slightly from the amount specified in the simulation, because of Monte Carlo error. Thus, the random effects have been included as separate columns in the dataset. See `examples` below for the code that generated the dataset.

Note

Before nadiv version 2.14.0, the `warcolak` dataset used a 0/1 coding for 'sex' and did not contain the random effects.

References

Fairbairn, D.J. & Roff, D.A. 2006. The quantitative genetics of sexual dimorphism: assessing the importance of sex-linkage. Heredity 97, 319-328.

Wolak, M.E. 2013. The Quantitative Genetics of Sexual Differences: New Methodologies and an Empirical Investigation of Sex-Linked, Sex-Specific, Non-Additive, and Epigenetic Effects. Ph.D. Dissertation. University of California Riverside.

Examples

```
## Not run:
 rm(list = ls())
 set.seed(101)
 library(nadiv)
 # create pedigree
 warcolak <- simPedDFC(F = 75, gpn = 4, fsn = 4, s = 2)
 names(warcolak)[1:3] <- c("ID", "Dam", "Sire")
 warcolak$trait2 <- warcolak$trait1 <- NA

 # Define covariance matrices for random effects:
 ## Autosomal additive genetic (trait1)
 Ga_t1 <- matrix(c(0.4, rep(0.399999, 2), 0.4), 2, 2)
 ## Autosomal additive genetic (trait2)
 Ga_t2 <- matrix(c(0.3, rep(0.299999, 2), 0.3), 2, 2)
 ## Sex-chromosomal additive genetic (trait2)
 Gs_t2 <- matrix(c(0.1, rep(0.099999, 2), 0.1), 2, 2)
 ## Autosomal dominance genetic
 Gd <- matrix(c(0.3, rep(0.299999, 2), 0.3), 2, 2)
 ## Environmental (or residual)
 ### Assumes no correlated environmental effects between sexes
 R <- diag(c(0.3, 0.3))

 ## define variables to be re-used
 pedn <- nrow(warcolak)
 # Female (homogametic sex chromosomes) in first column
 # Male (heterogametic sex chromosomes) in second column
 sexcol <- as.integer(warcolak$sex)
```

```
  # Create random effects
  ## Additive genetic
  ### trait1 autosomal
  tmp_a <- grfx(pedn, G = Ga_t1, incidence = makeA(warcolak[, 1:3]))
    var(tmp_a)
  warcolak$t1_a <- tmp_a[cbind(seq(pedn), sexcol)]
  ### trait2 autosomal
  tmp_a <- grfx(pedn, G = Ga_t2, incidence = makeA(warcolak[, 1:3]))
    var(tmp_a)
  warcolak$t2_a <- tmp_a[cbind(seq(pedn), sexcol)]
  ### trait2 sex-chromosomal
  tmp_s <- grfx(pedn, G = Gs_t2, incidence = makeS(warcolak[, 1:4],
heterogametic = "M", DosageComp = "ngdc", returnS = TRUE)$S)
    matrix(c(var(tmp_s[which(sexcol == 1), 1]),
rep(cov(tmp_s), 2), var(tmp_s[which(sexcol == 2), 2])), 2, 2)
    # NOTE above should be: covar = male var = 0.5* female var
  warcolak$t2_s <- tmp_s[cbind(seq(pedn), sexcol)]

  ## Dominance genetic
  ### trait1
  tmp_d <- grfx(pedn, G = Gd, incidence = makeD(warcolak[, 1:3], invertD = FALSE)$D)
    var(tmp_d)
  warcolak$t1_d <- tmp_d[cbind(seq(pedn), sexcol)]
  ### trait2
  tmp_d <- grfx(pedn, G = Gd, incidence = makeD(warcolak[, 1:3], invertD = FALSE)$D)
    var(tmp_d)
  warcolak$t2_d <- tmp_d[cbind(seq(pedn), sexcol)]

  ## Residual
  ### trait1
  tmp_r <- grfx(pedn, G = R, incidence = NULL) # warning of identity matrix
    var(tmp_r)
  warcolak$t1_r <- tmp_r[cbind(seq(pedn), sexcol)]
  ### trait2
  tmp_r <- grfx(pedn, G = R, incidence = NULL) # warning of identity matrix
    var(tmp_r)
  warcolak$t2_r <- tmp_r[cbind(seq(pedn), sexcol)]

  # Sum random effects and add sex-specific means to get phenotypes
  ## females have slightly greater mean trait values
  warcolak$trait1 <- 1 + (-1*sexcol + 2) + rowSums(warcolak[, c("t1_a", "t1_d", "t1_r")])
  warcolak$trait2 <- 1 + (-1*sexcol + 2) + rowSums(warcolak[, c("t2_a", "t2_s", "t2_d", "t2_r")])

## End(Not run)
```

Index