Package: breedR (via r-universe)

November 1, 2024

Type Package

Title Statistical Methods for Forest Genetic Resources Analysts

Version 0.12-7

Encoding UTF-8

Author Facundo Muñoz with libraries by Ignacy Misztal

Maintainer Facundo Muñoz <facundo.munoz@cirad.fr>

Description Statistical tools to build predictive models for the breeders community. It aims to assess the genetic value of individuals under a number of situations, including spatial autocorrelation, genetic/environment interaction and competition. It is under active development as part of the Trees4Future project, particularly developed having forest genetic trials in mind. But can be used for animals or other situations as well.

License GPL-3 | file LICENSE

Depends R (>= 3.1.2), sp

Imports ggplot2, graphics, grDevices, Matrix (>= 1.2.0), methods, nlme, pedigree, pedigreemm, splines, stats, utils

Suggests doParallel, fields, fmesher, GGally, grid, MASS, msm, plyr, dplyr, spam, testthat, knitr, rmarkdown, tidyr

SystemRequirements BLUPF90 programs will be downloaded from breedR website at install time.

LazyLoad yes

LazyData yes

URL https://github.com/famuvie/breedR

BugReports https://github.com/famuvie/breedR/issues

Additional_repositories http://inla.r-inla-download.org/R/testing

VignetteBuilder knitr

RoxygenNote 7.3.2

Repository https://famuvie.r-universe.dev

RemoteUrl https://github.com/famuvie/breedR

RemoteRef HEAD

RemoteSha bf52f982aaf86f13a55b54a2349420e13b7f0a56

Contents

breedR-package *Frequentist and Bayesian methods for breeders, quantitative geneticists and forest genetic resources analysts.*

Description

This package provides statistical tools to build predictive models for the breeders, quantitative genetists and forest genetic resources analysts communities. It aims to assess the genetic value of individuals under a number of situations, including spatial autocorrelation, genetic/environment interaction and competition. It is under active development as part of the Trees4Future project, particularly developed having forest genetic trials in mind. But can be used for animals or other situations as well.

Details

The package functionality builds up on a wrapping up of Ignacy Misztal's progsf90 suite of Fortran programs. Particularly, the function `reml` performs classical Restricted-Maximum Likelihood inference by interfacing Misztal's programs with several high-level options such as spatial components, etc. The Fortran back-end allows for fast inference on rather large datasets (hundreds of thousands of individuals) with complex pedigrees.

Author(s)

Maintainer: Facundo Muñoz <facundo.munoz@cirad.fr>

Authors:

- Leopoldo Sanchez

Other contributors:

- Ignacy Misztal [contributor]
- Pablo Cappa [contributor]
- Timothée Flutre [contributor]

References

Most functionality in the package is based on Ignacy Misztal's suite of Fortran programs for mixed model computations in breeding. http://nce.ads.uga.edu/wiki/doku.php

M. Lynch \& B. Walsh (1998). *Genetics and Analysis of Quantitative Traits*. Sinauer Associates, Inc.

See Also

pedigreemm

Examples

```
# Load, summarize and visualize data
data(m4)
summary(m4)
plot(m4)

# Fit Mixed Model using REML
res.f90 <- remlf90(fixed = phe_X ~ gen,
                   genetic = list(model = 'add_animal',
                                  pedigree = get_pedigree(m4),
                                  id = 'self'),
                   data = as.data.frame(m4))

# Summary of results
summary(res.f90)

# Observed phenotypes vs. Fitted values
library(ggplot2)
qplot(phe_X, fitted(res.f90), color=gen, data = as.data.frame(m4)) +
  geom_abline(intercept=0, slope=1)
```

`additive_genetic` *Build an additive_genetic model*

Description

Check conformity of arguments and return a `additive_genetic` object.

Usage

```
additive_genetic(pedigree, incidence)
```

Arguments

`pedigree`	object of class 'pedigree'
`incidence`	matrix-like object

Value

A list with elements `pedigree`, `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

Examples

```
ped <- pedigreemm::pedigree(sire = c(NA,NA,1, 1,4,5),
                            dam  = c(NA,NA,2,NA,3,2),
                            label= 1:6)
inc <- cbind(0, 0, diag(4))
breedR:::additive_genetic(ped, inc)
```

additive_genetic_animal

Build an additive-genetic animal model

Description

Given a pedigree, and an index vector of observations, build and `additive_genetic_animal` model.

Usage

```
additive_genetic_animal(pedigree, idx)
```

Arguments

`pedigree`	object of class 'pedigree'
`idx`	integer vector of observed individuals (in the original codification)

Details

`idx` must hold the index of observed individuals in the original codification. If recoding took place when building the pedigree, this function will convert the codes internally.

Value

A list with elements `pedigree`, `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

Examples

```
dat <- data.frame(id = 1:4,
                  sire = c(11, 11, 2, 3),
                  dam  = c(12, NA, 1, 12))
ped <- build_pedigree(1:3, data = dat)
breedR:::additive_genetic_animal(ped, dat$id)
```

additive_genetic_competition

Build an additive-genetic competition model

Description

Return incidence and structure for a `additive_genetic_competition` model, given the pedigree, the spatial coordinates and codes of the observations and the competition decay parameter.

Usage

```
additive_genetic_competition(pedigree, coordinates, id, decay, autofill = TRUE)
```

Arguments

pedigree	object of class 'pedigree'
coordinates	two-column matrix-like set of row and column coordinates of observational units
id	integer vector of numeric codes for observed individuals
decay	numeric. The positive value of the decay parameter α. Typically 1 or 2. See Details.
autofill	logical. If TRUE (default) it will try to fill missing rows or columns with missing observations. Otherwise, will treat individuals as neighbours even if they are across an empty line.

Details

id must hold the codes of observed individuals in the original codification. If recoding took place when building the pedigree, this function will handle the codes internally.

Value

A list with elements pedigree, incidence.matrix, structure.matrix and structure.type, which is a string indicating either covariance or precision.

Examples

```
dat <- data.frame(id   = 1:5,
                  sire = c(11, 11, 2, 3, 2),
                  dam  = c(12, NA, 1, 12, 1),
                  x    = c(rep(1:2, times = 2), 3),
                  y    = c(rep(1:2, each = 2), 3))
ped <- build_pedigree(1:3, data = dat)
breedR:::additive_genetic_competition(ped, coord = dat[, c('x', 'y')], dat$id, 2)
```

---

as.triplet *Represent a symmetric matrix in triplet format*

Description

It only gives the lower triangular elements, and **do not** check for symmetry.

Usage

```
as.triplet(x)
```

Arguments

x	matrix.

b.values *Breeding values*

Description

Breeding values

Usage

```
b.values(x)
```

Arguments

x	a metagene object.

bispline_incidence *Incidence Matrix of Bidimensional Splines*

Description

Compute the incidence matrix as a tensor product of B-spline bases, given the knots, coordinates and order.

Usage

```
bispline_incidence(knots, xx, ord, sparse)
```

Arguments

knots	list with numeric vectors of knot positions with non-decreasing values for each dimension
xx	2-column matrix of coordinates at which to evaluate the bidimensional splines
ord	integer order of the spline functions (equals degree + 1)
sparse	logical indicating if the result should be given in sparse format

Details

Need at least 2*ord -1 knots (typically, 7) but in fact, we need at least 2*ord unless we set outer.ok = TRUE in splineDesign (which we do not want)

References

Eilers, P H C and B D Marx (2003). Multivariate calibration with temperature interaction using two-dimensional penalized signal regression. *Chemometrics and Intelligent Laboratory Systems* 66(2), 159-174.

breedR.get.HOME *Determine the user's home directory*

Description

Relies on `Sys.getenv('HOME')`, or under windows, on `Sys.getenv("USERPROFILE"))` changing backslashes to slashes.

Usage

```
breedR.get.HOME()
```

breedR.get.USER *Determine the user name*

Description

Determine the user name

Usage

```
breedR.get.USER()
```

breedR.option *Set and get global options for breedR*

Description

Set and get global options for breedR. The options are stored in the variable `breedR.options` in the `.GlobalEnv`-environment, and will therefore persist during the session. If you want to set some options permanently do it in a file names `.breedRrc` in your home directory. See Examples.

Usage

```
breedR.getOption(
  option = c("ar.eval", "breedR.bin", "splines.nok", "default.initial.variance",
   "col.seq", "col.div", "cygwin", "cygwin.home", "ssh.auth.sock", "remote.host",
    "remote.user", "remote.port", "remote.bin", "ssh.options")
)

breedR.setOption(...)
```

Arguments

option	The option to get. If missing or NULL, then breedR.getOption will display the current defaults, otherwise, option must be one of ar.eval: numeric vector of values in (-1, 1) where the autoregressive parameters should be evaluated if not otherwise specified splines.nok: a function of the number of individuals in a row which gives the number of knots (nok) to be used for a splines model, if not otherwise specified default.initial.variance: a function of the numeric response vector or matrix which returns a default initial value for a variance component col.seq: a vector with the specification of default extreme breedR col for sequential scales in spatial quantitative plots. See Details. col.div: a vector with the specification of default extreme breedR col for diverging scales in spatial quantitative plots. See Details. cygwin: the home of the Cygwin installation (default "C:/cygwin") [Remote computing for Windows only] cygwin.home: the user's home in the Cygwin installation [Remote computing for Windows only] ssh.auth.sock: the ssh bind-address (value of $SSH_AUTH_SOCK in the Cygwin-shell). [Remote computing for Windows only] remote.host: (IP or DNS) address of a Linux server for remote computing remote.user: user name with ssh-keys access to the server (see details) breedR.bin: full path for breedR backend binaries remote.port: port for ssh connection remote.bin: path to the binaries directory in the remote installation of breedR. Usually the output of system.file('bin/linux',package='breedR') in a remote server's R-session. ssh.options: ssh options. You shouldn't need to change this.
...	Option and value, like option=value or option,value; see the Examples.

Details

Sequential scales are used for variables not necessarily centered such as a response variable, or the fitted values of a model. The colour scale is built as a gradient between two extreme colours which are specified as hex codes or colour names in the option col.seq.

Diverging scales are used for plots such as residuals, centered (hopefully) around zero, with positive and negative values represented with different colours whose intensity is linked to the magnitude. The option col.div is a vector of two hex codes or colour names of the most intense colours.

Examples

```
## Set default values for the autoregressive parameters
breedR.setOption("ar.eval", 3*(-3:3)/10)
## alternative format
breedR.setOption(ar.eval = 3*(-3:3)/10)
## check it
breedR.getOption("ar.eval")
```

```
## Not run:
# Set up some options permanently in $HOME/.breedRrc
writeLines(c("remote.host = '123.45.678.999'",
             "remote.user = 'uname'",
             "remote.bin  = 'remote/path/to/breedR/bin/linux'"),
           con = file.path(Sys.getenv('HOME'), '.breedRrc')

## End(Not run)
```

`breedR.os` *Check host OS*

Description

Identifies host operating system.

Usage

```
breedR.os(type = c("linux", "mac", "windows", "else"))
```

Arguments

`type`	character. OS to be checked.

Details

Relies on `.Platform$OS.type`, but distinguishes between linux or mac.

`breedR.os.32or64bit` *test 32/64 bits architecture*

Description

Give precedence to current R architecture

Usage

```
breedR.os.32or64bit()
```

Value

Either "32" or "64"

breedR.os.type *Return platform string*

Description

Return whether the OS is either `windows`, `linux` or `mac` Inspired in INLA's os.R functions

Usage

```
breedR.os.type()
```

breedR.remote *Perform a job remotely*

Description

Assumes that all the relevant files are in the current directory.

Usage

```
breedR.remote(jobid, breedR.call, verbose = TRUE)
```

Arguments

`jobid`	character. A string uniquely identifying the current job.
`breedR.call`	character. A full string path to the executable program in the server.
`verbose`	logical. If `TRUE` (default) it shows informative messages.

breedr_ar *Build an autoregressive model*

Description

Given the coordinates of the observations, the autocorrelation parameters and the autofill logical value, computes the incidence matrix B and the covariance matrix U

Usage

```
breedr_ar(coordinates, rho, autofill = TRUE, ...)
```

Arguments

coordinates	matrix(-like) of observation coordinates
rho	numeric. Vector of length two with the autocorrelation parameter in each dimension, with values in the open interval (-1, 1).
autofill	logical. If TRUE (default) it will try to fill gaps in the rows or columns. Otherwise, it will treat gaps the same way as adjacent rows or columns.
...	Not used.

Value

a list with - the autocorrelation parameters - the coordinates original coordinates as a data frame - the incidence matrix (encoded as a vector) - the covariance matrix (of the full grid, in triplet form)

---

breedr_blocks *Build a blocks model*

Description

Given the coordinates of the observations, the *factor* identifying blocks, and the logical autofill, build the incidence and covariance matrices of a blocks model.

Usage

```
breedr_blocks(coordinates, id, autofill = TRUE, ...)
```

Arguments

coordinates	matrix(-like) of observation coordinates
id	factor of the same length as observations, giving the block id for each observation.
autofill	logical. If TRUE (default) it will try to fill gaps in the rows or columns. Otherwise, it will treat gaps the same way as adjacent rows or columns.
...	Not used.

breedr_effect *Constructor for a generic breedR effect*

Description

The breedr_effect-class is virtual. No object should be directly created with this constructor. This constructor is to be called from within non-virtual subclasses like generic, diagonal, spatial or additive_genetic.

Usage

```
breedr_effect(incidence)

## S3 method for class 'breedr_effect'
dim(x)
```

Arguments

incidence	matrix-like object
x	A breedr_effect.

Details

This constructor performs the arguments checks. But the implementation details (i.e., storage format and handling) is left for the subclasses.

Value

A list with a single element incidence.matrix.

breedr_progsf90_repo *Default repository for PROGSF90 binaries*

Description

Default repository for PROGSF90 binaries

Usage

```
breedr_progsf90_repo()
```

`breedr_splines` *Build a splines model*

Description

Given the coordinates of the observations, and the degree, this function puts into place a sensible number of spline knots and computes the incidence matrix B and the covariance matrix U

Usage

```
breedr_splines(
  coordinates,
  n.knots = NULL,
  autofill = TRUE,
  degree = 3,
  sparse = TRUE,
  strategy = "uniformgrid",
  ...
)
```

Arguments

`coordinates`	matrix(-like) of observation coordinates
`n.knots`	numeric. Vector of length two with an integer number of knots in each dimension.
`autofill`	logical. If `TRUE` (default) it will try to fill gaps in the rows or columns. Otherwise, it will treat gaps the same way as adjacent rows or columns.
`degree`	integer. Degree of the B-spline polynomials.
`sparse`	logical. If `TRUE` (default) the incidence matrix will be stored in sparse format. Current implementation ignores a value of FALSE.
`strategy`	character. Strategy for placing spline knots. Only `uniformgrid` available for the moment.
`...`	Not used.

Details

Relies on `splines::splineDesign()`, which uses a C function call to compute the splines coefficients.

`sparse` matrices take less memory, but also take longer to compute with. This is probably convenient only for really big datasets in comparison with RAM size. The covariance matrix is always stored in sparse format, as it is particularly sparse.

Value

A list with elements `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

build_grid *Build an encompassing grid*

Description

Build the minimal regularly-spaced grid containing a given set of points.

Usage

```
build_grid(coordinates, autofill = TRUE)
```

Arguments

coordinates	two-column matrix-like set of row and column coordinates of observational units
autofill	logical. If TRUE (default) it will try to fill missing rows or columns with missing observations. Otherwise, will treat individuals as neighbours even if they are across an empty line.

Details

Note that `autofill = FALSE` virtually removes the empty lines, considering the spacing as constant.

Value

The parameters defining the grid, and the index of the observed coordinates in the grid seen as a vector. More specifically,

origin the coordinates of the *first* (with smallest row and column values) point in the grid

step the separation between rows and columns

length the number of points in each dimension

idx the index of each observation in the vectorized grid

build_pedigree *Build pedigree*

Description

Builds a full pedigree out of observed data with sorting and recoding.

Usage

```
build_pedigree(x, self = x[[1]], sire = x[[2]], dam = x[[3]], data)

## S3 method for class 'pedigree'
as.data.frame(x, ...)
```

Arguments

x	if given, a vector of length 3 with indices or names of columns in data corresponding to self, sire and dam codes
self	index or column name in data with codes of sires
sire	index or column name in data with codes of individuals
dam	index or column name in data with codes of dams
data	a dataframe or a list to take the individual codes from
...	Not used

Details

A full pedigree requires that all the individual codes for sires or dams are present as individuals themselves, possibly with unknown parents. Therefore, for using it in a statistical model, it is necessary to *complete* the pedigree by introducing new individuals with unknown parents.

Furthermore, the codes must be sorted in ascending and consecutive order beginning from 1, and the offspring must follow parents. All this is checked, and the pedigree is reordered and recoded if needed.

If recoding is needed, the function issues a warning and an attribute 'map' is attached to the pedigree, such that `map[i] = j` means that code `i` was renumbered as `j`. Therefore, if `x` is a vector with original codes, `map[x]` gives the new codes. Conversely, `match(y,map)` back-transforms to original codes.

Value

A well-formed 'pedigree'-class object. Possibly sorted and recoded.

Functions

- `as.data.frame(pedigree)`: Coerce to a data.frame. One row per individual, the first column being the identification code, and the other two columns are dad and mum codes respectively.

See Also

check_pedigree

Examples

```
# Founders are missing in the globulus dataset
data(globulus)
check_pedigree(globulus[,c('self', 'dad', 'mum')])
# build_pedigree completes the missing information
ped <- build_pedigree(c('self', 'dad', 'mum'), data = globulus)
check_pedigree(ped)
```

check_pedigree *Check the rules for a well-formed pedigree*

Description

This function performs sanity checks on a pedigree.

Usage

```
check_pedigree(ped)
```

Arguments

ped	a 'pedigree'-class object, a dataframe, a matrix, or anything that can be coerced into a 3-column, numeric dataframe.

Details

Rules are: the pedigree must be full (i.e. all the codes in the pedigree must appear once in the first column); the codes of the offspring must follow their parent's codes; the identity codes must be sorted in ascending order and this order must be consecutive beginning from 1.

If any check fails, the pedigree must be recoded/reordered to be used in analysis. `build_pedigree(1:3, data = ped)` should fix it.

Value

a named logical vector with the results of the checks

See Also

build_pedigree

Examples

```
# A well-formed pedigree
ped_ok <- data.frame(id   = 1:6,
                     dam  = c(NA,NA,1,1,4,4),
                     sire = c(NA,NA,2,2,3,5))
check_pedigree(ped_ok)  # passes all checks

# Sometimes founders are missing
(ped_notfull <- ped_ok[-c(1:2),])
check_pedigree(ped_notfull)  # fails full_ped

# Sometimes codes of parents are greater than their offspring
sw_23 <- c(1, 3, 2, 4:6)
(ped_messed <- as.data.frame(sapply(ped_ok, function(x) sw_23[x]))[sw_23,])
check_pedigree(ped_messed)        # fails offsp_follows
```

```
# Sometimes the pedigree is unordered
(ped_unordered <- ped_ok[sw_23, ])
check_pedigree(ped_unordered)   # fails codes_sorted

# Finally, sometimes codes are just not consecutive
(ped_notconsec <- transform(ped_ok, id = c(1:5, 10)))
check_pedigree(ped_notconsec)  # fails codes_consecutive

# Everything is fixed with build_pedigree()
check_pedigree(build_pedigree(1:3, data = ped_notfull))
check_pedigree(build_pedigree(1:3, data = ped_messed))
check_pedigree(build_pedigree(1:3, data = ped_unordered))
check_pedigree(build_pedigree(1:3, data = ped_notconsec))
```

`check_progsf90` *Checks installation of PROGSF90 binaries*

Description

Checks whether the binary dependencies are installed in the right directory. If not, allows calling the installer

Usage

```
check_progsf90(
  path = breedR.getOption("breedR.bin"),
  platform = breedR.os.type(),
  quiet = !interactive()
)
```

Arguments

`path`	directory to check for the presence of binaries. Default is defined in the package options, and it depends on the platform.
`platform`	Either "linux", "windows" or "mac". Default is current platform.
`quiet`	if TRUE, it won't ask whether to install missing binaries.

Details

This function does not check whether the binaries are for the right platform or architecture. It only checks the presence of files with the expected names.

check_var.ini *Check initial variances specification*

Description

If the user specified initial values, verify that all random effects were included. Otherwise, set default values. In any case, validate all initial values.

Usage

```
check_var.ini(x, random, response)
```

Arguments

x	list. user specification of var.ini (or NULL)
random	formula. user specification of random effects.
response	numeric vector or matrix.

Value

A list with initial covariance matrices for all random effects in the model. A logical attribute ‘var.ini.default‘ is TRUE if values were set by default.

matrix of observation values.

compare.plots *Compare two or more ggplots of the same kind*

Description

This function presents several ggplots of the same type side by side under the same scale, while keeping annotations.

Usage

```
compare.plots(plots)
```

Arguments

plots	List of ggplots with meaningful names

Details

The names of the objects in the list will be used for facet labels.

`competition` *Build a virtual competition model*

Description

Given the coordinates of a set of observations, a decay parameter and a structure matrix, compute the incidence matrix of competition, and return a random effect with the given structure.

Usage

```
competition(coordinates, covariance, precision, decay, autofill = TRUE)
```

Arguments

`coordinates`	two-column matrix-like set of row and column coordinates of observational units
`covariance`	matrix-like object
`precision`	matrix-like object
`decay`	numeric. The positive value of the decay parameter α. Typically 1 or 2. See Details.
`autofill`	logical. If TRUE (default) it will try to fill missing rows or columns with missing observations. Otherwise, will treat individuals as neighbours even if they are across an empty line.

Details

The competition model attributes to each individual a random effect of competition with variance $\sigma^2_{a_c}$, which impacts the phenotype of the neighbours rather than its own phenotype.

Conversely, the effect of the competition over one's phenotype is given by the additive-genetic competition effects of the neighbours, weighted by the relative distances. If α is the decay parameter and a_c is the random competition effect of a neighbour at distance d, then the Weighted Neighbour Competition effect over one's phenotype is given by

$$wnc = \sum_{\text{neighbours}} kd^{-\alpha}a_c,$$

where k is a normalizing constant which makes $Var(wnc) = \sigma^2_{a_c}$ and independent of the number of neighbours.

Value

An object inheriting from `spatial`.

coordinates_breedR *breedR coordinates methods*

Description

breedR coordinates methods

Usage

```
## S4 method for signature 'breedR'
coordinates(obj, ...)

## S4 replacement method for signature 'breedR'
coordinates(object) <- value

## S4 method for signature 'effect_group'
coordinates(obj, ...)

## S4 method for signature 'spatial'
coordinates(obj, ...)

## S4 method for signature 'metagene'
coordinates(obj, ...)

## S4 replacement method for signature 'metagene'
coordinates(object) <- value
```

Arguments

obj	an object of the corresponding class
...	not used.
object	an object of the corresponding class
value	2-column matrix or data frame with coordinates

default_initial_variance *Default initial value for variance components*

Description

A function of the response vector or matrix (multi-trait case) returning a SPD matrix of conforming dimensions.

Usage

```
default_initial_variance(
  x,
  dim = 1,
  cor.trait = NULL,
  cor.effect = 0.1,
  digits = NULL
)
```

Arguments

x	numeric vector or matrix with the phenotypic observations. Each trait in one column.
dim	integer. dimension of the random effect for each trait. Default is 1.
cor.trait	a number strictly in (-1, 1). The initial value for the correlation across traits. Default is NULL, which makes the function to take the value from the data. See Details.
cor.effect	a number strictly in (0, 1). The initial value for the correlation across the different dimensions of the random effect. Default is 0.1.
digits	numeric. If not NULL (as default), the resulting matrix is rounded up to 2 significant digits.

Details

The default initial covariance matrix across traits is computed as half the empirical covariance kronecker times a Positive-Definite matrix with Compound Symmetry Structure with a constant diagonal with value 1 and constant off-diagonal elements with the positive value given by `cor.effect`, i.e.

$$\Sigma = Var(x)/2\% * \%\psi(dim).$$

This implies that the default initial **correlations** across traits equal the empirical correlations, except if `cor.trait` is not `NULL`.

$\psi(dim)$ is intended to model correlated random effects within traits, and only has an effect when `dim` > 1.

If any column in `x` is constant (i.e. empirical variance of 0) then the function stops. It is better to remove this trait from the analysis.

Examples

```
## Initial covariance matrix for a bidimensional random effect
## acting independently over three traits
x <- cbind(rnorm(100, sd = 1), rnorm(100, sd = 2), rnorm(100, sd = 3))
breedR:::default_initial_variance(x, dim = 2, cor.effect = 0.5)
```

determine.n.knots *Determine a sensible number of knots*

Description

This function computes a reasonable number of inner knots to be used for a basis of unidimensional B-splines

Usage

```
determine.n.knots(n, cutoff = 4, rate = 0.3)
```

Arguments

n	an integer vector of sample sizes
cutoff	a numeric vector of cutoff values
rate	a numeric vector of rates at which the number of knots increases with the sample size

Value

An integer vector with the number of knots for each sample size

References

Ruppert, D. (2002). Selecting the number of knots for penalized splines. *Journal of Computational and Graphical Statistics* 11, 735–757.

diagonal *Constructor for a diagonal random effect*

Description

Builds a breedR random effect with an index incidence matrix and a diagonal covariance matrix, with one level for each value of x.

Usage

```
diagonal(x)
```

Arguments

x	a numeric covariate or a factor.

Details

Uses sparse storage. It does not support nesting (yet). So it is not possible to build random regression coefficients for each level of a grouping factor. This is in the TODO list.

Value

A random effect with element `incidence.matrix` and a diagonal `structure.matrix`

`distribute_knots_uniformgrid`

Distribute knots uniformly in a grid

Description

For each dimension, places the given number of knots evenly spaced covering the range of coordinates plus a small margin.

Usage

```
distribute_knots_uniformgrid(coordinates, n.knots, autofill)
```

Arguments

`coordinates`	matrix(-like) of observation coordinates
`n.knots`	numeric. Vector of length two with an integer number of knots in each dimension.
`autofill`	logical. If `TRUE` (default) it will try to fill gaps in the rows or columns. Otherwise, it will treat gaps the same way as adjacent rows or columns.

Details

The margin is calculated as half the median separation between observations. Furthermore, three more knots are added with equal spacing at each side, for each dimension.

douglas *Multi-site Douglas-fir dataset*

Description

Provenance test with measurements of height and circumference

Format

A dataframe with 1021 individuals and the following 9 variables

- `self` id of the tree
- `dad` id of sire or 0 if unknown
- `mum` id of dam or 0 if unknown
- `orig` factor origin of the seeds with 11 levels partially crossed with site
- `site` factor site with 3 levels
- `block` factor block with 127 levels nested within site
- `x, y` coordinates (in m)
- `H02:H05`heights as measured in 2002-2005
- `C13` circumference as measured in 2013
- `AN` factor angle quality with 5 levels
- `BR` factor branching quality with 5 levels

Details

The dataset includes measurements of height, circumference, angle and branching, for a progeny from 11 different origins which were planted in 1998 in 3 different sites.

Only 4 out of 11 origins are available in all the three sites.

Circumference was measured in all the three sites in 2013, but measurements of height were taken in 2002, 2003 and 2004 for site 3, and in 2005 for site 2. Measurements of angle and branching are only available for site 3.

The spacing between trees is different among sites, and in general between rows and columns. Coordinate units are meters.

Examples

```
data(douglas)

## Individuals by origin and site
with(douglas, table(orig, site))

## Measurements by variable and site
library(tidyr)
library(dplyr)
```

```
douglas %>%
 gather(variable, value, H02:BR, na.rm = TRUE) %>%
 with(., table(variable, site))

## Visualise trials
library(ggplot2)
ggplot(douglas, aes(x, y)) +
  geom_point() +
  facet_wrap(~ site)
```

`effect_group` *Constructor for a group of effects*

Description

Builds an `effect_group` from a list of `breer_effect` elements.

Usage

```
effect_group(x, cov.ini, ntraits)

## S3 method for class 'effect_group'
dim(x)
```

Arguments

`x`	list of breedr_effect elements
`cov.ini`	initial covariance matrix for the estimation algorithm
`ntraits`	number of traits in the model

Details

Temporarily, this takes the `cov.ini` argument and includes it in the object. In the future, the initial covariance matrix will be a matter of the inference engine, not inherent to the model.

The ‘ntraits‘ is used to check the dimension of the initial variance matrix.

Value

A list of `breedr_effect` elements.

effect_type *Type of a (group of) effect(s)*

Description

Generic function that returns whether an effect or a group of effects in a breedR mixed model is `fixed` or `random`

Usage

```
effect_type(x)

## S3 method for class 'effect_group'
effect_type(x)

## S3 method for class 'breedr_effect'
effect_type(x)
```

Arguments

x	object to be *translated* to progsf90

Methods (by class)

- `effect_type(effect_group)`: Type of an `effect_group`
- `effect_type(breedr_effect)`: Type of an `breedr_effect`

Extract.metagene *Extract or replace data in a metagene object*

Description

Extract or replace data in a metagene object

Extract columns directly from the dataframe

Write columns of the dataframe

Subset data

Usage

```
## S3 method for class 'metagene'
x$name

## S3 replacement method for class 'metagene'
x$name <- value

## S3 method for class 'metagene'
x[...]
```

Arguments

x	a metagene object.
name	character. A variable name.
value	a vector.
...	a vector of integer indices or names of columns in the dataset.

See Also

Other metagene: get_ntraits(), get_pedigree(), ngenerations(), nindividuals()

---

extract_block *Extract a block of lines from a section of the REML log*

Description

Extract a block of lines from a section of the REML log

Usage

```
extract_block(l, x)
```

Arguments

l	numeric. Line number where the block starts. Just *after* the section heading.
x	character vector. Lines of the log.

Value

character vector. Lines of text corresponding to numeric values under the section.

Examples

```
test_log <-
  c("REML log",
    "Some info",
    "Covariance matrix",
    "   1.23E2   4.56E-02   7.89E-03    ",
    "   1.23E2   4.56E-02   7.89E-03    "
    )
breedR:::extract_block(4, test_log)
```

fill_holes *Find and fill all the holes in a vector*

Description

Find and fill all the holes in a vector

Usage

```
fill_holes(x, label)
```

Arguments

x	numeric. Vector of increasing coordinates.
label	character. A name like 'rows' or 'x'.

fixed *Constructor for a fixed effect*

Description

Constructor for a fixed effect

Usage

```
fixed(x)
```

Arguments

x	a numeric covariate or a factor.

Value

A breedr_effect with element `incidence.matrix`.

fixef *Extract fixed-effects estimates*

Description

Extract the fixed-effects estimates

Usage

```
## S3 method for class 'remlf90'
fixef(object, ...)
```

Arguments

object	any fitted model object from which fixed effects estimates can be extracted.
...	not used

Details

Extract the estimates of the fixed-effects parameters from a fitted model.

Value

a named list of dataframes with the estimated coefficients and their standard error

Examples

```
res <- remlf90(phe_X ~ gg + bl, data = globulus)
fixef(res)
```

generic *Build a generic model*

Description

Check conformity of arguments and return a generic object.

Usage

```
generic(incidence, covariance, precision)
```

Arguments

incidence	matrix-like object
covariance	matrix-like object
precision	matrix-like object

Details

A generic random effect stores the incidence and structure matrices in Matrix form, which tries to take advantage of sparsity, if it exists.

Value

A list with elements `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

`genetic` *Build an genetic model*

Description

Check conformity of arguments and return a `genetic` object.

Usage

```
genetic(pedigree, incidence, covariance, precision)
```

Arguments

`pedigree`	object of class 'pedigree'
`incidence`	matrix-like object
`covariance`	matrix-like object
`precision`	matrix-like object

Details

This is a virtual class. No objects are expected to be created directly.

Value

A list with elements `pedigree`, `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

get_efnames *Get names of effects*

Description

Give the names of the (components) of the effects in a breedr_modelframe. Internal function. Not exported.

Usage

```
get_efnames(effects)
```

Arguments

effects	A breedr_modelframe.

get_ntraits *Extract the number of traits*

Description

Extract the number of traits

Usage

```
get_ntraits(x, ...)
```

Arguments

x	a metagene object.
...	Arguments to be passed to methods.

See Also

Other metagene: Extract.metagene, get_pedigree(), ngenerations(), nindividuals()

get_param *Parameters of a breedR component*

Description

Parameters of a breedR component

Usage

```
get_param(x)

## S3 method for class 'remlf90'
get_param(x)

## S3 method for class 'breedr_modelframe'
get_param(x)

## S3 method for class 'effect_group'
get_param(x)

## S3 method for class 'spatial'
get_param(x)
```

Arguments

x	Some breedR component.

Methods (by class)

- get_param(remlf90): Get the param from a remlf90 object
- get_param(breedr_modelframe): Get the param from a breedr_modelframe object. Internal function.
- get_param(effect_group): Get the param from a effect_group object. Internal function.
- get_param(spatial): Get the param from a spatial object

get_pedigree *Get the Pedigree from an object*

Description

Returns an object from the formal class pedigree.

Usage

```
get_pedigree(x, ...)

## S3 method for class 'metagene'
get_pedigree(x, ...)

## S3 method for class 'remlf90'
get_pedigree(x, ...)

## S3 method for class 'breedr_modelframe'
get_pedigree(x, ...)

## S3 method for class 'effect_group'
get_pedigree(x, ...)

## S3 method for class 'genetic'
get_pedigree(x, ...)
```

Arguments

x	object to extract pedigree from
...	Arguments to be passed to methods.

Methods (by class)

- `get_pedigree(metagene)`: Get the pedigree from a `metagene` object
- `get_pedigree(remlf90)`: Get the pedigree from a remlf90 object
- `get_pedigree(breedr_modelframe)`: Get the pedigree from a breedr_modelframe object. Internal function.
- `get_pedigree(effect_group)`: Get the pedigree from a effect_group object. Internal function.
- `get_pedigree(genetic)`: Get the pedigree from a `genetic` object

References

`pedigree-class` from package `pedigreemm`

See Also

Other metagene: `Extract.metagene`, `get_ntraits()`, `ngenerations()`, `nindividuals()`

get_structure *Covariance structure of a breedR component*

Description

This generic function returns the covariance or precision matrix of a breedR random effect or a group of effects.

Usage

```
get_structure(x)

## S3 method for class 'breedR'
get_structure(x)

## S3 method for class 'effect_group'
get_structure(x)

## S3 method for class 'breedr_effect'
get_structure(x)
```

Arguments

x	A breedr_effect.

Details

For effect_groups, it returns the common structure of all the elements in the group.

Methods (by class)

- get_structure(breedR): Return the structure matrices of all structured random effects
- get_structure(effect_group): Check that all elements share the same structure and return it.
- get_structure(breedr_effect): Return the structure matrix with an attribute indicating its type.

globulus *Eucalyptus Globulus dataset*

Description

Open-pollinated field test with one generation of 1021 individuals.

Format

A dataframe with 1021 individuals and the following 9 variables

- `self` id of the tree
- `dad` id of sire or 0 if unknown
- `mum` id of dam or 0 if unknown
- `gen` generation (there is only 1)
- `gg` genetic group
- `bl` block
- `phe_X`observed phenotype
- `x, y` coordinates (in m)

Details

The individuals are split into 14 genetic groups and arranged in blocks. The plantation is gridded with a separation of 3 m.

Examples

```
data(globulus)
```

install_progsf90 *Install PROGSF90 binary dependencies*

Description

Copy the binaries for the specified platform into a directory.

Usage

```
install_progsf90(
  url = breedr_progsf90_repo(),
  dest = system.file("bin", package = "breedR"),
  platform = breedR.os.type(),
  arch = breedR.os.32or64bit(),
  quiet = !interactive()
)
```

Arguments

url	where to download the files from
dest	destination directory for the binaries. Default is 'bin' under the current installation dir.
platform	what version of the binaries are to be installed. Default is current platform.
arch	Either "32" or "64". Coerced to string if necessary.
quiet	logical. Whether not to display messages.

Details

The url can be either of form http:// or of form file:// for local urls.

---

is_numericlog *Test whether a string from a REML log is numeric*

Description

A numeric string is a text string that contains only numbers that can be expressed in scientific format.

Usage

```
is_numericlog(x)
```

Arguments

x	character vector.

Details

Some alphabetic symbols are expected. For instance, in 1.23E-02. However, while spaces are expected, an empty line is not considered numeric.

Value

logical value.

Examples

```
breedR:::is_numericlog("  1.23E-02   1  ")
breedR:::is_numericlog("  var   1  ")
breedR:::is_numericlog("   ")
```

larix *Longitudinal Larix dataset*

Description

Repeated measurements of microdensity data along 16 years, with climatic covariates.

Format

A dataframe with 11897 observations of the following variables

- `self` id of the tree
- `dad` id of sire
- `mum` id of dam
- `x, y` coordinates (in rows/cols)
- `rep` factor replicate with 8 levels
- `bl` factor block with 40 levels nested within rep
- `yr` factor (growth) year with 16 ordered levels
- `map` mean annual precipitation
- `mat` mean annual temperature
- `mi` martone index
- `LAS` phenotype LAS
- `DOS` phenotype DOS

Details

Each one of the 8 replicate is composed of 5 incomplete blocks, which gives a nested variable with 40 levels.

Examples

```
library(tidyr)
library(dplyr)
library(ggplot2)
data(larix)

## N observations by year and replicate
with(larix, table(yr, rep))

## Mean response evolution by replicate
larix %>%
  group_by(yr, rep) %>%
  summarise(MLAS = mean(LAS)) %>%
  ggplot(aes(yr, MLAS, group = rep)) +
  geom_line()
```

```
## Visualise trial by year
ggplot(larix, aes(x, y)) +
  geom_tile(aes(fill = LAS)) +
  facet_wrap(~ yr)

## Correlations with environmental variables
if (require(GGally)) {
  ggpairs(larix[, c('map', 'mat', 'mi', 'LAS', 'DOS')])
}
```

`loc_grid` *Lattice of spatial locations*

Description

Returns a list row and column coordinates of observations

Usage

```
loc_grid(coordinates, autofill)
```

Arguments

`coordinates`	two-column matrix-like set of row and column coordinates of observational units
`autofill`	logical. If TRUE (default) it will try to fill missing rows or columns with missing observations. Otherwise, will treat individuals as neighbours even if they are across an empty line.

Value

list of row and column coordinates of spatial units

This functions converts observational coordinates into spatial coordinates. While in a observational dataset there can possibly be many or missing observations at one single location, this function returns the list of row and column coordinates of a grid which contains all the observations

m1 *A small Metagene synthesized dataset*

Description

Simulated progeny of one single generation under phenotypic selection and spatial arrangement. The dataset includes the true Breeding values, the environmental effect and the observed phenotype for 1760 individuals.

Details

The generation of founders (generation 0) consist in 80 independent couples, each of which produces 20 descendants (10 of each sex).

The full dataset contains data for the $2 \times 80 + 1600 = 1760$ simulated individuals. However, the Metagene program does not simulate phenotypes for the founders.

The 1600 descendants were arranged at random in a 40×40 spatial grid

Examples

```
# Load, summarize and visualize data
data(m1)
summary(m1)
plot(m1)

# Environmental component of the phenotype
# (spatially structured)
plot(m1, type = 'spatial')

# The phenotypes are noisy observations around the
# true Breeding values with a standard deviation of about 7.3.
library(ggplot2)
qplot(BV_X, phe_X-BV_X, colour = dad, data = as.data.frame(m1)) +
  geom_abline(intercept=0, slope=0, col='gray')
```

m4 *Metagene synthesized dataset with four generations*

Description

Simulated progeny of four generations under phenotypic selection and spatial arrangement. The dataset includes the true Breeding values, the environmental effect and the observed phenotype for the 6560 individuals.

Details

The generation of founders (generation 0) consist in 80 independent couples, each of which produces 20 descendants (10 of each sex). The second generation descend from 80 random couples of the best individuals among the 1600 members of the first generation. The same procedure is simulated to obtain the third and fourth generations.

The full dataset contains data for the $2 \times 80 + 4 \times 1600 = 6560$ simulated individuals. However, the Metagene program does not simulate phenotypes for the founders.

The 6400 individuals from generations 1 to 4 were arranged at random in a 80×80 spatial grid

Examples

```
# Load, summarize and visualize data
data(m4)
summary(m4)
plot(m4)

# Environmental component of the phenotype
# (spatially structured)
plot(m4, type = 'spatial')

# The phenotypes are noisy observations around the Breeding values
# with a standard deviation of about 10.
library(ggplot2)
qplot(BV_X, phe_X-BV_X, facets = .~gen, data = as.data.frame(m4)) +
  geom_abline(intercept=0, slope=0, col='gray')
```

`neighbours.at` *'move' an arrangement in a given direction*

Description

'move' an arrangement in a given direction

Usage

```
neighbours.at(x, dir)

## S3 method for class 'matrix'
neighbours.at(x, dir)

## S3 method for class 'list'
neighbours.at(x, dir)
```

Arguments

`x`	matrix or list of matrices
`dir`	a *direction* in ('N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW')

Methods (by class)

- `neighbours.at(matrix)`: Returns the matrix of neighbours in the specified direction
- `neighbours.at(list)`: Returns the list of matrices of neighbours in the specified direction

---

ngenerations *Number of generations*

Description

Number of generations

Usage

```
ngenerations(x, ...)
```

Arguments

x	a metagene object.
...	Arguments to be passed to methods.

See Also

Other metagene: `Extract.metagene`, `get_ntraits()`, `get_pedigree()`, `nindividuals()`

---

nindividuals *Number of individuals*

Description

Number of individuals

Usage

```
nindividuals(x, ...)
```

Arguments

x	a metagene object.
...	Arguments to be passed to methods.

See Also

Other metagene: `Extract.metagene`, `get_ntraits()`, `get_pedigree()`, `ngenerations()`

node2lattice_mapping *Define mapping between a lattice and nodes*

Description

Borrowed from INLA

Usage

```
node2lattice_mapping(nrow, ncol)
```

normalise_coordinates *Normalise coordinates specification*

Description

If checks succeed, returns a complete normalised specification.

Usage

```
normalise_coordinates(x, where = "")
```

Arguments

x	matrix-like object to be checked
where	string. Model component where coordinates were specified. For error messages only. E.g. `where = 'genetic component'`.

Value

a two-column data.frame, with numeric values.

parse.txtmat *Parse a matrix from a text output robustly*

Description

Each row of the matrix is a string. If rows are too long, they can continue in another line. Hence, the number of lines might be a multiple of the number of columns

Usage

```
parse.txtmat(x, names = NULL, square = TRUE)
```

Arguments

x	A character vector with space-separated numbers
names	A character vector with row and column names for the output matrix.
square	logical. Whether to assume that the matrix is square. If the matrix is not necessarily

permanent_environmental_competition *Build an permanent-environmental competition model*

Description

Given the coordinates of observations, and the competition decay parameter, build a permanent_environmental_competiti model.

Usage

```
permanent_environmental_competition(coordinates, decay, autofill = TRUE)
```

Arguments

coordinates	two-column matrix-like set of row and column coordinates of observational units
decay	numeric. The positive value of the decay parameter α. Typically 1 or 2. See Details.
autofill	logical. If TRUE (default) it will try to fill missing rows or columns with missing observations. Otherwise, will treat individuals as neighbours even if they are across an empty line.

Value

An object inheriting from competition with the incidence and structure matrices for the random effect.

Examples

```
dat <- data.frame(id   = 1:5,
                  sire = c(11, 11, 2, 3, 2),
                  dam  = c(12, NA, 1, 12, 1),
                  x    = c(rep(1:2, times = 2), 3),
                  y    = c(rep(1:2, each = 2), 3))
breedR:::permanent_environmental_competition(coord = dat[, c('x', 'y')], decay = 2)
```

---

pf90_code_missing *Determine a numeric code for missing observations*

Description

This function returns a code outside the range of variation of the observed values.

Usage

```
pf90_code_missing(x)
```

Arguments

x	a numeric vector.

Details

If the range of variation strictly excludes zero, then it is used as the code for missing observations (which is the default code in PROGSF90). Otherwise, returns 1 - the second power of 10 greater than the maximum observed magnitude. This ensures a code an order of magnitude greater than the observed values. E.g., for observations in the range -40 – 28, the code is -999.

Examples

```
breedR:::pf90_code_missing(rnorm(100))
```

pf90_default_heritability *Default formula for heritability*

Description

If all random effects are independent, and there is an additive-genetic effect, computes a default formula in PROGSF90 notation by dividing the genetic variance by the sum of all variance components plus the residual variance.

Usage

```
pf90_default_heritability(rglist, traits = NULL, quiet = FALSE)
```

Arguments

rglist	list of random groups in the parameters of a progsf90 object
traits	A character vector with trait names, or NULL for single trait.
quiet	logical. If FALSE, the function issues a message when it fails to build a formula.

Value

A character vector with one option specification per trait.

References

http://nce.ads.uga.edu/wiki/doku.php?id=readme.aireml#options

plot.remlf90 *Spatial plot of a model's fit components*

Description

Plots the predicted values of the component effects of the phenotype.

Usage

```
## S3 method for class 'remlf90'
plot(
  x,
  type = c("phenotype", "fitted", "spatial", "fullspatial", "residuals"),
  z = NULL,
  ...
)
```

Arguments

x	A breedR object
type	Character. Which component is to be represented in the map.
z	Optional. A numeric vector to be plotted with respect to the spatial coordinates. Overrides type.
...	Further layers passed to ggplot.

---

progsf90 *progsf90 class*

Description

This function parses a model frame and extracts the relevant fields that are to be written in the parameter, data and auxiliary files of the progsf90 programs.

Usage

```
progsf90(mf, weights, effects, opt = c("sol se"), res.var.ini = 10)
```

Arguments

mf	model.frame for fixed and diagonal random effects
weights	a vector of weights for the residual variance
effects	breedr_modelframe
opt	character. Options to be passed to Misztal's programs
res.var.ini	positive number. Initial value for the residual variance.

---

random *Constructor for a random effect*

Description

The random-class is virtual. No object should be directly created with this constructor. This constructor is to be called from within subclasses like generic, diagonal, spatial or additive_genetic.

Usage

```
random(incidence, covariance, precision)
```

Arguments

incidence	matrix-like object
covariance	matrix-like object
precision	matrix-like object

Details

This constructor performs the arguments and conformance checks. But the implementation details (i.e., storage format and handling) is left for the subclasses.

Value

A list with elements `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

---

ranef *Extract the modes of the random effects*

Description

Extract the conditional modes of the random effects from a fitted model object. For linear mixed models the conditional modes of the random effects are also the conditional means.

Usage

```
## S3 method for class 'remlf90'
ranef(object, ...)
```

Arguments

`object`	a fitted models with random effects of class `remlf90`.
`...`	not used

Details

This method is modeled a bit after `ranef`. However, it is independent and does not inherit from it. In particular, it always returns the conditional variance (argument condVar in lme4).

Value

An object of class `ranef.breedR` composed of a list of vectors or matrices (multitrait case), one for each random effect. The length of the vectors are the number of levels of the corresponding random effect.

Each random effect has an attribute called `"se"` which is a vector with the standard errors.

Additionally, depending of the nature of the random effect, there may be further attributes. The pedigree will be given for genetic random effects and the spatial prediction grid for the spatial random effects

Note

To produce a (list of) "caterpillar plots" of the random effects apply `plot` to the result of a call to `ranef`.

Examples

```
res <- remlf90(phe_X ~ bl,
               genetic = list(model = 'add_animal',
                              pedigree = globulus[, 1:3],
                              id = 'self'),
               data = globulus)
str(rr <- ranef(res))
plot(rr)
```

`read.metagene` *Metagene Data Input*

Description

A basic interface to the metagene simulator.

This function returns the data frame with the pedigree and phenotypes of the simulated individuals.

Usage

```
read.metagene(fname)

## S3 method for class 'metagene'
summary(object, ...)

## S3 method for class 'summary.metagene'
print(x, ...)

## S3 method for class 'metagene'
plot(x, type = c("default", "spatial"), ...)

## S3 method for class 'metagene'
get_ntraits(x, ...)

## S3 method for class 'metagene'
ngenerations(x, ...)

## S3 method for class 'metagene'
nindividuals(x, exclude.founders = FALSE, ...)

## S3 method for class 'metagene'
as.data.frame(x, ..., exclude.founders = TRUE)
```

Arguments

`fname`	file name of the second metagene output file (usually: insim.002)
`object`	a metagene object

...	not used
x	a metagene object
type	character. If 'default', the empirical density of the breeding and phenotypic values will be represented by generation. If 'spatial', the map of the spatial component will be plotted.
exclude.founders	logical: should the data.frame contain the genetic values of the founders?

Details

Read the output file (the one with the pedigree. Usually: insim.002) of the Metagene program, and build an object of class metagene.

Value

the data in the file as an object of class `metagene`

returns a data frame with one row per individual, with the spatial coordinates if applicable, the pedigree information, the generation, the true breeding value, the phenotype, the sex, the spatially structured component of the phenotype and other internal metagene variables.

Functions

- `summary(metagene)`: summary of a metagene object
- `print(summary.metagene)`: print summary method
- `plot(metagene)`: plots either genetic and phenotypic values, or the spatial component of the phenotype. Pass further `ggplot` layers in ...
- `get_ntraits(metagene)`: gets the number of traits
- `ngenerations(metagene)`: gets the number of generations
- `nindividuals(metagene)`: gets the number of individuals
- `as.data.frame(metagene)`: Coerce to a data.frame

References

http://www.igv.fi.cnr.it/noveltree/

---

remlf90 *Inference with REMLF90*

Description

Fits a Linear Mixed Model by Restricted Maximum Likelihood

Usage

```
remlf90(
  fixed,
  random = NULL,
  genetic = NULL,
  spatial = NULL,
  generic = NULL,
  data,
  var.ini = NULL,
  method = c("ai", "em"),
  breedR.bin = breedR.getOption("breedR.bin"),
  progsf90.options = NULL,
  weights = NULL,
  debug = FALSE
)
```

Arguments

`fixed`	an object of class formula (or one that can be coerced to that class): a symbolic description of the fixed effects of the model to be fitted. The details of model specification are given under 'Details'.
`random`	if not `NULL`, an object of class formula with the unstructured random effects.
`genetic`	if not `NULL`, a list with relevant parameters for an additive genetic effect; see 'Details'.
`spatial`	if not `NULL`, a list with relevant parameters for a spatial random effect; see 'Details'.
`generic`	if not `NULL`, a named list with an `incidence` matrix and either a `covariance` or a `precision` matrix; see 'Details'.
`data`	a data frame with variables and observations
`var.ini`	if not `NULL`, a named list with one item for each random component in the model. See 'Details'.
`method`	either 'ai' or 'em' for Average-Information or Expectation-Maximization REML respectively
`breedR.bin`	character. The local directory where the package binaries are stored, or any of 'remote' or 'submit' for remote computing. See 'Details'.
`progsf90.options`	character. Passed directly to OPTIONS field in PROGSF90. See available options for REMLF90 and for AIREMLF90. Option `sol se` is passed always and cannot be removed. No checks are performed, handle with care.
`weights`	numeric. A vector of weights for the residual variance.
`debug`	logical. If `TRUE`, the input files for blupf90 programs and their output are shown, but results are not parsed.

Details

If either `genetic` or `spatial` are not `NULL`, the model residuals are assumed to have an additive genetic effects or a spatially structured random effect, respectively. In those cases, `genetic` and `spatial` must be lists with named relevant parameters.

The `generic` component implements random effects with custom incidence and covariance (or precision) matrices. There can be any number of them, stored in a named list with custom unique names that will be used to identify and label the results. Each effect in the list must have an `incidence` argument and either a `covariance` or a `precision` matrix with conforming and suitable dimensions. Optionally, an initial variance for the REML algorithm can be specified in a third argument `var.ini`.

Genetic effect: The available models for the genetic effect are `add_animal` and `competition`. `add_animal` stands for an additive animal model with a given pedigree. `competition` includes the *direct* additive genetic effect and also a *competition* additive genetic effect possibly correlated with the former.

The minimum elements in the list of the `genetic` component are:

- `model` a string, either `add_animal` or `competition`
- `pedigree` either an object of class `pedigree` (see `build_pedigree`) or a *data.frame* with exactly three columns identifying the individual, his father and his mother respectively
- `id` either a vector of codes or the name of the variable with the individual identifier in the `data`

Optional common components are:

- `var.ini` a positive initial value for the variance component(s). For a `competition` model, the same initial value is used for both effects, with a negative initial correlation between them of half this value.

Finally, for model `competition` there are further mandatory and optional elements:

- mandatory elements
  - `coord` a matrix, list or data.frame with two columns for the rows and columns of the observations, respectively. This element is necessary even if duplicated in a `spatial` component.
  - `competition_decay` a positive number. The intensity of competition is weighted by the distance according to $1/d^{\alpha}$. This element specifies the exponent α to be used. Typically 1 or 2.
- optional elements
  - `pec` Permanent Environmental Competition effect. If present, this must be a named list with elements `present` which is either `TRUE` or `FALSE` and (optionally) `var.ini` specifying the initial variance for this effect.

The Permanent Environmental Competition (`pec`) effect is actually non-genetic in nature. However, it was included as an option to the (genetic) competition effect as it is usually used in conjunction with it.

Spatial effects: The available models for the spatial effect are `splines` and `AR1`. `splines` uses a two-dimensional tensor product of B-splines to represent the smooth spatially structured effect (Cappa and Cantet, 2007). `AR1` uses a kronecker product of autoregressive models for the rows and columns (Dutkowski et al., 2002).

In both cases, the minimum necessary components in the list are

- `model` a string, either `splines` or `AR`
- `coord` a matrix, list or data.frame with two columns for the rows and columns respectively.

Optional common components are

- `var.ini` a positive initial value for the variance component

Finally, optional model-dependent components are

- For model `splines`
  - `n.knots` a vector of two integers with the number of *internal* knots for the rows and columns of the spline design
- For model `AR`
  - `rho` a vector of two numbers strictly between -1 and 1 with the autoregressive parameters for the rows and columns respectively. Alternatively, a matrix or data.frame with two columns where every row contain a combination of autoregressive parameters to be tried.

The *internal* knots cover the region with observations at regular intervals (in each dimension). For the splines design, three additional knots are automatically added before the first internal knot, and other tree after the last one, in each dimension. As a result, if `n.knots = c(n1, n2)`, then the final number of parameters of the splines model is $(n1 + 2)(n2 + 2)$.

If `n.knots` is omitted, a sensible number of knots for each dimension is computed based on the number of observations in the experiment. See the internal function `determine.n.knots`. This is the default function that computes the default number of knots, but you can provide an alternative default function through `breedR.setOption`.

Due to limitations of the REML backend, we can only fit models with *fixed* autoregressive parameters. `remlf90()` will fit as many models as rows in `rho`, and return the results of the most likely. It will also return the list of log-likelihoods for each tried combination of autoregressive parameters, in the component `rho` of the `reml` object. This is useful for visualization, and further refinement of the search for appropriate parameters.

If any of the values in either column of `rho` is `NA`, then a default set of values for the corresponding dimension will be set. See `breedR.getOption('ar.eval')`, for the current defaults. You can set your own defaults with `breedR.setOption`. Each will be combined with every other value in the other column.

Omitting the specification of `rho` is equivalent to `rho = c(NA, NA)`.

Intercept: An intercept is automatically introduced in the model provided the user doesn't explicitly prevents it by using `0` or `-1` in the `fixed` formula (as conventional in `R`), *and* there are no other categorical covariates in `fixed`. The latter condition is actually a limitation of (ai)remlf90 backends, which would in any case return an estimate for each level of the categorical covariates while returning 0 for the intercept. It does not allow alternative parameterizations.

Initial variances: Initial variance components can also be specified through an additional argument `var.ini`. You can either use default initial values for the variance components (see `?breedR.options`) or specify custom values for *each* and *all* variance components in the model. In this case, `var.ini` must be a named list with one element for each term in `random` with matching names, plus one last element named `residual` for the initial residual variance. Furthermore if there are `genetic` or `spatial` effects, they must as well include a numeric element `var.ini` with the initial variance component specification for the corresponding effect.

Inference method: AI-REML is usually faster than EM-REML, and it provides more results. Namely, standard errors of the variance components estimates, and covariances as well. On the

other hand, is less robust than EM-REML and it usually gives extreme results when used with the splines spatial model (as in `spatial = list(model ='splines', ...)`).

Even when an effect accounts for no variance at all, EM-REML will always estimate a positive variance which will be determined by the starting value. If AI-REML does not converge but EM-REML does with the same dataset and model, re-run EM-REML with a small starting value for the effect. If the estimate does not change, it is likely that there is no variance.

Remote computing: If `breedR.bin = 'remote'`, the REML program will be run remotely and the results will be automatically transferred back automatically. While if `breedR.bin = 'submit'` the job will be submitted to the server, and the job-id and other relevant information about the model will be returned instantly. The returned object can be used to retrieve the results or check the status of the job. Several jobs can be submitted in parallel. See `?remote` to learn how to configure breedR for remote computing, and how to manage submitted jobs.

Value

An object of class 'remlf90' that can be further questioned by `fixef`, `ranef`, `fitted`, etc.

References

progsf90 wiki page: http://nce.ads.uga.edu/wiki/doku.php

E. P. Cappa and R. J. C. Cantet (2007). Bayesian estimation of a surface to account for a spatial trend using penalized splines in an individual-tree mixed model. *Canadian Journal of Forest Research* **37**(12):2677-2688.

G. W. Dutkowski, J. Costa e Silva, A. R. Gilmour, G. A. López (2002). Spatial analysis methods for forest genetic trials. *Canadian Journal of Forest Research* **32**(12):2201-2214.

See Also

`pedigree`

Examples

```
## Linear model
n <- 1e3
dat <- transform(data.frame(x = runif(n)),
                 y = 1 + 2*x + rnorm(n, sd = sqrt(3)))
res.lm <- remlf90(fixed = y ~ x, data = dat)
summary(res.lm)

## Linear Mixed model
f3 = factor(sample(letters[1:3], n, replace = TRUE))
dat <- transform(dat,
                 f3 = f3,
                 y = y + (-1:1)[f3])
res.lmm <- remlf90(fixed  = y ~ x,
                   random = ~ f3,
                   data   = dat)

## Generic model (used to manually fit the previous model)
```

```
inc.mat <- model.matrix(~ 0 + f3, dat)
cov.mat <- diag(3)
res.lmm2 <- remlf90(fixed  = y ~ x,
                    generic = list(f3 = list(inc.mat,
                                             cov.mat)),
                    data   = dat)
all.equal(res.lmm, res.lmm2, check.attributes = FALSE)  # TRUE

## Animal model
ped <- build_pedigree(c('self', 'dad', 'mum'),
                      data = as.data.frame(m1))
res.am <- remlf90(fixed   = phe_X ~ sex,
                  genetic = list(model    = 'add_animal',
                                 pedigree = ped,
                                 id       = 'self'),
                  data    = as.data.frame(m1))

## Not run:
## Same model with specification of initial variances
res.am <- remlf90(fixed   = phe_X ~ sex,
                  genetic = list(model    = 'add_animal',
                                 pedigree = ped,
                                 id       = 'self',
                                 var.ini  = 1),
                  data    = as.data.frame(m1),
                  var.ini = list(resid = 1))

## Animal-spatial models
gen.globulus <- list(model    = 'add_animal',
                     pedigree = globulus[, 1:3],
                     id       = 'self')
res.bm <- remlf90(fixed   = phe_X ~ gg,
                  genetic = gen.globulus,
                  spatial = list(model = 'blocks',
                                 coord = globulus[, c('x','y')],
                                 id    = 'bl'),
                  data    = globulus)

res.am <- remlf90(fixed   = phe_X ~ gg,
                  genetic = gen.globulus,
                  spatial = list(model = 'AR',
                                 coord = globulus[, c('x','y')],
                                 rho   = c(.85, .8)),
                  data    = globulus)

res.sm <- remlf90(fixed   = phe_X ~ gg,
                  genetic = gen.globulus,
                  spatial = list(model = 'splines',
                                 coord = globulus[, c('x','y')],
                                 n.knots = c(5, 5)),
                  data    = globulus,
                  method  = 'em')   # Necessary for splines models!!!
```

```
## Competition models

# This may take some minutes...
# and need to be fitted with 'em'
res.cm <- remlf90(fixed   = phe_X ~ 1,
                  genetic = list(model = 'competition',
                                 pedigree = globulus[, 1:3],
                                 id = 'self',
                                 coord = globulus[, c('x','y')],
                                 competition_decay = 1,
                                 pec = list(present = TRUE)),
                  method = 'em',
                  data = globulus)

## End(Not run)
```

`remote` *Control and view a remote breedR-queue of submitted jobs*

Description

Control and view a remote breedR-queue of submitted jobs

Usage

```
breedR.qget(id, remove = TRUE)

breedR.qdel(id)

breedR.qstat(id)

breedR.qnuke()

breedR.remote_load(retry = 5)
```

Arguments

`id`	The job-id which is the output from `breedR` when the job is submitted, the job-number or job-name. For `breedR.qstat`, `id` is optional and if omitted all the jobs will be listed.
`remove`	Logical. If FALSE, leave the job on the server after retrieval, otherwise remove it (default).
`retry`	numeric. In case of connection failure, number of times to retry before giving up and returning `NA`.

Details

breedR.qstat shows job(s) on the server, breedR.qget fetches the results (and by default remove the files on the server), breedR.qdel removes a job on the server and breedR.qnuke removes all jobs on the server.

Finally, breedR.remote_load returns the current load in the server, as a percent. This should be used to check whether jobs can be safely submitted, and this is left to the user.

The recommended procedure is to use r <- remlf90(..., breedR.bin="submit") and then do r <- breedR.qget(r) at a later stage. If the job is not finished, then r will not be overwritten and this step can be repeated. The reason for this procedure, is that some information usually stored in the result object does not go through the remote server, hence have to be appended to the results that are retrieved from the server. Hence doing r <- remlf90(..., breedR.bin="submit") and then later retrieve it using r <- breedR.qget(1), say, then r does not contain all the required information.

Value

breedR.qstat returns an breedR.q-object with information about current jobs.

Remote computing under Windows

You need to install cygwin and ssh beforehand.

Setup

You need to configure the client and server machines so that passwordless SSH authentication works. See for example here

Furthermore, you need to configure breedR by setting the options remote.host, remote.user, remote.port and remote.bin. You can permanently set these options in the file .breedRrc in your home directory. See ?breedR.setOption.

See Also

remlf90

Examples

```
## Not run:
r = remlf90(y~1, data = data.frame(y=rnorm(10)), breedR.bin = "submit")
summary(r)       # shows its status, same as breedR.qstat(r)
breedR.qstat()   # shows all jobs
r = breedR.qget(r, remove=FALSE)
breedR.qdel(1)
breedR.qnuke()
summary(r)   # results of the analysis

## End(Not run)
```

renderpf90 *Render a progsf90 effect*

Description

Translates breedR effects into progsf90 parameters and data.

Usage

```
renderpf90(x)

## Default S3 method:
renderpf90(x)

## S3 method for class 'fixed'
renderpf90(x)

## S3 method for class 'diagonal'
renderpf90(x)

## S3 method for class 'breedr_modelframe'
renderpf90(x, ntraits, weights)

## S3 method for class 'effect_group'
renderpf90(x)

## S3 method for class 'generic'
renderpf90(x)

## S3 method for class 'additive_genetic_animal'
renderpf90(x)

## S3 method for class 'additive_genetic_competition'
renderpf90(x)

## S3 method for class 'permanent_environmental_competition'
renderpf90(x)

## S3 method for class 'splines'
renderpf90(x)

## S3 method for class 'blocks'
renderpf90(x)

## S3 method for class 'ar'
renderpf90(x)
```

Arguments

x	object of class breedr_modelframe, effect_group or breedr_effect.
ntraits	integer. Number of traits in the model.
weights	logical. Whether there is an additional column of weights.

Details

This is an internal function. Not exported.

For the generic class, all matrices are converted to plain matrix-class, for exporting to files. The progsf90 model is either user_file or user_file_i depending on the type of structure matrix; i.e. respectively precision or covariance.

For the splines class, everything reduces to a generic effect with a covariance matrix

For the blocks class, everything reduces to a generic effect with a covariance matrix

For the ar class, everything reduces to a generic effect with a precision matrix

Value

The number of levels and type for each 'virtual' effect; the progsf90 model-name as appropriate; a file name and its content.

Methods (by class)

- renderpf90(default): For unknown classes, just returns the object untouched with a warning.
- renderpf90(fixed): For fixed effects.
- renderpf90(diagonal): For diagonal effects. Assumed grouping variables.
- renderpf90(breedr_modelframe): Render a full breedr_modelframe
- renderpf90(effect_group): Render groups of effects into pf90 code
- renderpf90(generic): Compute the parameters of a progsf90 representation of a generic effect.
- renderpf90(additive_genetic_animal): Compute the parameters of a progsf90 representation of a additive_genetic_animal effect.
- renderpf90(additive_genetic_competition): Compute the parameters of a progsf90 representation of a additive_genetic_competition effect.
- renderpf90(permanent_environmental_competition): Compute the parameters of a progsf90 representation of a permanent_environmental_competition effect. Has the same incidence matrix than the additive_genetic_competition effect but an unstructured covariance matrix.
- renderpf90(splines): Compute the parameters of a progsf90 representation of a splines effect.
- renderpf90(blocks): Compute the parameters of a progsf90 representation of a blocks effect.
- renderpf90(ar): Compute the parameters of a progsf90 representation of an AR effect.

See Also

Other renderpf90: `renderpf90.matrix()`

`renderpf90.matrix` *Render a sparse matrix into non-zero values and column indices*

Description

This function provides a representation of a sparse matrix suited to progsf90.

Usage

```
## S3 method for class 'matrix'
renderpf90(x)
```

Arguments

`x`	object of class breedr_modelframe, effect_group or breedr_effect.

Details

For each row, it keeps the non-zero elements and their respective column index. The gaps are filled with zeros.

Value

A matrix with a number of columns equal to twice the maximum number of non-zero elements in one row. The first half of the columns are the non-zero values (except for filling-in) while the second half are the column indices.

For indicator matrices (i.e. each row has at most one non-zero value of 1), it returns a one-column matrix of the corresponding column indices.

See Also

Other renderpf90: `renderpf90()`

retrieve_remote *Retrieve results stored in some remote directory*

Description

Use scp to transfer compressed files. Clean up afterwards.

Usage

```
retrieve_remote(rdir)
```

Arguments

rdir	string. Remote directory where the results are stored.

Value

dir name where the results are retrieved

sim.spatial *Simulate a spatial structure*

Description

Takes a metagene simulated dataset and distributes its individuals randomly into a more or less square spatial region. Furthermore, it takes part of the phenotypic noise and puts some spatial structure into it.

Usage

```
sim.spatial(meta, variance, range, ...)
```

Arguments

meta	A metagene object
variance	A number between 0 and 1. The variance of the spatial field as a proportion of the non-inheritable phenotypic variance. See Details.
range	A number between 0 and 1. The range of the spatial field, as a proportion of the region size.
...	Arguments to be passed to methods.

Details

Founders are not put into place, as they don't have phenotypic values. Therefore, they are returned without coordinates nor spatial values.

The variance of the spatial field is given as a proportion of the variance of the random noise that was added to the Breeding Values to produce the phenotypes. The phenotypes are afterwards recalculated to preserve the heritability.

The spatial unit is the distance between consecutive trees.

Value

Another metagene object with spatial structure given through an additional column `sp_X` with the spatially structured component of the phenotype, and a 'spatial' list element with coordinates and simulation information

---

`simulation` *Simulation of phenotypes and model components*

Description

These functions allow to draw samples from several models (spatial, genetic, competition, etc.) and to combine them to produce a simulated phenotype. The resulting dataset can then be fitted with breedR to compare the estimations with the true underlying parameters. `breedR.sample.phenotype` is the main function in the group, as it makes use of the rest to simulate a phenotype's components.

Usage

```
breedR.sample.phenotype(
  fixed = NULL,
  random = NULL,
  genetic = NULL,
  spatial = NULL,
  residual.variance = 1,
  N = NULL
)

breedR.sample.AR(size, rho, sigma2, N = 1)

breedR.sample.splines(coord, nkn, sigma2, N = 1)

breedR.sample.BV(ped, Sigma, N = 1)

breedR.sample.pedigree(Nobs, Nparents, check.factorial = TRUE)

breedR.sample.ranef(dim, var, Nlevels, labels = NULL, N = Nlevels, vname = "X")
```

Arguments

fixed	a numeric vector of regression coefficients.
random	a list of random effects specifications, where each element is itself a list with elements nlevels and sigma2.
genetic	a list with the additive genetic effect specifications. See Details.
spatial	a list with the spatial effect specifications. See Details.
residual.variance	is a positive number giving the value of the residual variance.
N	number of simulated individuals. If spatial is specified, N is overridden by the product of spatial$grid.size. Otherwise it is required. If genetic is specified, N is the size of the offspring only.
size	numeric. A vector of length two with the number of rows and columns in the field trial
rho	numeric. A vector of length two with the autocorrelation parameters for the row and column autoregressive processes
sigma2	numeric. The marginal variance
coord	numeric. A two-column matrix(-like) with spatial coordinates.
nkn	numeric. A vector of length two with the number of (inner) knots in each dimension
ped	a pedigree object
Sigma	numeric. The additive genetic variance. Either a variance for a single additive genetic effect, or a positive-definite matrix with the covariance structure for a set of correlated genetic effects
Nobs	numeric. Number of individuals to sample
Nparents	numeric. Vector of length two. Number of dams and sires to randomly mate.
check.factorial	logical. If TRUE (default), it checks whether all the possible matings had taken place at least once. If not, it stops with an error.
dim	numeric. Dimension of the effect (e.g. n. of traits)
var	numeric matrix. (Co)variance matrix
Nlevels	numeric. Number of individuals values to sample
labels	character vector of labels for each level.
vname	string. A name for the resulting variables

Details

The design matrix for the fixed effects (if given) is a column of ones and a matrix of random uniform values in (0, 1). Therefore, the first element in fixed gives the overall intercept.

genetic is a list with the following elements:

- model a character string, either 'add_animal' or 'competition'. In the former, a single breeding value per individual will be simulated, while in the latter *direct* and *competition* values are simulated.

- Nparents passed to breedR.sample.pedigree.
- sigma2_a numeric. For the add_animal model, the variance of the additive genetic effect. For the competition model, the 2×2 covariance matrix of direct and competition genetic effects. Passed to breedR.sample.BV as Sigma.
- check.factorial passed to breedR.sample.pedigree
- pec numeric. If present, and only under the competition model, it simulates a *Permanent Environmental Competition* effect with the given variance.
- relations character. If present and equals half-sibs it will generate a pedigree with unknown sires, so that relationships in the offspring are either unrelated or half-sibs are possible. Otherwise, both parents are known and full-sibs are also possible.

Note that only one generation is simulated.

spatial is a list with the following elements:

- model a character string, either 'AR' or 'splines'.
- grid.size a numeric vector of length two with the number of rows and columns of trees. Note that the spacing between trees is equal in both dimensions.
- rho/n.knots passed to breedR.sample.AR or to breedR.sample.splines as nkn.
- sigma2_s passed to breedR.sample.AR or to breedR.sample.splines as sigma2.

breedR.sample.AR simulates a two-dimensional spatial process as the kronecker product of first-order autoregressive processes in each dimension.

breedR.sample.splines simulates a two-dimensional spatial process as the kronecker product of B-splines processes in each dimension.

breedR.sample.BV simulates a set of breeding values (BV) given a pedigree

breedR.sample.pedigree simulates a one-generation pedigree from random mating of independent founders. Note that if check.factorial is FALSE, you can have some founders removed from the pedigree.

breedR.sample.ranef simulates a random effect with a given variance.

Examples

```
breedR.sample.phenotype(fixed   = c(mu = 10, x = 2),
                        random = list(u = list(nlevels = 3,
                                               sigma2  = 1)),
                        genetic = list(model    = 'add_animal',
                                       Nparents = c(10, 10),
                                       sigma2_a = 2,
                                       check.factorial = FALSE),
                        spatial = list(model     = 'AR',
                                       grid.size = c(5, 5),
                                       rho       = c(.2, .8),
                                       sigma2_s  = 1),
                        residual.variance = 1)
```

spatial *Build a spatial model*

Description

Check conformity of arguments and return a `spatial` object.

Usage

```
spatial(coordinates, incidence, covariance, precision)
```

Arguments

`coordinates`	two-column matrix-like set of row and column coordinates of observational units
`incidence`	matrix-like object
`covariance`	matrix-like object
`precision`	matrix-like object

Value

A list with elements `coordinates`, `incidence.matrix`, `structure.matrix` and `structure.type`, which is a string indicating either `covariance` or `precision`.

spatial.plot *Plot an spatially arranged continuous variable*

Description

Plot an spatially arranged continuous variable

Usage

```
spatial.plot(dat, scale = c("divergent", "sequential"))
```

Arguments

`dat`	A 3-column data.frame with names 'x', 'y' and 'z' where the first two are the spatial coordinates, and 'z' is the value to be represented
`scale`	Character. 'divergent' represents positive and negative values with different colours. 'sequential' uses a gradient scale of two colours.

splat *'Splat' arguments to a function*

Description

Wraps a function in do.call, so instead of taking multiple arguments, it takes a single named list which will be interpreted as its arguments.

Usage

```
splat(flat)
```

Arguments

flat	function to splat This is useful when you want to pass a function a row of data frame or array, and don't want to manually pull it apart in your function. Borrowed from splat

Value

a function

Examples

```
args <- replicate(3, runif(5), simplify = FALSE)
identical(breedR:::splat(rbind)(args), do.call(rbind, args))
```

validate_variance *Check properties for a covariance matrix*

Description

Check properties for a covariance matrix

Usage

```
validate_variance(
  x,
  dimension = dim(as.matrix(x)),
  what = "var.ini",
  where = ""
)
```

Arguments

x	number or matrix.
dimension	numeric vector with dimensions of the matrix
what	string. What are we validating
where	string. Model component where coordinates were specified. For error messages only. E.g. `where = 'competition specification'`.

Value

`TRUE` if all checks pass

---

variogram *Empirical variograms of residuals*

Description

Computes isotropic and anisotropic empirical variograms from the residuals of a breedR model.

Usage

```
variogram(
  x,
  plot = c("all", "isotropic", "anisotropic", "perspective", "heat", "none"),
  R,
  coord,
  z
)

## S3 method for class 'breedR.variogram'
print(x, minN = 30, ...)
```

Arguments

x	a breedR result.
plot	character. What type of variogram is to be plotted. The default is 'all'. Other options are 'isotropic', 'heat', 'perspective' and 'anisotropic'. See Details.
R	numeric. Radius of the variogram
coord	(optional) a two-column matrix with coordinates of observations
z	(optional) a numeric vector of values to be represented spatially
minN	numeric. Variogram values computed with less than minN pairs of observations are considered *unstable* and therefore are not plotted. Default: 30.
...	not used.

Details

An empirical variogram computes the mean squared differences between observations separated by a vector h, for all possible h. An `isotropic` variogram assumes that the underlying process depends only on the (euclidean) *distance* between points, but not on the orientation or direction of h. At the other end, an `anisotropic` variogram assumes that the process might depend on the orientation, but not on the direction. Finally, `heat` and `perspective` are different representations of a variogram which assumes that the process depends only on the absolute distance between rows and columns.

Unless `coord` or `z` are specified by the user, `variogram` builds the variogram with the residuals of the model fit in `x`. If `coord` or `z` are specified, then the spatial coordinates or the residuals are respectively overridden.

This function assumes that there is at most one observation per spatial location. Otherwise, are observations measured at different times? should a spatial-temporal variogram be fitted?

Functions

- `print(breedR.variogram)`: Print a breedR variogram

See Also

`vgram.matrix`

Examples

```
data(globulus)

# No spatial effect
res <- remlf90(fixed  = phe_X ~ 1,
               random = ~ gg,
               genetic = list(model = 'add_animal',
                              pedigree = globulus[, 1:3],
                              id = 'self'),
               data   = globulus)

Zd <- model.matrix(res)$genetic
PBV <- Zd %*% ranef(res)$genetic

# variogram() needs coordinates to compute distances
# either you use the \code{coord} argument, or you do:
coordinates(res) <- globulus[, c('x', 'y')]

# there is residual autocorrelation
# there is also spatial structure in the Breeding Values
variogram(res)
variogram(res, z = PBV)

# Autoregressive spatial effect
# eliminates the residual autocorrelation
res.sp<- remlf90(fixed  = phe_X ~ 1,
                 spatial = list(model = 'AR',
```

```
                              coord = globulus[, c('x', 'y')],
                              rho = c(.9, .9)),
                  genetic = list(model = 'add_animal',
                              pedigree = globulus[, 1:3],
                              id = 'self'),
                  data   = globulus)

Zd <- model.matrix(res.sp)$genetic
PBV <- Zd %*% ranef(res.sp)$genetic

variogram(res.sp)
variogram(res.sp, z = PBV)
```

`vcov.remlf90` *Covariance matrix of a fitted remlf90 object*

Description

Returns the variance-covariance matrix of the specified random effect.

Usage

```
## S3 method for class 'remlf90'
vcov(
  object,
 effect = c("spatial", "genetic", "genetic_direct", "genetic_competition", "pec"),
  ...
)
```

Arguments

`object`	a fitted model of class `remlf90`
`effect`	the structured random effect of interest
`...`	Not used.

`vgram.matrix` *Variogram of a matrix*

Description

This function is slightly adapted from the homonym function in package `fields`. As such, the all the credit is for their authors.

Usage

```
vgram.matrix(dat, R = 5, dx = 1, dy = 1)
```

Arguments

dat	A matrix.
R	Maximum radius in distance units for computing the variogram.
dx	The spacing between rows of the matrix, in distance units.
dy	The spacing between columns of the matrix, in distance units.

Note

fields, Tools for spatial data Copyright 2004-2013, Institute for Mathematics Applied Geosciences University Corporation for Atmospheric Research Licensed under the GPL – www.gpl.org/licenses/gpl.html

Index