Package: plantmix (via r-universe)

June 10, 2026

Title Genetic Study of Plant Mixtures

Version 1.0.2

Maintainer Timothee Flutre <timothee.flutre@inrae.fr>

Description Fit linear mixed models dedicated to the genetic study of plant mixtures, such as those based on general and specific mixing abilities (GMA-SMA) as well as direct and social breeding values (DBV-SBV), also known as direct and indirect genetic effects (DGE-IGE). More details in Forst et al (2019, <doi:10.1016/j.fcr.2019.107571>) for GMA-SMA models, and Salomon et al (2026, <doi:10.64898/2026.03.27.714849>) for DBV-SBV models. The package also provides functions to optimize experimental designs, simulate data sets and compute interaction indices.

Depends R (>= 4.0.0), ggplot2, lme4

Imports graphics, igraph, MASS, Matrix, methods, Rcpp (>= 1.0.13), stats, TMB (>= 1.9.17), utils

License AGPL-3

Copyright INRAE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests breedR, INLA, emmeans, knitr, MM4LMM, rmarkdown, testthat

Additional_repositories https://famuvie.github.io/breedR, https://inla.r-inla-download.org/R/stable

VignetteBuilder knitr

LinkingTo TMB, Rcpp, RcppEigen

NeedsCompilation yes

Author Timothee Flutre [aut, ctb, cre] (ORCID: <https://orcid.org/0000-0003-4489-4782>), INRAE [cph, fnd], Jerome Enjalbert [ctb] (ORCID: <https://orcid.org/0000-0003-0408-2253>), Emma Forst [ctb] (ORCID: <https://orcid.org/0000-0001-9557-3589>), Maxence

Remerand [ctb] (ORCID:
<https://orcid.org/0009-0007-3217-5613>), Jemay Salomon [ctb]
(ORCID: <https://orcid.org/0000-0003-4454-7916>)

Config/pak/sysreqs cmake libglpk-dev make libxml2-dev

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

Date/Publication 2026-06-09 15:50:07 UTC

RemoteUrl https://github.com/cran/plantmix

RemoteRef HEAD

RemoteSha 24a33b86e4e84f1075d13f853023e9dd3219a82d

Contents

`CC` *Change in contribution*

Description

Computes the change in contribution (CC) proposed by Williams and McCarthy (2001, doi:10.1046/j.14401703.2001.00368.x) (this function implements the equation for mixtures with any number of components as written in appendix V of the paper).

Usage

```
CC(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colY = "yield"
)
```

Arguments

`dat`	data frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the focal yield in each stand (see the example below); a stand with a single proportion equals to 1 will be interpreted as a monovarietal culture ("pure stand"), a mixture otherwise (and its proportions should sum to 1)
`colIDstand`	column name for the stand identifiers
`colIDfocal`	column name for the focal identifiers
`colProp`	column name for the proportions
`colY`	column name for the yield values

Value

input data.frame with an extra column named "CC"

Author(s)

Timothee Flutre

Examples

```
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield=c(50, 40, 25, 22)))
CC(dat)
```

`estimGRM` *Estimate a GRM*

Description

Estimates a genomic relationship matrix with the first estimator from VanRaden (2008).

Usage

```
estimGRM(M, AFs = NULL)
```

Arguments

`M`	matrix of SNP genotypes, with individuals in rows and SNPs in columns, encoded in allele dose in \[0,2\]
`AFs`	vector of allele frequencies; if NULL, will be estimated from `M`

Value

matrix

Author(s)

Timothee Flutre

See Also

`simulGenosDoseStruct`

Examples

```
strong_div_pops <- diag(3)
strong_div_pops[upper.tri(strong_div_pops)] <- 0.5
strong_div_pops[lower.tri(strong_div_pops)] <- strong_div_pops[upper.tri(strong_div_pops)]
strong_div_pops
M <- simulGenosDoseStruct(div_pops=strong_div_pops)
A <- estimGRM(M)
imageMat(A, "Strong divergence")
```

`estimRYMRep` *RYM with "rep" effect*

Description

Computes the relative yield of mixtures (RYMs) per replicate (e.g., blocks), then fits a linear model correcting for this rep effect, and quantifies the uncertainty around it.

Usage

```
estimRYMRep(
  data,
  colY,
  colID,
  colRep,
  mix2genos,
  conf_level = 0.95,
  plotDiag = FALSE,
  title = ""
)
```

Arguments

`data`	data frame
`colY`	column name specifying the response variable
`colID`	column name specifying the stand identifiers
`colRep`	column name specifying the replicate
`mix2genos`	named list with one component pe rmixture, each component being a vector of genotype names
`conf_level`	confidence level for the uncertainty intervals
`plotDiag`	if TRUE, diagnostics will be plotted
`title`	title for the diagnostics plots

Value

data frame with the inference summaries of RYM per mixture

Author(s)

Timothee Flutre [aut], Arnaud Gauffreteau [ctb]

See Also

`estimRYRep`

Examples

```
## generate fake data
set.seed(1234)
nbGenos <- 25
genos <- sprintf("g%02i", 1:nbGenos)
pairs <- t(combn(x=genos, m=2))
mixIDs <- sample(paste(pairs[,1], pairs[,2], sep="_"), size=75)
nbBlocks <- 3
blocks <- LETTERS[1:nbBlocks]
dat <- do.call(rbind, lapply(blocks, function(block){
  data.frame(ID=c(genos, mixIDs),
             block=block,
             stringsAsFactors=TRUE)
}))
listContr <- list(block="contr.sum")
X <- model.matrix(~ 1 + block, data=dat, contrasts=listContr)
Z_GMA <- mkZGMA(dat, "ID", "_")
Z_SMA <- mkZSMA(dat, "ID", "_", inc_SMA_ii="only_pur")
truth <- list("intercept"=100, "var_GMA"=10, "var_SMA"=4, "var_error"=1)
truth[["blockEffs"]] <- rnorm(n=nbBlocks - 1, mean=0, sd=2)
truth[["GMAs"]] <- rnorm(n=nbGenos, mean=0, sd=sqrt(truth$var_GMA))
truth[["SMAs"]] <- rnorm(n=nlevels(dat$ID), mean=0, sd=sqrt(truth$var_SMA))
truth[["errors"]] <- rnorm(n=nrow(dat), mean=0, sd=sqrt(truth$var_error))
y <- X %*% c(truth$intercept, truth$blockEffs) +
  Z_GMA %*% truth$GMAs +
  Z_SMA %*% truth$SMAs +
  truth$errors
dat$yield <- y[,1]
hist(dat$yield, breaks=20, las=1, main="Simulated data")
boxplot(yield ~ block, data=dat, las=1, main="Simulated data")

## compute the average yield per ID over blocks, and then compute the RYMs:
avg <- tapply(dat$yield, dat$ID, mean)
avg <- data.frame(ID=names(avg), yield=avg, stringsAsFactors=TRUE)
avg <- RYM(avg, colIDcomps="ID", colY="yield", sep="_")

## compute the RYMs per block, and then correct for the "block" effect:
mix2genos <- strsplit(levels(dat$ID), "_")
names(mix2genos) <- levels(dat$ID)
out <- estimRYMRep(dat, "yield", "ID", "block", mix2genos)
head(out)

## compare both approaches:
op <- par(mfrow=c(1,2))
hist(avg$RYM, las=1, main="Estimated RYM by averaging over blocks", xlab="RYM")
abline(v=1, lwd=2); abline(v=mean(avg$RYM, na.rm=TRUE), col="red", lwd=2)
hist(out$estim, las=1, main="Estimated RYM after correcting the 'block' effect", xlab="RYM")
abline(v=1, lwd=2); abline(v=mean(out$estim), col="red", lwd=2)
par(op)

out$avgRYM <- avg[rownames(out), "RYM"]
plot(out$avgRYM, out$estim, las=1, type="n",
```

```
     xlab="RYM averaged over blocks",
     ylab="RYM after correcting the 'block' effect",
     main="Estimated RYMs")
idx <- which(out$pv > 0.05)
points(out$avgRYM[idx], out$estim[idx], pch=19, col="black")
idx <- which(out$pv <= 0.05)
points(out$avgRYM[idx], out$estim[idx], pch=19, col="red")
abline(a=0, b=1, h=1, v=1, lty=2)
segments(x0=as.numeric(out$avgRYM), y0=out$cil,
         x1=as.numeric(out$avgRYM), y1=out$ciu,
         col="grey", lty=2, lwd=2)
```

`estimRYRep` *RY with "rep" effect*

Description

Computes the relative yields (RYs) per replicate (e.g., blocks), then fits a linear model correcting for this rep effect, and quantifies the uncertainty around it.

Usage

```
estimRYRep(
  data,
  colY,
  colIDstand,
  colIDfocal,
  colRep,
  mix2genos,
  conf_level = 0.95,
  plotDiag = FALSE
)
```

Arguments

`data`	data frame
`colY`	column name specifying the response variable
`colIDstand`	column name specifying the stand identifiers
`colIDfocal`	column name for the focal identifiers
`colRep`	column name specifying the replicate
`mix2genos`	named list with one component per mixture, each component being a vector of genotype names; see `getMixtureList`
`conf_level`	confidence level for the uncertainty intervals
`plotDiag`	if TRUE, diagnostic plots

Value

data frame with the inference summaries of RY per mixture

Author(s)

Timothee Flutre [aut], Arnaud Gauffreteau [ctb]

See Also

estimRYMRep

Examples

```
(dat <- data.frame(ID=c("geno1", "geno1",
                        "geno2", "geno2",
                        "mixg1g2", "mixg1g2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno1",
                           "geno2", "geno2",
                           "geno1", "geno2",
                           "geno1", "geno2"),
                   prop=c(1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5),
                   block=c("A","B","A","B","A","A","B","B"),
                   yield=c(51,45, 39,43, 25,22,26,25)))
estimRYRep(dat, "yield", "ID", "focal", "block", list("mixg1g2"=c("geno1","geno2")))
```

`fitDBVSBVinter` *Fit DBV-SBV models for interspecific mixtures*

Description

Fits DBV-SBV models for interspecific mixtures.

Usage

```
fitDBVSBVinter(
  listY,
  listX,
  listZ,
  listVCov,
  REML = TRUE,
  lOptions = NULL,
  verbose = FALSE
)
```

Arguments

listY	named list of response variables (a matrix Y_IC and, optionally, a vector y_SC_f and a vector y_SC_t)
listX	named list of design matrices for the fixed-effect explanatory factors (a matrix X_IC and, optionally, a matrix X_SC_f and a matrix X_SC_t)
listZ	named list of design matrices of random-effect explanatory factors
listVCov	named list of square, symmetric matrices used, after re-scaling, as variance-covariance matrices for the random-effect factors; named "K" for the BVs (DBVs and SBVs) and "Kmixpair" for the DBVxSBVs; these matrices must have dimnames (both rows and columns) coherent with the column names of the design matrices in listZ
REML	logical
lOptions	named list of options (for experts)
verbose	verbosity level

Value

list

Author(s)

Jemay Salomon, Timothee Flutre

See Also

simulDBVSBVinter

Examples

```
## simulate a data set with both sole crops and intercrops:
GRMs <- list("S1" = diag(100),
             "S2" = diag(2))
dimnames(GRMs$S1) <- list(paste0("gS1_", 1:100), paste0("gS1_", 1:100))
dimnames(GRMs$S2) <- list(paste0("gS2_", 1:2), paste0("gS2_", 1:2))
out <- simulDBVSBVinter(GRMs)
names(out)
datW <- out$datW
str(datW)

## fit the model using intercrops only:
idxIC <- which(!is.na(datW$geno_S1) & !is.na(datW$geno_S2))
datW_IC <- droplevels(datW[idxIC, ])
listY <- list(Y_IC = datW_IC[, c("yield_S1", "yield_S2")])
listX <- list(X_IC = model.matrix(~ 1 + block + geno_S2, datW_IC,
              contrasts.arg = list("block" = "contr.sum",
                                   "geno_S2" = "contr.sum")))
listZ <- list(Z_DS_f = model.matrix(~ 0 + geno_S1, datW_IC))
colnames(listZ$Z_DS_f) <- gsub("^geno_S1", "", colnames(listZ$Z_DS_f))
listVCov <- list(K = GRMs$S1[levels(datW_IC$geno_S1), levels(datW_IC$geno_S1)])
```

```
fitTmb <- fitDBVSBVinter(listY, listX, listZ, listVCov,
                         lOptions = list(iter.max = 20), REML = TRUE)
names(fitTmb)
fitTmb$sdr

## see the third vignette for more details
```

`fitGMASMA` *Fit GMA-SMA models*

Description

Fits GMA-SMA models.

Usage

```
fitGMASMA(
  formFix,
  data,
  listZ,
  pkg = "lme4",
  listVCov,
  contrasts = NULL,
  REML = TRUE,
  ...
)
```

Arguments

`formFix`	formula whose right-hand side should only include the fixed effects, the random effects being deduced based on the names of `"listZ"`
`data`	data frame; missing data will be removed
`listZ`	named list of design matrices for the random effects; its names should be `"GMA"` (compulsory) and, optionally, one of `"SMA"`, `"SMA_ij"` or `"SMA_ii"`
`pkg`	package used to fit the model among lme4, MM4LMM, TMB, INLA and breedR; lme4 and INLA do not use `"listVCov"`; breedR does not use `"contrasts"`
`listVCov`	named list of variance-covariance matrices for the random effects; the names of this list should be the same as the names of `"listZ"`
`contrasts`	see `stats::model.matrix()`
`REML`	logical
`...`	additional arguments specific to each package

Value

depends on the chosen package

Author(s)

Timothee Flutre

See Also

mkZGMA, mkZSMA

Examples

```
## see the first and second vignettes
```

`getDesignBinaryCropMix` *Optimize design for binary crop mixtures*

Description

Optimizes a design for binary crop mixtures

Usage

```
getDesignBinaryCropMix(
  levGenos1,
  levGenos2,
  nbMixes,
  seed = NULL,
  showPlots = FALSE,
  verbose = FALSE
)
```

Arguments

`levGenos1`	character vector with the names of the genotypes of the first species
`levGenos2`	character vector with the names of the genotypes of the second species
`nbMixes`	number of mixtures
`seed`	seed for the pseudo-random number generator
`showPlots`	logical indicating if plots should be showed
`verbose`	verbosity level

Value

list

Author(s)

Timothee Flutre

Examples

```
nbGenos1 <- 30
levGenos1 <- sprintf(fmt=paste0("S1_%0", floor(log10(nbGenos1))+1, "i"),
                     1:nbGenos1)
nbGenos2 <- 8
levGenos2 <- sprintf(fmt=paste0("S2_%0", floor(log10(nbGenos2))+1, "i"),
                     1:nbGenos2)
nbMixes <- 60 # only binary and balanced
design <- getDesignBinaryCropMix(levGenos1, levGenos2, nbMixes, seed=12345)
plotDesignCropMix(design$graph, levGenos1, levGenos2)
ggplot(design$entropies) + aes(x=iteration, y=entropy, group=type) + geom_point() + geom_line() +
  geom_hline(yintercept = design$max_ent) + theme_bw() + facet_wrap(~ type)
```

---

getDesignBinaryVarMix *Optimize design for binary varietal mixtures*

Description

Optimizes a design for binary varietal mixtures

Usage

```
getDesignBinaryVarMix(
  levGenos,
  nbMixes,
  seed = NULL,
  showPlots = FALSE,
  verbose = FALSE
)
```

Arguments

levGenos	character vector with the names of the genotypes
nbMixes	number of mixtures
seed	seed for the pseudo-random number generator
showPlots	logical indicating if plots should be showed
verbose	verbosity level

Value

list

Author(s)

Timothee Flutre

Examples

```
nbGenos <- 25
levGenos <- sprintf(fmt=paste0("geno%0", floor(log10(nbGenos))+1, "i"),
                    1:nbGenos)
nbMixes <- 75 # only binary and balanced
design <- getDesignBinaryVarMix(levGenos, nbMixes, seed=12345)
plotDesignVarMix(design$graph, levGenos)
ggplot(design$entropies) + aes(x=iteration, y=entropy) + geom_point() + geom_line() +
  geom_hline(yintercept = design$max_ent) + theme_bw()
```

---

getMixtureList *Reformat composition of mixtures into a list*

Description

Reformats composition of mixtures into a list.

Usage

```
getMixtureList(mixtures, sep = NULL)
```

Arguments

`mixtures`	vector, matrix, data.frame or list containing of mixtures
`sep`	required only if `mixtures` is a vector; will be given to `strsplit`

Value

list of vectors, with one component per mixture, the vector containing the mixture components

Author(s)

Timothee Flutre

See Also

`getMixturesPerGeno`

Examples

```
nbGenos <- 25
levGenos <- sprintf(fmt=paste0("geno%0", floor(log10(nbGenos))+1, "i"),
                    1:nbGenos)
nbMixes <- 75 # only binary and balanced
design <- getDesignBinaryVarMix(levGenos, nbMixes, seed=12345)
mixtures <- getMixtureList(design$combs)
str(mixtures, list.len=10)
```

getMixturesPerGeno *Get mixture(s) per genotype*

Description

Returns the list of mixtures per genotype.

Usage

```
getMixturesPerGeno(mix2genos)
```

Arguments

mix2genos	list with one component per mixture, listing the genotypes it contains (output of getMixtureList)

Value

list with one vector per genotype

Author(s)

Timothee Flutre

See Also

getMixtureList

Examples

```
nbGenos <- 25
levGenos <- sprintf(fmt=paste0("geno%0", floor(log10(nbGenos))+1, "i"),
                    1:nbGenos)
nbMixes <- 75 # only binary and balanced
design <- getDesignBinaryVarMix(levGenos, nbMixes, seed=12345)
mixtures <- getMixtureList(design$combs)
str(mixtures, list.len=10)
geno2mixes <- getMixturesPerGeno(mixtures)
geno2mixes[c(1,2)]
table(sapply(geno2mixes, length))
```

imageMat *Image of a matrix*

Description

Plots an image of a matrix.

Usage

```
imageMat(mat, title, col, breaks)
```

Arguments

mat	matrix
title	optional title of the plot; if missing, it will be the matrix name and dimensions
col	optional vector of colors; if missing, it will be a grey color gradient
breaks	optional vector of breaks to go with col

Value

nothing

Author(s)

Timothee Flutre

Examples

```
M <- diag(c(rep(2, 3), rep(15, 5)))
imageMat(M) # see how the the "2" values are ignored!
imageMat(M, col=c("lightgrey","red","blue"), breaks=c(-1,1,3,20))
```

infoCriterion *Information criterion*

Description

Returns an information criterion, among AIC, AICc and BIC. Caution: choosing k and n may not be straightforward for certain models, such as mixed models.

Usage

```
infoCriterion(k, lnLmax, n = NULL, type = "AIC")
```

Arguments

k	number of parameters, sometimes also called "effective degrees of freedom"
lnLmax	value of the log-likelihood when maximized
n	sample size
type	specific information criterion to be returned

Value

numeric with attributes k and n (if not NULL)

Author(s)

Timothee Flutre

---

invvec *Inverse vec operator*

Description

Applies the inverse vec operator to a vector.

Usage

```
invvec(x, n_col, n_row = NULL, sep = NULL)
```

Arguments

x	vector
n_col	number of columns of the output matrix
n_row	number of rows of the output matrix
sep	separator used to retrieve row and column names, only if x has names

Value

matrix

Author(s)

Timothee Flutre

See Also

vec

Examples

```
mat <- matrix(c(1, 2, 3,
                4, 5, 6),
              nrow = 2, ncol = 3, byrow = TRUE,
              dimnames = list(as.character(1:2), letters[1:3]))
mat
(x <- vec(mat))
invvec(x, n_col = 3, sep = "-")
```

---

invvecMixes *Inverse vec for mixtures*

Description

Tranforms a vector of component observations into a matrix with one row per mixture and components in colums. Requires all mixtures to have the same number of components, and all the obervationss from the same mixture to be one after the others, in the same order for each mixture.

Usage

```
invvecMixes(x, nb_comps)
```

Arguments

x	vector; make sure that it is correctly sorted beforehand
nb_comps	number of components in each mixture

Value

matrix with nb_comps columns

Author(s)

Timothee Flutre

Examples

```
datL <- data.frame(ID = c("g1+g2", "g1+g2", "g1+g2", "g1+g2"),
  focal = c("g1", "g2", "g1", "g2"),
  neighbor = c("g2", "g1", "g2", "g1"),
  block = c("A", "A", "B", "B"),
  yield = c(1, 2, 3, 4)
)
invvecMixes(datL$yield, 2)
```

LER *Land equivalent ratio (LER)*

Description

Computes the land equivalent ratio (LER) of Willey and Osiru (1972, doi:10.1017/S0021859600025909) interpreted as the relative land area required under sole cropping to produce the same yield as under intercropping. It corresponds to equation 28 in Weigelt and Jolliffe (2003, doi:10.1046/j.1365-2745.2003.00805.x), and is equivalent to the index called "relative yield total" (RYT) corresponding to equation 26 of the same paper. Missing values are forbidden.

Usage

```
LER(x)
```

Arguments

x	data frame with species in rows (with species names as row names) and at least two columns, the first being the performance under sole-cropping and the second under inter-cropping

Value

list with partial and total LERs

Author(s)

Timothee Flutre

See Also

RYT, RY

Examples

```
(dat <- data.frame(solecrop=c(5, 15),
                   intercrop=c(4, 9),
                   row.names=c("grain", "fruit")))
LER(dat)
```

mixSowingWeight *Sowing*

Description

Computes the seed weight per genotype for each stand, assuming equal sowing proportions.

Usage

```
mixSowingWeight(pureSowingWeights, stands)
```

Arguments

pureSowingWeights	vector which names are genotypes and values are sowing weights in pure stands
stands	vector which names are stand identifiers and values are genotype(s), a single one if it is a pure stand, several ones if it is a mixed stand (separated by a dash "-", e.g., "var1-var8")

Value

list which components are stands and values are named vectors with sowing weight per genotype for each stand

Author(s)

Timothee Flutre

See Also

nbSeedsToSownInPure

Examples

```
sowingArea <- 9.52
sowingDensity <- 160
(tmp <- nbSeedsToSownInPure(sowingArea, sowingDensity))
TKWs <- c("var1"=38.605, "var2"=40.051, "var6"=36.251, "var8"=33.368)
pureSowingWeights <- TKWs * tmp
stands <- c("var1"="var1", "mix8"="var1-var2", "mix34"="var1-var8",
            "mix50"="var1-var6-var8")
mixSowingWeight(pureSowingWeights, stands)
```

mkAllZSMA *Design matrices of SMAs for GMA-SMA models*

Description

Makes the design matrices with the contribution of specific mixing abilities (SMAs). More details in the help of mkZSMA and in the vignette.

Usage

```
mkAllZSMA(
  df,
  col,
  sep,
  models = c("2p", "2", "2pp", "3", "3p"),
  verbose = FALSE
)
```

Arguments

df	data frame
col	name of the column containing the names of pure and mixed stands
sep	separator used to separate the genotype names in "col"
models	vector of model identifiers; see the vignette
verbose	verbosity level

Value

list of SMA design matrices

Author(s)

Timothee Flutre

Examples

```
dat <- data.frame(mix=paste0("mix", 1:3),
                  varieties=c("var3","var1-var3","var1-var2"),
                  pheno=c(10, 7, 9))
(listZSMA <- mkAllZSMA(df=dat, col="varieties", sep="-"))
```

mkZGMA *Design matrix of GMAs for GMA-SMA models*

Description

Makes a design matrix with the contribution of general mixing abilities (GMAs). The design is based on the names of pure and mixed stands, in which the genotype names are separated by a specific symbol.

Usage

```
mkZGMA(df, col, sep = ",")
```

Arguments

df	data frame
col	name of the column containing the names of pure and mixed stands
sep	separator used to separate the genotype names in "col"

Value

matrix

Author(s)

Timothee Flutre

See Also

mkZGMA, fitGMASMA

Examples

```
dat <- data.frame(mix=paste0("mix", 1:3),
                  varieties=c("var3","var1-var3","var1-var2"),
                  pheno=c(10, 7, 9))
(Z_GMA <- mkZGMA(df=dat, col="varieties", sep="-"))
```

`mkZinterspe` *Design matrices for DBV-SBV models in an inter-specific trial*

Description

Makes the design matrices for direct and social breeding values per species when the trial includes binary intercrops and, possibly, sole crops.

Usage

```
mkZinterspe(df, genosPerSp, colIDfocal = "focal", colIDneighbors = "neighbors")
```

Arguments

`df`	data frame with one row per sole-crop plot or two rows per binary intercrop plot, with a factor column corresponding to the focal identifiers ("DBV" a.k.a. "DGE" or "Pr") and a column corresponding to the "neighbor(s)" identifiers ("SBV" a.k.a. "IGE" or "As")
`genosPerSp`	list of two components, one per species, each being a vector with the genotype identifiers of the given species (they will be sorted)
`colIDfocal`	name of the column containing the identifiers of the focal genotypes
`colIDneighbors`	name of the column containing the identifiers of the neighbour genotypes

Value

list of two components, one per species, each being a list of two matrices, the first for the DBVs and the second for the SBVs; the column names of these matrices correspond to the genotype identifiers after sorting

Author(s)

Timothee Flutre

Examples

```
(dat <- data.frame(focal=c("wheat01", "pea02",
                           "wheat01", "pea01",
                           "wheat01"),
                   neighbors=c("pea02", "wheat01",
                               "pea01", "wheat01",
                               "wheat01"),
                   name=c("wheat01-pea02", "wheat01-pea02",
                          "wheat01-pea01", "wheat01-pea01",
                          "wheat01-wheat01"),
                   stand = c("inter", "inter", "inter", "inter", "sole"),
                   species = c("wheat", "pea", "wheat", "pea", "wheat")))
(out <- mkZinterspe(dat,
                    list("wheat"="wheat01",
                         "pea"=c("pea01","pea02"))))
```

mkZSMA *Design matrix of SMAs for GMA-SMA models*

Description

Makes a design matrix with the contribution of specific mixing abilities (SMAs). Before Forst et al (2019), there was only one kind of SMA, the inter-genotypic SMA (SMA_ij). Forst et al (2019) introduced a second kind of SMA, the intra-genotypic SMA (SMA_ii). As a result, various design matrices can be made depending on which kinds of SMAs are modeled. The design is based on the names of pure and mixed stands, in which the genotype names are separated by a specific symbol.

Usage

```
mkZSMA(df, col, sep = ",", inc_SMA_ii = "no", skipUnusedCols = TRUE)
```

Arguments

df	data frame
col	name of the column containing the names of pure and mixed stands
sep	separator used to separate the genotype names in "col"
inc_SMA_ii	specify how the intra-genotypic SMA should be included, with "no" meaning no SMA_ii, "only_pur" meaning that the SMA_ii is only included in the pure stands (this is model 2 of Forst et al, 2019), "pur_mix" meaning that the SMA_ii is included in both the pure and the mixed stands (this is model 3 of Forst et al, 2019)
skipUnusedCols	skip unused columns, i.e., full of zeroes and not present in the data frame

Value

matrix

Author(s)

Timothee Flutre

See Also

mkZGMA, fitGMASMA

Examples

```
dat <- data.frame(mix=paste0("mix", 1:3),
                  varieties=c("var3","var1-var3","var1-var2"),
                  pheno=c(10, 7, 9))
(Z_SMA <- mkZSMA(df=dat, col="varieties", sep="-", inc_SMA_ii="only_pur")) # model 2
(Z_SMA <- mkZSMA(df=dat, col="varieties", sep="-", inc_SMA_ii="pur_mix")) # model 3
```

`nbSeedsToSownInPure` *Sowing*

Description

Computes the number of seeds of a given genotype to be sowned as pure stand based on a given sowing area and density.

Usage

```
nbSeedsToSownInPure(sowingArea = 8.4, sowingDensity = 160)
```

Arguments

`sowingArea`	area to be sowned in square meters
`sowingDensity`	number of seeds per square meter

Value

number of seeds (in thousands)

Author(s)

Timothee Flutre

See Also

`mixSowingWeight`

Examples

```
sowingArea <- 9.52
sowingDensity <- 160
nbSeedsToSownInPure(sowingArea, sowingDensity)
sowingArea <- 8.66
sowingDensity <- 200
nbSeedsToSownInPure(sowingArea, sowingDensity)
```

normBiasError *Normalized bias error*

Description

Returns the normalized bias error (NBE).

Usage

```
normBiasError(estim, truth, perc = TRUE)
```

Arguments

estim	numeric vector of estimates
truth	numeric vector of true values
perc	logical; if TRUE, the return value will be in percentage

Value

numeric vector

Author(s)

Timothee Flutre

paramBoot4TMB *Parametric bootstrap with TMB*

Description

Performs parametric bootstrap with TMB and `nlminb` on a fitted model to allow uncertainty quantification.

Usage

```
paramBoot4TMB(fit, nb_boot = 5, mc.cores = 1)
```

Arguments

fit	list corresponding to a model fitted with TMB
nb_boot	number of parametric bootstraps to perform
mc.cores	the number of cores to use, i.e. at most how many child processes will be run simultaneously (see the "parallel" package)

Value

list with as many components as nb_boot

Author(s)

Jemay Salomon, Timothee Flutre

Examples

```
## see the example of `fitDBVSBVinter`
## then run (slow if nb_boot is high!): paramBoot4TMB(fitTmb)
```

`pivotMixData2Long` *Pivot mixture data into the long format*

Description

Pivots mixture data into the long format, from one row per stand into one row per component of each stand (i.e., several rows if the stand is a mixture).

Usage

```
pivotMixData2Long(
  df,
  genos,
  colC,
  sep = "-",
  prefixY = NULL,
  sepY = "_",
  colIDfocal = "focal",
  colIDneighbors = "neighbor"
)
```

Arguments

df	data frame
genos	named list of vectors, one per species, containing all the identifiers of the genotypes of the given species
colC	name of the column containing the component(s) of each stand
sep	separator used to separate the genotype names in "col"; see the "split" argument of strsplit
prefixY	prefix of the columns containing the yield data (one column per mixture component); it is assumed that the components in colC are in the same order as the yield data
sepY	separator used in the name of the new "yield" columns made by concatenating colY and colIDfocal or colIDneighbors

colIDfocal	name of the column in the output that will contain the identifiers of the focal genotypes
colIDneighbors	name of the column in the output that will contain the identifiers of the neighbor genotypes; for monovarietal stands, the entries in this neighbor column will be identical to the entries in the focal column

Value

data.frame with more rows

See Also

pivotMixData2Wide

Examples

```
## example of species mixtures:
(dat <- data.frame(name=c("wheat01-pea02","wheat01-pea01", "wheat01"),
                   stand=c("inter","inter","sole"),
                   x=c(1, 1, 1),
                   y=c(1, 2, 3),
                   "yield_cereal" = c(10, 11, 20),
                   "yield_legume" = c(9, 8, NA),
                   check.names = FALSE,
                   stringsAsFactors=TRUE))
pivotMixData2Long(df=dat, colC="name", sep="-", prefixY="yield",
                  genos=list("cereal"=c("wheat01"),
                             "legume"=c("pea01","pea02")))

## example of varietal mixtures:
(dat <- data.frame(name=c("g1+g2","g1+g2","g1","g2"),
                   "yield_focal" = c(10, 12, 20, 18),
                   "yield_neighbor" = c(11, 9, NA, NA),
                   check.names = FALSE,
                   stringsAsFactors=TRUE))
pivotMixData2Long(df=dat, colC="name", sep="+", prefixY="yield",
                  genos=list(c("g1","g2")))
```

---

pivotMixData2Wide *Pivot mixture data into the wide format*

Description

Pivots mixture data into the wide format, from one row per component of each stand (i.e., several rows if the stand is a mixture) into one row per stand.

Usage

```
pivotMixData2Wide(
  df,
  colIDstand = "ID",
  colIDfocal = "focal",
  colIDneighbors = "neighbor",
  colPlot = c("x", "y"),
  colY = "yield",
  sepY = "_",
  sepFocalNeighbors = NULL
)
```

Arguments

df	data frame
colIDstand	name of the column containing the identifier of each stand
colIDfocal	name of the column containing the identifiers of the focal genotypes
colIDneighbors	optional name of the column containing the identifiers of the neighbor genotypes
colPlot	name of the column(s) allowing to uniquely identify a plot
colY	name of the column containing the yield data
sepY	separator used in the name of the new "yield" columns made by concatenating colY and colIDfocal or colIDneighbors
sepFocalNeighbors	in case colIDneighbors is unspecified or does not exist in df, the distinction between focal and neighbor(s) will be retrieved by splitting colIDstand using sepFocalNeighbors

Value

data.frame

Author(s)

Timothee Flutre

See Also

pivotMixData2Long

Examples

```
## only binary mixtures:
dat0 <- data.frame(
  focal = c("wheat01", "pea02", "wheat01", "pea01"),
  neighbor = c("pea02", "wheat01", "pea01", "wheat01"),
  name = c(
    rep("wheat01-pea02", 2),
    rep("wheat01-pea01", 2)
```

```
  ),
  stand = rep("inter", 4),
  x = 1,
  y = c(1, 1, 2, 2),
  yield = c(10, 11, 12, 13),
  stringsAsFactors = TRUE
)
dat0
(dat1 <- pivotMixData2Wide(dat0, colIDstand="name"))

## binary mixtures and a monovarietal:
dat0 <- data.frame(
  focal = c("wheat01", "pea02", "wheat01", "pea01", "wheat01"),
  neighbor = c("pea02", "wheat01", "pea01", "wheat01", "wheat01"),
  name = c(
    rep("wheat01-pea02", 2),
    rep("wheat01-pea01", 2),
    "wheat01"
  ),
  stand = c(rep("inter", 4), "sole"),
  x = 1,
  y = c(1, 1, 2, 2, 3),
  yield = c(10, 11, 12, 13, 20.5),
  stringsAsFactors = TRUE
)
dat0
(dat1 <- pivotMixData2Wide(dat0, colIDstand="name"))

## reverse conversion
dat1v2 <- dat1
colnames(dat1v2)[colnames(dat1v2) == "yield_focal"] <- "yield-cereal"
colnames(dat1v2)[colnames(dat1v2) == "yield_neighbor"] <- "yield-legume"
dat1v2$yield <- apply(dat1v2[,c(7,8)], 1, sum, na.rm=TRUE)
(dat1v2 <- dat1v2[,-c(1,2)])
(dat2 <- pivotMixData2Long(dat1v2, colC="name", sepY="-",
                           genos=list("cereal"=c("wheat01","wheat02"),
                                      "legume"=c("pea01","pea02"))))
all.equal(dat2, dat0)
```

`plotDesignCropMix` *Plot design for crop mixtures*

Description

Plots design for crop mixtures, as a graph and as a diallel.

Usage

```
plotDesignCropMix(graph, levGenos1, levGenos2, main = NULL)
```

Arguments

graph	graph
levGenos1	character vector with the names of the genotypes of the first species
levGenos2	character vector with the names of the genotypes of the second species
main	main title for the graph plot

Value

nothing

Author(s)

Timothee Flutre

Examples

```
nbGenos1 <- 30
levGenos1 <- sprintf(fmt=paste0("S1_%0", floor(log10(nbGenos1))+1, "i"),
                     1:nbGenos1)
nbGenos2 <- 8
levGenos2 <- sprintf(fmt=paste0("S2_%0", floor(log10(nbGenos2))+1, "i"),
                     1:nbGenos2)
nbMixes <- 60 # only binary and balanced
design <- getDesignBinaryCropMix(levGenos1, levGenos2, nbMixes, seed=12345)
plotDesignCropMix(design$graph, levGenos1, levGenos2)
```

---

plotDesignVarMix *Plot design for varietal mixtures*

Description

Plots design for varietal mixtures, as a graph and as a diallel.

Usage

```
plotDesignVarMix(
  graph,
  levGenos,
  main = NULL,
  subplots = c("graph", "diallel")
)
```

Arguments

graph	graph
levGenos	character vector with the names of the genotypes
main	main title for the graph plot
subplots	character vector indicating which object(s) to plot

Value

nothing

Author(s)

Timothee Flutre

Examples

```
nbGenos <- 25
levGenos <- sprintf(fmt=paste0("geno%0", floor(log10(nbGenos))+1, "i"),
                    1:nbGenos)
nbMixes <- 75 # only binary and balanced
design <- getDesignBinaryVarMix(levGenos, nbMixes, seed=12345)
plotDesignVarMix(design$graph, levGenos)
```

---

plotDiallel *Plot diallel*

Description

Plots a diallel matrix.

Usage

```
plotDiallel(diallel, main = NULL)
```

Arguments

diallel	matrix
main	title

Value

nothing

Author(s)

Timothee Flutre

Examples

```
nbGenos <- 25
levGenos <- sprintf(fmt=paste0("geno%0", floor(log10(nbGenos))+1, "i"),
                    1:nbGenos)
nbMixes <- 75 # only binary and balanced
design <- getDesignBinaryVarMix(levGenos, nbMixes, seed=12345)
plotDiallel(design$diallel)
```

RII *Relative interaction index (RII)*

Description

Computes the relative interaction index (RII) as defined in Armas et al (2004, doi:10.1890/030650): RII_i = (Y_i(j) - Y_i) / (Y_i(j) + Y_i) where Y_i(j) is the yield per plant of i when mixed with j, and Y_i is the yield per plant of i when growned in isolation. Missing values are forbidden.

Usage

```
RII(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colY = "yield",
  avgIfReps = FALSE
)
```

Arguments

dat	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the focal yield in each stand (see the example below); a stand with a single proportion equals to 1 will be interpreted as a monovarietal culture ("pure stand"), a mixture otherwise (and its proportions should sum to 1)
colIDstand	column name for the stand identifiers
colIDfocal	column name for the focal identifiers
colProp	column name for the proportions
colY	column name for the yield values
avgIfReps	if TRUE, replicates of monovarietals will be averaged; if FALSE, the presence of replicates will return an error

Value

input data.frame with an extra column named "RII"

Author(s)

Timothee Flutre

See Also

RIInet, RY, RYT, RYP, RYM

Examples

```
sow_density <- 200
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield_qt_ha=c(50, 40, 25, 22)))
dat$yield_g_m2 <- (dat$yield_qt_ha / 10^4) * 10^6
dat$yield_g_plant <- dat$yield_g_m2 / (sow_density * dat$prop)
RII(dat, colY = "yield_g_plant")
```

`RIInet` *Net Relative interaction index (RIInet)*

Description

Computes the net relative interaction index (RII) as defined in Stefan et al (2022): RIInet = sum_i prop_i RII_i. Missing values are forbidden.

Usage

```
RIInet(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colRII = "RII"
)
```

Arguments

`dat`	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the relative interaction index of the focal.
`colIDstand`	column name for the stand identifiers
`colIDfocal`	column name for the focal identifiers
`colProp`	column name for the proportions
`colRII`	column name for the RII values

Value

data.frame with columns `colIDstand` and "RIInet"

Author(s)

Timothee Flutre

See Also

RII

Examples

```
sow_density <- 200
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield_qt_ha=c(50, 40, 25, 22)))
dat$yield_g_m2 <- (dat$yield_qt_ha / 10^4) * 10^6
dat$yield_g_plant <- dat$yield_g_m2 / (sow_density * dat$prop)
dat <- RII(dat, colY = "yield_g_plant")
RIInet(dat)
```

---

rmatnorm *Matrix-variate Normal distribution*

Description

Random generation for the matrix-variate Normal distribution. See https://en.wikipedia.org/wiki/Matrix_normal_distribution.

Usage

```
rmatnorm(n = 1, M, U, V, pivot = c(U = "auto", V = "auto"))
```

Arguments

n	number of observations
M	mean matrix; if missing, will be replaced by a matrix of zeroes; if dimnames(M) is not NULL, the column and row names will be compared to those of U and V (if any) and, in the absence of conflict, propagated to the output
U	between-row covariance matrix
V	between-column covariance matrix
pivot	2-element vector with values TRUE/FALSE/"auto", where TRUE (FALSE) means using pivoting (or not) for Choleski decomposition of U and/or V (see chol); useful when U and/or V are singular; with "auto", this will be automatically determined

Value

array

Author(s)

Timothee Flutre

Examples

```
set.seed(1859)
Sigma <- matrix(c(3,2,2,4), nrow=2, ncol=2)
rho <- Sigma[2,1] / prod(sqrt(diag(Sigma)))
samples <- rmatnorm(n=100, M=matrix(0, nrow=10^3, ncol=2),
                    U=diag(10^3), V=Sigma)
tmp <- t(apply(samples, 3, function(mat){
  c(var(mat[,1]), var(mat[,2]), cor(mat[,1], mat[,2]))
}))
summary(tmp) # corresponds well to Sigma
```

RY *Relative yield (RY)*

Description

Computes the "relative yield" (RY) as defined in Fowler (1982, doi:10.2307/2259865) citing de Wit (1960): RY_ij = Y_ij / Y_i where Y_ij is the yield of i when mixed with j and Y_i is the yield of i in monovarietal culture. Note that equation 1 of table 2 in Williams and McCarthy (2001, doi:10.1046/j.14401703.2001.00368.x) is called "RY" but in fact it corresponds to "RYP" (relative yield per plant). Note also that the equation for "RY" in Reiss and Drinkwater (2018, doi:10.1002/eap.1629) in fact corresponds to "RYM" (relative yield of mixture). Missing values are forbidden.

Usage

```
RY(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colY = "yield",
  avgIfReps = FALSE
)
```

Arguments

dat	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the focal yield in each stand (see the example below); a stand with a single proportion equals to 1 will be interpreted as a monovarietal culture ("pure stand") and a mixture otherwise; note that contrary to RYP, the proportions will not be used for anything else
colIDstand	column name for the stand identifiers
colIDfocal	column name for the focal identifiers
colProp	column name for the proportions

colY	column name for the yield values
avgIfReps	if TRUE, replicates of monovarietals will be averaged; if FALSE, the presence of replicates will return an error

Value

input data.frame with an extra column named "RY"

Author(s)

Timothee Flutre

See Also

estimRYRep, RYP, RYT, RYM

Examples

```
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield=c(50, 40, 25, 22)))
RY(dat)
```

RYM *Relative yield of mixture (RYM)*

Description

Computes the "relative yield of mixture" (RYM) as defined initially by Wilson (1988, doi:10.2307/2403626) for binary mixtures sowed in equal proportions, and extended by Williams and McCarthy (2001, doi:10.1046/j.14401703.2001.00368.x) to binary mixtures of unequal proportions. A RYM above 1 means that the mixture yielded better than the sum of the pure-stand yields, each weighted by their proportion in the mixture. It corresponds to equation 35 in Weigelt and Jolliffe (2003, doi:10.1046/j.13652745.2003.00805.x), as well as equation 25b (and not to 25a!) of the same paper where the "control" treatment is the expected mixture yield calculated based on the weighted component monovarietal yields. Note that the RYM was unfortunately called "relative yield" (RY) by Reiss and Drinkwater (2018, doi:10.1002/eap.1629). Missing values are forbidden.

Usage

```
RYM(
  dat,
  colIDstand = NULL,
  colIDcomps = "comps",
  colProps = NULL,
  colY = "yield",
  sep = "-",
  colOut = "RYM"
)
```

Arguments

dat	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifiers of the components of each stand (separated by a specific symbol in the case of mixtures), the sowing (plant) proportion of the components in each stand (separated by the same symbol in the case of mixtures, and the yield of each stand (see the example below); a stand with a single component should have its proportion equal to 1 and will be interpreted as a monovarietal culture ("pure stand"), a mixture otherwise (and its proportions should sum to 1)
colIDstand	column name for the stand identifiers (they should be unique); if NULL, the code will attempt to use "colIDcomps"
colIDcomps	column name for the identifiers of the stand components (separated by sep)
colProps	column name for the stand proportions (separated by sep and summing to 1); if NULL, the code will assume all components of a given mixture are equiprobable
colY	column name for the yield values
sep	separator
colOut	column name for the output RYM

Value

input data.frame with an extra column named from "colOut"

See Also

estimRYMRep, RYP, RY, RYT

Examples

```
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2"),
                   comps=c("geno1", "geno2", "geno1-geno2"),
                   props=c("1", "1", "0.6-0.4"),
                   yield=c(50, 40, 47)))
RYM(dat)
```

RYP *Relative yield per plant (RYP)*

Description

Computes the relative yield per plant (RYP) as defined in Fowler (1982, doi:10.2307/2259865): RYP_ij = Y_ij / (p Y_i) where Y_ij is the yield of i when mixed with j at a proportion of p, and Y_i is the yield of i in monovarietal culture. Note that, unfortunately, the equation of RYP was called "relative yield" (RY) in table 2 of Williams and McCarthy (2001, doi:10.1046/j.1440-1703.2001.00368.x), although these authors do have the right equation, i.e., they only dropped the "per plant". Missing values are forbidden.

Usage

```
RYP(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colY = "yield"
)
```

Arguments

dat	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the focal yield in each stand (see the example below); a stand with a single proportion equals to 1 will be interpreted as a monovarietal culture ("pure stand"), a mixture otherwise (and its proportions should sum to 1); note that contrary to RY, the proportions will be used to compute the values of RYP
colIDstand	column name for the stand identifiers
colIDfocal	column name for the focal identifiers
colProp	column name for the proportions
colY	column name for the yield values

Value

input data.frame with an extra column named "RYP"

Author(s)

Timothee Flutre

See Also

RY, RYT, RYM

Examples

```
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield=c(50, 40, 25, 22)))
RYP(dat)
```

RYT *Relative yield total (RYT)*

Description

Computes the "relative yield total" (RYT) of a mixture from de Wit and Den Bergh (1965) as the sum of all relative yields (RY) of the mixture. It corresponds to equation 26 (based on 25a and not 25b!) in Weigelt and Jolliffe (2003, doi:10.1046/j.13652745.2003.00805.x), and is equivalent to the index called "land equivalent ratio" (LER) corresponding to equation 28 of the same paper. Missing values are forbidden.

Usage

```
RYT(
  dat,
  colIDstand = "ID",
  colIDfocal = "focal",
  colProp = "prop",
  colY = "yield"
)
```

Arguments

dat	data.frame with one yield value per row and with at least four columns: the identifier of each stand, the identifier of the focal genotype or species in each stand, the sowing (plant) proportion of the focal in each stand, and the focal yield in each stand (see the example below); a stand with a single proportion equals to 1 will be interpreted as a monovarietal culture ("pure stands") and a mixture otherwise; note that contrary to RYP, the proportions will not be used for anything else
colIDstand	column name for the stand identifiers
colIDfocal	column name for the focal identifiers
colProp	column name for the proportions
colY	column name for the yield values

Value

input data.frame with an extra column named "RYT"

See Also

RYP, RY, RYM, LER

Examples

```
(dat <- data.frame(ID=c("geno1", "geno2", "mixg1g2", "mixg1g2"),
                   focal=c("geno1", "geno2", "geno1", "geno2"),
                   prop=c(1, 1, 0.5, 0.5),
                   yield=c(50, 40, 25, 22)))
RYT(dat)
```

`simulDBVSBVinter` *Simulate w.r.t. a DBV-SBV model for interspecific mixtures*

Description

Simulates an incomplete (sparse) yet balanced, tester-based design with respect to a DBV-SBV model for interspecific mixtures.

Usage

```
simulDBVSBVinter(
  GRMs,
  parsGenetics = list(mu = list(S1 = c(SC = 65, IC = 32), S2 = c(SC = 30, IC = 27)),
   sdBlocks = 4, CV_g = c(S1 = 0.08, S2 = 0.08), prop_var_SBV = c(S1 = 0.2, S2 = 0.2),
    prop_var_SIGV = c(S1 = 0.5, S2 = 0), cor_DBV_SBV = c(S1 = -0.9, S2 = -0.9),
   prop_var_DBVxSBV = c(S1 = 0, S2 = 0), use_GRM_for_DBVxSBV = TRUE, cor_DxS = 0, H2_SC
    = c(S1 = 0.7, S2 = 0.7), T2 = c(S1 = 0.7, S2 = 0.7), cor_err_IC = -0.2),
  parsDesign = list(nbBlocks = 2, nbPlotsPerBl = 500, nbYs = 10, sowingDensities =
    list(S1 = c(SC = 300, IC = 150), S2 = c(SC = 40, IC = 40)), testersS2 = TRUE,
    incomplete_balanced = TRUE),
  sep = "+",
  seed = NULL
)
```

Arguments

`GRMs`	named list with a genomic relationship matrix per species, named S1 and S2; should be a diagonal for S2 if it is a tester (see `parsDesign`)
`parsGenetics`	list of genetic parameters
`parsDesign`	list of design parameters
`sep`	character separating the names of a mixture components
`seed`	seed for the generation of pseudo-random numbers

Value

list

Author(s)

Timothee Flutre

See Also

fitDBVSBVinter

Examples

```
## simulate a data set with both sole crops and intercrops:
GRMs <- list("S1" = diag(100),
             "S2" = diag(2))
dimnames(GRMs$S1) <- list(paste0("gS1_", 1:100), paste0("gS1_", 1:100))
dimnames(GRMs$S2) <- list(paste0("gS2_", 1:2), paste0("gS2_", 1:2))
out <- simulDBVSBVinter(GRMs)
names(out)
str(out$datW)
str(out$datL)

## see the third vignette for more details
```

---

simulGenosDoseStruct *Simulate SNP genotypes*

Description

Simulates SNP genotypes as allele dose additively encoded, i.e. 0,1,2, using correlated allele frequencies to mimick genetic structure.

Usage

```
simulGenosDoseStruct(
  nb_genos = c(100, 120, 80),
  nb_snps = 1000,
  div_pops = diag(3) * 0.5 + 0.5,
  geno_IDs = NULL,
  snp_IDs = NULL
)
```

Arguments

nb_genos	number of genotypes (i.e. individuals) per population
nb_snps	number of SNPs
div_pops	matrix of divergence among populations, with a diagonal of 1's; the closer off-diagonal values are from 1; the weaker the divergence; the further, the stronger
geno_IDs	vector of genotype identifiers (if NULL, will be "geno001", etc)
snp_IDs	vector of SNP identifiers (if NULL, will be "snp001", etc)

Value

matrix with genotypes in rows and SNPs in columns

Author(s)

Timothee Flutre thanks to code from Andres Legarra

See Also

estimGRM

Examples

```
## weak divergences among populations:
weak_div_pops <- diag(3)
weak_div_pops[upper.tri(weak_div_pops)] <- 0.9
weak_div_pops[lower.tri(weak_div_pops)] <- weak_div_pops[upper.tri(weak_div_pops)]
weak_div_pops

## strong divergences among populations:
strong_div_pops <- diag(3)
strong_div_pops[upper.tri(strong_div_pops)] <- 0.5
strong_div_pops[lower.tri(strong_div_pops)] <- strong_div_pops[upper.tri(strong_div_pops)]
strong_div_pops

M <- simulGenosDoseStruct(div_pops=weak_div_pops)
A <- estimGRM(M)
imageMat(A, "Weak divergence")

M <- simulGenosDoseStruct(div_pops=strong_div_pops)
A <- estimGRM(M)
imageMat(A, "Strong divergence")
```

---

summarizeGMASMAs *Model summary*

Description

Summarizes the GMA-SMA models.

Usage

```
summarizeGMASMAs(fits)
```

Arguments

fits	named list of "merMod" objects returned by lmer, as is done by fitGMASMA with pkg="lme4"

Value

matrix

Author(s)

Timothee Flutre

See Also

fitGMASMA

Examples

```
## see the first vignette
```

---

vec *Vec operator*

Description

Applies the vec operator to a matrix, i.e., concatenate its columns, and save the names, too, if any.

Usage

```
vec(mat, sep = "-")
```

Arguments

mat	matrix
sep	separator of column and row names; used only if mat has dimnames

Value

vector, possibly with names as <rowname><sep><colname>

Author(s)

Timothee Flutre

See Also

invvec

Examples

```
mat <- matrix(c(1, 2, 3,
                4, 5, 6),
              nrow = 2, ncol = 3, byrow = TRUE,
              dimnames = list(as.character(1:2), letters[1:3]))
mat
vec(mat)
```

Index