Package 'FieldSimR'

August 30, 2024

Title Simulation of Plot Errors and Phenotypes in Plant Breeding Field Trials

Version 1.4.0

Date 2024-08-30

Maintainer Christian Werner <werner.christian@proton.me>

Description Simulates plot data in multi-environment field trials with one or more traits. Its core function generates plot errors that capture spatial trend, random error (noise), and extraneous variation, which are combined at a user-defined ratio. Phenotypes can be generated by combining the plot errors with simulated genetic values that capture genotype-by-environment (GxE) interaction using wrapper functions for the R package `AlphaSimR`.

License GPL (>= 3)

URL https://github.com/crWerner/fieldsimr, https://crwerner.github.io/fieldsimr/

Encoding UTF-8

Language en-GB

LazyData true

Imports ggplot2, interp, lattice, Matrix, mbend, cluster, grDevices, RColorBrewer

Suggests AlphaSimR, knitr, rmarkdown

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

VignetteBuilder knitr

BugReports https://github.com/crWerner/fieldsimr/issues

NeedsCompilation no

Author Christian Werner [aut, cre] (<https://orcid.org/0000-0001-9400-5061>), Daniel Tolhurst [aut] (<https://orcid.org/0000-0002-4787-080X>), Jon Bancic [ctb]

Repository CRAN

Date/Publication 2024-08-30 15:00:06 UTC

Contents

`compsym_asr_input` *Simulate genetic values based on a compound symmetry model for GxE interaction - 'AlphaSimR' input parameters*

Description

Creates a list of input parameters for 'AlphaSimR' to simulate genetic values in multiple environments for one or more traits based on a compound symmetry model for genotype-by-environment (GxE) interaction.
This function utilises the ability of 'AlphaSimR' to simulate correlated traits. The wrapper function `compsym_asr_input()` is used to specify the input parameters required in 'AlphaSimR'. After simulating the genetic values, the wrapper function compsym_asr_output can be used to generate a data frame with output values.

Usage

```
compsym_asr_input(
  ntraits = 1,
  nenvs = 2,
  mean = 0,
  var = 1,
```

```
  prop.main = 0.5,
  corA = NULL,
  meanDD = NULL,
  varDD = NULL,
  prop.mainDD = NULL,
  corDD = NULL,
  relAA = NULL,
  prop.mainAA = NULL,
  corAA = NULL
)
```

Arguments

ntraits	Number of traits to be simulated.
nenvs	Number of environments to be simulated (minimum of two).
mean	A vector of mean genetic values for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same mean.
var	A vector of genetic variances for each trait. **Note:** When useVarA = TRUE is specified in 'AlphaSimR' (default), the values in var represent the additive genetic variances, otherwise they represent the total (additive + non-additive) genetic variances.
prop.main	A vector defining the proportion of main effect variance for each trait. If only one value is specified, all traits will be assigned the same proportion. **Note:** 0 < prop.main < 1.
corA	A matrix of additive genetic correlations between traits. By default, a diagonal matrix is constructed.
meanDD	A vector of mean dominance degrees for each environment-within-trait combination (similar to mean). If only one value is specified, all combinations will be assigned the same mean. By default, meanDD = NULL and dominance is not simulated.
varDD	A vector of dominance degree variances for each trait.
prop.mainDD	A vector defining the proportion of dominance degree main effect variance for each trait (similar to prop.main). If only one value is specified, all traits will be assigned the same proportion. **Note:** 0 < prop.mainDD < 1.
corDD	A matrix of dominance degree correlations between traits (similar to corA). If not specified and dominance is simulated, a diagonal matrix is constructed.
relAA	A vector defining the relative magnitude of additive-by-additive (epistatic) variance to additive genetic variance for each trait, that is in a diploid organism with allele frequency of 0.5. If only one value is specified, all traits will be assigned the same relative magnitude.
prop.mainAA	A vector defining the proportion of epistatic main effect variance for each trait (similar to prop.main). If only one value is specified, all traits will be assigned the same proportion. **Note:** 0 < prop.mainAA < 1.
corAA	A matrix of epistatic correlations between traits (similar to corA). If not specified and epistasis is simulated, a diagonal matrix is constructed.

Details

The compound symmetry model assumes the same genetic variance for each environment and the same genetic covariance between each pair of environments. New functionality is being implemented which relaxes the former assumption (also see unstr_asr_output).

Note: 'AlphaSimR' can simulate different biological effects (see: SimParam).

- For additive traits use `addTraitA()`.
- For additive + dominance traits use `addTraitAD()`.
- For additive + epistatic traits use `addTraitAE()`.
- For additive + dominance + epistatic traits use `addTraitADE()`.

Check the `useVarA` argument of these functions when simulating non-additive traits.

Value

A list with input parameters for 'AlphaSimR', which are used to simulate correlated genetic values based on a compound symmetry model for GxE interaction.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive + dominance traits
# in two environments based on a compound symmetry model.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values and mean dominance degrees.
mean <- c(4.9, 5.4, 235.2, 228.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
meanDD <- c(0.4, 0.4, 0.1, 0.1) # Trait 1 and 2, same value for both environments

# Additive genetic variances and dominance degree variances.
var <- c(0.08, 13) # Different values for Traits 1 and 2
varDD <- 0.2 # Same value for Traits 1 and 2

# Proportion of additive and dominance degree main effect variances.
prop.main <- c(0.4, 0.6) # Different values for Traits 1 and 2
prop.mainDD <- 0.4 # Same value for Traits 1 and 2

# Additive and dominance degree correlations between the two simulated traits.
corA <- matrix(c(
  1.0, 0.5,
  0.5, 1.0
), ncol = 2)
corDD <- diag(2) # Assuming independence

input_asr <- compsym_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
  var = var,
  prop.main = prop.main,
  corA = corA,
```

```
  meanDD = meanDD,
  varDD = varDD,
  prop.mainDD = prop.mainDD,
  corDD = corDD
)
```

`compsym_asr_output` *Simulate genetic values based on a compound symmetry model for GxE interaction - Simulation with 'AlphaSimR'*

Description

Creates a data frame of simulated genetic values in multiple environments for one or more traits based on a compound symmetry model for genotype-by-environment (GxE) interaction. The wrapper function `compsym_asr_output()` requires an 'AlphaSimR' population object generated with compsym_asr_input.

Usage

```
compsym_asr_output(pop, ntraits = 1, nenvs, nreps = 1, return.effects = FALSE)
```

Arguments

`pop`	An 'AlphaSimR' population object (Pop-class or HybridPop-class) generated with compsym_asr_input.
`ntraits`	Number of traits specified in compsym_asr_input.
`nenvs`	Number of environments specified in compsym_asr_input.
`nreps`	A vector defining the number of replicates in each environment. If only one value is specified, all environments will be assigned the same number.
`return.effects`	When `TRUE` (default is `FALSE`), a list is returned with additional entries containing the genotype main effects and GxE interaction effects for each trait.

Value

A data frame with columns 'env', genotype 'id', and 'rep', followed by the simulated genetic values for each trait. When `return.effects = TRUE`, a list is returned with additional entries containing the genotype main effects and GxE interaction effects for each trait.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive + dominance traits
# in two environments based on a compound symmetry model.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values and mean dominance degrees.
mean <- c(4.9, 5.4, 235.2, 228.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
```

```
meanDD <- c(0.4, 0.4, 0.1, 0.1) # Trait 1 and 2, same value for both environments

# Additive genetic variances and dominance degree variances.
var <- c(0.08, 13) # Different values for Traits 1 and 2
varDD <- 0.2 # Same value for Traits 1 and 2

# Proportion of additive and dominance degree main effect variances.
prop.main <- c(0.4, 0.6) # Different values for Traits 1 and 2
prop.mainDD <- 0.4 # Same value for Traits 1 and 2

# Additive and dominance degree correlations between the two simulated traits.
corA <- matrix(c(
  1.0, 0.5,
  0.5, 1.0
), ncol = 2)
corDD <- diag(2) # Assuming independence

input_asr <- compsym_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
  var = var,
  prop.main = prop.main,
  corA = corA,
  meanDD = meanDD,
  varDD = varDD,
  prop.mainDD = prop.mainDD,
  corDD = corDD
)


# 2. Use input_asr to simulate genetic values with 'AlphaSimR' based on a
# compound symmetry model.

library("AlphaSimR")
FOUNDERPOP <- quickHaplo(
  nInd = 10,
  nChr = 1,
  segSites = 20
)

SP <- SimParam$new(FOUNDERPOP)



SP$addTraitAD(
  nQtlPerChr = 20,
  mean = input_asr$mean,
  var = input_asr$var,
  corA = input_asr$corA,
  meanDD = input_asr$meanDD,
  varDD = input_asr$varDD,
  corDD = input_asr$corDD,
```

```
  useVarA = TRUE
)

# By default, the variances in 'var' represent additive genetic variances.
# When useVarA = FALSE, the values represent total genetic variances.

pop <- newPop(FOUNDERPOP)


# 3. Create a data frame with simulated genetic values for the two traits in
# the two environments, with two replicates of each genotype.

gv_ls <- compsym_asr_output(
  pop = pop,
  ntraits = 2,
  nenvs = 2,
  nreps = 2,
  return.effects = TRUE
)
```

`error_df_bivar` *Plot errors - Example data frame*

Description

An example data frame with simulated plot errors for two traits in three environments. Environments 1 and 2 comprise two blocks, while Environment 3 comprises three blocks. The blocks are aligned in the column direction (side-by-side) and comprise 5 columns and 20 rows. The data frame was generated using the function field_trial_error with bivariate interpolation. The simulation is demonstrated in the vignette Simulation of plot errors and phenotypes in a plant breeding field trial.

Usage

```
error_df_bivar
```

Format

A data frame with 700 rows and 6 columns:

env Environment number

block Block number

col Column number

row Row number

e.Trait1 Simulated plot errors for Trait 1

e.Trait2 Simulated plot errors for Trait 2

`field_trial_error` *Simulate plot errors in plant breeding field trials*

Description

Creates a data frame of simulated plot errors in multi-environment field trials for one or more traits. The plot errors capture spatial trend, random error (noise), and extraneous variation. Spatial trend is simulated using bivariate interpolation or a separable first-order autoregressive (AR1) process. Random error is simulated using an independent process. Extraneous variation is simulated using random or zig-zag ordering between neighbouring columns and/or rows. The three error components are combined at a user-defined ratio.

Correlated plot errors can be simulated between traits by setting different correlation structures for each error component. A separable structure is assumed between traits and plots within environments, but different error variances can be specified for each environment-within-trait combination.

Usage

```
field_trial_error(
  ntraits = 1,
  nenvs = 1,
  nblocks = 2,
  block.dir = "col",
  ncols = 10,
  nrows = 20,
  varR = 1,
  ScorR = NULL,
  RcorR = NULL,
  EcorR = NULL,
  spatial.model = "Bivariate",
  complexity = NULL,
  plot.length = 8,
  plot.width = 2,
  col.cor = 0.5,
  row.cor = 0.7,
  prop.spatial = 0.5,
  ext.ord = "random",
  ext.dir = "row",
  prop.ext = 0,
  return.effects = FALSE
)
```

Arguments

`ntraits`	Number of traits to be simulated.
`nenvs`	Number of environments to be simulated.
`nblocks`	A vector defining the number of blocks in each environment. If only one value is specified, all environments will be assigned the same number.

`block.dir` A vector defining the block direction in each environment. Use 'col' for side-by-side (default), 'row' for above-and-below, or NA if only one block is simulated. If only one value is specified, all environments will be assigned the same block direction.

`ncols` A vector defining the number of columns in each environment. If only one value is specified, all environments will be assigned the same number.

`nrows` A vector defining the number of rows in each environment. If only one value is specified, all environments will be assigned the same number.

`varR` A vector of error variances for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same error variance.

`ScorR` A matrix of spatial error correlations between traits. If not specified and spatial trend is simulated, a diagonal matrix is constructed.

`RcorR` A matrix of random error correlations between traits. If not specified and random error is simulated, a diagonal matrix is constructed.

`EcorR` A matrix of extraneous error correlations between traits. If not specified and extraneous variation is simulated, a diagonal matrix is constructed.
Note: the same correlation between traits is used for the column and row errors. Currently only implemented when `ext.ord = "random"`.

`spatial.model` A character string defining the model used to simulate spatial trend. Use 'Bivariate' for bivariate interpolation (default) or 'AR1' for a separable first-order autoregressive process. Bivariate interpolation is implemented with the `interp` function of the R package 'interp'.

`complexity` A vector defining the complexity of the simulated spatial trend in each environment when `spatial.model = "Bivariate"`. If only one value is specified, all environments will be assigned the same complexity. If not specified and `spatial.model = "Bivariate"`, the complexity is set to half the maximum number of columns and rows in each environment.

`plot.length` A vector of plot lengths for each environment (column direction). If only one value is specified, all environments will be assigned the same plot length. Only required when `spatial.model = "Bivariate"`.

`plot.width` A vector of plot widths for each environment (row direction). If only one value is specified, all environments will be assigned the same plot width. Only required when `spatial.model = "Bivariate"`.

`col.cor` A vector of column autocorrelations for each environment. If only one value is specified, all environments will be assigned the same column autocorrelation. Only required when `spatial.model = "AR1"`.

`row.cor` A vector of row autocorrelations for each environment. If only one value is specified, all environments will be assigned the same row autocorrelation. Only required when `spatial.model = "AR1"`.

`prop.spatial` A vector defining the proportion of spatial trend for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the proportion.

`ext.ord` A character string defining the method used to simulate extraneous variation. Use 'random' (default) for random variation between neighbouring columns and/or rows or 'zig-zag' for alternating positive and negative values.

`ext.dir` A vector defining the direction of extraneous variation for each environment. Use 'row' (default) for row variation, 'col' for column variation, 'both' for variation in both directions, or NA if no extraneous variation is simulated. When `ext.dir = "both"`, half the variance is assigned to the columns and half is assigned to the rows. If only one value is specified, all environments will be assigned the same direction.

`prop.ext` A vector defining the proportion of extraneous variation for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same proportion.

`return.effects` When `TRUE` (default is `FALSE`), a list is returned with additional entries containing the spatial, random, and extraneous error terms for each trait.

Value

A data frame with columns 'env', 'block', 'col', and 'row', followed by the simulated plot errors for each trait. When `return.effects = TRUE`, a list is returned with additional entries containing the spatial, random, and extraneous error terms for each trait.

Examples

```
# Simulate plot errors for two traits in two environments using an AR1 model
# for spatial variation.

# Error variances for the four environment-within-trait combinations.
varR <- c(0.2, 0.4, 10, 15) # Trait 1 x 2 environments, Trait 2 x 2 environments

# Spatial error correlations between the two simulated traits.
ScorR <- matrix(c(
  1.0, 0.2,
  0.2, 1.0
), ncol = 2)

error_ls <- field_trial_error(
  ntraits = 2,
  nenvs = 2,
  nblocks = 2,
  block.dir = "row",
  ncols = 10,
  nrows = 20,
  varR = varR,
  ScorR = ScorR,
  spatial.model = "AR1",
  col.cor = 0.5,
  row.cor = 0.7,
  prop.spatial = 0.4,
  ext.ord = "zig-zag",
  ext.dir = "row",
```

```
  prop.ext = 0.2,
  return.effects = TRUE
)
```

`group_cor_mat` *Simulate a reduced rank correlation matrix with multiple groups*

Description

Creates a symmetric correlation matrix with user-defined structure, rank and groupings.

Usage

```
group_cor_mat(
  n = c(5, 5),
  within.cor = 0.5,
  between.cor = 0.2,
  range = NULL,
  rank = 4,
  skew = 0,
  pos.def = FALSE,
  small.positive = NULL,
  return.groups = FALSE
)
```

Arguments

`n`	A vector defining the size of each group.
`within.cor`	A vector defining the baseline correlation within each group. If only one value is supplied, all groups will be assigned the same correlation. **Note:** `-1 < within.cor < 1`.
`between.cor`	A scalar defining the baseline correlation between groups. **Note:** `between.cor <= within.cor`.
`range`	A scalar defining the range of correlations around the baseline. By default, `range = 1 - max(within.cor)` which ensures the matrix is positive semi-definite. **Note:** `max(within.cor) + range <= 1`.
`rank`	A scalar defining the rank of the correlation matrix.
`skew`	A scalar defining the skewness imposed on the correlations. **Note:** `-1 < skew < 1`.
`pos.def`	When `TRUE` (default is `FALSE`), the function bend of the R package 'mbend' is used to bend a non-positive (semi)-definite matrix to be positive (semi)-definite.
`small.positive`	Argument passed to bend when `pos.def = TRUE` (default is 1e-8). Eigenvalues smaller than `small.positive` are replaced by this. **Note:** `0 < small.positive < 0.1`.
`return.groups`	When `TRUE` (default is `FALSE`), a list is returned with additional entries containing the members of each group.

Value

A symmetric correlation matrix with defined rank and groupings. When `pos.def = TRUE`, the correlation matrix is guaranteed to be positive (semi)-definite. When `return.groups = TRUE`, a list is returned with additional entries containing the group members.

Examples

```
# Simulate and visualise a correlation matrix with 2 groups containing 5 and 10 members,
# correlations of 0.4 within groups and 0 between groups and rank equal to 4
cor_ls <- group_cor_mat(
  n = c(5, 10),
  within.cor = 0.4,
  between.cor = 0,
  rank = 4,
  return.groups = TRUE
)

plot_matrix(
  mat = cor_ls$cor.mat,
  group.df = cor_ls$group.df,
  order = TRUE
)
```

`gv_df_unstr` *Genetic values - Example data frame*

Description

An example data frame with simulated genetic values of 100 genotypes for two traits in three environments. Environments 1 and 2 comprise two replicates of each genotype, while Environment 3 comprises three replicates. The data frame was generated using the wrapper functions unstr_asr_input and unstr_asr_output, which simulate correlated genetic values with 'AlphaSimR'. The simulation is demonstrated in the vignette Simulation of genetic values based on an unstructured model for GxE interaction.

Usage

```
gv_df_unstr
```

Format

A data frame with 700 rows and 5 columns:

env Environment number

id Genotype identifier

rep Replicate number

gv.Trait1 Simulated genetic values for Trait 1

gv.Trait2 Simulated genetic values for Trait 2

make_phenotypes *Generate phenotypes - Combine genetic values and plot errors*

Description

Creates a data frame of phenotypes by combining genetic values with plot errors generated with the function field_trial_error. Requires genetic values generated with the functions compsym_asr_output or unstr_asr_output, or any data frame matching the description below.

Usage

```
make_phenotypes(
  gv.df,
  error.df,
  design.df = NULL,
  randomise = TRUE,
  return.effects = FALSE
)
```

Arguments

gv.df	A data frame of genetic values. Must contain the columns 'env', genotype 'id', 'rep', and the genetic values for each trait.
error.df	A data frame of plot errors. Must contain the columns 'env', 'block', 'col', 'row', and the plot errors for each trait.
design.df	A optional data frame of frequencies for generating incomplete block designs. Must contain the columns 'env', 'id', and 'nreps' indicating the number of replicates per individual for each environment.
randomise	When TRUE (default), genotypes are randomly allocated to plots according to a randomized complete (or incomplete) block design. **Note:** Other experimental designs are being implemented and should be generated externally.
return.effects	When TRUE (default is FALSE), a list is returned with additional entries containing the genetic values and plot errors for each trait.

Value

A data frame with columns 'env', 'block', 'column', 'row', genotype 'id', 'rep', and the phenotypes for each trait. When return.effects = TRUE, a list is returned with additional entries containing the genetic values and plot errors for each trait.

Examples

```
# Generate and visualise phenotypes by combining the genetic values and plot errors provided
# in the two example data frames gv_df_unstr and error_df_bivar.
```

```
pheno_ls <- make_phenotypes(
  gv.df = gv_df_unstr,
  error.df = error_df_bivar,
  randomise = TRUE,
  return.effects = TRUE
)

plot_effects(
  df = pheno_ls$pheno.df[pheno_ls$pheno.df$env == 1, ],
  effect = "y.Trait1",
  labels = TRUE,
)
```

`multi_asr_input` *Simulate genetic values based on a multiplicative model for GxE interaction - 'AlphaSimR' input parameters*

Description

Creates a list of input parameters for 'AlphaSimR' to simulate genetic values in multiple environments for one or more traits based on a (reduced rank) multiplicative model for genotype-by-environment (GxE) interaction.
This function utilises the ability of 'AlphaSimR' to simulate correlated traits. The wrapper function `multi_asr_input()` is used to specify the input parameters required in 'AlphaSimR'. After simulating the genetic values, the wrapper function multi_asr_output can be used to generate a data frame with output values.

Usage

```
multi_asr_input(
  ntraits = 1,
  nenvs = 2,
  mean = 0,
  var = 1,
  corA = NULL,
  nterms = NULL
)
```

Arguments

`ntraits`	Number of traits to be simulated.
`nenvs`	Number of environments to be simulated (minimum of two).
`mean`	A vector of mean genetic values for each trait or each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same mean.

var	A vector of additive genetic variances for each trait or each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same variance.
corA	A matrix of additive genetic correlations between environment-within-trait combinations. By default, a diagonal matrix is constructed.
nterms	A scalar defining the number of multiplicative terms to be simulated. By default, the number of terms is set to the number of environment-within-trait combinations. **Note:** when nterms is less than the number of environment-within-trait combinations, the values in mean will be approximated.

Details

Currently supports additive traits only, but other (non-additive) traits are being implemented.

Value

A list with input parameters for 'AlphaSimR', which are used to simulate correlated genetic values based on a multiplicative model for GxE interaction. Covariates are also supplied for use in multi_asr_output.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive traits in two
# environments based on a multiplicative model with three terms.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values.
mean <- c(5, 240) # Trait 1, Trait 2

# Additive genetic variances.
var <- c(0.086, 0.12, 15.1, 8.5) # Trait 1 x 2 environments, Trait 2 x 2 environments

# Additive genetic correlations between the two simulated traits.
TcorA <- matrix(c(
  1.0, 0.6,
  0.6, 1.0
), ncol = 2)

# Additive genetic correlations between the two simulated environments.
EcorA <- matrix(c(
  1.0, 0.2,
  0.2, 1.0
), ncol = 2)

# Construct separable additive genetic correlation matrix.
corA <- kronecker(TcorA, EcorA)

input_asr <- multi_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
```

```
  var = var,
  corA = corA,
  nterms = 3
)
```

`multi_asr_output` *Simulate genetic values based on a multiplicative model for GxE interaction - Simulation with 'AlphaSimR'*

Description

Creates a data frame of simulated genetic values in multiple environments for one or more traits based on a (reduced rank) multiplicative model for genotype-by-environment (GxE) interaction. This function requires an 'AlphaSimR' population object generated with multi_asr_input.

Usage

```
multi_asr_output(
  pop,
  ntraits = 1,
  nenvs,
  nreps = 1,
  cov.mat,
  return.effects = FALSE
)
```

Arguments

`pop`	An 'AlphaSimR' population object (Pop-class or HybridPop-class) generated with multi_asr_input.
`ntraits`	Number of traits specified in multi_asr_input.
`nenvs`	Number of environments specified in multi_asr_input.
`nreps`	A vector defining the number of replicates in each environment. If only one value is specified, all environments will be assigned the same number.
`cov.mat`	A matrix of covariates that will be used to construct the genetic values, typically generated with multi_asr_input.
`return.effects`	When `TRUE` (default is `FALSE`), a list is returned with additional entries containing the genotype slopes for each multiplicative term.

Value

A data frame with columns 'env', genotype 'id', and 'rep', followed by the simulated genetic values for each trait. When `return.effects = TRUE`, a list is returned with additional entries containing the genotype slopes for each multiplicative term.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive traits in two
# environments based on a multiplicative model with three terms.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values.
mean <- c(5, 240) # Trait 1, Trait 2

# Additive genetic variances.
var <- c(0.086, 0.12, 15.1, 8.5) # Trait 1 x 2 environments, Trait 2 x 2 environments

# Additive genetic correlations between the two simulated traits.
TcorA <- matrix(c(
  1.0, 0.6,
  0.6, 1.0
), ncol = 2)

# Additive genetic correlations between the two simulated environments.
EcorA <- matrix(c(
  1.0, 0.2,
  0.2, 1.0
), ncol = 2)

# Construct separable additive genetic correlation matrix
corA <- kronecker(TcorA, EcorA)

input_asr <- multi_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
  var = var,
  corA = corA,
  nterms = 3
)


# 2. Use input_asr to simulate genetic values in 'AlphaSimR' based on a
# multiplicative model with three terms.

library("AlphaSimR")
FOUNDERPOP <- quickHaplo(
  nInd = 10,
  nChr = 1,
  segSites = 20
)

SP <- SimParam$new(FOUNDERPOP)



SP$addTraitA(
  nQtlPerChr = 20,
```

```
  mean = input_asr$mean,
  var = input_asr$var,
  corA = input_asr$corA
)

pop <- newPop(FOUNDERPOP)


# 3. Create a data frame with simulated genetic values for the two traits in the two
# environments, with two replicates of each genotype.

# The covariates are obtained from input_asr.

gv_ls <- multi_asr_output(
  pop = pop,
  ntraits = 2,
  nenvs = 2,
  nreps = 2,
  cov.mat = input_asr$cov.mat,
  return.effects = TRUE
)
```

`plot_effects` *Graphics for plot effects*

Description

Creates a graphical field array for a set of plot effects (e.g., phenotypes, genetic values, or plot errors). Requires a data frame generated with the functions field_trial_error or make_phenotypes, or any data frame matching the description below.

Usage

```
plot_effects(df, effect, blocks = TRUE, labels = TRUE)
```

Arguments

`df`	A data frame with the columns 'col', 'row', and the effects to be plotted.
`effect`	The name of the effects to be plotted.
`blocks`	When `TRUE` (default), the field array is split into blocks. This requires an additional column 'block' in the data frame.
`labels`	When `TRUE` (default), column and row labels are displayed.

Value

A graphical field array with x- and y-axes displaying the column and row numbers, and colour gradient ranging from red (low value) to green (high value).

Examples

```
# Display the simulated plot errors in the example data frame 'error_df_bivar'
# for Trait 1 in Environment 1.

error_df <- error_df_bivar[error_df_bivar$env == 1, ]

plot_effects(
  df = error_df,
  effect = "e.Trait1",
  labels = TRUE,
)
```

`plot_hist` *Histogram of values*

Description

Creates a histogram of user-defined values (e.g., effects, correlations, or covariances).

Usage

```
plot_hist(df, value = NULL, bins = 30, density = FALSE)
```

Arguments

`df`	A data frame or vector with the values to be plotted.
`value`	The name of the values to be plotted. Ignored when 'df' is a vector.
`bins`	Argument passed to `ggplot2` (default is `30`). Controls the number of bins in the histogram.
`density`	When `TRUE` (default is `FALSE`), a density curve is superimposed onto the histogram.

Value

A histogram with x- and y-axes displaying the values and their frequency, respectively. When `density = TRUE`, a density curve is superimposed onto the histogram.

Examples

```
# Histogram of the simulated plot errors in the example data frame 'error_df_bivar'
# for Trait 1 in Environment 1.
error_df <- error_df_bivar[error_df_bivar$env == 1, ]
plot_hist(
  df = error_df,
  value = "e.Trait1",
  density = TRUE
)
```

plot_matrix *Graphics for matrices*

Description

Creates a heatmap of a symmetric matrix (e.g., correlation or covariance matrix).

Usage

```
plot_matrix(mat, order = FALSE, group.df = NULL, labels = TRUE)
```

Arguments

mat	A symmetric matrix.
order	When TRUE (default is FALSE), the function agnes of the R package 'cluster' is used with default arguments to order the matrix based on a dendrogram.
group.df	An optional data frame with columns containing the variable names followed by the group numbers. When supplied, the heatmap is split into groups and then ordered (when order = TRUE).
labels	When TRUE (default), variable labels are displayed.

Value

A heatmap with x- and y-axes displaying the variable numbers, and colour gradient ranging from blue (low value) to red (high value).

Examples

```
# Display a random correlation matrix.

cor_mat <- rand_cor_mat(
  n = 10,
  min.cor = -1,
  max.cor = 1
)

# Define groups.
group_df <- data.frame(variable = 1:10, group = c(1, 1, 1, 1, 2, 2, 2, 3, 3, 4))

plot_matrix(
  mat = cor_mat,
  group.df = group_df,
  order = TRUE,
  labels = TRUE
)
```

qq_plot *Q-Q plot*

Description

Creates a normal quantile-quantile (Q-Q) plot for a set of effects (e.g., phenotypes, genetic values, or plot errors).

Usage

```
qq_plot(df, effect, labels = FALSE)
```

Arguments

df	A data frame or vector with the effects to be plotted.
effect	The name of the effects to be plotted. Ignored when 'df' is a vector.
labels	When TRUE (default is FALSE), column and row labels are displayed. This requires additional columns 'col' and 'row' in the data frame.

Value

A Q-Q plot with x- and y-axes displaying the theoretical and sample quantiles of the effects, respectively.

Examples

```
# Q-Q plot of the simulated plot errors in the example data frame 'error_df_bivar'
# for Trait 1 in Environment 1.

error_df <- error_df_bivar[error_df_bivar$env == 1, ]

qq <- qq_plot(
  df = error_df,
  effect = "e.Trait1",
  labels = TRUE
)

# Q-Q plot
qq

# Extract the data frame with the theoretical and sample quantiles of the
# user-defined effects.
qq_df <- qq$data
```

`rand_cor_mat` *Simulate a random correlation matrix*

Description

Creates a symmetric `n` x `n` correlation matrix with user-defined minimum and maximum correlations based on a continuous uniform distribution.

Usage

```
rand_cor_mat(
  n = 5,
  min.cor = -1,
  max.cor = 1,
  pos.def = FALSE,
  small.positive = NULL
)
```

Arguments

`n`	A scalar defining the dimensions of the correlation matrix.
`min.cor`	A scalar defining the minimum correlation.
`max.cor`	A scalar defining the maximum correlation. **Note:** `-1` < `min.cor` < `max.cor` < `1`.
`pos.def`	When `TRUE` (default is `FALSE`), the function `bend` of the R package 'mbend' is used to bend a non-positive (semi)-definite matrix to be positive (semi)-definite.
`small.positive`	Argument passed to `bend` when `pos.def` = `TRUE` (default is 1e-8). Eigenvalues smaller than `small.positive` are replaced by this. **Note:** `0` < `small.positive` < `0.1`.

Value

A symmetric `n` x `n` correlation matrix. When `pos.def` = `TRUE`, the correlation matrix is guaranteed to be positive (semi)-definite.

Examples

```
# Simulate and visualise a random correlation matrix with 10 columns and rows.
cor_mat <- rand_cor_mat(
  n = 10,
  min.cor = -0.2,
  max.cor = 0.8,
  pos.def = TRUE
)

plot_matrix(
  mat = cor_mat,
```

```
  order = TRUE
)
```

`rand_diag_mat` *Simulate a random diagonal variance matrix*

Description

Creates a diagonal `n` x `n` variance matrix with user-defined minimum and maximum variances based on a continuous uniform distribution.

Usage

```
rand_diag_mat(n = 5, min.var = 0, max.var = 1)
```

Arguments

`n`	A scalar defining the dimensions of the variance matrix.
`min.var`	A scalar defining the minimum variance.
`max.var`	A scalar defining the maximum variance. **Note:** `0` < `min.var` < `max.var`.

Value

A diagonal `n` x `n` variance matrix.

Examples

```
# Simulate a random diagonal matrix with 10 columns and rows.
diag_mat <- rand_diag_mat(
  n = 10,
  min.var = 0,
  max.var = 0.2
)
```

`sample_met` *Sample environments from a target population*

Description

Creates a list of environments sampled from a population with user-defined sample size.

Usage

```
sample_met(
  ntraits = 1,
  nenvs = 1000,
  nsamples = 10,
  sample.size = 20,
  replace = TRUE,
  cov.mat = NULL
)
```

Arguments

`ntraits`	A scalar defining the number of traits.
`nenvs`	A scalar defining the number of environments in the target population.
`nsamples`	A scalar defining the number of samples to be taken.
`sample.size`	A vector defining the number of environments in each sample. When only one value is specified, all samples will be assigned the same number.
`replace`	When `TRUE` (default), samples are taken with replacement. Ignored when `nsamples = 1`.
`cov.mat`	An optional matrix of environmental covariates for one or more traits. When supplied, the covariates are sampled and printed.

Value

A list with elements given by the sample of environments taken from the target population. When `cov.mat` is supplied, additional entries are given containing the sampled environmental covariates for each trait.

Examples

```
# Sample environments from a target population of 1000, with each sample containing 20 environments.
cov_ls <- sample_met(
  nenvs = 1000,
  nsamples = 10,
  sample.size = 20,
  replace = TRUE
)
```

sample_variogram *Sample variogram*

Description

Creates a sample variogram for a set of effects (e.g., plot errors).

Usage

```
sample_variogram(df, effect, min.np = 30)
```

Arguments

df	A data frame with the columns 'col', 'row', and the effects to be plotted.
effect	The name of the effects to be plotted.
min.np	Minimum number of pairs for which semivariances are displayed (default is 30).

Value

A sample variogram with x- and y-axes displaying the row and column displacements, and the z-axis displaying the average semivariances (variogram ordinates) for the effects.

Examples

```
# Sample variogram of plot errors simulated using a separable first order
# autoregressive (AR1) process.

error_df <- field_trial_error(
  ntraits = 1,
  nenvs = 1,
  spatial.model = "AR1"
)

variogram <- sample_variogram(
  df = error_df,
  effect = "e.Trait1"
)

# Sample variogram
variogram

# Extract the data frame with the column and row displacements, and the
# average semivariances.
variogram_df <- variogram$data
```

skew_diag_mat *Simulate a skewed diagonal variance matrix*

Description

Creates a diagonal n x n variance matrix with user-defined skewness based on a gamma or inverse gamma distribution.

Usage

```
skew_diag_mat(n = 5, shape = 1.5, scale = 1, inverse = FALSE, mean.var = NULL)
```

Arguments

n	A scalar defining the dimensions of the variance matrix.
shape	A scalar defining the shape of the distribution.
scale	A scalar defining the scale of the distribution.
inverse	When TRUE (default is FALSE), the variances are sampled from the inverse gamma distribution instead of the gamma distribution.
mean.var	An optional scalar defining the mean variance. . When supplied, the variances are scaled to achieve the defined mean.

Value

A diagonal n x n variance matrix.

Examples

```
# Simulate a random diagonal matrix with 10 columns and rows, and negatively skewed variances
# scaled to a mean of 0.1.
diag_mat <- skew_diag_mat(
  n = 10,
  shape = 1.5,
  scale = 1,
  mean.var = 0.1
)
```

struc_cor_mat *Simulate a structured correlation matrix with reduced rank*

Description

Creates a symmetric n x n correlation matrix with user-defined structure and rank.

Usage

```
struc_cor_mat(
  n = 5,
  base.cor = 0.5,
  range = NULL,
  rank = 3,
  skew = 0,
  base.mat = NULL,
  pos.def = FALSE,
  small.positive = NULL
)
```

Arguments

n	A scalar defining the dimensions of the correlation matrix.
base.cor	A scalar defining the baseline correlation. **Note:** -1 < base.cor < 1.
range	A scalar defining the range of correlations around the baseline. By default, range = 1 - base.cor which ensures the matrix is positive semi-definite with defined rank. **Note:** base.cor + range <= 1.
rank	A scalar defining the rank of the correlation matrix.
skew	A scalar defining the skewness imposed on the correlations. **Note:** -1 < skew < 1.
base.mat	An optional n x n base correlation matrix. When supplied, base.cor and skew are ignored and noise is simulated based on rank.
pos.def	When TRUE (default is FALSE), the function bend of the R package 'mbend' is used to bend a non-positive (semi)-definite matrix to be positive (semi)-definite.
small.positive	Argument passed to bend when pos.def = TRUE (default is 1e-8). Eigenvalues smaller than small.positive are replaced by this. **Note:** 0 < small.positive < 0.1.

Value

A symmetric n x n correlation matrix with defined rank. When pos.def = TRUE, the correlation matrix is guaranteed to be positive (semi)-definite.

Examples

```
# Simulate and visualise a correlation matrix with 10 columns and rows, rank equal to 4 and
# negatively skewed correlations.
cor_mat <- struc_cor_mat(
  n = 10,
  base.cor = 0.3,
  range = 0.7,
  rank = 4,
  skew = -0.5
)

plot_matrix(
  mat = cor_mat,
  order = TRUE
)
```

---

theoretical_variogram *Theoretical variogram*

Description

Creates a theoretical variogram for a separable first order autoregressive (AR1) process.

Usage

```
theoretical_variogram(
  ncols = 10,
  nrows = 20,
  varR = 1,
  col.cor = 0.5,
  row.cor = 0.7,
  prop.spatial = 1
)
```

Arguments

ncols	A scalar defining the number of columns.
nrows	A scalar defining the number of rows.
varR	A scalar defining the error variance.
col.cor	A scalar defining the column autocorrelation,
row.cor	A scalar defining the row autocorrelation.
prop.spatial	A scalar defining the proportion of spatial trend.

Value

A theoretical variogram with x- and y-axes displaying the row and column displacements, and the z-axis displaying the semivariances (variogram ordinates) for a separable autoregressive process.

Examples

```
# Theoretical variogram for a field trial with 10 columns and 20 rows, based
# on column and row autocorrelations of 0.5 and 0.7, and a proportion of
# spatial trend of 0.5. The remaining proportion represents random error.

variogram <- theoretical_variogram(
  ncols = 10,
  nrows = 20,
  varR = 1,
  col.cor = 0.5,
  row.cor = 0.7,
  prop.spatial = 0.5
)

# Theoretical variogram
variogram

# Extract the data frame with the column and row displacements, and the
# theoretical semivariances.
variogram_df <- variogram$data
```

---

unstr_asr_input *Simulate genetic values based on an unstructured model for GxE interaction - 'AlphaSimR' input parameters*

Description

Creates a list of input parameters for 'AlphaSimR' to simulate genetic values in multiple environments for one or more traits based on an unstructured model for genotype-by-environment (GxE) interaction.
This function utilises the ability of 'AlphaSimR' to simulate correlated traits. The wrapper function `unstr_asr_input()` is used to specify the input parameters required in 'AlphaSimR', and can handle separable and non-separable structures between traits and environments (see below). After simulating the genetic values, the wrapper function unstr_asr_output can be used to generate a data frame with output values.

Usage

```
unstr_asr_input(
  ntraits = 1,
  nenvs = 2,
  mean = 0,
  var = 1,
  Tvar = NULL,
  Evar = NULL,
  corA = NULL,
  TcorA = NULL,
```

```
  EcorA = NULL,
  meanDD = NULL,
  varDD = NULL,
  TvarDD = NULL,
  EvarDD = NULL,
  corDD = NULL,
  TcorDD = NULL,
  EcorDD = NULL,
  relAA = NULL,
  TrelAA = NULL,
  ErelAA = NULL,
  corAA = NULL,
  TcorAA = NULL,
  EcorAA = NULL
)
```

Arguments

`ntraits`	Number of traits to be simulated.
`nenvs`	Number of environments to be simulated (minimum of two).
`mean`	A vector of mean genetic values for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same mean.
`var`	A vector of genetic variances for each environment-within-trait combination. If only one value is specified, all combinations will be assigned the same variance. **Alternatively**, if a separable structure between traits and environments is desired, `Tvar` and `Evar` can be specified.
`Tvar`	A vector of genetic variances for each trait. Must be provided in combination with `Evar`. **Alternatively**, `var` can be specified.
`Evar`	A vector of genetic variances for each environment. Must be provided in combination with `Tvar`. **Alternatively**, `var` can be specified.
`corA`	A matrix of additive genetic correlations between environment-within-trait combinations. By default, a diagonal matrix is constructed. **Alternatively**, `TcorA` and `EcorA` can be specified.
`TcorA`	A matrix of additive genetic correlations between traits. Must be provided in combination with `EcorA`. **Alternatively**, `corA` can be specified.
`EcorA`	A matrix of additive genetic correlations between environments. Must be provided in combination with `TcorA`. **Alternatively**, `corA` can be specified.
`meanDD`	A vector of mean dominance degrees for each environment-within-trait combination (similar to `mean`). If only one value is specified, all combinations will be assigned the same mean. By default, `meanDD = NULL` and dominance is not simulated.

varDD — A vector of dominance degree variances for each environment-within-trait combination (similar to var). If only one value is specified, all combinations will be assigned the same variance.
Alternatively, if a separable structure between traits and environments is desired, TvarDD and EvarDD can be specified.

TvarDD — A vector of dominance degree variances for each trait (similar to Tvar). Must be provided in combination with EvarDD.
Alternatively, varDD can be specified.

EvarDD — A vector of dominance degree variances for each environment (similar to Evar). Must be provided in combination with TvarDD.
Alternatively, varDD can be specified.

corDD — A matrix of dominance degree correlations between environment-within-trait combinations (similar to corA). If not specified and dominance is simulated, a diagonal matrix is constructed.
Alternatively, TcorDD and EcorDD can be specified.

TcorDD — A matrix of dominance degree correlations between traits (similar to TcorA). Must be provided in combination with EcorDD.
Alternatively, corDD can be specified.

EcorDD — A matrix of dominance degree correlations between environments (similar to EcorA). Must be provided in combination with TcorDD.
Alternatively, corDD can be specified.

relAA — A vector defining the relative magnitude of additive-by-additive (epistatic) variance to additive genetic variance for each environment-within-trait combination, that is in a diploid organism with allele frequency of 0.5. If only one value is specified, all environment-within-trait combinations will be assigned the same value. By default, relAA = NULL and epistasis is not simulated.
Alternatively, if a separable structure between traits and environments is desired, TrelAA and ErelAA can be specified.

TrelAA — A vector defining the relative magnitude of epistatic variance to additive genetic variance for each trait. Must be provided in combination with ErelAA.
Alternatively, relAA can be specified.

ErelAA — A vector defining the relative magnitude of epistatic variance to additive genetic variance for each environment. Must be provided in combination with TrelAA.
Alternatively, relAA can be specified.

corAA — A matrix of epistatic correlations between environment-within-trait combinations (similar to corA). If not specified and epistasis is simulated, a diagonal matrix is constructed.
Alternatively, TcorAA and EcorAA can be specified.

TcorAA — A matrix of epistatic correlations between traits (similar to TcorA). Must be provided in combination with EcorAA.
Alternatively, corAA can be specified.

EcorAA — A matrix of epistatic correlations between environments (similar to EcorA). Must be provided in combination with TcorAA.
Alternatively, corAA can be specified.

Details

`unstr_asr_input` can handle separable and non-separable structures between traits and environments.

- For separable structures, provide (1) `Tvar` & `Evar`, and (2) `TcorA` & `EcorA`.
- For non-separable structures, provide (1) `var`, and (2) `corA`.

Note: 'AlphaSimR' can simulate different biological effects (see: SimParam).

- For additive traits use `addTraitA()`.
- For additive + dominance traits use `addTraitAD()`.
- For additive + epistatic traits use `addTraitAE()`.
- For additive + dominance + epistatic traits use `addTraitADE()`.

Check the `useVarA` argument of these functions when simulating non-additive traits.

Value

A list with input parameters for 'AlphaSimR', which are used to simulate correlated genetic values based on an unstructured model for GxE interaction.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive + dominance traits
# in two environments based on an unstructured model.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values and mean dominance degrees.
mean <- c(4.9, 5.4, 235.2, 228.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
meanDD <- c(0.4, 0.4, 0.1, 0.1) # Trait 1 and 2, same value for both environments

# Additive genetic variances and dominance degree variances.
var <- c(0.086, 0.12, 15.1, 8.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
varDD <- 0.2 # Same value for all environment-within-trait combinations

# Additive genetic correlations between the two simulated traits.
TcorA <- matrix(c(
  1.0, 0.6,
  0.6, 1.0
), ncol = 2)

# Additive genetic correlations between the two simulated environments.
EcorA <- matrix(c(
  1.0, 0.2,
  0.2, 1.0
), ncol = 2)

# Dominance degree correlations between the four environment-within-trait combinations.
corDD <- diag(4) # Assuming independence
```

```
input_asr <- unstr_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
  var = var,
  TcorA = TcorA,
  EcorA = EcorA,
  meanDD = meanDD,
  varDD = varDD,
  corDD = corDD
)
```

`unstr_asr_output` *Simulate genetic values based on an unstructured model for GxE interaction - Simulation with 'AlphaSimR'*

Description

Creates a data frame of simulated genetic values in multiple environments for one or more traits based on an unstructured model for genotype-by-environment (GxE) interaction. The wrapper function `unstr_asr_output` requires an 'AlphaSimR' population object generated with unstr_asr_input.

Usage

```
unstr_asr_output(pop, ntraits = 1, nenvs, nreps = 1)
```

Arguments

`pop`	An 'AlphaSimR' population object (Pop-class or HybridPop-class) generated with unstr_asr_input.
`ntraits`	Number of simulated traits specified in unstr_asr_input.
`nenvs`	Number of simulated environments specified in unstr_asr_input.
`nreps`	A vector defining the number of replicates in each environment. If only one value is specified, all environments will be assigned the same number.

Value

A data frame with columns 'env', genotype 'id', and 'rep', followed by the simulated genetic values for each trait.

Examples

```
# Simulate genetic values with 'AlphaSimR' for two additive + dominance traits
# in two environments based on an unstructured model.

# 1. Define the genetic architecture of the simulated traits.
# Mean genetic values and mean dominance degrees.
```

```
mean <- c(4.9, 5.4, 235.2, 228.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
meanDD <- c(0.4, 0.4, 0.1, 0.1) # Trait 1 and 2, same value for both environments

# Additive genetic variances and dominance degree variances.
var <- c(0.086, 0.12, 15.1, 8.5) # Trait 1 x 2 environments, Trait 2 x 2 environments
varDD <- 0.2 # Same value for all environment-within-trait combinations

# Additive genetic correlations between the two simulated traits.
TcorA <- matrix(c(
  1.0, 0.6,
  0.6, 1.0
), ncol = 2)

# Additive genetic correlations between the two simulated environments.
EcorA <- matrix(c(
  1.0, 0.2,
  0.2, 1.0
), ncol = 2)

# Dominance degree correlations between the four environment-within-trait combinations.
corDD <- diag(4) # Assuming independence

input_asr <- unstr_asr_input(
  ntraits = 2,
  nenvs = 2,
  mean = mean,
  var = var,
  TcorA = TcorA,
  EcorA = EcorA,
  meanDD = meanDD,
  varDD = varDD,
  corDD = corDD
)


# 2. Use input_asr to simulate genetic values with 'AlphaSimR' based on an
# unstructured model.

library("AlphaSimR")
FOUNDERPOP <- quickHaplo(
  nInd = 10,
  nChr = 1,
  segSites = 20
)

SP <- SimParam$new(FOUNDERPOP)



SP$addTraitAD(
  nQtlPerChr = 20,
  mean = input_asr$mean,
  var = input_asr$var,
```

```
  corA = input_asr$corA,
  meanDD = input_asr$meanDD,
  varDD = input_asr$varDD,
  corDD = input_asr$corDD,
  useVarA = TRUE
)

# By default, the variances in 'var' represent additive genetic variances.
# When useVarA = FALSE, the values represent total genetic variances.

pop <- newPop(FOUNDERPOP)


# 3. Create a data frame with simulated genetic values for the two traits in
# the two environments, with two replicates of each genotype.

gv_df <- unstr_asr_output(
  pop = pop,
  ntraits = 2,
  nenvs = 2,
  nreps = 2
)
```

Index