Package 'simtrait'

January 6, 2023

Title Simulate Complex Traits from Genotypes

Version 1.1.3

Description

Simulate complex traits given a SNP genotype matrix and model parameters (the desired heritability, number of causal loci, and either the true ancestral allele frequencies used to generate the genotypes or the mean kinship for a real dataset). Emphasis on avoiding common biases due to the use of estimated allele frequencies. The code selects random loci to be causal, constructs coefficients for these loci and random independent non-genetic effects, and can optionally generate random group effects. Traits can follow three models: random coefficients, fixed effect sizes, and infinitesimal (multivariate normal). GWAS method benchmarking functions are also provided. Described in Yao and Ochoa (2022) <doi:10.1101/2022.03.25.485885>.

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Imports PRROC

Suggests popkin, testthat, knitr, rmarkdown, bnpsd, BEDMatrix

VignetteBuilder knitr

URL https://github.com/OchoaLab/simtrait

BugReports https://github.com/OchoaLab/simtrait/issues

NeedsCompilation no

Author Alejandro Ochoa [aut, cre] (<https://orcid.org/0000-0003-4928-3403>)

Maintainer Alejandro Ochoa <alejandro.ochoa@duke.edu>

Repository CRAN

Date/Publication 2023-01-06 21:10:02 UTC

R topics documented:

allele_freqs 2
cov_trait 3
herit_loci 4

pval_aucpr 5
pval_infl 6
pval_power_calib 7
pval_srmsd 8
pval_type_1_err 10
rmsd 11
simtrait 11
sim_trait 13
sim_trait_mvn 16

Index **19**

`allele_freqs` *Compute locus allele frequencies*

Description

On a regular matrix, this is essentially a wrapper for `colMeans()` or `rowMeans()` depending on `loci_on_cols`. On a BEDMatrix object, the locus allele frequencies are computed keeping memory usage low.

Usage

```
allele_freqs(X, loci_on_cols = FALSE, fold = FALSE, m_chunk_max = 1000)
```

Arguments

`X`	The genotype matrix (regular R matrix or BEDMatrix object). Missing values are ignored in averages.
`loci_on_cols`	If `TRUE`, `X` has loci on columns and individuals on rows; if false (the default), loci are on rows and individuals on columns. If `X` is a BEDMatrix object, code assumes loci on columns (`loci_on_cols` is ignored).
`fold`	If `TRUE`, allele frequencies are converted to minor allele frequencies. Default is to return frequencies for the given allele counts in `X` (regardless of whether it is the minor or major allele).
`m_chunk_max`	BEDMatrix-specific, sets the maximum number of loci to process at the time. If memory usage is excessive, set to a lower value than default (expected only for extremely large numbers of individuals).

Value

The vector of allele frequencies, one per locus. Names are set to the locus names, if present.

Examples

```
# Construct toy data
X <- matrix(
    c(0, 1, 2,
      1, 0, 1,
      1, NA, 2),
    nrow = 3,
    byrow = TRUE
)

# row means
allele_freqs(X)
c(1/2, 1/3, 3/4)

# row means, in minor allele frequencies
allele_freqs(X, fold = TRUE)
c(1/2, 1/3, 1/4)

# col means
allele_freqs(X, loci_on_cols = TRUE)
c(1/3, 1/4, 5/6)
```

---

cov_trait *The model covariance matrix of the trait*

Description

This function returns the expected covariance matrix of trait vectors simulated via `sim_trait()` and `sim_trait_mvn()`. Below there are `n` individuals.

Usage

```
cov_trait(kinship, herit, sigma_sq = 1, labs = NULL, labs_sigma_sq = NULL)
```

Arguments

`kinship`	The `n`-by-`n` kinship matrix of the individuals. These values should be scaled such that an outbred individual has 1/2 self-kinship, the parent-child relationship is 1/4, etc (which is half the values sometimes defined for kinship).
`herit`	The desired heritability (proportion of trait variance due to genetics).
`sigma_sq`	The desired parametric variance factor of the trait (scalar, default 1). Corresponds to the variance of an outbred individual.
`labs`	Optional labels assigning individuals to groups, to simulate group effects. If vector, length must be number of individuals. If matrix, individuals must be along rows, and levels along columns (for multiple levels of group effects). The levels are not required to be nested (as the name may falsely imply). Values can be numeric or strings, simply assigning the same values to individuals in the same group. If this is non-`NULL`, then `labs_sigma_sq` must also be given!

`labs_sigma_sq` Optional vector of group effect variance proportions, one value for each level given in `labs` (a scalar if `labs` is a vector, otherwise its length should be the number of columns of `labs`). Ignored unless `labs` is also given. As these are variance proportions, each value must be non-negative and `sum(labs_sigma_sq) + herit <= 1` is required so residual variance is non-negative.

Value

The n-by-n trait covariance matrix, which under no environment effects equals `sigma_sq * ( herit * 2 * kinship + sigma_sq_residual * I )`, where `I` is the n-by-n identity matrix and `sigma_sq_residual = 1 - herit`. If there are labels, covariance will include the specified block diagonal effects and `sigma_sq_residual = 1 - herit - sum(labs_sigma_sq)`.

See Also

`sim_trait()`, `sim_trait_mvn()`

Examples

```
# create a dummy kinship matrix
kinship <- matrix(
    data = c(
        0.6, 0.1, 0.0,
        0.1, 0.6, 0.1,
        0.0, 0.1, 0.6
    ),
    nrow = 3,
    byrow = TRUE
)
# covariance of simulated traits
V <- cov_trait(kinship = kinship, herit = 0.8)
```

---

herit_loci *Per-locus heritability contribution from allele frequency and causal coefficient*

Description

Calculates the vector of per-locus heritability values, with each causal locus `i` calculated as `h_i^2 = 2 * p_i * ( 1 - p_i ) * beta_i^2 / sigma_sq`, where `p_i` is the ancestral allele frequency, `beta_i` is the causal coefficient, and `sigma_sq` is the trait variance scale. These are all assumed to be true parameters (not estimated). These per-locus heritabilities equal per-locus effect sizes divided by `sigma_sq`.

Usage

```
herit_loci(p_anc, causal_coeffs, causal_indexes = NULL, sigma_sq = 1)
```

Arguments

p_anc	The ancestral allele frequency vector.
causal_coeffs	The vector of causal coefficients.
causal_indexes	The optional vector of causal indexes. If NULL (default), p_anc and causal_coeffs are assumed to be for causal loci only (must be the same length). If non-NULL, causal_loci is used to subset both p_anc and causal_coeffs as needed: if each of these vectors is longer than causal_loci, then it is subset; otherwise they must have equal lengths as causal_loci or an error is thrown.
sigma_sq	The parametric variance factor of the trait (default 1). This factor corresponds to the variance of an outbred individual.

Value

The vector of per-locus heritability contributions. The sum of these values gives the overall heritability. This value can be greater than one (or wrong, more generally) if sigma_sq is misspecified.

See Also

sim_trait() generates random traits by drawing causal loci and their coefficients to fit a desired heritability. cov_trait() calculates the covariance structure of the random traits.

Examples

```
# create toy random data
m_loci <- 10
# ancestral allele frequencies
p_anc <- runif( m_loci )
# causal loci
causal_coeffs <- rnorm( m_loci ) / m_loci
# resulting heritability contributions vector
herit_loci( p_anc, causal_coeffs )
```

---

pval_aucpr *Area under the precision-recall curve*

Description

Calculates the Precision-Recall (PR) Area Under the Curve (AUC) given a vector of p-values and the true classes (causal (alternative) vs non-causal (null)). This is a wrapper around PRROC::pr.curve(), which actually calculates the AUC (see that for details).

Usage

```
pval_aucpr(pvals, causal_indexes, curve = FALSE)
```

Arguments

pvals	The vector of association p-values to analyze. NA values are allowed in input, are internally set to 1 (worst score) prior to AUC calculation (to prevent methods to get good AUCs by setting more cases to NA). Non-NA values outside of [0,1] will trigger an error.
causal_indexes	The vector of causal indexes, defining the true classes used for AUC calculation. Values of causal_indexes as returned by sim_trait work. There must be at least one causal index and at least one non-causal case.
curve	If FALSE (default), only scalar AUC is returned. If TRUE, then curve = TRUE is passed to PRROC::pr.curve() and the full object (class PRROC) is returned (see below).

Value

If curve = FALSE, returns the PR AUC scalar value. If curve = TRUE, returns the PRROC object as returned by PRROC::pr.curve(), which can be plotted directly, and which contains the AUC under the named value auc.integral.

However, if the input pvals is NULL (taken for case of singular association test, which is rare but may happen), then the returned value is NA.

See Also

PRROC::pr.curve(), which is used internally by this function.

pval_power_calib() for calibrated power estimates.

Examples

```
# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# for toy example, take the first two p-values to be truly causal
causal_indexes <- 1:2
# calculate desired measure
pval_aucpr( pvals, causal_indexes )
```

---

pval_infl *Calculate inflation factor from p-values*

Description

The inflation factor is defined as the median association test statistic divided by the expected median under the null hypothesis, which is typically assumed to have a chi-squared distribution. This function takes a p-value distribution and maps its median back to the chi-squared value (using the quantile function) in order to compute the inflation factor in the chi-squared scale. The full p-value distribution (a mix of null and alternative cases) is used to calculate the desired median value (the true causal_loci is not needed, unlike pval_srmsd()).

Usage

```
pval_infl(pvals, df = 1)
```

Arguments

pvals	The vector of association p-values to analyze. This function assumes all p-values are provided (a mix of null and alternative tests). NA values are allowed in input and removed. Non-NA values outside of [0, 1] will trigger an error.
df	The degrees of freedom of the assumed chi-squared distribution (default 1).

Value

The inflation factor

See Also

pval_srmsd(), a more robust measure of null p-value accuracy, but which requires knowing the true causal loci.

pval_type_1_err() for classical type I error rate estimates.

Examples

```
# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# calculate desired measure
pval_infl( pvals )
```

---

pval_power_calib *Estimate calibrated power*

Description

Given a significance level alpha and p-values with known causal status, this function estimates the calibrated power. First it estimates the p-value threshold at which the desired type I error of alpha is achieved, then it uses this p-value threshold (not alpha) to estimate statistical power. Note that these simple empirical estimates are likely to be inaccurate unless the number of p-values is much larger than 1/alpha.

Usage

```
pval_power_calib(pvals, causal_indexes, alpha = 0.05)
```

Arguments

pvals	The vector of association p-values to analyze. This function assumes all p-values are provided (a mix of null and alternative tests). NA values are allowed in input and removed. Non-NA values outside of [0, 1] will trigger an error.
causal_indexes	The vector of causal indexes, defining the true classes used for calibrated power estimation. Values of causal_indexes as returned by sim_trait work. There must be at least one causal index and at least one non-causal case.
alpha	The desired significance level (default 0.05). May be a vector.

Value

The calibrated power estimates at each alpha

See Also

pval_aucpr(), a robust proxy for calibrated power that integrates across significance thresholds.

Examples

```
# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# for toy example, take the first two p-values to be truly causal
causal_indexes <- 1:2
# estimate desired measure
pval_power_calib( pvals, causal_indexes )
```

---

pval_srmsd *Signed RMSD measure of null p-value uniformity*

Description

Quantifies null p-value uniformity by computing the RMSD (root mean square deviation) between the sorted observed null (truly non-causal) p-values and their expected quantiles under a uniform distribution. Meant as a more robust alternative to the "inflation factor" common in the GWAS literature, which compares median values only and uses all p-values (not just null p-values). Our signed RMSD, to correspond with the inflation factor, includes a sign that depends on the median null p-value: positive if this median is <= 0.5 (corresponds with test statistic inflation), negative otherwise (test statistic deflation). Zero corresponds to uniform null p-values, which arises in expectation only if test statistics have their assumed null distribution (there is no misspecification, including inflation).

Usage

```
pval_srmsd(pvals, causal_indexes, detailed = FALSE)
```

Arguments

pvals	The vector of association p-values to analyze. This function assumes all p-values are provided (a mix of null and alternative tests). NA values are allowed in input and removed. Non-NA values outside of [0, 1] will trigger an error.
causal_indexes	The vector of causal indexes, whose p-values will be omitted. Values of causal_indexes as returned by sim_trait work. This parameter is required to prevent use of this function except when the true status of every test (null vs alternative) is known. Set to NULL if all loci are truly null (non-causal). Otherwise, causal_indexes must have at least one causal index.
detailed	If FALSE (default) only SRMSD is returned. If TRUE, sorted null p-values without NAs and their expectations are returned (useful for plots).

Value

If detailed is FALSE, returns the signed RMSD between the observed p-value order statistics and their expectation under true uniformity. If detailed is TRUE, returns data useful for plots, a named list containing:

- srmsd: The signed RMSD between the observed p-value order statistics and their expectation under true uniformity.
- pvals_null: Sorted null p-values (observed order statistics). If any input null p-values were NA, these have been removed here (removed by sort()).
- pvals_unif: Expected order statistics assuming uniform distribution, same length as pvals_null.

If the input pvals is NULL (taken for case of singular association test, which is rare but may happen), then the returned value is NA if detailed was FALSE, or otherwise the list contains NA, NULL and NULL for the above three items.

See Also

rmsd() for the generic root-mean-square deviation function.

pval_infl() for the more traditional inflation factor, which focuses on the median of the full distribution (combination of causal and null cases).

pval_type_1_err() for classical type I error rate estimates.

Examples

```
# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# for toy example, take the first p-value to be truly causal (will be ignored below)
causal_indexes <- 1
# calculate desired measure
pval_srmsd( pvals, causal_indexes )
```

pval_type_1_err *Estimate type I error rate*

Description

Given a significance level and p-values with known causal status, this function estimates the type I error rate, defined as the proportion of null p-values that are below or equal to the threshold. Note that these simple empirical estimates are likely to be zero unless the number of p-values is much larger than `1/alpha`.

Usage

```
pval_type_1_err(pvals, causal_indexes, alpha = 0.05)
```

Arguments

`pvals`	The vector of association p-values to analyze. This function assumes all p-values are provided (a mix of null and alternative tests). `NA` values are allowed in input and removed. Non-`NA` values outside of [0, 1] will trigger an error.
`causal_indexes`	The vector of causal indexes, whose p-values will be omitted. Values of `causal_indexes` as returned by `sim_trait` work. This parameter is required to prevent use of this function except when the true status of every test (null vs alternative) is known. Set to `NULL` if all loci are truly null (non-causal). Otherwise, `causal_indexes` must have at least one causal index.
`alpha`	The desired significance level (default 0.05). May be a vector.

Value

The type I error rate estimates at each `alpha`

See Also

`pval_srmsd()` to directly quantify null p-value uniformity, a more robust alternative to type I error rate.

`pval_infl()` for the more traditional inflation factor, which focuses on the median of the full distribution (combination of causal and null cases).

Examples

```
# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# for toy example, take the first p-value to be truly causal (will be ignored below)
causal_indexes <- 1
# estimate desired measure
pval_type_1_err( pvals, causal_indexes )
```

rmsd *Root mean square deviation*

Description

Calculates the euclidean distance between two vectors x and y divided by the square root of the lengths of the vectors. NA values are ignored by default when calculating the mean squares (so the denominator is the number of non-NA differences).

Usage

```
rmsd(x, y, na.rm = TRUE)
```

Arguments

x	The first vector to compare (required).
y	The second vector to compare (required). Lengths of x and y must be equal.
na.rm	If TRUE (default), NA values are removed before calculating the mean square difference. If FALSE, any missing values in either x or y result in NA returned. Passed to mean(), see that for more info.

Value

the square root of the mean square difference between x and y, after removing NA comparisons (cases where either is NA).

Examples

```
x <- rnorm(10)
y <- rnorm(10)
rmsd( x, y )
```

simtrait *simtrait: simulate complex traits from genotypes*

Description

This package enables simulation of complex (polygenic and continuous) traits from a simulated or real genotype matrix. The focus is on constructing the mean and covariance structure of the data to yield the desired heritability. The main function is sim_trait(), which returns the simulated trait and the vector of causal loci (randomly selected) and their coefficients. The causal coefficients are constructed under two models: *random coefficients* (RC) and *fixed effect sizes* (FES). The function cov_trait() computes the expected covariance matrix of the trait given the model parameters (namely the desired heritability and the true kinship matrix). Infinitesimal traits (without causal loci) can also be simulated using sim_trait_mvn().

Details

Package also provides some functions for evaluating genetic association approaches. `pval_srmsd()` and `pval_infl()` quantify null p-value accuracy, while `pval_aucpr()` quantifies predictive power.

The recommended inputs are simulated genotypes with known ancestral allele frequencies. The `bnpsd` package simulates genotypes for admixed individuals, resulting in a complex population structure.

For real data it is necessary to estimate the kinship matrix. `popkin::popkin()`‘ provides high-accuracy kinship estimates.

Author(s)

Maintainer: Alejandro Ochoa <alejandro.ochoa@duke.edu> (ORCID)

See Also

Useful links:

- https://github.com/OchoaLab/simtrait
- Report bugs at https://github.com/OchoaLab/simtrait/issues

Examples

```
# construct a dummy genotype matrix
X <- matrix(
    data = c(
        0, 1, 2,
        1, 2, 1,
        0, 0, 1
    ),
    nrow = 3,
    byrow = TRUE
)
# made up ancestral allele frequency vector for example
p_anc <- c(0.5, 0.6, 0.2)
# desired heritability
herit <- 0.8
# create a dummy kinship matrix for example
# make sure it is positive definite!
kinship <- matrix(
    data = c(
        0.6, 0.1, 0.0,
        0.1, 0.5, 0.0,
        0.0, 0.0, 0.5
    ),
    nrow = 3
)

# create simulated trait and associated data
# default is *random coefficients* (RC) model
obj <- sim_trait(X = X, m_causal = 2, herit = herit, p_anc = p_anc)
```

```
# trait vector
obj$trait
# randomly-picked causal locus indeces
obj$causal_indexes
# regression coefficient vector
obj$causal_coeffs

# *fixed effect sizes* (FES) model
obj <- sim_trait(X = X, m_causal = 2, herit = herit, p_anc = p_anc, fes = TRUE)

# either model, can apply to real data by replacing `p_anc` with `kinship`
obj <- sim_trait(X = X, m_causal = 2, herit = herit, kinship = kinship)

# covariance of simulated traits
V <- cov_trait(kinship = kinship, herit = herit)

# draw simulated traits (matrix of replicates) from infinitesimal model
traits <- sim_trait_mvn( rep = 10, kinship = kinship, herit = herit )
traits

# Metrics for genetic association approaches

# simulate truly null p-values, which should be uniform
pvals <- runif(10)
# for toy example, take these p-value to be truly causal
causal_indexes <- c(1, 5, 7)

# calculate desired measures
# this one quantifies p-value uniformity
pval_srmsd( pvals, causal_indexes )
# related, calculates inflation factors
pval_infl( pvals )
# this one quantifies predictive power
pval_aucpr( pvals, causal_indexes )
```

`sim_trait` *Simulate a complex trait from genotypes*

Description

Simulate a complex trait given a SNP genotype matrix and model parameters, which are minimally: the number of causal loci, the heritability, and either the true ancestral allele frequencies used to generate the genotypes or the mean kinship of all individuals. An optional minimum marginal allele frequency for the causal loci can be set. The output traits have by default a zero mean and unit variance (for outbred individuals), but those parameters can be modified. The code selects random loci to be causal, constructs coefficients for these loci (scaled appropriately) and random Normal independent non-genetic effects and random group effects if specified. There are two models for constructing causal coefficients: random coefficients (RC; default) and fixed effect sizes (FES; i.e.,

coefficients roughly inversely proportional to allele frequency; use `fes = TRUE`). Suppose there are `m` loci and `n` individuals.

Usage

```
sim_trait(
  X,
  m_causal,
  herit,
  p_anc = NULL,
  kinship = NULL,
  mu = 0,
  sigma_sq = 1,
  labs = NULL,
  labs_sigma_sq = NULL,
  maf_cut = NA,
  loci_on_cols = FALSE,
  m_chunk_max = 1000,
  fes = FALSE
)
```

Arguments

`X`	The `m`-by-`n` genotype matrix (if `loci_on_cols = FALSE`, transposed otherwise), or a BEDMatrix object. This is a numeric matrix consisting of reference allele counts (in `c(0, 1, 2, NA)` for a diploid organism).
`m_causal`	The desired number of causal loci.
`herit`	The desired heritability (proportion of trait variance due to genetics).
`p_anc`	The length-`m` vector of true ancestral allele frequencies. Optional but recommended for simulations. Either this or `kinship` must be specified.
`kinship`	The mean kinship value of the individuals in the data. The `n`-by-`n` kinship matrix of the individuals in the data is also accepted. Optional but recommended for real data. Either this or `p_anc` must be specified.
`mu`	The desired parametric mean value of the trait (scalar, default 0).
`sigma_sq`	The desired parametric variance factor of the trait (scalar, default 1). Corresponds to the variance of an outbred individual.
`labs`	Optional labels assigning individuals to groups, to simulate group effects. If vector, length must be number of individuals. If matrix, individuals must be along rows, and levels along columns (for multiple levels of group effects). The levels are not required to be nested (as the name may falsely imply). Values can be numeric or strings, simply assigning the same values to individuals in the same group. If this is non-`NULL`, then `labs_sigma_sq` must also be given!
`labs_sigma_sq`	Optional vector of group effect variance proportions, one value for each level given in `labs` (a scalar if `labs` is a vector, otherwise its length should be the number of columns of `labs`). Ignored unless `labs` is also given. As these are variance proportions, each value must be non-negative and `sum(labs_sigma_sq) + herit <= 1` is required so residual variance is non-negative.

maf_cut
: The optional minimum allele frequency threshold (default NA, no threshold). This prevents rare alleles from being causal in the simulation. Threshold is applied to the *sample* allele frequencies and not their true parametric values (p_anc), even if these are available.

loci_on_cols
: If TRUE, X has loci on columns and individuals on rows; if FALSE (the default), loci are on rows and individuals on columns. If X is a BEDMatrix object, loci are always on the columns (loci_on_cols is ignored).

m_chunk_max
: BEDMatrix-specific, sets the maximum number of loci to process at the time. If memory usage is excessive, set to a lower value than default (expected only for extremely large numbers of individuals).

fes
: If TRUE, causal coefficients are inversely proportional to the square root of p_anc * (1 - p_anc) (estimated when p_anc is unavailable), which ensures *fixed effect sizes* (FES) per causal locus. Signs (+/-) are drawn randomly with equal probability. If FALSE (the default), *random coefficients* (RC) are drawn from a standard Normal distribution. In both cases coefficients are rescaled to result in the desired heritability.

Details

To center and scale the trait and locus coefficients vector correctly to the desired parameters (mean, variance, heritability), the parametric ancestral allele frequencies (p_anc) must be known. This is necessary since in the heritability model the genotypes are random variables (with means given by p_anc and a covariance structure given by p_anc and the kinship matrix), so these genotype distribution parameters are required. If p_anc are known (true for simulated genotypes), then the trait will have the specified mean and covariance matrix in agreement with cov_trait(). To simulate traits using real genotypes, where p_anc is unknown, a compromise that works well in practice is possible if the mean kinship is known (see package vignette). We recommend estimating the mean kinship using the popkin package!

Value

A named list containing:

- trait: length-n vector of the simulated trait
- causal_indexes: length-m_causal vector of causal locus indexes
- causal_coeffs: length-m_causal vector of coefficients at the causal loci
- group_effects: length-n vector of simulated group effects, or 0 (scalar) if not simulated

However, if herit = 0 then causal_indexes and causal_coeffs will have zero length regardless of m_causal.

See Also

cov_trait(), sim_trait_mvn()

Examples

```
# construct a dummy genotype matrix
X <- matrix(
    data = c(
        0, 1, 2,
        1, 2, 1,
        0, 0, 1
    ),
    nrow = 3,
    byrow = TRUE
)
# made up ancestral allele frequency vector for example
p_anc <- c(0.5, 0.6, 0.2)
# made up mean kinship
kinship <- 0.2
# desired heritability
herit <- 0.8

# create simulated trait and associated data
# default is *random coefficients* (RC) model
obj <- sim_trait(X = X, m_causal = 2, herit = herit, p_anc = p_anc)

# trait vector
obj$trait
# randomly-picked causal locus indexes
obj$causal_indexes
# regression coefficients vector
obj$causal_coeffs

# *fixed effect sizes* (FES) model
obj <- sim_trait(X = X, m_causal = 2, herit = herit, p_anc = p_anc, fes = TRUE)

# either model, can apply to real data by replacing `p_anc` with `kinship`
obj <- sim_trait(X = X, m_causal = 2, herit = herit, kinship = kinship)
```

`sim_trait_mvn` *Simulate traits from a kinship matrix under the infinitesimal model*

Description

Simulate matrix of trait replicates given a kinship matrix and model parameters (the desired heritability, group effects, total variance scale, and mean). Although these traits have the covariance structure of genetic traits, and have heritabilities that can be estimated, they do not have causal loci (an association test against any locus should fail). Below `n` is the number of individuals.

Usage

```
sim_trait_mvn(
```

```
  rep,
  kinship,
  herit,
  mu = 0,
  sigma_sq = 1,
  labs = NULL,
  labs_sigma_sq = NULL,
  tol = 1e-06
)
```

Arguments

rep	The number of replicate traits to simulate. Simulating all you need at once is more efficient than simulating each separately (the kinship matrix is eigendecomposed once per run, shared across replicates).
kinship	The n-by-n kinship matrix of the individuals to simulate from.
herit	The desired heritability (proportion of trait variance due to genetics).
mu	The desired parametric mean value of the trait (scalar, default 0).
sigma_sq	The desired parametric variance factor of the trait (scalar, default 1). Corresponds to the variance of an outbred individual.
labs	Optional labels assigning individuals to groups, to simulate group effects. If vector, length must be number of individuals. If matrix, individuals must be along rows, and levels along columns (for multiple levels of group effects). The levels are not required to be nested (as the name may falsely imply). Values can be numeric or strings, simply assigning the same values to individuals in the same group. If this is non-NULL, then labs_sigma_sq must also be given!
labs_sigma_sq	Optional vector of group effect variance proportions, one value for each level given in labs (a scalar if labs is a vector, otherwise its length should be the number of columns of labs). Ignored unless labs is also given. As these are variance proportions, each value must be non-negative and sum(labs_sigma_sq) + herit <= 1 is required so residual variance is non-negative.
tol	Tolerance factor for an internal test of positive semi-definiteness of the trait covariance matrix. Procedure fails if any eigenvalues are smaller than -tol times the absolute value of the largest eigenvalue. Increase this value only if you are getting errors but you're sure your covariance matrix (the output of cov_trait()) is positive semi-definite.

Value

A rep-by-n matrix containing the simulated traits along the rows, individuals along the columns.

See Also

cov_trait(), sim_trait()

Examples

```
# create a dummy kinship matrix
# make sure it is positive definite!
kinship <- matrix(
    data = c(
        0.6, 0.1, 0.0,
        0.1, 0.5, 0.0,
        0.0, 0.0, 0.5
    ),
    nrow = 3
)
# draw simulated traits (matrix)
traits <- sim_trait_mvn( rep = 10, kinship = kinship, herit = 0.8 )
traits
```

Index

allele_freqs, 2

colMeans(), *2*
cov_trait, 3
cov_trait(), *5*, *11*, *15*, *17*

herit_loci, 4

mean(), *11*

popkin::popkin(), *12*
PRROC::pr.curve(), *5*, *6*
pval_aucpr, 5
pval_aucpr(), *8*, *12*
pval_infl, 6
pval_infl(), *9*, *10*, *12*
pval_power_calib, 7
pval_power_calib(), *6*
pval_srmsd, 8
pval_srmsd(), *6*, *7*, *10*, *12*
pval_type_1_err, 10
pval_type_1_err(), *7*, *9*

rmsd, 11
rmsd(), *9*
rowMeans(), *2*

sim_trait, 13
sim_trait(), *3–5*, *11*, *17*
sim_trait_mvn, 16
sim_trait_mvn(), *3*, *4*, *11*, *15*
simtrait, 11
simtrait-package *(*simtrait*)*, 11
sort(), *9*