

Package: simtrait (via r-universe)

September 23, 2026

Title Simulate Complex Traits from Genotypes

Version 1.1.11.9000

Description Simulate complex traits given a SNP genotype matrix and model parameters (the desired heritability, optional environment group effects, 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 is 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 (2023) <[doi:10.7554/eLife.79238](https://doi.org/10.7554/eLife.79238)>.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports PRROC, stats

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

VignetteBuilder knitr

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

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

Repository <https://ochoalab.r-universe.dev>

Date/Publication 2025-07-08 20:14:32 UTC

RemoteUrl <https://github.com/ochoalab/simtrait>

RemoteRef HEAD

RemoteSha 89b19084517ff563c79b8a490cec9fb1d1925b95

Contents

allele_freqs	2
cov_trait	3
herit_loci	5
inv_var_est_bayesian	6
p_anc_est_beta_mle	7
pval_aucpr	8
pval_gc	9
pval_infl	10
pval_power_calib	11
pval_srmsd	12
pval_type_1_err	14
rmsd	15
sim_trait	15
sim_trait_model	19
sim_trait_mvn	21
simtrait	23
Index	26

allele_freqs	<i>Compute locus allele frequencies</i>
--------------	---

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,
  subset_ind = NULL,
  want_counts = FALSE
)
```

Arguments

<code>X</code>	The genotype matrix (regular R matrix or <code>BEDMatrix</code> object). Missing values are ignored in averages.
<code>loci_on_cols</code>	If <code>TRUE</code> , <code>X</code> has loci on columns and individuals on rows; if <code>false</code> (the default), loci are on rows and individuals on columns. If <code>X</code> is a <code>BEDMatrix</code> object, code assumes loci on columns (<code>loci_on_cols</code> 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).
subset_ind	Optionally subset individuals by providing their indexes (negative indexes to exclude) or a boolean vector (in other words, the usual ways to subset matrices). Most useful for BEDMatrix inputs, to subset chunks and retain low memory usage.
want_counts	If TRUE (default FALSE), raw allele counts are also returned. Note fold option has no effect on these counts.

Value

If `want_counts = FALSE`, the vector of estimated ancestral allele frequencies, one per locus. Names are set to the locus names, if present. If `want_counts = TRUE`, a named list containing both the estimated ancestral allele frequencies `p_anc_est` and the allele counts matrix, with loci along the rows (also with names if present), and alleles along the columns.

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)
```

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 environment group effects. Values can be numeric or strings, simply assigning the same values to individuals in the same group. If vector (single environment), length must be number of individuals. If matrix (multiple environments), individuals must be along rows, and environments along columns. The environments are not required to be nested. If this is non-NULL, then <code>labs_sigma_sq</code> must also be given!
labs_sigma_sq	Optional vector of environment effect variance proportions, one value for each environment given in <code>labs</code> (a scalar if <code>labs</code> is a vector, otherwise its length should be the number of columns of <code>labs</code>). Ignored unless <code>labs</code> is also given. As these are variance proportions, each value must be non-negative and $\text{sum}(\text{labs_sigma_sq}) + \text{herit} \leq 1$ is required so residual variance is non-negative.

Value

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

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	<i>Per-locus heritability contribution from allele frequency and causal coefficient</i>
------------	---

Description

Calculates the vector of per-locus heritability values, with each causal locus i calculated as $h_i^2 = 2 * p_i * (1 - p_i) * \beta_i^2 / \sigma^2$, where p_i is the ancestral allele frequency, β_i is the causal coefficient, and σ^2 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 σ^2 .

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 σ^2 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 )
```

inv_var_est_bayesian	<i>Calculate expectations of inverse variance terms over the posterior distribution of the ancestral allele frequency</i>
----------------------	---

Description

This function assumes that each sample allele frequency x in the input has a Beta distribution with mean p and variance $p*(1-p)*kinship$, where p is the true allele frequency and is unknown, while $kinship$ is the mean kinship. The function returns a vector of expectations of $1 / (p * (1 - p))^g$, where $p * (1 - p)$ corresponds to the Bernoulli variance, and g is some desired exponent. Note

Usage

```
inv_var_est_bayesian(p_anc_est, kinship, g)
```

Arguments

p_anc_est	A vector of sample allele frequencies (each referred to as x above)
kinship	The mean kinship coefficient of the data
g	The exponent of the variance term

Value

A vector of expectations of $1 / (p * (1 - p))^g$ over the marginalized, unknown true ancestral allele frequencies p , weighted by how likely they are in the posterior given each input sample allele frequency. The length of the vector is the same as the input `p_anc_est`, and each element in the output was estimated from each corresponding element in the input only.

See Also

[p_anc_est_beta_mle\(\)](#) for another way to get less biased estimates of the ancestral allele frequencies and related terms.

Examples

```
# select a relatively high value for example
kinship <- 0.1

# try a grid of values, including edge cases (0,1)
m_loci <- 1000
p_anc_est <- 0 : (m_loci - 1) / (m_loci - 1)

# calculate the desired estimates!
# here g=0.5 estimates  $\sqrt{1 / \sqrt{p * (1 - p)}}$ 
inv_var_est <- inv_var_est_bayesian( p_anc_est, kinship, 0.5 )
```

p_anc_est_beta_mle	<i>Calculate maximum likelihood estimates of allele frequencies from Beta model</i>
--------------------	---

Description

For each sample allele frequency x in the input, this function calculates the maximum likelihood estimate of the true allele frequency assuming that the sample was drawn from a Beta distribution with mean p and variance g , where $kinship$ is the mean kinship of the sample and is known. Note that, oddly, this function estimates the unknown p from a single data point x . Nevertheless, this procedure results in favorable shrinkage of estimate towards 0.5.

Usage

```
p_anc_est_beta_mle(p_anc_est, kinship)
```

Arguments

p_anc_est	A vector of sample allele frequencies (each referred to as x above)
kinship	The mean kinship coefficient of the data

Value

A vector of maximum likelihood estimates of allele frequencies. The length of the vector is the same as the input `p_anc_est`, and each element in the output was estimated from each corresponding element in the input only.

See Also

[inv_var_est_bayesian\(\)](#) for another way to get unbiased estimates of a specific class of inverse variance terms of interest in this project.

Examples

```
# select a relatively high value for example
kinship <- 0.1

# try a grid of values, including edge cases (0,1)
m_loci <- 1000
p_anc_est <- 0 : (m_loci - 1) / (m_loci - 1)

# calculate the desired estimates!
p_anc_mle <- p_anc_est_beta_mle( p_anc_est, kinship )

# notice values tend to be shrunk towards 0.5,
# except values near the edges of the range are shrunk less
plot( p_anc_est, p_anc_mle )
abline(0,1)
```

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 <code>sim_trait</code> 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 <code>PRROC::pr.curve()</code> 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_gc

Apply genomic control transformation to a list of p-values

Description

Genomic control (GC) is a procedure designed to correct the calibration of p-values calculated from test statistics that are supposed to have a Chi-squared distribution but are misspecified and have a different median than desired, in which case the test statistics are divided by the inflation factor (the ratio between the observed and desired medians). This function applies the correction using p-values only, calculating the underlying test statistics assuming they are Chi-squared distributed with known degrees of freedom. The function guarantees that the corrected p-values have a median of 0.5, as desired if they were calibrated, but otherwise there is no guarantee that these corrected p-values will have a reasonable distribution.

Usage

```
pval_gc(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

A list with the following named elements:

- pvals: the vector of GC-corrected p-values
- lambda: the inflation factor of the input p-values, which was used to calculate the corrected p-values

See Also

`pval_infl()`, which is used internally to calculate the inflation factor.

Examples

```
# a simulated set of highly inflated p-values, skewed toward zero
pvals <- rbeta( 100, 1, 10 )
hist( pvals )

# calculate the GC-corrected p-values
obj <- pval_gc( pvals )
pvals2 <- obj$pvals
# and get the inflation factor of the original data:
obj$lambda

# note GC-corrected p-values are often not uniform:
hist( pvals2 )
# but they have a median of 0.5
median( pvals2 )
```

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.

[pval_gc\(\)](#) to perform genomic control correction of p-values.

Examples

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

pval_power_calib	<i>Estimate calibrated power</i>
------------------	----------------------------------

Description

Given a significance level α 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 α is achieved, then it uses this p-value threshold (not α) 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 <code>sim_trait</code> 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	<i>Estimate type I error rate</i>
-----------------	-----------------------------------

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 <code>sim_trait</code> 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	<i>Root mean square deviation</i>
------	-----------------------------------

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

<i>x</i>	The first vector to compare (required).
<i>y</i>	The second vector to compare (required). Lengths of <i>x</i> and <i>y</i> must be equal.
<i>na.rm</i>	If TRUE (default), NA values are removed before calculating the mean square difference. If FALSE, any missing values in either <i>x</i> or <i>y</i> 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 )
```

sim_trait	<i>Simulate a complex trait from genotypes</i>
-----------	--

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 environment 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,
  mac_cut = NA,
  loci_on_cols = FALSE,
  m_chunk_max = 1000,
  fes = FALSE,
  fes_kinship_method = c("original", "mle", "bayesian"),
  causal_indexes = NULL,
  old_center_scale = FALSE,
  old_sample_var = FALSE
)

```

Arguments

<code>X</code>	The m-by-n genotype matrix (if <code>loci_on_cols = FALSE</code> , transposed otherwise), or a <code>BEDMatrix</code> object. This is a numeric matrix consisting of reference allele counts (in <code>c(0, 1, 2, NA)</code> for a diploid organism).
<code>m_causal</code>	The desired number of causal loci. Ignored if <code>causal_indexes</code> is provided.
<code>herit</code>	The desired heritability (proportion of trait variance due to genetics).
<code>p_anc</code>	The length-m vector of true ancestral allele frequencies. Optional but recommended for simulations. Either this or <code>kinship</code> must be specified.
<code>kinship</code>	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 <code>p_anc</code> must be specified.
<code>mu</code>	The desired parametric mean value of the trait (scalar, default 0).
<code>sigma_sq</code>	The desired parametric variance factor of the trait (scalar, default 1). Corresponds to the variance of an outbred individual.
<code>labs</code>	Optional labels assigning individuals to groups, to simulate environment group effects. Values can be numeric or strings, simply assigning the same values to individuals in the same group. If vector (single environment), length must be number of individuals. If matrix (multiple environments), individuals must be along rows, and environments along columns. The environments are not required to be nested. If this is non-NULL, then <code>labs_sigma_sq</code> must also be given!
<code>labs_sigma_sq</code>	Optional vector of environment effect variance proportions, one value for each environment given in <code>labs</code> (a scalar if <code>labs</code> 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 $\text{sum}(\text{labs_sigma_sq}) + \text{herit} \leq 1$ 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 <i>sample</i> allele frequencies and not their true parametric values (p_{anc}), even if these are available. Ignored if causal_indexes is provided.
mac_cut	The optional minimum allele count threshold (default NA, no threshold). If both this and maf_cut are provided, both thresholds are applied (they are not redundant under missingness). Ignored if causal_indexes is provided.
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 $p_{\text{anc}} * (1 - p_{\text{anc}})$ (estimated when p_{anc} is unavailable), which ensures <i>fixed effect sizes</i> (FES) per causal locus. Signs (+/-) are drawn randomly with equal probability. If FALSE (the default), <i>random coefficients</i> (RC) are drawn from a standard Normal distribution. In both cases coefficients are rescaled to result in the desired heritability.
fes_kinship_method	String specifying the bias correction method for the case when fes = TRUE and kinship is provided while p_{anc} is absent (option is ignored otherwise). original corresponds to a simple formula with substantial bias. mle replaces sample allele frequencies with maximum likelihood estimates assuming a Beta distribution with a known variance scale given by kinship (see p_anc_est_beta_mle()). bayesian calculates the expectation over the posterior of a desired inverse variance term, assuming the same Beta distribution as the mle option (see inv_var_est_bayesian()).
causal_indexes	If provided, will use the loci at these indexes as causal loci. When NULL (default), causal indexes are drawn randomly. Thus, parameters m_loci and maf_cut are ignored and have no effect if this parameter is set.
old_center_scale	If TRUE (default FALSE), simulates traits as its more frequently done in the literature, namely genotypes are centered and scaled ("standardized") and coefficients are drawn from a normal distribution with mean zero and variance equal to the heritability times sigma_sq divided by the number of causal loci. True ancestral allele frequencies p_{anc} are used if available, otherwise sample estimates are used. This mode is for comparisons only, it is not recommended as it can result in misspecified heritabilities, particularly when true ancestral allele frequencies are not available.
old_sample_var	If TRUE (default FALSE), old_center_scale = TRUE is required, in which case sample variance estimates are used to scale the genotypes. Otherwise, if old_sample_var = FALSE and old_center_scale = TRUE, genotype variance is calculated using the genetics-motivated formula $2 * p_{\text{anc}} * (1 - p_{\text{anc}})$ or the sample frequency plug-in estimate if p_{anc} is not available.

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.
- `group_effects`: length-`n` vector of simulated environment group effects, or 0 (scalar) if not simulated.
- `causal_indexes`: length-`m_causal` vector of causal locus indexes.
- `causal_coeffs`: length-`m_causal` vector of coefficients at the causal loci. They are in the scale of the given genotypes.
- `alpha`: scalar parameter used to shift genotype effect (taking away genotype mean effects and adding desired mean).
- `herit`, `sigma_sq`, `labs_sigma_sq`: same values specified in input, stored to be able to simulate the same trait from new individuals later if desired.
- `sigma_sq_residual`: residual variance proportion, equal to $1 - \text{herit} - \text{sum}(\text{labs_sigma_sq})$.

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

See Also

`sim_trait_model()` for simulating the same trait (with the same causal variants and other parameters) for new individuals.

`cov_trait()` for constructing the covariance matrix matching these simulations.

`sim_trait_mvn()` for simulating idealized infinitesimal traits with no causal variants.

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_model

Simulate a complex trait from a fixed trait model

Description

This function requires a trait model, which contains values for all of its variance components and intercept coefficient, as well as pre-determined causal loci indexes along with their coefficients, and simply simulates the traits with genetic effects and random non-genetic effects, for the individuals provided. This is useful to simulate traits for new individuals from a pre-existing model that was constructed using some previous data, so that the trait is fundamentally the same in all individuals (rather than a new random trait as `sim_trait` does for each call).

Usage

```

sim_trait_model(
  model,
  X,
  labs = NULL,
  loci_on_cols = FALSE,
  X_subset = FALSE,
  skip_checks = FALSE
)

```

Arguments

model	A list returned by a previous <code>sim_trait</code> call. This list must have scalar values with names <code>sigma_sq</code> , <code>herit</code> , <code>sigma_sq_residual</code> , <code>alpha</code> ; if <code>herit > 0</code> , then <code>causal_indexes</code> and <code>causal_coeffs</code> vectors are also required in this list. Lastly, either <code>labs</code> and <code>model\$labs_sigma_sq</code> must be both missing, or both must be present.
X	The m-by-n genotype matrix (if <code>loci_on_cols = FALSE</code> , transposed otherwise), or a <code>BEDMatrix</code> object. This is a numeric matrix consisting of reference allele counts (in <code>c(0, 1, 2, NA)</code> for a diploid organism).
labs	Optional labels assigning individuals to groups, to simulate environment group effects. Values can be numeric or strings, simply assigning the same values to individuals in the same group. If vector (single environment), length must be number of individuals. If matrix (multiple environments), individuals must be along rows, and environments along columns. The environments are not required to be nested. If this is non-NULL, then <code>labs_sigma_sq</code> must also be given!
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 <code>BEDMatrix</code> object, loci are always on the columns (<code>loci_on_cols</code> is ignored).
X_subset	If TRUE, X is treated as if it were already subset to the desired causal loci. Otherwise (default) X will be subset internally using <code>model\$causal_indexes</code> .
skip_checks	If TRUE, skips validations of input model and other parameters. Not recommended for regular users; this is provided for <code>sim_trait()</code> internal use only, which calls <code>sim_trait_model()</code> , since these checks are redundant with <code>sim_trait()</code> 's own validations.

Value

The same type of named list returned by `sim_trait()`, please see that for more details. This includes the desired trait values, as well as copies of the input model parameters.

See Also

`sim_trait()` for simulating traits from scratch, picking random causal variants and their coefficients from various models.

Examples

```
# make two genotype matrices, one to construct the initial trait,
# the second one is a new dataset from which we want to simulate the same trait.
# They should have the same loci (number of rows), but individuals should be different!
X <- matrix(
  data = c(
    0, 1, 2,
    1, 2, 1,
    0, 0, 1
  ),
  nrow = 3,
```

```

        byrow = TRUE
    )
    X2 <- matrix(
        data = c(
            2, 1,
            2, 0,
            1, 1
        ),
        nrow = 3,
        byrow = TRUE
    )

    # here's a minimal `sim_trait` call that simulates a trait of interest from the first dataset
    model <- sim_trait( X = X, m_causal = 2, herit = 0.7, kinship = 0.2 )

    # use the same model to simulate traits for the new individuals!
    obj <- sim_trait_model( model, X2 )
    # trait vector for new individuals:
    obj$trait

```

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 environment group effects. Values can be numeric or strings, simply assigning the same values to individuals in the same group. If vector (single environment), length must be number of individuals. If matrix (multiple environments), individuals must be along rows, and environments along columns. The environments are not required to be nested. If this is non-NULL, then labs_sigma_sq must also be given!
labs_sigma_sq	Optional vector of environment effect variance proportions, one value for each environment 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 $\text{sum}(\text{labs_sigma_sq}) + \text{herit} \leq 1$ 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 $-\text{tol}$ 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 <code>cov_trait()</code>) 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
```

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](#))

Authors:

- Grace E. Rhodes <grace.e.rhodes@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 )
```

Index

allele_freqs, [2](#)

colMeans(), [2](#)

cov_trait, [3](#)

cov_trait(), [5](#), [18](#), [22](#), [23](#)

herit_loci, [5](#)

inv_var_est_bayesian, [6](#)

inv_var_est_bayesian(), [7](#), [17](#)

mean(), [15](#)

p_anc_est_beta_mle, [7](#)

p_anc_est_beta_mle(), [6](#), [17](#)

popkin::popkin(), [23](#)

PRROC::pr.curve(), [8](#), [9](#)

pval_aucpr, [8](#)

pval_aucpr(), [12](#), [23](#)

pval_gc, [9](#)

pval_gc(), [11](#)

pval_infl, [10](#)

pval_infl(), [10](#), [13](#), [14](#), [23](#)

pval_power_calib, [11](#)

pval_power_calib(), [9](#)

pval_srmsd, [12](#)

pval_srmsd(), [10](#), [11](#), [14](#), [23](#)

pval_type_1_err, [14](#)

pval_type_1_err(), [11](#), [13](#)

rmsd, [15](#)

rmsd(), [13](#)

rowMeans(), [2](#)

sim_trait, [15](#)

sim_trait(), [4](#), [5](#), [20](#), [22](#), [23](#)

sim_trait_model, [19](#)

sim_trait_model(), [18](#), [20](#)

sim_trait_mvn, [21](#)

sim_trait_mvn(), [4](#), [18](#), [23](#)

simtrait, [23](#)

simtrait-package (simtrait), [23](#)

sort(), [13](#)