

Package: simplePHENOTYPES (via r-universe)

October 9, 2026

Date 2026-09-29

Type Package

Version 2.0.0.9003

Title Simulation of Pleiotropic, Linked and Epistatic Phenotypes

Description The number of studies involving correlated traits and the availability of tools to handle this type of data has increased considerably in the last decade. With such a demand, we need tools for testing hypotheses related to single and multi-trait (correlated) phenotypes based on many genetic settings. Thus, we implemented various options for simulation of pleiotropy and Linkage Disequilibrium under additive, dominance and epistatic models. The simulation currently takes a marker data set as an input and then uses it for simulating multiple traits as described in Fernandes and Lipka (2020)
<[doi:10.1186/s12859-020-03804-y](https://doi.org/10.1186/s12859-020-03804-y)>.

License MIT + file LICENSE

Copyright The package authors, except the Rust crates vendored in src/rust/vendor.tar.xz; see file inst/COPYRIGHTS for their authors and licences.

Encoding UTF-8

LazyData true

Depends R (>= 4.2)

Imports data.table, mvtnorm, stats, utils, rlang

Suggests SNPRelate, gdsfmt, jsonlite, knitr, pkgload, rmarkdown, stringi, testthat (>= 3.0.0), withr

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

URL <https://github.com/samuelbfernandes/simplePHENOTYPES>

BugReports <https://github.com/samuelbfernandes/simplePHENOTYPES/issues>

NeedsCompilation yes
Config/rextendr/version 0.5.0
Config/testthat/edition 3
SystemRequirements Cargo (Rust's package manager), rustc >= 1.71.0, xz
Config/roxygen2/version 8.1.0
Config/pak/sysreqs xz-utils libclang-dev
Repository <https://samuelbfernandes.r-universe.dev>
Date/Publication 2026-10-09 06:30:42 UTC
RemoteUrl <https://github.com/samuelbfernandes/simplephenotypes>
RemoteRef HEAD
RemoteSha 3c000630a5e566640b5eadfd429d0fcbfd5349b9

Contents

[.Population	3
a_matrix	4
additive	5
additive_value	9
as_numeric	11
as_population	14
breed_composition	16
bulk	17
c.Population	18
coheritability	19
combining_ability	20
complex_phenotypes	23
cor_ar1	24
create_phenotypes	25
cross	35
cross_usefulness	38
crossbreed	40
dominance	41
dosages	44
double_haploid	45
epistasis	46
families	48
filter_geno	49
founders_coalescent	53
g_matrix	55
genetic_values	56
genotypic_value	57
gxe_value	59
haplotypes	60
heterosis	61

liability_threshold	62
mabc_select	64
marker_select	67
mate	69
mating_design	70
mediation_split	72
n_individuals	73
observe_counts	74
optimum_contribution	75
parentage	77
pedigree	78
phenotype_value	81
phenotypes_long	84
phenotypes_wide	85
plot.phenotype_sim	85
population_from_haplotypes	86
population_trait	88
predict.transcriptome_sim	90
predict_ebv	91
prediction_accuracy	93
progeny_test	94
qtn_table	97
recurrent_parent_recovery	98
recurrent_selection	99
sample_parents	101
select_ind	102
selection_methods	108
selfcross	109
simulate_phenotype	110
simulate_transcriptome	117
single_seed_descent	121
synthetic_map	122
template_effects	124
transcriptome	125
vqtl	127
write_phenotypes	129
write_qtn_table	132

Index**135**

[.Population

*Subset the individuals of a Population***Description**

Subset the individuals of a Population

Usage

```
## S3 method for class 'Population'
x[i]
```

Arguments

x a Population.

i individuals to keep, by name, position or logical mask. A repeated subscript selects the same individual again: the copies get unique ids ("P1", "P1_1", ...) and share one pedigree key (they are the same individual). An empty subscript gives an empty Population.

Value

A Population with the selected individuals.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04)
pop[1:2]
```

a_matrix

Numerator relationship matrix from the recorded pedigree

Description

The additive relationship matrix A of a Population's individuals, built from its recorded pedigree (see [parentage\(\)](#)) by the tabular method (Emik & Terrill 1949) on coefficients of relationship (Wright 1922): processing individuals parents-first, $A_{ij} = (A_{i,m(j)} + A_{i,f(j)})/2$ and $A_{jj} = 1 + F_j$ with $F_j = A_{m(j)f(j)}/2$. A self has both parents equal, so $F = (1 + F_P)/2$; a doubled haploid is fully inbred, $A_{jj} = 2$, and relates to others as its parent does. Ancestors not in pop still enter through the pedigree.

Usage

```
a_matrix(pop, ids = NULL, founder_f = 0)
```

Arguments

pop a Population.

ids optional character ids (of pop's individuals) to return; default all.

founder_f inbreeding coefficient of the founders, in $[0, 1]$: one value for every founder (default 0, non-inbred), or a numeric vector named by founder (founders not named keep 0). A name is a pedigree key (the key column of [parentage\(pop, ancestors = TRUE\)](#), unique by construction) or a founder's display id. Display ids may repeat across founder pools, so an id shared by several founders is ambiguous and is an error naming the pedigree keys to use instead; names must be distinct and identify pedigree founders. 1 is a fully inbred line.

Details

Founders are the base of the pedigree: unrelated to each other, and by default **non-inbred** (`founder_f` = 0, $A_{jj} = 1$), which is right for outbred (random-mating) founders. Inbred founders – e.g. the bundled maize inbred lines – are not: their inbreeding coefficient enters the base as $A_{jj} = 1 + F_j$ and their relationships with descendants, so pass `founder_f` = 1 for fully inbred lines (a founder P crossed to another inbred line then has $A_{P,F1} = 1$ and the two founders' $F1$ sibs $A = 1$, against 0.5 with the default). Leaving the default on inbred founders halves those relationships and disagrees systematically with the genomic matrix of `g_matrix()`, whose founder diagonal is close to 2 for inbred lines. Downstream individuals follow from the recursion; selfing and doubled-haploid rules are not altered by `founder_f`. Note that a Population in which the same individual appears twice (e.g. `c(pop, pop[1])`) lists that individual under two ids with a relationship equal to its own diagonal; `a_matrix()` does not refuse it.

Value

A symmetric matrix with dimnames ids.

References

Wright S (1922) Coefficients of inbreeding and relationship. *The American Naturalist* 56:330–338. doi:10.1086/279872

Emik LO, Terrill CE (1949) Systematic procedures for calculating inbreeding coefficients. *Journal of Heredity* 40:51–55. doi:10.1093/oxfordjournals.jhered.a105986

See Also

`parentage()`, `predict_ebv()`, `g_matrix()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:2)
f1 <- cross(pop[1], pop[2], n = 2, seed = 1)
s1 <- selfcross(f1[1], n = 2, seed = 2)
a_matrix(c(f1, s1))
# the bundled lines are inbred: declare it (A between a line and its F1 is 1)
a_matrix(c(pop, f1), founder_f = 1)
```

additive

Add an additive variance component

Description

Add an additive variance component

Usage

```
additive(
  sim,
  prop = NULL,
  n_qtn = NULL,
  qtn = NULL,
  effect = NULL,
  phase = c("coupling", "repulsion"),
  dist = "geometric",
  orthogonal = FALSE,
  a = NULL,
  d = NULL
)
```

Arguments

sim	a phenotype_sim.
prop	proportion of total phenotypic variance (scalar or length n_traits). May be omitted when h2 was set in simulate_phenotype() , in which case the layer takes whatever of that budget is unspent – so a single additive() layer gets prop = h2. When prop is given and h2 is set, the layer prop values must sum to h2.
n_qtn	number of additive QTNs; overrides the baseline n_qtn from simulate_phenotype() (with a warning when both are given). Ignored, with a warning, when qtn fixes the loci.
qtn	optional user-supplied QTNs for this layer, so you can fix the causal loci for one effect type while the others are drawn at random. Give marker names (matched against the map) or column indices; a vector is used for every trait, or a length-n_traits list sets each trait's loci separately (every element the same length). n_qtn is then taken from what you supply. Fixed loci are not redrawn by vary_qtn. (For epistasis() , qtn describes interacting sets; see there.) Every architecture accepts qtn, and each keeps its construction: under "independent" the loci carry the effect series; under "pleiotropy" (multi-trait) every locus affects every trait, so give a single vector (or the same loci for each trait in a list); loci that affect only some traits are partial pleiotropy, built with complex_phenotypes() . If a correlation is controlled (cor, pi, ...) the correlated draw sets the effects and the genetic correlation is its target; otherwise the correlation is just an outcome of the shared loci, and an explicit effect sets the effects (without it the default draw does, with implicit cor = 0). pi only matters for controlling a correlation, and pi < 1 cannot be used with fixed shared loci. Under "ld" (ld_type = "direct" only) give qtn = list(trait1_loci, trait2_loci) of equal length – the i-th locus of each is a linked pair on one chromosome, no marker may be causal for both traits, and each pair's r2 is reported (by qtn_table()) and warned about when outside [r2_min, r2_max]. A marker that is monomorphic (or heterozygous in every individual) carries no variance: it is accepted with a warning, except where it makes the architecture undefined (an "ld" pair, or a pleiotropy layout with no varying locus), which is an error.

effect	optional geometric base (scalar) or explicit effect series (length <code>n_qtn</code>), used for every trait; or a length- <code>n_traits</code> list of these, one per trait – e.g. to re-score per-trait effects frozen from an earlier simulation on fixed <code>qtn</code> in a single layer. Under <code>architecture = "pleiotropy"</code> (multi-trait) it is accepted only when no correlation is controlled (no <code>cor</code> , <code>pi</code> , ...): it then sets the effects of the shared loci (the same series for every trait unless a per-trait list is given) and the genetic correlation is an outcome. Without effect (or with <code>cor / pi</code>) the pleiotropy draw sets the effects. Not available with <code>orthogonal = TRUE</code> (use <code>a</code>).
phase	QTN effect-sign pattern, "coupling" (default) or "repulsion". Under "repulsion" the effect signs alternate across the layer's QTNs in draw order (+, -, +, -, ...); coupling leaves the drawn signs untouched. The realized genetic variance is still <code>prop</code> (variance is pinned by the partition, not by phase); what changes is the sign structure of the per-QTN effects. This is a convenient stand-in for a repulsion architecture, not a phase derived from the haplotypes: it alternates by position in the draw, and does not inspect the sign of LD between the loci. It therefore induces genuine repulsion-style cancellation only when consecutively drawn QTNs happen to be in positive LD (same chromosome, coupling-phase alleles); for effectively unlinked QTNs it merely relabels which allele is "increasing" and leaves the cross-locus covariance unchanged. <code>phase</code> is architecture-independent – it acts on whichever QTNs the additive layer draws. For a genuine haplotype-derived phase between two linked traits use <code>architecture = "ld"</code> with its <code>ld_phase</code> argument (see simulate_phenotype()): it reads the sign of the dosage correlation of each linked pair and sets trait 2's effect sign so that every pair's linkage-induced covariance is positive ("coupling") or negative ("repulsion"). The two may be combined: <code>phase</code> is applied first, then <code>ld_phase</code> re-signs trait 2's effects pair by pair, so under "ld" with a non-default <code>ld_phase</code> the positional alternation is only guaranteed to survive on trait 1.
dist	within-layer effect distribution (default "geometric").
orthogonal	use the orthogonal genotypic model instead of the variance-partition coding (default FALSE). When TRUE, give per-locus additive effects <code>a</code> and dominance deviations <code>d</code> ; the additive and dominance components are orthogonal under random mating (see Details).
a	additive effect per locus for <code>orthogonal = TRUE</code> : a scalar geometric base or a length- <code>n_qtn</code> vector (the counterpart of effect). The locus value is $-a / +d / +a$ for gene content 0 / 1 / 2.
d	dominance deviation per locus for <code>orthogonal = TRUE</code> : a single value (same at every locus) or a length- <code>n_qtn</code> vector. $d = 0$ (default) is a purely additive locus. The heterozygote value is <code>d</code> , and the degree of dominance is $d / \text{abs}(a)$ (abs because <code>a</code> may be negative, directly or through <code>phase = "repulsion"</code>): magnitude 0 = additive, 1 = complete dominance, > 1 = overdominance. Dominance is toward the $+a$ homozygote when <code>d</code> and <code>a</code> share a sign and toward the $-a$ homozygote when they differ – it is the ratio d/a , not the sign of <code>d</code> alone, that sets the direction.

Details

The additive value is the $-1/0/1$ dosage weighted by the QTN effects, centered, and scaled to `prop` of the phenotypic variance. This is a simulation convention, not Fisher's average-effect decomposition;

for an additive-only model prop equals the narrow-sense h^2 under Hardy-Weinberg.

Combining this layer with `dominance()` on the same loci (the default) makes the realized genetic variance differ from $\text{prop_A} + \text{prop_D}$ by $2\text{Cov}(c_A, c_D)$ (with one additive and one dominance layer; several layers of one type add their mutual covariance to $\text{Var}(c_A)$ or $\text{Var}(c_D)$), a structural term that depends on the allele frequencies and on which allele is coded +1 (see `simulate_phenotype()`); the object reports the realized $\text{Var}(A)$, $\text{Var}(D)$ and $2\text{Cov}(A,D)$, and $\text{Var}(c_A)$, $\text{Var}(c_D)$ and $2\text{Cov}(c_A,c_D)$, in `$ad_report` and in `print()`. Use `orthogonal = TRUE` for a Fisher-orthogonal additive/dominance model.

Value

the updated `phenotype_sim`.

Orthogonal genotypic model (`orthogonal = TRUE`)

Instead of the dosage coding above, the layer builds each locus's genotypic value from an additive effect a and a dominance deviation d (value $-a / +d / +a$ for gene content $0 / 1 / 2$), and the whole genotypic value is scaled to prop of the phenotypic variance. Its additive and dominance **variances then emerge** from a , d and the allele frequencies rather than from separate layer proportions: the additive (breeding-value) component uses the average effect of substitution $\alpha_j = a_j + d_j(1 - 2p_j)$, and the dominance deviation is the realized residual $D = g - A$. The additive and dominance components are orthogonal ($\text{Cov}(A, D) = 0$) in expectation under **random mating**, which puts each locus in Hardy-Weinberg proportions; α above is the one-generation transmitting-ability form. This does *not* require linkage equilibrium – between-locus LD is compatible with orthogonality in this no-epistasis model – but per-locus HWE alone is not sufficient under arbitrary nonrandom multilocus genotype association (e.g. selection or population structure), which can correlate one locus's gene content with another's heterozygosity. A finite or structured sample therefore carries a (usually small) $\text{Cov}(A, D)$ that breeders customarily ignore; this implementation does not, and the variance budget reports the **realized** shares $\text{Var}(A)/\text{Var}(g)$ and $\text{Var}(D)/\text{Var}(g)$ on separate additive and dominance rows, plus an `add_dom_cov` row for $2\text{Cov}(A, D)/\text{Var}(g)$, so the three sum to the layer prop (the `add_dom_cov` row is ≈ 0 for a large random-mating or F2 sample and non-zero otherwise). The realized narrow-sense heritability follows from the degree of dominance $d / \text{abs}(a)$ (magnitude 1 = complete dominance, > 1 = overdominance; abs because a may be negative – which the variance-partition grammar cannot express, because per-component scaling washes a global degree out), and the breeding value used by `select_ind()` / `optimum_contribution()` picks up the dominance-induced average effect automatically. Because the effects are fixed, this mode does not draw a separate `dominance()` layer, is incompatible with `vary_qtn`, and cannot be used with the correlation-controlling "pleiotropy" (multi-trait) or "ld" architectures (their effects come from the correlated draw; `qtn = alone` is accepted).

References

Fisher RA (1918) The correlation between relatives on the supposition of Mendelian inheritance. *Trans. R. Soc. Edinb.* 52:399-433; Falconer DS (1985) A note on Fisher's 'average effect' and 'average excess'. *Genet. Res.* 46(3):337-347 (the average effect of substitution $\alpha = a + d(q - p)$ and its dependence on random-mating / Hardy-Weinberg proportions); Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow; Lynch M, Walsh B (1998) *Genetics and Analysis of Quantitative Traits*. Sinauer, Sunderland, MA (additive / dominance decomposition).

See Also

`dominance()`, `epistasis()`, `vqtl()` for the other layers.

Examples

```
data("SNP55K_maize282_maf04")

# Additive trait: 3 QTNs explaining half the phenotypic variance,
# so h2 = 0.5 and the remaining 0.5 is residual.
ph <- simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3)
ph

head(phenotypes_long(ph))

# A major QTN with a set share of the variance: stack two additive layers,
# one locus explaining 20% of V_P and 50 equal-effect loci another 30%
g <- SNP55K_maize282_maf04
major <- simulate_phenotype(g, seed = 1) |>
  additive(prop = 0.2, qtn = g$snp[100]) |>
  additive(prop = 0.3, n_qtn = 50, effect = rep(1, 50))
tb <- qtn_table(major)
head(tb[order(-tb$var_explained), c("snp", "var_explained")], 3)

# Effect sizes follow a geometric series by default. `effect` sets its base,
# so QTN effects here are 0.5, 0.25, 0.125, ...
simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3, effect = 0.5)

# Or give the effects explicitly, one per QTN.
simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3, effect = c(0.6, 0.3, 0.1))

# Fix the additive QTNs yourself by marker name.
simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.5, qtn = c("ss196442916", "ss196439337", "ss196480535"))

# Orthogonal genotypic model: give per-locus a and d; the additive and
# dominance variances emerge (see the var_budget). Dominance needs
# heterozygotes, so simulate on a segregating F2.
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
f2 <- selfcross(cross(pop[1], pop[2], n = 1, seed = 1), n = 40, seed = 2)
og <- simulate_phenotype(f2, h2 = 0.6, seed = 3) |>
  additive(orthogonal = TRUE, a = 0.5, d = 0.5, n_qtn = 8)
og$var_budget
```

Description

Scores each individual by $\sum_j \text{dosage}_{ij} \text{effect}_j$ over a **given, frozen architecture** (loci qtn and their effects), using the -1/0/1 dosages directly with **no per-population centring or rescaling**.

Usage

```
additive_value(x, qtn, effect)
```

Arguments

x	a Population, a Population-backed phenotype_sim, or a marker x individual dosage matrix coded -1/0/1 (marker names as row names when qtn is given by name; individual ids as column names).
qtn	the causal loci, as marker names (matched against the map) or integer marker (row) indices.
effect	a finite numeric vector of per-locus additive effects, one per entry of qtn and in the same order.

Details

This is the accessor to use when comparing populations *across generations* – e.g. to show a selection response as the mean additive value climbs. It differs from [genetic_values\(\)](#), which re-centres and re-scales every layer to its target prop on whatever population is being scored: that makes [genetic_values\(\)](#) read roughly mean 0 / variance prop at *every* generation, so it cannot express a cross-generational trend. Here the loci and effects are fixed by the caller, so the scale is stable and the numbers are comparable from one generation to the next. Effects are on the same -1/0/1 dosage scale as an [additive\(\)](#) layer's effect (and as [qtn_table\(\)](#)'s effect column).

Value

a named numeric vector of additive values, one per individual.

See Also

[genetic_values\(\)](#), [dosages\(\)](#), [qtn_table\(\)](#), [select_ind\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
# Score on a fixed 3-locus architecture; the scale does not depend on the set.
av <- additive_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2))
head(av)
```

as_numeric

*Convert genotypes to numeric coding***Description**

Converts character SNP genotypes to numeric coding, the format every simulation function in the package expects. HapMap, VCF, GDS and PLINK bed/ped input are recognized automatically, from a file path or from an object already in memory.

Usage

```
as_numeric(x, ...)
```

Arguments

x genotype data: a file path, or an object in memory (for example a HapMap-style data frame).

... further options controlling the conversion:

- to_r** return the result as an R object (default TRUE for in-memory input, FALSE for file input).
- to_file** write the result to a file; **file_name** sets the path.
- code_as** "-101" (major = 1, het = 0, minor = -1; default) or "012" (major = 2, het = 1, minor = 0).
- method** allele orientation: "frequency" (default) or "reference", the latter requiring **ref_allele**.
- ref_allele** reference allele per marker, for method = "reference".
- impute** missing-data handling: "None" (default), "Middle", "Minor" or "Major".
- model** "Add" (default), "Dom", "Left" or "Right".
- from** input format, when automatic detection is not wanted.
- verbose** print progress messages.
- counted_column** FALSE (default) or TRUE: add the counted column (the allele coded +1 at each marker) after cm; see "Which allele is counted".

Details

When **to_file** = TRUE and no **file_name** is given, the default output name is <label>_numeric.txt, where the label is the name of the object passed as **x** (for a file input, the input file's name without its extension). A label that is not already a portable file name (other characters replaced, or longer than 100 characters) is sanitized and gets a short hash of the original label appended; this makes accidental clashes unlikely but does not make names unique. Different inputs can still map to the same default name (labels that differ only in case on a case-insensitive file system, files such as same.hmp.txt and same.txt in one directory, or two inline objects of the same class and size, whose generic label is built from class and dimensions). A default-named file that already exists is overwritten, with a warning naming the file; an explicit **file_name** is used as given, without a warning. Pass **file_name** to choose the output path.

Value

The genotypes in numeric format: five metadata columns (snp, allele, chr, pos, cm), plus the optional counted column when `counted_column = TRUE`, followed by one column per individual. Returned as a data frame when `to_r = TRUE`, otherwise written to file.

Which allele becomes 1

By default (`method = "frequency"`) the **most frequent allele is the reference**: it is coded 1, the minor allele -1, and the heterozygote 0. The choice is made per marker from the observed genotype calls, so it does not depend on how the input file ordered its alleles – with one exception, the tie rule below.

Tie rule (minor-allele frequency exactly 0.5). When the two homozygote classes are equally frequent at a marker (for example 50 of 100 doubled-haploid lines carry each allele), no allele is more frequent, and the coding follows the allele order instead: the first-listed allele ("allele 1") is coded as the reference (1, or 2 under `code_as = "012"`), and the second-listed as the minor allele (this is the strict $>$ comparison in the numericalization kernel: the alleles are swapped only when the second allele is strictly more frequent). "First-listed" means the first allele of the input's allele pair: REF for VCF, the first allele of the alleles column for HapMap, A1 (as SNPRelate lists it) for PLINK bed/ped, and the first allele of `snp.allele` in a GDS file. For inputs that declare no allele pair (nucleotide tables and Illumina FinalReport) the alphabetically first allele is allele 1. Every reader applies the same rule, so the same VCF gives identical dosage columns from a file path and from a data frame, tied markers included.

The alleles column of HapMap input is checked, not trusted. If the calls at a marker contain alleles that are not in its declared pair (a stale or opposite-strand alleles column), a warning names the affected rows and the marker is oriented from the calls; the returned allele column then shows the alleles actually observed.

The returned allele column lists the two alleles as "allele1/allele2" in the order above (for a marker whose second allele was never observed, only allele 1). The column itself records the raw orientation, not which allele was coded 1.

Which allele is counted. The allele coded 1 (or 2 under `code_as = "012"`) at each marker is recorded in the "counted_allele" attribute of the returned data frame, a character vector with one allele per marker (NA where it is not known, and absent under `model = "Dom"`, which counts no allele). No dosage value depends on it: it only lets `cross()`, `c.Population()` and the selection schemes see whether two separately converted panels count the same allele. Two panels that both carry it and count different alleles at a marker are refused when crossed or pooled (e.g. an all-AA panel and an all-GG panel would otherwise both encode +1 and cross to an F1 of +1 instead of the heterozygote 0). The attribute is a convenience of the R object: a numeric text file cannot carry it, and `[]` keeps it unchanged when it subsets or reorders the rows of the data frame (so it no longer lines up with them). The durable form is the optional counted column: with `counted_column = TRUE` the result, and the file written, gain a character column counted immediately after `cm` (before the individuals), holding the same allele per marker (NA where it is not known). It travels with its marker through files and row subsetting, `as_population()`, `filter_geno()` and `as_numeric()` itself accept it, and when present it is the record used, so two panels read back from files, or subsetted, are still checked against each other. The default (`counted_column = FALSE`) writes exactly the five metadata columns and changes no value. `counted_column = TRUE` is an error under `model = "Dom"` (no allele is counted) and for numeric input that carries neither the column nor the attribute; numeric input that already has the column keeps it. A numeric data frame without the column (or

with a numeric sixth column, which is an individual) has no record and gets the older label-only check: convert panels that will be crossed **jointly**, or with the same `ref_allele`, to be safe.

Calls are matched case-insensitively. A marker with more than two alleles is set to missing with a warning that counts the affected markers; missing marker IDs in VCF / PLINK input are filled with `chr:pos`, and duplicated marker IDs or sample names are an error (downstream functions index by name).

VCF calls. Only complete, biallelic diploid genotype calls are used, from a file path and from an in-memory VCF table alike: haploid calls ("`0`", "`1`"), partially missing calls ("`./1`") and calls that name a second ALT allele ("`0/2`", "`2/2`") are set to missing, and one warning per kind counts them. The marker's orientation is then chosen from the remaining calls.

Set `method = "reference"` with `ref_allele` to fix the reference allele yourself instead — for example to keep the coding consistent with an external panel, where the major allele in your sample may not be the one you want coded 1.

`code_as = "012"` switches to counting the reference allele (`2/1/0`) rather than centering on the heterozygote.

The output, and the `cm` column

The result has five metadata columns — `snp`, `allele`, `chr`, `pos`, `cm` — followed by one column per individual (with `counted_column = TRUE`, a sixth metadata column `counted` follows `cm`; see "Which allele is counted"). The types are the same for every input format: `snp`, `allele` and `chr` are character, `pos` is integer (double only if a position is not a whole number below 2^{31} , and integer NA when the input carries no positions), `cm` is double, and the genotype columns are integer. Numeric-format input (a data frame or a file written by `as_numeric()`) is brought to this schema too (logical and whole-number double genotype columns become integer, values unchanged), so converting a written file again returns the same table.

`cm` is the genetic map in centiMorgans, and it is NA unless the input format carries one. HapMap and VCF have no genetic-distance field, so `cm` is NA for them; PLINK `.bim` / `.map` files do (column 3), and it is kept when populated. This matters because meiosis needs recombination distances: [as_population\(\)](#) and the crossing functions refuse to run on an all-NA map. Fill one in from the physical positions when you have no real map:

```
num <- as_numeric("my_genotypes.hmp.txt", to_r = TRUE)
num$cm <- synthetic_map(num$chr, num$pos)
```

Note also that `aggregate(cm ~ chr, ...)` returns "no rows to aggregate" on an all-NA map, since the formula interface drops missing values.

See Also

[synthetic_map\(\)](#) to supply a genetic map, [as_population\(\)](#) and [cross\(\)](#) for what that map is needed for.

Examples

```
# A small HapMap table: 11 metadata columns, then one column per individual.
hmp <- data.frame(
  `rs#` = c("snp1", "snp2", "snp3"),
```

```

alleles = c("C/T", "A/G", "G/T"),
chrom   = c(1, 1, 2),
pos     = c(100, 500, 200),
strand  = "+", `assembly#` = NA, center = NA,
protLSID = NA, assayLSID = NA, panelLSID = NA, QCcode = NA,
L1 = c("C", "A", "G"),
L2 = c("T", "G", "T"),
L3 = c("C", "A", "T"),
L4 = c("T", "G", "G"),
check.names = FALSE, stringsAsFactors = FALSE
)

# to_r = TRUE returns the result instead of writing it to disk.
num <- as_numeric(hmp, to_r = TRUE, verbose = FALSE)
num

# 0 / 1 / 2 coding instead of -1 / 0 / 1:
as_numeric(hmp, to_r = TRUE, code_as = "012", verbose = FALSE)

# Orient the coding against a reference allele rather than by frequency:
as_numeric(hmp, to_r = TRUE, verbose = FALSE,
           method = "reference", ref_allele = c("C", "A", "G"))

# From a file, the format is detected from its contents:
## Not run:
num <- as_numeric("my_genotypes.hmp.txt", to_r = TRUE)
num <- as_numeric("my_genotypes.vcf", to_r = TRUE)

## End(Not run)

```

as_population

Create a population of phased individuals from genotype data

Description

Meiosis needs to know which allele sits on which of an individual's two homologous chromosomes, information that a -1/0/1 dosage matrix does not carry. `as_population()` builds that phased representation so the result can be passed to `cross()`, `selfcross()` and `double_haploid()`.

Usage

```
as_population(geno, individuals = NULL, pool = NA_character_)
```

Arguments

geno	a numeric-format data frame whose first five columns are <code>c("snp", "allele", "chr", "pos", "cm")</code> , as returned by <code>as_numeric()</code> , with the remaining columns individuals coded -1/0/1. An optional character column counted right after cm (see <code>as_numeric(counted_column =)</code>) is read as the counted-allele record, not as an individual; it must name one allele per marker (NA = unknown), consistent
------	--

	with the allele label. When the "counted_allele" attribute is also present (same length), it fills the column's unknown entries, and a disagreement between the two records is an error.
individuals	optional character or numeric vector selecting which individuals to keep, in the order given. Defaults to all of them.
pool	optional label for the founder pool these individuals come from (e.g. a breed or heterotic group), recorded in the pedigree so progeny can be traced to it (see parentage()). Default NA. A founder is identified by its pool label, id and haplotypes, so the same genotypes imported twice under the same label (or none) are the same individuals; give each breed or pool its own label when individuals in different pools may share an id and genotype. The empty string and "<unassigned>" (the column breed_composition() uses for founders without a label) are reserved.

Details

A Population is also accepted anywhere [simulate_phenotype\(\)](#) takes geno, so a simulated pedigree can be phenotyped directly.

Value

A Population. Its individuals are recorded as pedigree founders.

Phasing of heterozygotes

Homozygous genotypes (-1 and 1) determine both strands exactly. Heterozygotes do not, and are assigned arbitrarily as allele-1 on the first strand and allele-2 on the second. For inbred panels such as [SNP55K_maize282_maf04](#) this is nearly lossless because heterozygotes are rare, but for an outbred sample the phase of each heterozygote is a guess, and linkage between heterozygous sites in the first generation will not be realistic: every heterozygous site of such a founder sits in perfect coupling on the first strand, so its first-generation gametes carry the maximum coupling-phase linkage disequilibrium. The current public API does not import external haplotype phase, so [as_population\(\)](#) should not be used for multi-generation recombination studies of substantially heterozygous, unphased founders.

Genetic map, units and chromosome order

The cm column must be a genetic map in **centiMorgans** (it is divided by 100 to give Morgans for meiosis). A map whose largest position is at most 5 across 20 or more markers looks like Morgans (or a proportion) and draws a warning: crossovers would be 100 times too rare. cm may start anywhere on a chromosome: the expected number of crossovers on a chromosome (Poisson with interference = "poisson", gamma-spaced by default; see [cross\(\)](#)) equals its **last** map position in Morgans (the isqg convention), which is not its span $\max(\text{cm}) - \min(\text{cm})$ when the first marker is not at 0. Chromosomes are processed, and their random draws are consumed, in a fixed **canonical order that does not depend on the storage type of chr or on the locale**: labels that are numbers first, in numeric order (1, 2, 10), then other labels by their non-numeric prefix in byte order (uppercase before lowercase) and then by their trailing number (chr1, chr2, chr10, chrX). So the same map with chr stored as integers or as text gives the same seeded progeny. (Text labels that

used to sort as "1", "10", "2" are now ordered 1, 2, 10; a seeded run on such a map draws its chromosomes in the new order.)

Allele orientation (crossing populations built from separate files)

The -1/0/1 coding is relative: +1 is the allele that `as_numeric()` considered the reference (by default the most frequent allele *of that data set*). Two populations made from **separately** converted panels can therefore code opposite alleles as +1 at a marker, and crossing them (or pooling them with `c.Population()`) would then silently mix up the alleles. Convert the panels **jointly**, or give each the same reference alleles with `as_numeric(method = "reference", ref_allele = )`. `as_numeric()` records the allele it counted as +1 at every marker (the "counted_allele" attribute of its result, see `as_numeric()`), and `as_population()` keeps it; the optional counted column of `as_numeric(counted_column = TRUE)` (placed after `cm`) carries the same record through text files and row subsetting and, when present, is the record used. When both populations carry that record, `cross()` and `c.Population()` compare it per marker and **stop** if the two panels count different alleles as +1. Where a record is missing on either side (numeric data read back from a text file or subsetting without the counted column, other software), the allele label is compared instead: the check warns when the two panels list a marker's alleles in opposite order (a sign the orientation may differ) and stops when they share no allele. The check only sees what these records show: it cannot detect a difference that neither records.

See Also

`cross()`, `selfcross()`, `double_haploid()`, `synthetic_map()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = c("33-16", "38-11"))
pop
```

breed_composition

Expected breed composition from the pedigree

Description

Each individual's expected fraction of genes from each founder pool (breed, heterotic group), traced through its recorded pedigree: a founder is 1 for its own pool (the pool given to `as_population()`), and a progeny is the mean of its parents (a self or doubled haploid equals its parent). These are **expected** fractions – the realized fraction of an individual's genome from each breed varies around them and is not tracked (a Population carries no per-locus founder origin).

Usage

```
breed_composition(pop)
```

Arguments

pop a Population.

Value

A matrix, individuals by pools, rows summing to 1. Founders imported without a pool label count toward a column "<unassigned>".

See Also

[crossbreed\(\)](#), [heterosis\(\)](#), [parentage\(\)](#)

Examples

```
g <- data.frame(snp = paste0("m", 1:10), allele = "A/G", chr = 1,
               pos = 1:10, cm = seq(0, 90, by = 10), P1 = 1L, P2 = -1L)
A <- as_population(g[, 1:6], pool = "A"); B <- as_population(g[, -6], pool = "B")
f1 <- cross(A, B, n = 2, seed = 1)
bc <- cross(f1[1], A, n = 2, seed = 2)
breed_composition(bc)                # 3/4 A, 1/4 B
```

bulk	<i>Bulk advance</i>
------	---------------------

Description

Advances a population as an undivided bulk: each generation every plant is selfed and the progeny are pooled, then *n* seeds are drawn at random from the pooled progeny to form the next generation (Bernardo 2020). Every plant is assumed to set ample seed, so the draw is uniform over the current plants: the number of advanced seeds each plant contributes is $\text{multinomial}(n, 1/N)$ (mean n / N , variance about n / N), not fixed – including when *n* is a multiple of the current size *N*. Line identity is not tracked.

Usage

```
bulk(x, generations = 5L, n = NULL, seed = NULL, interference = NULL)
```

Arguments

- x* a Population, or a phenotype_sim built on one (its genotypes are used).
- generations* number of selfing generations.
- n* bulk size carried to the next generation (default: keep the current size); it may be smaller or larger than the current size.
- seed* optional RNG seed for the whole scheme: NULL (default; the ambient RNG stream is used) or one non-negative whole number. When given, the caller's RNG state is restored on exit, so a seeded scheme does not disturb the surrounding random stream.

interference crossover interference model of `cross()` (see its section "Crossover interference"): NULL (default: the option `simplePHENOTYPES.interference` if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the `isqg` stream, the default before 2.0.0.9003) or `list(nu = , p = )`. It is applied to every meiosis the scheme runs, in every generation, so a whole scheme can use one meiosis model.

Value

a Population (the final bulk).

References

Bernardo R (2020) *Breeding for Quantitative Traits in Plants*, 3rd ed. Stemma Press, Woodbury, Minnesota.

See Also

`single_seed_descent()`, `pedigree()`, `recurrent_selection()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:10)
f1 <- cross(pop[1], pop[2], n = 30, seed = 1)
bk <- bulk(f1, generations = 4, n = 30, seed = 2)
n_individuals(bk)
```

c.Population

Pool populations that share a genetic map

Description

Column-binds several Populations (e.g. the progeny of many crosses) into one. All inputs must share the same marker map: identical snp names and chromosomes, and pos and cm equal to a numerical tolerance (absolute $1e-8$ for values below 1, relative $1e-8$ above – so round-off noise is accepted, any real map difference is not). The map of the first input is kept. Individual ids are made unique across the pooled set.

Usage

```
## S3 method for class 'Population'
c(...)
```

Arguments

... Population objects with identical maps.

Value

a single Population.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:6)
a <- selfcross(pop[1], n = 3, seed = 1)
b <- selfcross(pop[2], n = 3, seed = 2)
c(a, b)
```

coheritability	<i>Co-heritability between traits</i>
----------------	---------------------------------------

Description

The realized co-heritability matrix of a multi-trait simulation: $\text{Cov}(G_i, G_j) / \sqrt{V_{P_i} V_{P_j}} = r_{G_{ij}} h_i h_j$, the genetic covariance standardized by the phenotypic standard deviations. G are the total genetic values, so this is the **broad-sense** co-heritability and the diagonal the realized broad-sense heritability $\text{Var}(G_i) / \text{Var}(P_i)$. Only for a purely additive model does it equal the narrow-sense co-heritability $r_{A_{ij}} h_i h_j$ that predicts the correlated response to phenotypic selection, $CR_j = i r_{A_{ij}} h_i h_j \sigma_{P_j}$ (Falconer & Mackay 1996); with dominance or epistasis the non-additive covariance is not transmitted (e.g. pure dominance at $p = 0.5$ gives a positive value here but no response). G are [genetic_values\(\)](#); for a [liability_threshold\(\)](#) trait the liability is used, the scale on which its genetic model is defined.

Usage

```
coheritability(sim, rep = 1L)
```

Arguments

sim	a phenotype_sim.
rep	replication (default 1).

Value

a symmetric $n_traits \times n_traits$ matrix with Trait_k dimnames.

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow (correlated response to selection).

See Also

[genetic_values\(\)](#), [liability_threshold\(\)](#)

Examples

```
data("SNP55K_maize282_maf04")
sim <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2,
                          architecture = "pleiotropy", cor = 0.6,
                          h2 = c(0.5, 0.3), n_qtn = 20, seed = 1)
coheritability(sim)
```

combining_ability	<i>Combining ability: GCA, SCA and testcross merit</i>
-------------------	--

Description

Scores candidates by their merit **in crosses** – general combining ability (GCA, the average merit of a parent in its crosses) and specific combining ability (SCA, the deviation of a particular cross from what its parents' GCAs predict), following the two-way model of Sprague & Tatum (1942) and, for the diallel, Griffing (1956). This is the criterion of half-sib / reciprocal recurrent selection and hybrid development. It is not the per se value ([genotypic_value\(\)](#)) and, in general, not the transmissible breeding value.

Usage

```
combining_ability(
  candidates,
  testers = NULL,
  qtn,
  a,
  d = 0,
  design = c("topcross", "factorial", "diallel"),
  method = c("expected", "simulated"),
  n_progeny = NULL,
  h2 = NULL,
  var_e = NULL,
  ref = NULL,
  seed = NULL,
  interference = NULL
)
```

Arguments

candidates	a Population of candidates, each individual once (the same individual under a second id is an error).
testers	a Population of testers on the same marker map (NULL for "diallel"), each individual once.
qtn, a, d	frozen loci and their additive effects and dominance deviations, as for genotypic_value() . d may be a single value for every locus (default 0, purely additive).
design	"topcross", "factorial" or "diallel".

method	"expected" or "simulated".
n_progeny	progeny per cross for "simulated".
h2, var_e, ref	optional residual for "simulated": give at most one of h2 / var_e (none: no residual); ref (the reference population for h2) is an error without h2.
seed	optional RNG seed for "simulated".
interference	crossover interference model of <code>cross()</code> (see its section "Crossover interference"): NULL (default: the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the isqg stream, the default before 2.0.0.9003) or <code>list(nu = , p =)</code> . Used for the meioses of method = "simulated"; giving it with "expected", which runs no meiosis, is an error.

Details

The architecture is frozen, as for `genotypic_value()`: loci qtn with additive effects a and dominance deviations d on the realized scale, so a locus has genotypic value $-a / d / +a$ at gene content $0 / 1 / 2$ (use `template_effects()` to take them from a simulation). Epistasis is outside this model.

Value

A combining_ability object: a list with `gca` (named, over candidates), `gca_testers` (factorial only), `sca` (matrix: candidates x testers, or the symmetric candidate x candidate matrix for a diallel, NA on its diagonal), `cross_means` (the Y matrix), `grand_mean` (μ), `design`, `method`, and for "simulated" also `n_progeny`, `var_e` and `progeny` (the progeny Population, pedigree recorded).

Expected (no simulation)

method = "expected" gives the conditional expectation of each cross's progeny value given the parents' genotypes – the idealized truth a simulation knows. It uses no random numbers. At one locus a parent with gene content x transmits the counted allele with probability $x / 2$, so for parents with $g_i = x_i / 2$ and $g_k = x_k / 2$ the progeny is AA, Aa or aa with probabilities $g_i g_k$, $g_i(1 - g_k) + (1 - g_i)g_k$ and $(1 - g_i)(1 - g_k)$, and its expected value is (this package's derivation, summed over loci)

$$E[G_{ik}] = g_i g_k a + [g_i(1 - g_k) + (1 - g_i)g_k]d - (1 - g_i)(1 - g_k)a.$$

Averaged over a set of testers this is linear in the candidate's gene content, with slope half the tester-referenced average effect $\alpha_T = a + d(1 - 2p_T)$ (p_T the testers' mean gamete frequency): Falconer & Mackay's (1996) average effect $\alpha = a + d(q - p)$ at the testers' allele frequencies. So a candidate's GCA depends on the tester, and with testers drawn from the candidates' own population ($p_T = p$) it equals half its breeding value; with $d = 0$ every *expected* SCA is exactly zero. Expectations depend only on the parents' gamete frequencies, so linkage does not enter them (it enters the variance of a realized family).

Simulated (an estimate)

method = "simulated" realizes `n_progeny` progeny of every cross with the crossing core (`mate()`), scores them with `genotypic_value()`, optionally adds an independent residual per progeny (exactly one of `h2 / var_e`, as in `phenotype_value()`, `h2` being the heritability of the total genotypic value in `ref`, default the progeny), and averages per cross. It is a finite-sample estimate of

the expected values, with Mendelian (and, with a residual, environmental) sampling error. Consequently a *simulated* SCA is not zero for $d = 0$ when the parents are heterozygous (segregation among the n_{progeny} progeny of a cross is a sampling deviation from the expectation, shrinking with n_{progeny}); it is zero for $d = 0$ only in expectation, and exactly in a simulation from fully inbred parents (identical gametes, so no Mendelian sampling) **that has no residual**: an independent residual per progeny is environmental sampling error, which inbred parents do not remove, so with h^2 / var_e the simulated SCA is not zero even then (it shrinks with n_{progeny}). `ref`, like `h2`, only scales the residual and is an error without `h2`. With `seed` the caller's random-number stream is left as it was found.

Designs and centering

- "topcross": every candidate crossed to every tester; GCA over candidates.
- "factorial": the same crosses (North Carolina Design II), with GCA reported for candidates and for testers.
- "diallel": every unordered pair of distinct candidates (testers unused; Griffing's method 4 layout, at least three candidates). With Y the cross means and μ their mean, the factorial / topcross GCA is the row (column) mean minus μ and $\text{SCA} = Y - \mu - g_i - g_k$. In the diallel each cross involves two candidates, so this package's least-squares solution is $g_i = (m_i - \mu)(p - 1) / (p - 2)$, with m_i the mean of candidate i 's $p - 1$ crosses; again $\text{SCA} = Y - \mu - g_i - g_k$. In every design GCAs sum to zero and each candidate's SCAs sum to zero. Only the diallel *without reciprocals* is modelled (Griffing's method 4, one cross per unordered pair): reciprocal or maternal effects are outside the model and the cross matrix is symmetric. Individuals are identified by pedigree key, which includes the founders' pool label: two populations built from the same genotypes under different `as_population(pool = )` labels are different individuals (a cross between them is recorded as a cross, not a self, although its expected value is the same).

References

- Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow – the genotypic values $-a, d, a$ and the average effect $\alpha = a + d(q - p)$.
- Sprague GF, Tatum LA (1942) General vs. specific combining ability in single crosses of corn. *Journal of the American Society of Agronomy* (now *Agronomy Journal*) 34:923–932. doi:10.2134/agronj1942.00021962003400100008x
- Griffing B (1956) Concept of general and specific combining ability in relation to diallel crossing systems. *Australian Journal of Biological Sciences* 9:463–493. doi:10.1071/B19560463

See Also

`template_effects()`, `genotypic_value()`, `mate()`, `select_ind()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:8)
q <- c("ss196442916", "ss196439337", "ss196480535")
ca <- combining_ability(pop[1:5], pop[6:8], qtn = q, a = c(1, 0.5, 0.25),
                        d = c(0.5, 0.5, 0))
```

```
ca$gca
ca$sca
```

complex_phenotypes	<i>Combine single-architecture models under a common heritability</i>
--------------------	---

Description

`complex_phenotypes()` merges two or more realized `phenotype_sim` objects built on the same genotypes with the same number of traits. The inputs' genetic values are summed (so each contributes in proportion to its genetic variance), the sum is scaled to the requested genetic variance share `h2`, and a common residual is added; the inputs' individual residuals are discarded. Realized `H2` is reported from the resulting sample and can differ slightly because the generated genetic and residual vectors need not be exactly orthogonal. This recreates partial pleiotropy by combining, for example, a "pleiotropy" model with an "independent" model.

Usage

```
complex_phenotypes(
  ...,
  h2,
  reps = 1,
  resid_cor = NULL,
  residual_mode = c("fixed", "random")
)
```

Arguments

<code>...</code>	two or more <code>phenotype_sim</code> objects.
<code>h2</code>	requested genetic variance share (scalar or length <code>n_traits</code>), on the single-record scale $V_G / (V_G + V_E)$.
<code>reps</code>	number of independent records averaged into each entry's phenotype (default 1; entry-mean replication, as in simulate_phenotype()): a positive whole number or one per trait. The common residual drawn here is divided by <code>sqrt(reps)</code> , so the phenotype is an entry mean. <code>h2</code> stays the single-record target share $V_G / (V_G + V_E)$; the entry-mean target is $V_G / (V_G + V_E / \text{reps})$. The realized <code>H2</code> reported is $\text{Var}(G) / \text{Var}(\bar{y})$ from the realized values, with $\text{Var}(\bar{y}) = V_G + V_E / \text{reps} + 2 \text{Cov}(G, e) / \text{sqrt}(\text{reps})$ (record scale $V_G + V_E + 2 \text{Cov}(G, e)$): it equals the allocation formula only when the sample covariance $\text{Cov}(G, e)$ is zero in fixed mode. In random mode the residual sample variance can also differ from its parameter. The inputs' own <code>reps</code> are not used (their residuals are discarded); with <code>reps = 1</code> the result is bit-identical to a call without it.
<code>resid_cor</code>	target residual correlation between traits of the common residual: <code>NULL</code> (default, independent residuals, bit-identical to a call without it), one value in <code>[-1, 1]</code> for every pair, or an <code>n_traits x n_traits</code> symmetric positive semi-definite

matrix with unit diagonal. Built as in `simulate_phenotype()`: each trait keeps exactly its $1 - h^2$ residual sample variance in fixed mode. Random mode uses this as a distribution variance parameter instead.

`residual_mode` "fixed" (default) or "random", as in `simulate_phenotype()`. Applies to the newly drawn common residual, independently of the modes of the input models.

Details

Every input must be a **complete** model, exactly as for `phenotypes_long()`: if an input set an h^2 , its layer proportions must fill it (otherwise `complex_phenotypes()` errors with the same "Incomplete h^2 allocation" message). The result is **terminal**: it has no layers, so `additive()`, `dominance()`, `epistasis()`, `vqtl()` and `transcriptome()` refuse to extend it (build the full model first, then combine). It carries no per-input state: `mediation_split()` is NULL for a combined model (the environmental expression-mediated part of an input is discarded with the input's residual, so no split of the combined phenotype is defined), and breeding-value selection methods that need per-locus effects refuse it.

Value

a combined `phenotype_sim` (architecture "complex").

Examples

```
data("SNP55K_maize282_maf04")
pleio <- simulate_phenotype(SNP55K_maize282_maf04, architecture = "pleiotropy",
                           n_traits = 2, seed = 10, cor = 0.5)
pleio <- additive(pleio, prop = 0.4, n_qtn = 3)
indep <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, seed = 11)
indep <- additive(indep, prop = 0.3, n_qtn = 3)
both <- complex_phenotypes(pleio, indep, h2 = 0.5)
```

<code>cor_ar1</code>	<i>AR(1) correlation matrix</i>
----------------------	---------------------------------

Description

The first-order autoregressive correlation matrix $R_{ij} = \rho^{|i-j|}$: adjacent traits correlate ρ , and the correlation decays geometrically with the distance between them. Use it for repeated measurements of one trait over time points or environments (high-throughput phenotyping), as the genetic correlation target `cor` of architecture = "pleiotropy" and/or the residual correlation `resid_cor` of `simulate_phenotype()`.

Usage

```
cor_ar1(n_traits, rho)
```

Arguments

`n_traits` number of traits (time points), at least 2.
`rho` correlation between adjacent traits, in $(-1, 1)$.

Value

an `n_traits` x `n_traits` positive-definite correlation matrix.

See Also

[simulate_phenotype\(\)](#)

Examples

```
cor_ar1(4, 0.8)
data("SNP55K_maize282_maf04")
# five time points: genetic and residual correlations decay with time lag
sim <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 5,
                          architecture = "pleiotropy", cor = cor_ar1(5, 0.9),
                          resid_cor = cor_ar1(5, 0.5), h2 = 0.4, n_qtn = 30,
                          seed = 1)
```

create_phenotypes	<i>Simulation of single/multiple traits under different models and genetic architectures.</i>
-------------------	---

Description

Superseded. `create_phenotypes()` is the frozen original engine, retained unchanged for backward compatibility (full v1 signature, legacy seed arithmetic, and `RNGversion("3.5.1")`). It is bugfix-only and is *not* a wrapper over the v2 grammar (the two implementations coexist). For new work prefer the composable grammar: [simulate_phenotype\(\)](#) with [additive\(\)](#), [dominance\(\)](#), [epistasis\(\)](#), [vqtl\(\)](#) and [complex_phenotypes\(\)](#).

Usage

```
create_phenotypes(
  geno_obj = NULL,
  geno_file = NULL,
  geno_path = NULL,
  QTN_list = list(add = list(NULL), dom = list(NULL), epi = list(NULL), var = list(NULL)),
  prefix = NULL,
  rep = NULL,
  ntraits = 1,
  h2 = NULL,
  mean = NULL,
  model = NULL,
```

```
architecture = "pleiotropic",
add_QTN_num = NULL,
dom_QTN_num = NULL,
epi_QTN_num = NULL,
var_QTN_num = NULL,
epi_type = NULL,
epi_interaction = 2,
pleio_a = NULL,
pleio_d = NULL,
pleio_e = NULL,
trait_spec_a_QTN_num = NULL,
trait_spec_d_QTN_num = NULL,
trait_spec_e_QTN_num = NULL,
add_effect = NULL,
dom_effect = NULL,
epi_effect = NULL,
var_effect = NULL,
remove_add_effect = FALSE,
same_add_dom_QTN = FALSE,
same_mv_QTN = FALSE,
big_add_QTN_effect = NULL,
degree_of_dom = 1,
type_of_ld = "indirect",
ld_min = 0.2,
ld_max = 0.8,
ld_method = "composite",
sim_method = "geometric",
vary_QTN = FALSE,
cor = NULL,
cor_res = NULL,
QTN_variance = FALSE,
seed = NULL,
home_dir = NULL,
output_dir = NULL,
export_gt = FALSE,
output_format = "long",
to_r = FALSE,
out_genotype = NULL,
chr_prefix = "chr",
remove_QTN = FALSE,
warning_file_saver = TRUE,
constraints = list(maf_above = NULL, maf_below = NULL, hets = NULL),
maf_cutoff = NULL,
nrows = Inf,
na_string = "NA",
SNP_effect = "Add",
SNP_impute = "Middle",
quiet = FALSE,
```

```

    verbose = TRUE,
    RNGversion = "3.5.1"
)

```

Arguments

geno_obj	Marker data set loaded as an R object. Currently either HapMap or numericalized files (code as aa = -1, Aa = 0 and AA = 1, e.g. data("SNP55K_maize282_maf04")) are accepted. These and other file formats (VCF, GDS, and Plink Bed/Ped files) may be read from file with geno_file or geno_path. Only one of geno_obj, geno_file or geno_path should be provided.
geno_file	Name of a marker data set to be read from a file. If in a different folder, the whole path should be provided. Formats accepted are Numeric, HapMap, VCF, GDS, and Plink Bed/Ped files. Notice that the major allele will always be 1 and the minor allele -1. Thus, when using Plink Bed files, the dosage information will be converted to the opposite value.
geno_path	Path to a folder containing the marker data set file/files (e.g., separated by chromosome). Formats accepted are: Numeric, HapMap, VCF, GDS, and Plink Bed/Ped files
QTN_list	A list of specific markers to be used as QTNs. If one wants to specify the QTNs instead of selecting them randomly, at least one of the following elements should be provided: QTN_list\$add, QTN_list\$dom, and/or QTN_list\$epi. The element \$add, \$dom, and \$epi are lists containing a vector of markers for each of the traits to be simulated. For example, to simulate 2 traits controlled by 1 pleiotropic and 2 trait-specific additive QTNs, the user would create a list of marker names marker_list <- list(add = list(trait1 = c("marker1", "marker2", "marker3"), trait2 = c("marker1", "marker4", "marker5"))) and set QTN_list = marker_list. On the other hand, to simulate a single trait controlled by 1 additive and 2 dominance QTNs, the marker list would be marker_list <- list(add = list("marker1"), dom = list(c("marker2", "marker3"))). Notice that these vectors with maker names is used in the order they appear. For instance, in the list marker_list <- list(add = list(trait9 = c("marker1"), trait4 = c("marker5"))), the vector names itself ("trait9" and "trait4") are ignored and "trait9" will be the vector of markers used to simulate the first trait and "trait4" will be the vector of markers used to simulate the second trait. Also, when using QTN_list, many parameters used for selecting QTNs will be ignored (e.g., constraints). QTN_list requires ntraits >= 2 (a single trait cannot be simulated from a marker list: this combination is not supported and stops with an error) and cannot be combined with same_add_dom_QTN = TRUE (list the same markers in QTN_list\$add and QTN_list\$dom instead) or with architecture = "LD". Every marker must exist in the marker data.
prefix	If geno_path points to a folder with files other than the marker data set, a part of the data set name may be used to select the desired files (e.g., prefix = "Chr" would read files Chr1.hmp.txt, ..., Chr10.hmp.txt but not HapMap.hmp.txt).
rep	The number of experiments (replicates of a trait with the same genetic architecture) to be simulated.

ntraits	The number of multi-trait phenotypes to simulate under pleiotropic, partially pleiotropic, and LD (spurious pleiotropy) architectures (see architecture). If not assigned, a single trait will be simulated. Currently, the only option for the LD architecture is ntraits = 2.
h2	The heritability for each traits being simulated. It could be either a vector with length equals to ntraits, or a matrix with ncol equals to ntraits. If the later is used, the simulation will loop over the number of rows and will generate a result for each row. If a single trait is being simulated and h2 is a vector, one simulation of each heritability value will be conducted. Either none or all traits are expected to have h2 = 0. Values must lie in [0, 1]. Only the first trait's heritability of a row enters the residual seed, and positive values of at most 0.05 (exactly 0.05 included, since round(10 * 0.05) is 0 in R) are rejected when rep > 1: use h2 > 0.05 (see seed). With vQTL = TRUE, h2 is met for an individual at the median vQTL standard deviation, not as the population variance ratio, which is $1/(1+k^2\text{mean}(\sigma^2))$ with $k^2 = (1/h^2 - 1)/\text{median}(\sigma^2)$: lower than, equal to or higher than h2 as $\text{mean}(\sigma^2)$ is greater than, equal to or less than $\text{median}(\sigma^2)$.
mean	A vector with the mean (intercept) value for each of the simulated traits. If omitted, the simulated traits will be centered to zero.
model	The genetic model to be assumed. The options are "A" (additive), "D" (dominance), "E" (epistatic), "V" (variance QTL) as well as any combination of those models such as "AE" or "AD" (upper case). For a single trait (ntraits = 1) the combinations "DE" and "ADE" are not supported and stop with an error (use two or more traits, "AE"/"AD", or simulate_phenotype()).
architecture	The genetic architecture to be simulated. Should be provided if ntraits > 1. Possible options are: 'pleiotropic' (default), for traits being controlled by the same QTNs; 'partially', for traits being controlled by pleiotropic and trait-specific QTNs; 'LD', for traits being exclusively controlled by different QTNs in "direct" or "indirect" (See type_of_ld, ld_min, and ld_max below) linkage disequilibrium. Currently the only option for architecture = "LD" is ntraits = 2.
add_QTN_num	The number of additive quantitative trait nucleotides (QTNs) to be simulated.
dom_QTN_num	The number of dominance QTNs to be simulated.
epi_QTN_num	The number of epistatic (Currently, only additive x additive epistasis are simulated) QTNs to be simulated.
var_QTN_num	The number of variance quantitative trait nucleotides (QTNs) to be simulated.
epi_type	reserved for future epistasis types; currently unused.
epi_interaction	Number of markers that compose an epistatic QTN. If epi_interaction = 2 (default), a 2-way interaction (marker1 x marker2) will be used to simulate epistatic QTNs. If epi_interaction = 3 a 3-way interaction (marker1 x marker2 x marker3) will be used instead.
pleio_a	The number of pleiotropic additive QTNs to be used if architecture = "partially". When sim_method = custom (see below), the first effects will be assigned to the pleiotropic QTNs and the last to the trait-specific ones. For instance, in a scenario where ntraits = 2, pleio_a = 2, trait_spec_a_QTN_num = 1, and add_effect

	= list(trait1 = c(0.1, 0.2, 0.3), trait2 = c(0.4, 0.5, 0.6)), the trait-specific QTNs for trait 1 and trait 2 will be 0.3 and 0.6, respectively. The first two allelic effects will be assigned to the pleiotropic QTNs.
pleio_d	The number of pleiotropic dominance QTNs to be used if architecture = "partially" (See pleio_a for details).
pleio_e	The number of pleiotropic epistatic QTNs to be used if architecture = "partially" (See pleio_a for details).
trait_spec_a_QTN_num	The number of trait-specific additive QTNs if architecture = "partially". It should be a vector of length equals to ntraits.
trait_spec_d_QTN_num	The number of trait-specific dominance QTNs if architecture = "partially". It should be a vector of length equals to ntraits.
trait_spec_e_QTN_num	The number of trait-specific epistatic QTNs if architecture = "partially". It should be a vector of length equals to ntraits.
add_effect	Additive effect size to be simulated. It may be either a vector (assuming ntraits = 1 or one allelic effect per trait to create a geometric series, sim_method = "geometric") or a list of length = ntraits, i.e., if ntraits > 1, a list with one vector of additive effects should be provided for each trait. Unless big_add_QTN_effect is provided, the length of each vector should be equal to the number of additive QTNs being simulated. A single value per trait is the base of a geometric series; a vector with one value per QTN is used as is. All effect classes (add_effect, dom_effect, epi_effect, var_effect) must use the same style: mixing one style for one class and the other for another class, or any other length, is rejected.
dom_effect	Similar to the add_effect, it could be either a vector or a list. Optional if same_add_dom_QTN = TRUE.
epi_effect	Epistatic (additive x additive) effect size to be simulated. Similar to the add_effect, it could be either a vector or a list.
var_effect	Similar to the add_effect, it could be either a vector or a list.
remove_add_effect	Experimental option for simulating variance QTL without mean effect.
same_add_dom_QTN	A boolean for selecting markers to be both additive and dominance QTNs. Default FALSE. Not supported for a single trait (ntraits = 1) or together with QTN_list.
same_mv_QTN	A boolean for selecting markers to be both additive and variance QTNs. Default FALSE.
big_add_QTN_effect	Additive effect size for one possible major effect quantitative trait nucleotide. If ntraits > 1, big_add_QTN_effect should have length equals ntraits. If add_QTN_num > 1, this large effect will be assigned to the first QTN.
degree_of_dom	If the same set of QTNs are being used for simulating additive and dominance effects, the dominance allelic effect could be a proportion of the additive allelic effect. In other words, degree_of_dom equals to 0.5, 1, 1.5 will simulate, partial dominance, complete dominance and overdominance, respectively.

type_of_ld	Type of LD used to simulate spurious pleiotropy. If "indirect" (default), an intermediate marker is selected from which two adjacent markers (one upstream and another downstream) will be chosen based on its LD with the intermediate marker to be the QTNs. Optionally, in the "direct" method, one marker is selected to be a QTN for trait 1, and a second marker is selected based on its LD with the first selected marker to be the QTN for trait 2.
ld_min	Minimum Linkage disequilibrium for selecting QTNs when architecture = LD. The default is $ld_min = 0.2$ (markers should have a minimum LD of 0.2 to be used as QTNs). LD is compared as an <i>absolute value</i> on the scale of ld_method (for the default "composite" and for "corr" this is a correlation, not r^2). The window $[ld_min, ld_max]$ is inclusive and $ld_min \leq ld_max < 1$ is required. Only the randomly drawn markers are subject to constraints; the linked partner markers are found by walking along the marker order and are not filtered.
ld_max	Maximum Linkage disequilibrium for selecting QTNs when architecture = LD. The default is $ld_max = 0.8$ (markers should have an LD of at maximum 0.8 to be used as QTNs). It must be smaller than 1.
ld_method	Four methods can be used to calculate linkage disequilibrium values: "composite" for LD composite measure (Default), "r" for R coefficient (by EM algorithm assuming HWE, it could be negative), "dprime" for D' , and "corr" for correlation coefficient (see snpgdsLDpair from package SNPRelate).
sim_method	Provide the method of simulating allelic effects. The options available are "geometric" and "custom". For multiple QTNs, a geometric series may be simulated, i.e., if $add_effect = 0.5$, the effect size of the first QTNs will be 0.5, the effect size of the second QTN will be 0.5^2 , and the effect of the n^{th} QTN will be 0.5^n .
vary_QTN	A boolean that determines if the same set of quantitative trait nucleotide (QTN) should be used to generate genetic effects for each experiment ($vary_QTN = FALSE$) or if a different set of QTNs should be used for each replication ($vary_QTN = TRUE$).
cor	Option to simulate traits with a predefined genetic correlation. It should be a symmetric, positive-definite correlation matrix with a number of rows = ntraits (other matrices are rejected; the frozen eigenvalue clamp that used to repair them realized a different correlation and inflated the genetic variance). It is ignored, with a warning, when ntraits = 1. Default = NULL. Notice that when opting for controlling the correlation, the genetic effects are transformed using Cholesky decomposition. In this case, the correlation of genetic effects for different traits will be as provided, but due to the transformation, the actual allelic effects of correlated traits may be different than the input allelic effect.
cor_res	Option to simulate traits with a predefined residual correlation. It should be a correlation matrix with number of rows = ntraits (symmetric, unit diagonal, positive semi-definite). If NULL, an identity matrix (independent residuals) will be used. The "Residual Correlation" printed in the log is this input matrix, not an estimate from the simulated residuals.
QTN_variance	Whether or not the percentage of the phenotypic variance explained by each QTN (QTN variance / phenotypic variance) should be exported. The default is

	FALSE. Notice that this is calculated prior to any transformation, such as the whitening/coloring transformation used to assign user-specified correlation to the genetic effect. It may not reflect the actual variance explained when the data is transformed.
seed	Value to be used by <code>set.seed</code> . If NULL (default), a master seed is drawn with <code>runif(1, 0, 1000000)</code> from the caller's random-number stream; the caller's RNG kind (including <code>sample.kind</code>) and state are restored on exit, the stream being advanced by that one draw only. The master seed is saved at the top of the log file. Seeds for the individual sampling steps are derived from <code>seed</code> by adding replicate (and effect-class) offsets, and, unless <code>verbose = FALSE</code> , the seed actually used in every simulation is exported with the simulated phenotypes (<code>Seed_number_for_*</code> and <code>Seed_num_for_*</code> files). The <i>residual</i> (environmental) seed of replicate <code>r</code> for row <code>i</code> of <code>h2</code> is as <code>integer((seed + r) * round(10 * h2[i, 1]))</code> , i.e. it uses the heritability of the <i>first trait</i> only, and <code>seed + r</code> when all heritabilities are 0. For example, <code>seed = 123</code> , <code>h2 = 0.5</code> and replicate 10 use $(123 + 10) * 5 = 665$. Consequences: (1) <code>h2 <= 0.05</code> (positive) gives <code>round(10 * h2) = 0</code> (R rounds 0.5 to even, so exactly 0.05 is included), hence seed 0 for every replicate and identical replicates, so it is rejected when <code>rep > 1</code> ; the accepted range is <code>h2 > 0.05</code> ; (2) heritabilities that round to the same multiple of 0.1 (for example 0.26 and 0.34) share their random stream, so their residuals are perfectly correlated across rows of <code>h2</code> ; (3) every derived seed must fit R's integer range, so very large seeds are rejected up front. For <code>architecture = "LD"</code> the marker search also uses retry seeds <code>seed * s + ...</code> with <code>s</code> up to 10, so the bound is about <code>.Machine\$integer.max / 10</code> (about 214 million; the error message reports the exact limit for the call).
home_dir	Directory where files should be saved. It may be <code>home_dir = getwd()</code> .
output_dir	Name of the folder created inside <code>home_dir</code> to hold this run's output files. A single run writes several files (phenotypes, selected QTNs, genetic values and a log), so they are always collected in a folder of their own rather than written loose. Defaults to <code>"simplePHENOTYPES_output"</code> ; if that folder already exists, a numbered variant such as <code>simplePHENOTYPES_output(1)</code> is created so earlier results are never overwritten. Use <code>output_dir = ""</code> to write directly into <code>home_dir</code> , which was the behavior before version 2.0. If a run fails, the run folder created by the call is removed (with <code>output_dir = ""</code> the new entries of <code>home_dir</code> are removed instead).
export_gt	If TRUE genotypes of selected QTNs will be saved at file. If FALSE (default), only the QTN information will be saved.
output_format	Four options are available for saving simulated phenotypes: <code>'multi-file'</code> , saves each simulation in a separate file; <code>'long'</code> (default for multiple traits), appends each experiment (rep) to the last one (by row); <code>'wide'</code> , saves experiments by column (default for single trait) and <code>'gemma'</code> , saves <code>.fam</code> files to be used by <code>gemma</code> or other software that uses <code>plink</code> bed files. (renaming <code>.fam</code> file with the same name of the <code>bim</code> and <code>bed</code> files is necessary). <code>'wide'</code> with <code>ntraits > 1</code> requires <code>rep >= 2</code> .
to_r	Option for outputting the simulated results as an R data.frame in addition to saving it to file. If TRUE, results need to be assigned to an R object (see vignette).

	If <code>ntraits = 1</code> and <code>length(h2) > 1</code> , results for each <code>h2</code> will be saved in a list. Several rows of <code>h2</code> with <code>to_r = TRUE</code> are rejected when <code>ntraits > 1</code> or <code>vary_QTN = TRUE</code> (only the last row would be returned); read the output files instead.
<code>out_geno</code>	Optionally saves the numericalized genotype either as "numeric" (see vignettes for an example data), "BED" or "gds". The default is NULL.
<code>chr_prefix</code>	If input file format is VCF and <code>out_geno = "BED"</code> , and a prefix is used in the chromosomes names, <code>chr_prefix</code> may be used to avoid issues in converting to bed files (e.g., <code>chr_prefix = "chr"</code> in "chr01").
<code>remove_QTN</code>	Whether or not a copy of the genotypic file should be saved without the simulated QTNs. The default is FALSE. If <code>vary_QTN = TRUE</code> , the question "Are you sure that you want to save one genotypic file/rep (<code>remove_QTN = TRUE</code> and <code>vary_QTN = TRUE</code>) (type yes or no) ?" will pop up to avoid saving multiple large files unintentionally
<code>warning_file_saver</code>	Skips the interactive question and saves all files when <code>remove_QTN = TRUE</code> and <code>vary_QTN = TRUE</code> .
<code>constraints</code>	Set constraints for QTN selection. Currently, the options are <code>maf_above</code> (the minimum value of minor allele frequency, a double between 0 - 0.5), <code>maf_below</code> (the maximum value of minor allele frequency, a double between 0 - 0.5; both comparisons are strict, i.e. a marker whose MAF equals the bound is excluded), and <code>hets</code> ('include' and 'remove'). They filter only the randomly drawn QTNs, not the partner markers of the LD architecture. All of these options are NULL by default ('list(maf_above = NULL, maf_below = NULL, hets = NULL)'). For instance, if the parameters used are <code>constraints = list(maf_above = 0.3, maf_below = 0.44, hets = "include")</code> , only heterozygote markers with minor allele frequency between 0.3 and 0.44 will be selected to be QTNs. The option "remove" would only select homozygote markers to be QTNs.
<code>maf_cutoff</code>	Option for filtering the data set based on minor allele frequency (Not to be confounded with the constraints option which will only filter possible QTNs). It may be useful when outputting the genotypic data set.
<code>nrows</code>	Option for loading only part of a data set. Used when marker data is in numeric or HapMap format. Please see <code>data.table::fread</code> for details.
<code>na_string</code>	Tell <code>create_phenotypes</code> what character represents missing data (default is "NA"). Used when the input marker data is numeric or HapMap.
<code>SNP_effect</code>	Parameter used for numericalization. The options are: Add (AA = 1, Aa = 0, aa = -1), Dom (AA = -1, Aa = 0, aa = -1), Left (AA = 1, Aa = -1, aa = -1), Right (AA = 1, Aa = 1, aa = -1). The default option is Add. It is used to numericalize file input (HapMap, VCF, PLINK, GDS) and is rejected, when not "Add", for a <code>geno_obj</code> that is already numeric.
<code>SNP_impute</code>	Naive imputation for HapMap numericalization. The options are: Major (NA <- 1), Middle (NA <- 0), and Minor (NA <- -1). Numeric <code>geno_obj</code> values must be coded -1/0/1 (or 0/1/2 dosage that includes the value 2); a 0/1-only matrix is ambiguous and rejected.
<code>quiet</code>	Whether or not the log file should pop up into R once the simulation is done.

verbose	If FALSE, suppress all prints and suppress individual seed numbers from being saved to file. The master seed (unique value required to reproduce results) is saved at the top of the log file.
RNGversion	Parameter to set the random number generator. Different R versions may be selected, the default value is 3.5.1.

Value

Single or multi-trait phenotypes in one of many formats. Numericalized marker data set with or without the selected QTNs. Diagnostic files (log, QTN information, summary of LD between QTNs, proportion of phenotypic variance explained by each QTN).

Rejected inputs and unsupported combinations

The frozen engine validates its arguments up front and stops with a message that names the argument and the remedy, instead of failing cryptically or returning wrong numbers. Valid inputs produce exactly the numbers of earlier releases. Inner errors are re-signalled (earlier releases printed the message and returned NULL), and on error only the run folder created by the call is removed. The following are rejected:

- QTN_list with ntraits = 1 (a single trait cannot be simulated from a user-specified marker list; use random QTNs or `simulate_phenotype()` with qtn =), with same_add_dom_QTN = TRUE, with architecture = "LD", or with a number of trait-specific marker vectors different from ntraits.
- Single-trait models combining dominance and epistasis ("DE", "ADE") and single-trait same_add_dom_QTN = TRUE.
- architecture = "LD" with ntraits > 2, with ld_max >= 1 (an absolute LD of 1 is met by a marker paired with itself) or ld_min > ld_max, and model = "D" with type_of_ld = "indirect". When the LD search cannot deliver distinct, linked marker pairs inside [ld_min, ld_max] for the chosen seed (indirect: a marker selected for both traits or a duplicated QTN; direct: a pair outside the window, on two chromosomes or paired with itself), the call stops with an "LD contract" error: change seed, the LD window or type_of_ld. For type_of_ld = "direct" the search is first run exactly as in earlier releases and, only when that attempt fails, repeated with derived seeds (section "Direct-LD search retries"), so the error is now rare.
- h2 outside [0, 1]; h2 positive but not above 0.05 (that is, $0 < h2 \leq 0.05$, including exactly 0.05) with rep > 1 (see seed); to_r = TRUE with several rows of h2 unless a single trait is simulated with vary_QTN = FALSE.
- cor that is not a symmetric positive-definite ntraits x ntraits matrix (it is no longer repaired silently) and cor_res that is not a symmetric, positive semi-definite correlation matrix.
- Effect sizes given as a mixture of one value per QTN and a single geometric-series base, or with a length that is neither 1 nor the number of QTNs.
- Variance QTL ("V") with ntraits > 1, with h2 = 0, or with var_effect values that make the standard-deviation multiplier negative, or with a constant additive baseline (for example model = "AV" with add_effect = 0: the variance multiplier is undefined, so the call stops instead of returning NaN phenotypes).
- Unknown output_format, output_format = "wide" with ntraits > 1 and rep = 1, rep/model/seed that are missing or malformed, and more QTNs than markers.

Duplicated marker positions

Duplicated marker positions (chr_pos, the same chromosome and position on several rows) are *accepted* by the fully pleiotropic and the partially pleiotropic architectures: QTN columns are addressed by row position, never by the Chr_<chr>_<pos> name, so duplicated names cannot change which markers enter a genetic value (an epistatic interaction, for example, multiplies the intended columns). The architecture = "LD" searches are stricter: a duplicated QTN or a QTN pair sharing a chromosome position is reported as an "LD contract" error (see above).

Direct-LD search retries

With architecture = "LD" and type_of_ld = "direct" the marker search of every replicate first runs exactly as in earlier releases (same seeds, same random draws); whenever that attempt meets the LD contract (distinct markers on one chromosome with an absolute LD inside the inclusive window [ld_min, ld_max], and the reported LD equal to the recomputed one) its output is unchanged, bit for bit. Only when it does not (the contract check fails, the search stops with "None of the selected SNPs met the minimum/maximum LD threshold", or it runs past the first/last marker) the replicate is searched again, up to 50 attempts in total, from the seed seed - (a - 1) * 1000003 for attempt a (seed + (a - 1) * 1000003 when seed <= 0); the retry seeds are then combined with the replicate and QTN offsets exactly like the original one. Retry attempts also reset the neighbour pointers after a marker is re-drawn, which the earlier dominance walks (model containing "D") did not: they kept pairing the new marker with the neighbours of the old one, so most pairs failed the contract. The window itself is never relaxed: every accepted pair is verified against [ld_min, ld_max] and the chromosome constraint. After 50 failed attempts the "LD contract" error is raised as before; a retry attempt that uses up every candidate marker without finding a pair inside the window (a window too narrow for the data, which used to end in the cryptic "invalid first argument") stops the search at once with an informative error. The seed of the attempt that succeeded is the one written to the Seed_num_for_* file and, when verbose = TRUE, a message names the attempt. Indirect LD is not retried. A dominance-only model ("D") can still stop with the "All individuals are homozygote for the selected dominance QTNs" error, which is a separate check on the selected markers.

Author(s)

Samuel B Fernandes and Alexander E Lipka. Last update: Jan 19, 2021

References

Fernandes, S.B., and Lipka, A.E., 2020 simplePHENOTYPES: SIMulation of pleiotropic, linked and epistatic SIMulation of Pleiotropic, Linked and Epistatic PHENOTYPES. BMC Bioinformatics 21(1):491, doi:[10.1186/s12859-020-03804-y](https://doi.org/10.1186/s12859-020-03804-y)

Examples

```
# Simulate 50 replications of a single phenotype.
data("SNP55K_maize282_maf04")
pheno <-
  create_phenotypes(
    geno_obj = SNP55K_maize282_maf04,
```

```

    add_QTN_num = 3,
    add_effect = 0.2,
    big_add_QTN_effect = 0.9,
    rep = 10,
    h2 = 0.7,
    model = "A",
    to_r = TRUE,
    home_dir = tempdir(),
    quiet = T
  )
# For more examples, please run the following:
# vignette("simplePHENOTYPES")

```

cross

Cross two individuals

Description

Produces n progeny from a biparental cross. Each progeny receives one recombinant gamete from each parent, so the two parents contribute one homologue apiece.

Usage

```
cross(mother, father, n = 1, seed = NULL, interference = NULL)
```

Arguments

mother, father	single-individual Populations (use [to select one). Their roles are symmetric apart from which homologue a progeny inherits first; there is no sex-specific recombination. Crossing an individual with itself is a self: it draws what selfcross() draws and the pedigree records it as one (design = "self", see parentage()).
n	number of progeny.
seed	optional RNG seed. All randomness is drawn in R, so <code>set.seed()</code> before the call works equally well. With a seed the caller's RNG state is restored on exit (a seeded call does not disturb the ambient stream); with seed = NULL the draws consume the ambient stream.
interference	crossover interference model (see the section "Crossover interference"). NULL (default): the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$ (AlphaSimR's default). "poisson": Poisson crossovers, no interference, the isgq random stream (the default before 2.0.0.9003). <code>list(nu = , p =)</code> : the two-pathway gamma model with these values; $1 \leq \nu \leq 1e6$ is the interference strength (1 = none), p in $[0, 1]$ (default 0 when omitted) the share of chiasmata that do not interfere. The expected number of crossovers per Morgan is the same under every model.

Details

Each gamete carries on average one crossover per Morgan of a chromosome's length. By default their spacing follows the gamma model of crossover interference ($\nu = 2.6$, $p = 0$, AlphaSimR's default; see the section "Crossover interference"). With `interference = "poisson"` recombination follows the count-location model of `isqg`: the number of crossovers on a chromosome is Poisson with mean equal to its length in Morgans, and their positions are uniform along it. That length is the chromosome's **last** map position (`cm / 100`), not its span `max(cm) - min(cm)`: positions are used as given and are not rebased to a zero origin (the `isqg` convention). The two agree when a chromosome's first marker is at 0; when it is not, the extra crossovers fall upstream of the first marker and only swap the whole chromosome, so the recombination fraction between markers is still Haldane's, $(1 - e^{-2d})/2$. Chromosomes assort independently. The genetic map is taken from the `cm` column of the population's marker map, so it must not be missing - see `synthetic_map()` if you only have physical positions.

Both parents must carry the same marker map (marker names, chromosomes and positions; the allele column is not part of it). The dosages are relative to the allele coded +1, which `as_numeric()` chooses per data set; crossing populations built from **separately** converted panels can therefore mix up the alleles. Convert the panels together, or with `as_numeric(method = "reference", ref_allele = )`. Where both populations record the allele column, a marker whose alleles are listed in opposite order draws a warning and markers with no allele in common an error (see `as_population()`).

Chromosomes are processed, and their random draws consumed, in a canonical order that does not depend on the storage type of `chr` or on the locale (see `as_population()`).

Value

A Population of n progeny.

Crossover interference

By default (`interference = NULL`, with the option below unset) crossovers follow a two-pathway gamma model of interference with $\nu = 2.6$ and $p = 0$: AlphaSimR's default (its `SimParam` fields `v` and `p`), which approximates Kosambi's map function. Give `interference = list( $\nu =$  ,  $p =$  )` for other values, or `interference = "poisson"` for Poisson crossovers (no interference), whose random draws are exactly `isqg`'s. Before version 2.0.0.9003 Poisson was the default; seeded results of every function that runs meiosis changed with the new default (see `NEWS.md`). The gamma model:

- The four-strand bivalent carries chiasmata at an average of 2 per Morgan. A gamete takes part in each chiasma independently with probability 1/2 (no chromatid interference), so it carries one crossover per Morgan on average: **the genetic map (`cm`) keeps its meaning for every ν and p , as under Haldane.**
- A share p of the chiasmata (intensity $2p$ per Morgan) is non-interfering: a Poisson process.
- The rest (intensity $2(1-p)$) is interfering: a stationary renewal process whose gaps are gamma distributed with shape ν and rate $2\nu(1-p)$ per Morgan (mean gap $1/(2(1-p))$ Morgans). $1 \leq \nu \leq 1e6$ is the interference strength (larger values are rejected: the gamma gaps are then numerically a fixed spacing); $\nu = 1$ is exponential gaps, i.e. no interference. The first chiasma is drawn from the stationary (equilibrium) distribution, so the process does not "start" at the chromosome end.

- The two pathways are independent; the gamete's crossovers are the union.

The recombination fraction between two loci d Morgans apart is then $r(d) = [1 - P_0(d)]/2$, with $P_0(d)$ the probability of no chiasma in the interval: $P_0(d) = e^{-2pd}\{1 - F_{\nu+1}(y) - (y/\nu)[1 - F_{\nu}(y)]\}$, $y = 2\nu(1 - p)d$, F_a the Gamma(a , 1) distribution function. With $\nu = 1$, or $p = 1$, the model is Poisson in distribution (Haldane), although it then uses its own random stream, not `isqg`'s. p is in $[0, 1]$; ν must be in $[1, 1e6]$ (negative interference is not modelled). The gamma model of interference is that of McPeck and Speed (1995) and the two-pathway extension that of Housworth and Stahl (2003) (see References; the formulas above were derived and are checked numerically in this package's tests, not copied from either paper or from AlphaSimR). The draws are made in R; the Rust core only applies the drawn crossovers and is unchanged.

To use one interference model for a whole session or scheme without passing `interference =` to every call, set the option `simplePHENOTYPES.interference` to "poisson" (the old default, e.g. to reproduce results of earlier versions) or to a list(`nu =`, `p =`) (NULL, the default, is the gamma model above). An interference argument that is NULL is then replaced by the option, in `cross()`, `selfcross()`, `double_haploid()`, `mate()`, `crossbreed()` and in every function that forwards it (`single_seed_descent()`, `bulk()`, `pedigree()`, `recurrent_selection()`, `cross_usefulness()`, `combining_ability()`, `progeny_test()`), so the precedence is: an explicit non-NULL argument, then the option, then the default gamma model. The option is validated by the same rules as the argument, and an invalid value is an error that names the option. A call overrides the option with an explicit value (`interference = "poisson"` or a list). With the option set, the draws are exactly those made when the same value is passed to every call.

References

- Toledo, F.H., Perez-Rodriguez, P., Crossa, J. and Burgueno, J. (2019). `isqg`: A Binary Framework for in Silico Quantitative Genetics. *G3 Genes|Genomes|Genetics* 9(8), 2425–2428. doi:10.1534/g3.119.400373
- McPeck, M.S. and Speed, T.P. (1995). Modeling interference in genetic recombination. *Genetics* 139(2), 1031–1044. doi:10.1093/genetics/139.2.1031
- Housworth, E.A. and Stahl, F.W. (2003). Crossover interference in humans. *American Journal of Human Genetics* 73(1), 188–197. doi:10.1086/376610

See Also

`selfcross()`, `double_haploid()`, `as_population()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = c("33-16", "38-11"))
f1 <- cross(pop[1], pop[2], n = 5, seed = 1)
f1
```

cross_usefulness	<i>Predict the usefulness of candidate crosses</i>
------------------	--

Description

For each candidate biparental cross, simulates a progeny family and reports its additive genetic mean, genetic standard deviation, and *usefulness* $U = \mu + i \sigma$ (Zhong & Jannink 2007; Lehermeier et al. 2017). A cross with a lower mean but more genetic variance can outrank a safer one, which is the information parent-mean ranking misses.

Usage

```
cross_usefulness(
  sim,
  pairs = NULL,
  scheme = c("dh", "selfcross", "cross"),
  n_progeny = 100L,
  generations = 5L,
  select_top = 0.1,
  trait = 1L,
  direction = c("high", "low"),
  seed = NULL,
  interference = NULL
)
```

Arguments

<code>sim</code>	a realized, Population-backed phenotype_sim whose candidate parents are its individuals and whose additive layer supplies the scoring model.
<code>pairs</code>	a two-column matrix or data frame of parent pairs (ids, or whole positions), at least one row. NULL (default) uses all pairwise combinations of the parents — guard the count yourself for large panels.
<code>scheme</code>	how the progeny family is derived: "dh" (doubled haploids, fully inbred in one step), "selfcross" (self to near-inbred over generations), or "cross" (single-cross progeny; only informative for non-inbred parents).
<code>n_progeny</code>	family size simulated per cross.
<code>generations</code>	selfing generations for scheme = "selfcross".
<code>select_top</code>	fraction of progeny whose intensity <i>i</i> sets the usefulness horizon: one value in (0, 1) (default 0.1; smaller = a more elite target).
<code>trait</code>	trait index to evaluate (default 1).
<code>direction</code>	"high" (default) adds <i>i</i> *sigma; "low" subtracts it.
<code>seed</code>	optional RNG seed for the whole evaluation: one non-negative whole number. The caller's RNG state is restored on exit; with seed = NULL the ambient RNG is used (and advanced).

interference crossover interference model of `cross()` (see its section "Crossover interference"): NULL (default: the option `simplePHENOTYPES.interference` if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the `isqg` stream, the default before 2.0.0.9003) or `list(nu = , p = )`. It is applied to every meiosis of every simulated family (the F1 and each progeny generation), so the family variance, and with it `sd` and `usefulness`, reflects interference.

Details

$U = \mu + i\sigma$ is a **normal-theory criterion**: it equals the expected genetic value of the best `select_top` fraction of progeny only when that family's genetic values are approximately normally distributed, with i the standardized selection intensity for the fraction. For a large polygenic family this holds well; for a small family or one driven by a few large-effect QTNs the distribution is discrete/skewed and U is an approximation that can exceed the realized top-fraction mean (and even the family maximum). Use it as a *ranking* criterion under that assumption; read the realized family from the returned `mean/sd` (or simulate the family directly with `cross()/selfcross()`) if you need the exact tail mean.

The intensity is the infinite-population normal value $i = \varphi(\Phi^{-1}(1 - p))/p$ for `select_top = p` (1.755 at $p = 0.1$); the mean of the top p of a *finite* family is slightly smaller (about 1.73 for the top 10 of 100), and a family with `n_progeny * select_top < 1` selects less than one progeny, so U is then only a normal-theory ranking score. Prefer `n_progeny * select_top >= 1`, in practice a few tens of progeny at least.

Heterozygous parents and "dh" / "selfcross". These schemes derive each family from a *single* F1 individual of the cross (one draw of the two parental gametes), then double or self it. For inbred (homozygous) parents that F1 is unique, so the family-generating distribution is that of the cross itself (no conditioning on a random F1). The returned `mean` and `sd` are nevertheless Monte Carlo estimates from `n_progeny` simulated progeny (meiosis and, for "selfcross", the selfing draws remain random), so they still vary with seed and converge to the cross's moments only as `n_progeny` grows. For a **heterozygous** parent the F1 is also one random draw, so the reported `mean` and `sd` are additionally conditional on which F1 was drawn: they change with seed for that reason as well, and the standard deviation `sd` (its square, the family variance, likewise) is on average lower than the F1-averaged (many-F1) value. The function warns once per call when a parent in pairs is heterozygous at a scoring locus under these schemes. Use inbred parents, or `scheme = "cross"` (which draws every progeny from an independent meiosis of the parents), when the parents are not inbred; the sampling scheme itself is unchanged.

Progeny are scored on the additive QTN effects already realized in `sim` (the template), applied on a common scale, so the means and standard deviations are directly comparable between crosses. Only the additive component enters the score (the breeding-value basis of the usefulness criterion); dominance and epistasis layers are ignored, and doubled-haploid / inbred families carry no dominance anyway.

Value

a data frame with `parent1`, `parent2`, `mean`, `sd`, `usefulness`, sorted best first, carrying attribute `intensity` (the i used).

References

Zhong S, Jannink J-L (2007) Using quantitative trait loci results to discriminate among crosses on the basis of their progeny mean and variance. *Genetics* 177(1):567–576. doi:10.1534/genetics.107.075358; Lehermeier C, Teyssedre S, Schon C-C (2017) Genetic gain increases by applying the usefulness criterion with improved variance prediction in selection of crosses. *Genetics* 207(4):1651–1661. doi:10.1534/genetics.117.300403

See Also

[cross\(\)](#), [single_seed_descent\(\)](#), [select_ind\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:6)
sim <- simulate_phenotype(pop, h2 = 0.5, seed = 1) |> additive(n_qtn = 30)
u <- cross_usefulness(sim, scheme = "dh", n_progeny = 30, seed = 2)
head(u) # crosses ranked by expected best-progeny value
```

crossbreed

Crossbreeding systems

Description

Runs a classical crossbreeding system on pure-breed Populations and returns the final crossbred generation, its pedigree traced to the breeds (see [breed_composition\(\)](#)). Each generation mates `n_progeny` random dam-sire pairs (one progeny each) with [mate\(\)](#); sex is not modelled, so "dam" and "sire" are the roles of the two pools in the plan, not sexes.

Usage

```
crossbreed(
  breeds,
  system = c("two_way", "backcross", "three_way", "terminal", "rotational"),
  n_progeny,
  generations = 1L,
  sire_breed = NULL,
  seed = NULL,
  interference = NULL
)
```

Arguments

<code>breeds</code>	a named list of pure-breed Populations (founder pools named as the list).
<code>system</code>	one of the systems above.
<code>n_progeny</code>	progeny per generation.
<code>generations</code>	for "rotational": generations after the F1.

sire_breed	for "terminal": the name of the terminal sire breed (default the last breed).
seed	optional RNG seed; the caller's RNG state is restored on exit.
interference	crossover interference model of <code>cross()</code> (see its section "Crossover interference"): NULL (default: the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the <code>isqg</code> stream, the default before 2.0.0.9003) or <code>list(nu = , p =)</code> . Applied to every generation.

Details

- "two_way": breed 1 dams x breed 2 sires (an F1).
- "backcross": F1 dams x breed-1 sires.
- "three_way": F1 (breed 1 x breed 2) dams x breed-3 sires.
- "terminal": F1 (breed 1 x breed 2) dams x sires of `sire_breed` (a breed not in the F1); the products are not bred further.
- "rotational": F1 dams, then each generation's crossbred dams x sires of the next breed in rotation (breed 1, 2, ... then back to 1), for generations after the F1. The expected composition converges (for two breeds, toward alternating 2/3 : 1/3).

Value

The final generation as a Population, with attribute history: a data frame with one row per generation (generation, `sire_breed` and the mean expected breed fraction per breed).

See Also

`breed_composition()`, `heterosis()`, `mate()`

Examples

```
g <- data.frame(snp = paste0("m", 1:10), allele = "A/G", chr = 1,
               pos = 1:10, cm = seq(0, 90, by = 10))
mk <- function(pool, v) as_population(cbind(g, matrix(v, 10, 4,
  dimnames = list(NULL, paste0(pool, 1:4))))), pool = pool)
br <- list(A = mk("A", 1L), B = mk("B", -1L))
rot <- crossbreed(br, "rotational", n_progeny = 20, generations = 6, seed = 1)
attr(rot, "history")
```

dominance

Add a dominance variance component

Description

Add a dominance variance component

Usage

```
dominance(
  sim,
  prop = NULL,
  same_as_add = TRUE,
  n_qtn = NULL,
  qtn = NULL,
  dist = "geometric",
  effect = NULL
)
```

Arguments

sim	a <code>phenotype_sim</code> .
prop	proportion of total phenotypic variance (scalar or length <code>n_traits</code>). May be omitted when <code>h2</code> was set in <code>simulate_phenotype()</code> , in which case the layer takes whatever of that budget is unspent – so a single additive() layer gets <code>prop = h2</code> . When <code>prop</code> is given and <code>h2</code> is set, the layer <code>prop</code> values must sum to <code>h2</code> .
same_as_add	reuse the additive layer's QTNs (default <code>TRUE</code>). Supplying <code>qtn</code> fixes the loci instead, so <code>same_as_add</code> is then treated as <code>FALSE</code> (the layer records and prints it as such).
n_qtn	number of dominance QTNs for a fresh draw (<code>same_as_add = FALSE</code>); ignored, with a warning, when the loci are reused (<code>same_as_add = TRUE</code>) or fixed by <code>qtn</code> .
qtn	optional user-supplied QTNs for this layer, so you can fix the causal loci for one effect type while the others are drawn at random. Give marker names (matched against the map) or column indices; a vector is used for every trait, or a length- <code>n_traits</code> list sets each trait's loci separately (every element the same length). <code>n_qtn</code> is then taken from what you supply. Fixed loci are not redrawn by <code>vary_qtn</code> . (For <code>epistasis()</code> , <code>qtn</code> describes interacting sets; see there.) Every architecture accepts <code>qtn</code> , and each keeps its construction: under "independent" the loci carry the effect series; under "pleiotropy" (multi-trait) every locus affects every trait, so give a single vector (or the same loci for each trait in a list); loci that affect only some traits are partial pleiotropy, built with <code>complex_phenotypes()</code> . If a correlation is controlled (<code>cor</code> , <code>pi</code> , ...) the correlated draw sets the effects and the genetic correlation is its target; otherwise the correlation is just an outcome of the shared loci, and an explicit effect sets the effects (without it the default draw does, with implicit <code>cor = 0</code>). <code>pi</code> only matters for controlling a correlation, and <code>pi < 1</code> cannot be used with fixed shared loci. Under "ld" (<code>ld_type = "direct"</code> only) give <code>qtn = list(trait1_loci, trait2_loci)</code> of equal length – the <i>i</i> -th locus of each is a linked pair on one chromosome, no marker may be causal for both traits, and each pair's <code>r2</code> is reported (by <code>qtn_table()</code>) and warned about when outside <code>[r2_min, r2_max]</code> . A marker that is monomorphic (or heterozygous in every individual) carries no variance: it is accepted with a warning, except where it makes the architecture undefined (an "ld" pair, or a pleiotropy layout with no varying locus), which is an error.

dist	within-layer effect distribution (default "geometric").
effect	optional geometric base (scalar) or explicit effect series (length <code>n_qtn</code> , the number of dominance loci – the additive layer's <code>n_qtn</code> when <code>same_as_add = TRUE</code>), used for every trait; or a length- <code>n_traits</code> list of these, one per trait (the counterpart of <code>v1 dom_effect</code>). Under <code>architecture = "pleiotropy"</code> (multi-trait) accepted only when no correlation is controlled (no <code>cor</code> , <code>pi</code> , ...), as in additive() .

Details

Dominance is modelled as a deviation applied to heterozygotes (the `het` indicator), with its share of phenotypic variance set by `prop`. In this variance-partition grammar there is no separate "degree of dominance" argument (there is no `degree` =): the ratio of the *scaled components*' variances is `prop_dominance / prop_additive`, set through the layer proportions. (A single degree-of-dominance scalar would be washed out by the per-component variance scaling and is therefore not offered; for a per-locus degree of dominance `d / abs(a)` use `additive(orthogonal = TRUE, a =, d =)`.)

Additive + dominance on the same loci is not orthogonal. The dosage and the heterozygote indicator covary by $-(2p-1)2pq$ at each locus, so the realized genetic variance is `prop_A + prop_D + 2Cov(c_A, c_D)` for one additive and one dominance layer (several layers of a type add their mutual covariance to `Var(c_A)` or `Var(c_D)`) – a structural, allele-coding-dependent bias, not a finite-sample effect; its per-locus sign is that of $1 - 2p$ for the counted-allele frequency `p`, so it is one-signed across loci only when those frequencies are on one side of 0.5 – and the classical `Va:Vd` of the block differs from `prop_A:prop_D`. The object reports the realized `Var(A)`, `Var(D)` and `2Cov(A,D)`, and `Var(c_A)`, `Var(c_D)` and `2Cov(c_A,c_D)` (`$ad_report`, and the note printed by `print()`); use `additive(orthogonal = TRUE, a =, d =)` for a Fisher-orthogonal partition.

Dominance needs heterozygotes to be identifiable: it is identically zero at a locus with no heterozygous individuals. If the selected loci carry none (common on a near-inbred panel such as the bundled maize lines, ~0.4% heterozygous), `dominance()` **errors** rather than silently substituting loci; if only some of them carry none, it warns that the rest absorb `prop`. Pre-filter to heterozygous markers with `filter_geno(hets = "include")`, fix het-bearing loci with `qtn =`, or simulate dominance on an outbred or F2-type population instead.

Under `architecture = "pleiotropy"` the dominance effects are drawn with the same covariance as the additive layer (DECISION-023): shared loci carry correlated effects and trait-specific loci independent ones, split by `pi`, so the dominance component targets `cor` – its realized correlation converges to `cor` as the loci and individuals grow, for loci in approximate linkage equilibrium; see [simulate_phenotype\(\)](#) `cor` – rather than every trait getting one identical effect series. With `same_as_add = TRUE` the additive layer's shared and trait-specific loci are reused. Each locus's effect is scaled by the realized standard deviation of its heterozygote indicator. Fixing loci with `qtn =` works under every architecture: under "pleiotropy" every locus affects every trait (the same loci for each trait), with an explicit effect setting the effects when no correlation is controlled; under "ld" dominance reuses the additive layer's linked loci (`same_as_add = TRUE`) or takes `qtn = list(trait1_loci, trait2_loci)` of disjoint linked pairs (no marker may already be causal for the other trait in an earlier layer), since a fresh random draw could make a marker causal for both traits across layers.

Value

the updated `phenotype_sim`.

See Also

`additive()`, `epistasis()`, `vqtl()`, `filter_geno()`.

Examples

```
data("SNP55K_maize282_maf04")

# Dominance reuses the additive QTNs by default, so the trait has
# h2 = 0.4 + 0.1 = 0.5 spread over one set of loci.
ph <- simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.4, n_qtn = 5) |>
  dominance(prop = 0.1)
ph

# same_as_add = FALSE draws separate dominance-only loci instead.
simulate_phenotype(SNP55K_maize282_maf04, seed = 2) |>
  additive(prop = 0.4, n_qtn = 5) |>
  dominance(prop = 0.1, same_as_add = FALSE, n_qtn = 3)

# Give the dominance effects yourself: a geometric base, or one value per QTN.
simulate_phenotype(SNP55K_maize282_maf04, seed = 2) |>
  additive(prop = 0.4, n_qtn = 5) |>
  dominance(prop = 0.1, same_as_add = FALSE, n_qtn = 3,
    effect = c(0.6, 0.3, 0.1))
```

<code>dosages</code>	<i>Dosage matrix of a Population</i>
----------------------	--------------------------------------

Description

Dosage matrix of a Population

Usage

`dosages(x)`

Arguments

`x` a Population.

Value

An integer matrix of markers by individuals, coded -1/0/1.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:3)
dim(dosages(pop))
```

double_haploid	<i>Produce doubled haploids from an individual</i>
----------------	--

Description

Produces n doubled-haploid progeny: a single recombinant gamete is drawn and then doubled, so every individual is completely homozygous at every marker. This is the one-generation route to a fully inbred line, and the result contains no heterozygotes at all.

Usage

```
double_haploid(parent, n = 1, seed = NULL, interference = NULL)
```

Arguments

parent	a single-individual Population.
n	number of progeny.
seed	optional RNG seed. All randomness is drawn in R, so <code>set.seed()</code> before the call works equally well. With a seed the caller's RNG state is restored on exit (a seeded call does not disturb the ambient stream); with <code>seed = NULL</code> the draws consume the ambient stream.
interference	crossover interference model (see the section "Crossover interference"). <code>NULL</code> (default): the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$ (AlphaSimR's default). "poisson": Poisson crossovers, no interference, the isqg random stream (the default before 2.0.0.9003). <code>list(nu = , p =)</code> : the two-pathway gamma model with these values; $1 \leq \nu \leq 1e6$ is the interference strength (1 = none), p in $[0, 1]$ (default 0 when omitted) the share of chiasmata that do not interfere. The expected number of crossovers per Morgan is the same under every model.

Value

A Population of n fully homozygous progeny.

Many doubled-haploid families in one call

`double_haploid()` takes one parent. To make doubled haploids from many parents (or many families from one) call `mate()` with a plan whose rows have `design = "dh"`, e.g. `mate(data.frame(mother = ids, father = ids, n = 100, design = "dh"), pop)`: all rows are executed in one pass through the Rust core, with the same random draws, in the same order, and the same progeny and pedigree as calling `double_haploid()` on each parent in turn under one seed, at a fraction of the cost per call (see `mate()`).

References

Toledo, F.H., Perez-Rodriguez, P., Crossa, J. and Burgueno, J. (2019). isqg: A Binary Framework for in Silico Quantitative Genetics. *G3 Genes|Genomes|Genetics* 9(8), 2425–2428. doi:10.1534/g3.119.400373

See Also

`cross()`, `selfcross()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = c("33-16", "38-11"))
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
dh <- double_haploid(f1, n = 10, seed = 3)
# No heterozygotes by construction:
any(dosages(dh) == 0)
```

epistasis

Add an epistatic variance component

Description

Add an epistatic variance component

Usage

```
epistasis(
  sim,
  prop = NULL,
  n_pairs = NULL,
  interaction = 2,
  interaction_type = "a",
  qtn = NULL,
  effect = NULL,
  dist = "geometric"
)
```

Arguments

<code>sim</code>	a <code>phenotype_sim</code> .
<code>prop</code>	proportion of total phenotypic variance (scalar or length <code>n_traits</code>). May be omitted when <code>h2</code> was set in <code>simulate_phenotype()</code> , in which case the layer takes whatever of that budget is unspent – so a single <code>additive()</code> layer gets <code>prop = h2</code> . When <code>prop</code> is given and <code>h2</code> is set, the layer <code>prop</code> values must sum to <code>h2</code> .
<code>n_pairs</code>	number of interacting QTN sets.
<code>interaction</code>	number of markers per epistatic QTN (default 2, pairwise).
<code>interaction_type</code>	how each marker in an interacting set contributes: "a" (additive – the centered dosage) or "d" (dominance – the centered heterozygote indicator). A single value applies to every position, or a length- <code>interaction</code> vector sets each position: <code>c("a", "a")</code> is additive-by-additive (default), <code>c("a", "d")</code> additive-by-dominance, <code>c("d", "d")</code> dominance-by-dominance.

qtn	optional user-supplied interacting sets (marker names or column indices): a matrix with one row per set and interaction columns, or a vector whose length is a multiple of interaction (filled by row). A matrix or vector is used for every trait. A list must have one element per trait (as in <code>additive()</code>), each element that trait's sets (a matrix or vector as above), and every trait must get the same number of sets – a list of length <code>n_traits</code> is never read as "one set per element". Under <code>architecture = "pleiotropy"</code> every set affects every trait (the same sets for each trait; sets match as ordered tuples; sets for only some traits are partial pleiotropy, <code>complex_phenotypes()</code>). Not available under <code>architecture = "ld"</code> .
effect	optional geometric base (scalar) or explicit effect series (length <code>n_pairs</code> , one value per interacting set), used for every trait; or a length- <code>n_traits</code> list of these, one per trait (the counterpart of <code>v1 epi_effect</code>). Under <code>architecture = "pleiotropy"</code> (multi-trait) accepted only when no correlation is controlled (no <code>cor</code> , <code>pi</code> , ...), as in <code>additive()</code> .
dist	within-layer effect distribution (default "geometric").

Details

The epistatic value is the product of the interacting markers' design terms times a per-set effect, then centered and scaled to prop. Each design term is centered per locus before multiplying, which removes each locus's mean so the product is not dominated by a constant offset. This does **not** orthogonalize the interaction against the additive/dominance main effects: the centered product can still be correlated with the constituent dosages – strongly so when the interacting loci are in LD or away from allele frequency 0.5 (with two perfectly linked loci it can be collinear with each additive term). It is therefore not a Fisher/NOIA-orthogonal variance component (that is a planned genotypic-model mode), so the "epistatic proportion" is a simulation quantity, not an orthogonal share of variance.

`interaction_type` values of "d" depend on heterozygotes and are near-degenerate on a fully inbred panel (see `dominance()`); use "a" (the default) for inbred data, or an outbred/F2-type population for "d".

Under `architecture = "pleiotropy"` the interacting sets are split by `pi` into shared sets (common to every trait, with correlated effects) and trait-specific sets (independent effects), drawn with the same covariance as the additive layer, so the epistatic component targets `cor` (its realized correlation converges as the sets and individuals grow, for loci in approximate linkage equilibrium; DECISION-023). Each set's effect is scaled by the realized standard deviation of its centered-product design column; `effect =` is rejected there while a correlation is controlled (without `cor / pi` it sets the effects, and fixed `qtn = sets` are accepted). `epistasis()` is not supported under `architecture = "ld"`: that architecture makes each trait's causal loci distinct markers in LD within the `r2` window, and epistatic sets have no such linked construction – they would be unlinked and could reuse a marker another layer made causal for the other trait.

Value

the updated `phenotype_sim`.

See Also

[additive\(\)](#), [dominance\(\)](#), [vqtl\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")

# Two pairs of interacting loci contributing 10% of the variance,
# on top of an additive background.
ph <- simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.4, n_qtn = 5) |>
  epistasis(prop = 0.1, n_pairs = 2)
ph

# interaction = 3 makes each epistatic term a three-way interaction
# rather than the default pairwise one.
simulate_phenotype(SNP55K_maize282_maf04, seed = 3) |>
  additive(prop = 0.3, n_qtn = 4) |>
  epistasis(prop = 0.2, n_pairs = 2, interaction = 3)

# Additive-by-dominance and dominance-by-dominance interactions depend on
# heterozygotes, so run them on a segregating F2 (the maize282 panel is
# inbred and has none).
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
f2 <- selfcross(cross(pop[1], pop[2], n = 1, seed = 1), n = 40, seed = 2)
simulate_phenotype(f2, seed = 4) |>
  additive(prop = 0.3, n_qtn = 4) |>
  epistasis(prop = 0.1, n_pairs = 2, interaction_type = c("a", "d")) |>
  epistasis(prop = 0.1, n_pairs = 2, interaction_type = c("d", "d"))
```

families

Family groupings from the recorded pedigree

Description

Groups a Population's individuals by shared parents, for [select_ind\(\)](#)'s `family` = argument (method = "within_family", "among_family" or "combined") or for progeny testing.

Usage

```
families(
  x,
  by = c("full_sib", "maternal_half_sib", "paternal_half_sib", "selfed")
)
```

Arguments

x a `Population`.

by "full_sib" (same two parents, in either order), "maternal_half_sib" (same mother), "paternal_half_sib" (same father) or "selfed" (progeny of the same selfed or doubled-haploid parent).

Value

A factor with one element per individual of `x`, named by `id`; levels are named by the parent `id(s)`. Founders, and for "selfed" any individual that is not a self or doubled haploid, are `NA`.

See Also

`parentage()`, `select_ind()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:3)
fs <- c(cross(pop[1], pop[2], n = 4, seed = 1),
        cross(pop[1], pop[3], n = 4, seed = 2))
families(fs, "full_sib")
families(fs, "maternal_half_sib")
```

filter_genotype

Filter a genotype object by MAF, heterozygosity and LD

Description

One place to apply every marker filter `simplePHENOTYPES` implements, so a genotype can be trimmed once and then reused across simulations. It takes the `simplePHENOTYPES` numeric format (a data frame whose first five columns are `snp`, `allele`, `chr`, `pos`, `cm` (and optionally the counted column of `as_numeric(counted_column = TRUE)`), followed by one column per individual, e.g. `SNP55K_maize282_maf04`), an individuals-by-markers numeric matrix coded -1/0/1, or a `Population` (founders or the progeny of `cross()`, `selfcross()` and `double_haploid()`), and returns the same kind of object with the failing markers dropped. Every genotype input that `simulate_phenotype()` accepts can therefore be filtered, so the same MAF, heterozygosity and LD constraints apply whatever the architecture or layer (additive, dominance, epistasis, vQTL) drawn from it afterwards.

Usage

```
filter_genotype(
  geno,
  maf_above = NULL,
  maf_below = NULL,
  hets = c("any", "include", "remove"),
```

```

remove_monomorphic = TRUE,
indep_pairwise = NULL,
indep_pairphase = NULL,
indep = NULL,
blocks = FALSE,
block_max_kb = 500,
window_unit = c("variants", "kb"),
code_as = c("-101", "012"),
verbose = TRUE
)

```

Arguments

geno	the genotype object: a numeric-format data frame, an individuals-by-markers $-1/0/1$ matrix, or a Population. Convert other formats with <code>as_numeric()</code> first. A Population is filtered on its dosages (always $-1/0/1$, so <code>code_as</code> must stay <code>"-101"</code>); its genetic map, individuals, ids and pedigree are kept, only the failing markers are dropped. The result has a different marker map from the input, so it can no longer be crossed with, or pooled by <code>c.Population()</code> with, an unfiltered population: filter the founders once, before crossing.
maf_above	keep markers with minor-allele frequency $\geq$ maf_above. A single finite number in $[0, 0.5]$ (a vector or NA is an error).
maf_below	keep markers with minor-allele frequency $\leq$ maf_below. A single finite number in $[0, 0.5]$, and not smaller than maf_above.
hets	"any" (no heterozygosity filter, default), "include" (keep only markers that have at least one heterozygote – needed for a dominance layer on a near-inbred panel), or "remove" (keep only markers with no heterozygotes).
remove_monomorphic	drop markers with no variation (default TRUE); must be TRUE or FALSE.
indep_pairwise	optional <code>c(window, step, r2)</code> for composite- r^2 pairwise pruning (see the LD section). NULL skips it. window and step are positive whole numbers of markers (window may be fractional in kilobases when <code>window_unit = "kb"</code>), <code>r2</code> is in $(0, 1]$; the same holds for <code>indep_pairphase</code> , and <code>indep</code> takes a vif of at least 1.
indep_pairphase	optional <code>c(window, step, r2)</code> for haplotypic- r^2 pairwise pruning (EM-phased; see the LD section). NULL skips it.
indep	optional <code>c(window, step, vif)</code> for variance-inflation-factor pruning (PLINK's <code>--indep</code> ; see the LD section). NULL skips it.
blocks	TRUE to define Gabriel et al. (2002) haplotype blocks and keep one tag marker per block (default FALSE); must be TRUE or FALSE.
block_max_kb	maximum block span in kilobases for blocks (default 500).
window_unit	"variants" (default) or "kb" – the unit of the pruning window. "kb" needs marker positions: the numeric-format data frame or a Population (their pos column), not a bare matrix.

code_as	the genotype coding of geno: "-101" (default; major = 1, het = 0, minor = -1) or "012" (major = 2, het = 1, minor = 0). This is declared rather than guessed, because a marker with only 0/1 present is ambiguous between the two schemes; values outside the declared set are an error. It sets how minor-allele frequency and heterozygosity are computed.
verbose	report how many markers each filter removed (default TRUE).

Details

Filters are applied in order: monomorphic removal, minor-allele frequency, heterozygosity, then LD pruning. A marker must pass every requested filter to be kept. There is no call-rate filter (PLINK's `--geno` / `--mind`) and no Hardy-Weinberg test: a marker with no called genotype is kept unless `remove_monomorphic = TRUE` (its minor-allele frequency is then 0), so drop or impute such markers beforehand if they matter.

Value

the filtered genotype object, in the same format as `geno` (a `Population` stays a `Population`).

LD pruning and haplotype blocks

The three sliding-window pruners take the same three-number specification used by PLINK (`c(window, step, threshold)`); `window_unit` sets whether window counts markers (default) or kilobases ("`kb`", using the `pos` column). `indep_pairwise`, `indep` and `indep_pairphase` reproduce PLINK 1.9's `--indep-pairwise`, `--indep` and `--indep-pairphase` **marker-for-marker for ordinary use**: on complete-call **autosomal** genotypes, with `step <= window`, the kept marker set (and hence the removed set) matches PLINK 1.9. The committed evidence is a set of golden PLINK v1.9.0-b.8 results on the bundled `SNP55K_maize282_maf04` panel (nine prune configurations genome-wide, plus `--blocks no-pheno-req` at 200 and 500 kb), compared in the same marker order by `tests/testthat/test-filter-genotype-plink-parity.R`, together with hard-coded threshold-epsilon and monomorphic corner cases; there is no PLINK run on other panels or on randomly generated genotypes. This holds for variant windows (all three) and kb windows (`indep_pairwise`); `blocks = TRUE` likewise reproduces the block definitions of PLINK's `--blocks no-pheno-req` at a matching `--blocks-max-kb` (simplePHENOTYPES imposes no phenotype requirement, and its `block_max_kb` defaults to 500, versus PLINK's 200-kb default – so a bare PLINK `--blocks` will differ unless run with `no-pheno-req` and the same span). All loci are treated as autosomal diploid: simplePHENOTYPES has no sex chromosome, so PLINK's haploid chromosome-X weighting is neither modelled nor matched.

Beyond that envelope the match is best-effort, not guaranteed, in a few pathological corners that do not arise in normal genotype panels: non-overlapping windows (`step > window`) combined with markers pruned mid-window; `indep` (VIF) when markers outnumber individuals, so the marker correlation matrix is rank-deficient and pruning turns on PLINK's exact matrix-inversion tie handling; `indep` when a pair is near-perfectly *negatively* correlated in PLINK's minor-allele coding (the VIF collinearity scan compares $|r|$, which matches every real panel tested but not that hand-built sign case); `indep_pairphase` at a threshold within floating-point tolerance of a realised haplotypic r^2 (that r^2 comes from a transcendental cubic solver – `acos/cos/log` – so, unlike the integer-exact composite r^2 , it can differ from PLINK's in the last bit); and `kb + indep`. PLINK computes LD from *founders only*; `filter_genotype()` does not apply that rule: it computes LD from **every individual it is given**, including the non-founders of a crossing `Population` (its pedigree is

not consulted; the parity is verified on founder-only data frames) – pass a founder-only genotype set, e.g. the founders before crossing, to reproduce a PLINK run that had non-founders. Missing calls (NA) are handled per pair on the complete observations (a pairwise.complete.obs correlation for indep_pairwise/indep, complete pairs only for indep_pairphase) and are **not** claimed identical to PLINK's missing-data path; drop or impute missing genotypes first for PLINK parity. Variant windows (window_unit = "variants") tolerate missing chr/pos (for example genotypes converted from a nucleotide table): markers with a missing chr form one group, ordered as supplied. window_unit = "kb" and blocks = TRUE need a physical map, so they stop with an error when chr or pos is missing for any marker.

indep_pairwise = c(window, step, r2) PLINK --indep-pairwise: drop one marker of every pair whose composite genotype r^2 (squared correlation of the $-1/0/1$ dosage) exceeds r2, removing the **lower-MAF** marker of the pair (ties keep the earlier one), e.g. c(50, 5, 0.2).

indep_pairphase = c(window, step, r2) PLINK --indep-pairphase: like indep_pairwise, but r^2 is the *haplotypic* (phased) r^2 of Hill & Robertson (1968), with the double-heterozygote phase resolved per pair by PLINK's exact two-locus maximum-likelihood solver, rather than the composite genotype r^2 , e.g. c(50, 5, 0.2).

indep = c(window, step, vif) PLINK --indep: multi-marker pruning by variance inflation factor – within each window the largest-VIF marker (VIF = the diagonal of the inverse marker correlation matrix) is dropped while any VIF exceeds vif, e.g. c(50, 5, 2) (Purcell et al. 2007).

blocks = TRUE PLINK --blocks: define haplotype blocks by the method of Gabriel et al. (2002) – markers are grouped where the D' confidence intervals show strong LD and few recombination-consistent pairs – then keep one tag marker (highest MAF) per block. The block definitions reproduce PLINK 1.9's --blocks no-pheno-req (Haploview algorithm) block-for-block on the MAF ≥ 0.05 markers Haploview considers, at a matching span; block_max_kb (default 500) is PLINK's --blocks-max-kb (default 200). The tag-per-block collapse is a simple-PHENOTYPES convenience on top of PLINK's block definitions.

The VIF pruner emits a one-per-session citation for PLINK (Purcell et al. 2007), and the block method one for Gabriel et al. (2002); the two pairwise pruners need no separate citation.

References

- Purcell, S. *et al.* (2007). PLINK: a tool set for whole-genome association and population-based linkage analyses. *Am. J. Hum. Genet.* 81, 559–575. doi:10.1086/519795 (the VIF pruner).
 Gabriel, S.B. *et al.* (2002). The structure of haplotype blocks in the human genome. *Science* 296, 2225–2229. doi:10.1126/science.1069424 (the haplotype-block definition).
 Hill, W.G. and Robertson, A. (1968). Linkage disequilibrium in finite populations. *Theor. Appl. Genet.* 38, 226–231. doi:10.1007/BF01245622 (the phased r^2 used by indep_pairphase).
 Gaunt, T.R., Rodriguez, S. and Day, I.N.M. (2007). Cubic exact solutions for the estimation of pairwise haplotype frequencies. *BMC Bioinformatics* 8, 428. doi:10.1186/147121058428 (the exact cubic solution for two-locus haplotype frequencies behind the phased r^2 of indep_pairphase).

See Also

[as_numeric\(\)](#), [simulate_phenotype\(\)](#), [dominance\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")

# Common-variant, heterozygote-bearing markers (e.g. before a dominance model)
g <- filter_geno(SNP55K_maize282_maf04, maf_above = 0.1, hets = "include")

# Pairwise r^2 pruning in 50-marker windows
p <- filter_geno(SNP55K_maize282_maf04, indep_pairwise = c(50, 5, 0.2))
```

founders_coalescent	<i>Founder haplotypes from a coalescent with recombination</i>
---------------------	--

Description

Simulates founders with historical linkage disequilibrium and mutation-drift allele frequencies, as AlphaSimR's `runMacs()` does with MaCS, natively and reproducibly. Each chromosome is walked with a sequential Markov coalescent (SMC', Marjoram & Wall 2006) – the model AlphaSimR runs, since it calls MaCS with a one-base history window – under a piecewise-constant population-size history, with infinite-sites mutation. The haplotypes become a [Population](#) ready for `cross()`, `selfcross()`, `double_haploid()` and the selection engine.

Usage

```
founders_coalescent(
  n_ind,
  n_chr = 1,
  seg_sites = NULL,
  inbred = FALSE,
  species = c("GENERIC", "MAIZE", "WHEAT", "CATTLE"),
  split = NULL,
  theta = NULL,
  rho = NULL,
  history = NULL,
  morgans = NULL,
  bp = NULL,
  pool = NA_character_,
  seed = NULL
)
```

Arguments

<code>n_ind</code>	number of founder individuals (diploid).
<code>n_chr</code>	number of chromosomes.
<code>seg_sites</code>	segregating sites kept per chromosome (one value or one per chromosome), a uniform random subset of those simulated; NULL keeps them all. Asking for more sites than the coalescent produced is an error (as in <code>runMacs()</code>).

inbred	TRUE for fully inbred founders: each individual carries one simulated haplotype twice (n_ind haplotypes are simulated, not $2 \times n_ind$).
species	one of "GENERIC", "MAIZE", "WHEAT", "CATTLE".
split	optional number of generations ago at which the founders' two halves split into isolated subpopulations (no migration): individuals $1..n_ind/2$ and the rest. Converted to coalescent time as $split / (4 N_e) + 1e-6$, as <code>runMacs()</code> does. Needs an even n_ind , so that both haplotypes of every individual come from one subpopulation.
theta, rho	optional per-chromosome $4N_0\mu$ and $4N_0r$, overriding the preset.
history	optional data frame with numeric columns time ($4 N_0$ generations, increasing, in $(0, 1e12]$) and size (relative to N_0 , in $[1e-9, 1e9]$); NULL keeps the preset's.
morgans	optional genetic length of each chromosome in Morgans (one value or one per chromosome).
bp	optional physical length of each chromosome in base pairs, used for the pos column of the map.
pool	founder pool label passed to <code>population_from_haplotypes()</code> .
seed	optional seed (NULL or a whole number): with it the founders are reproducible and the caller's RNG state is restored; without it they follow the ambient stream (<code>set.seed()</code> also reproduces them).

Value

a Population of n_ind founders. Allele 1 (the counted allele) is the derived allele. Its attribute "coalescent" records the parameters, the per-chromosome seeds and the number of segregating sites simulated before subsetting (n_total).

Model and units

As in `ms` and `MaCS`, time is in units of $4N_0$ generations, $\theta = 4N_0\mu$ and $\rho = 4N_0r$ are per chromosome, and `history` gives the population size relative to N_0 from each time back ($ms - eN$). A pair of lineages coalesces at rate $2/\lambda(u)$; along the chromosome, mutations and recombination points arrive at rates θL and ρL for a marginal tree of total length L . The genetic map is linear in physical position, morgans long, and starts at 0 cM at the first segregating site (as `runMacs()` rebases it).

Species presets

`species` reproduces the demographic commands `runMacs()` builds in AlphaSimR 2.1.0 (read from its source; parameter values chosen by AlphaSimR, not estimates made here):

GENERIC 10^8 bp, $\theta = 1000$, $\rho = 400$, 1 Morgan, $N_e = 100$, sizes 5 / 15 / 60 / 120 / 1000 from times 0.25 / 2.5 / 25 / 250 / 2500.

MAIZE 2×10^8 bp, $\theta = 1000$, $\rho = 800$, 2 Morgans, $N_e = 100$, a 15-step history.

WHEAT 8×10^8 bp, $\theta = 320$, $\rho = 288$, 1.43 Morgans, $N_e = 50$, a 26-step history.

CATTLE $2.8 \times 10^9/30$ bp per chromosome, mutation rate 9.4×10^{-9} and recombination rate 9.26×10^{-9} per bp, $N_e = 90$, a 12-step history in generations.

theta, rho, history, morgans and bp override the preset (the preset's N_e still converts split to coalescent time).

References

Marjoram P, Wall JD (2006) Fast "coalescent" simulation. *BMC Genet* 7:16 (PMID 16539698).
 Chen GK, Marjoram P, Wall JD (2009) Fast and flexible simulation of DNA sequence data. *Genome Res* 19:136-142 (PMID 19029539).
 Hudson RR (2002) Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics* 18:337-338 (PMID 11847089).
 Gaynor RC, Gorjanc G, Hickey JM (2021) AlphaSimR: an R package for breeding program simulations. *G3* 11(2):jkaa017. doi:10.1093/g3journal/jkaa017 (the runMacs() presets).

See Also

[population_from_haplotypes\(\)](#), [as_population\(\)](#), [cross\(\)](#).

Examples

```
# 50 founders, 2 chromosomes, 200 segregating sites each (GENERIC preset)
f <- founders_coalescent(50, n_chr = 2, seg_sites = 200, seed = 1)
f
# inbred lines, e.g. for doubled-haploid programs
lines <- founders_coalescent(20, seg_sites = 100, inbred = TRUE, seed = 2)
all(dosages(lines) != 0)
```

g_matrix

Genomic relationship matrix (VanRaden 2008)

Description

Builds the additive genomic relationship matrix $G = ZZ' / (2 \sum p_j(1 - p_j))$ from marker dosages, where $Z = M - 2p$ centres the 0/1/2 genotype matrix by twice the allele frequency (VanRaden 2008, method 1).

Usage

```
g_matrix(x, ridge = 0, base_freq = NULL)
```

Arguments

x	a Population, a Population-backed phenotype_sim, or a marker x individual dosage matrix coded -1/0/1 with individual column names.
ridge	optional blend toward the identity, $G^* = (1 - \text{ridge}) G + \text{ridge} I$, for a positive-definite matrix when one is needed (default 0).
base_freq	optional fixed base allele frequencies, one per marker (in the marker order of x), each in [0, 1] and matching the allele the 0/1/2 coding counts (dosage 2 = homozygote for it). When supplied, G uses this fixed base instead of the current-sample frequencies, and only markers with base_freq strictly in (0, 1) are kept. Default NULL (current-sample base).

Details

By default p is the allele frequency of the current individuals (`base_freq = NULL`), and markers monomorphic *in this set* are dropped. Then G is a moving-base matrix: relatedness and the diagonal are expressed relative to this sample, so $F_i = G_{ii} - 1$ is inbreeding relative to the current set, not to a fixed founder base – across cycles of selection the base shifts, and a marker fixed by selection ($p \rightarrow 0$ or 1) drops out rather than reading as fixation. For a fixed reference (e.g. recurrent selection or founder-relative inbreeding) pass `base_freq`: markers polymorphic in the base are kept even when currently monomorphic, and $2p$ centring uses the base frequencies.

Value

an individuals x individuals matrix with id dimnames.

References

VanRaden PM (2008) Efficient methods to compute genomic predictions. *Journal of Dairy Science* 91(11):4414–4423. doi:10.3168/jds.20070980

See Also

`optimum_contribution()`, `dosages()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
G <- g_matrix(pop)
dim(G)
# Genomic inbreeding of the first few lines:
round(diag(G)[1:3] - 1, 3)
```

genetic_values

Genetic values behind the simulated phenotypes

Description

The genetic component of each individual's phenotype: the sum of the scaled additive, dominance and epistatic layers, before the residual is added. This is the quantity the v1 engine wrote to `Genetic_values.txt`.

Usage

```
genetic_values(sim, rep = 1L)
```

Arguments

<code>sim</code>	a <code>phenotype_sim</code> .
<code>rep</code>	replication to retrieve (default 1).

Details

When a phenotype includes a genome-**derived** transcriptome() layer, the genetic-mediated part of that expression component (the share tracing to the genome through gene expression) is also included, so it counts toward heritability. A **real** expression source contributes nothing here: its genetic content is not asserted, so it is treated as an environmental predictor and excluded from the genetic value.

Variance QTL are deliberately excluded: a vqt1() layer modulates the spread of the residual rather than contributing a genetic value, so it has no place in this matrix. By default this returns replication 1. When the simulation used vary_qtn = TRUE, use rep to retrieve the architecture and genetic values belonging to another replication.

Value

An individuals-by-traits numeric matrix, with individual IDs as row names and trait names as column names.

See Also

[qtn_table\(\)](#) for the loci and effects that produce these values, [phenotypes_long\(\)](#) for the phenotypes themselves.

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, h2 = 0.5,
  seed = 1) |>
  additive(n_qtn = 5)

g <- genetic_values(ph)
head(g)

# Realized genetic correlation between the two traits
round(cor(g[, 1], g[, 2]), 3)

# Realized heritability: genetic variance over phenotypic variance
pheno <- phenotypes_wide(ph)
round(var(g[, 1]) / var(pheno$Trait_1), 3)
```

genotypic_value	<i>Total genotypic value on a fixed, cross-generational scale</i>
-----------------	---

Description

The additive-plus-dominance counterpart of [additive_value\(\)](#). Scores each individual by its **total genotypic value**

$$G_i = \sum_j [a_j \text{dosage}_{ij} + d_j \mathbf{1}(\text{dosage}_{ij} = 0)]$$

over a **given, frozen architecture**: a per-locus additive effect a and dominance deviation d , on the $-1/0/1$ dosage scale (so the heterozygote is dosage 0), with **no per-population centring or rescaling**. The per-locus term is $-a / +d / +a$ for gene content 0 / 1 / 2 – exactly the raw genotypic value an `additive(orthogonal = TRUE, a = , d =)` layer and a `dominance()` layer build – but held fixed so it is comparable across generations, like `additive_value()`.

Usage

```
genotypic_value(x, qtn, a, d)
```

Arguments

<code>x</code>	a Population, a Population-backed phenotype_sim, or a marker x individual dosage matrix coded $-1/0/1$ (marker names as row names when qtn is given by name; individual ids as column names).
<code>qtn</code>	the causal loci, as marker names (matched against the map) or integer marker (row) indices.
<code>a</code>	a finite numeric vector of per-locus additive effects (departure of the homozygote from the midpoint), one per entry of qtn and in the same order (as in <code>additive_value()</code> 's effect).
<code>d</code>	a finite numeric vector of per-locus dominance deviations – the value of the heterozygote above the homozygous midpoint – one per entry of qtn and in the same order. Use 0 at a purely additive locus.

Details

This scores each individual's own **per se** total genotypic value $G = A + D$. It is **not** a parental combining ability: a per se genotypic value is not a parent's testcross merit – at a pure-dominance locus ($a = 0, d > 0$) a heterozygous Aa parent scores above a homozygous AA parent, yet crossed to an aa tester it is the AA parent whose progeny mean is higher, so per se scores can reverse testcross ranking. A reciprocal-recurrent or hybrid program therefore uses this by scoring the **realized testcross / hybrid progeny** (score the progeny population and dominance drives its mean), not the parents per se.

It is also **not** the transmissible breeding value. For selection on breeding value use `select_ind()` with `on = "bv"`, which scores the average-effect breeding value $A_i = \sum_j \alpha_j (x_{ij} - 2p_j)$ with $\alpha_j = a_j + d_j(1 - 2p_j)$. Passing those α to `additive_value()` yields only a **ranking-equivalent** score (α_j dosage_{*ij*}, which differs from A_i by an additive constant), not the population-centred breeding value.

Value

a named numeric vector of total genotypic values, one per individual.

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow – the genotypic-value parameters a (departure of the homozygote from the midpoint) and d (dominance deviation of the heterozygote), and the average effect $\alpha = a + d(q - p)$.

See Also

[additive_value\(\)](#), [phenotype_value\(\)](#), [genetic_values\(\)](#), [select_ind\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
# a = additive effects, d = dominance deviations (complete dominance at locus 5).
gv <- genotypic_value(pop, qtn = c(1, 5, 9),
                      a = c(0.5, -1, 2), d = c(0, 1, 0))
head(gv)
```

gxe_value

*Genotype-by-environment slope on a fixed, cross-generational scale***Description**

Scores each individual's **G x E slope** (its response to the environmental covariate) on a frozen architecture:

$$s_i = \text{intercept} + \sum_j \text{dosage}_{ij} b_j,$$

with the -1/0/1 dosages and **no per-population centring or rescaling**, like [additive_value\(\)](#). This is the additive-by-environment trait of AlphaSimR (`SimParam$addTraitAG()`): its per-locus `gxeEff` are the effect here and its `gxeInt` the intercept, and AlphaSimR's centred genotype $x - 1$ ($x = 0/1/2$ copies of the counted allele) is this package's dosage. [phenotype_value\(\)](#) adds $s_i * w$ to the phenotype, where w is the environmental covariate of the trial (see its `gxe / env / var_env` arguments).

Usage

```
gxe_value(x, qtn, effect, intercept = 0)
```

Arguments

x	a Population, a Population-backed <code>phenotype_sim</code> , or a marker x individual dosage matrix coded -1/0/1 (marker names as row names when <code>qtn</code> is given by name; individual ids as column names).
qtn	the causal loci, as marker names (matched against the map) or integer marker (row) indices.
effect	a finite numeric vector of per-locus G x E effects (slope effects), one per entry of <code>qtn</code> and in the same order.
intercept	a single finite number added to every slope (AlphaSimR <code>gxeInt</code>). AlphaSimR sets it so the mean slope of the founder population is 1 when the trait has an environmental variance (<code>varEnv > 0</code> , so the covariate is also a main effect of the environment) and 0 otherwise.

Value

a named numeric vector of slopes, one per individual.

References

Gaynor RC, Gorjanc G, Hickey JM (2021) AlphaSimR: an R package for breeding program simulations. *G3* 11(2):jkaa017. doi:10.1093/g3journal/jkaa017 (the additive-by-environment trait additive-by-environment trait). The exact slope scaling and the phenotype formula reproduced here are those of the AlphaSimR 2.1.0 source (SimParam\$addTraitAG(), calcPheno()), not equations printed in the paper.

See Also

[phenotype_value\(\)](#), [additive_value\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
s <- gxe_value(pop, qtn = c(1, 5, 9), effect = c(0.2, -0.1, 0.3))
head(s)
```

haplotypes

The phased haplotypes of a population

Description

The inverse of [population_from_haplotypes\(\)](#): the two homologous strands of every individual, 1 marking the counted (+1) allele (so that dosages(pop) == cis + trans - 1).

Usage

```
haplotypes(x)
```

Arguments

x a Population.

Value

A list with the integer matrices cis and trans, markers by individuals (rows named by map\$snp, columns by the individual ids).

See Also

[population_from_haplotypes\(\)](#), [dosages\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:3)
h <- haplotypes(pop)
dim(h$scis)
stopifnot(identical(h$scis + h$trans - 1L, dosages(pop)))
```

heterosis

Heterosis on a frozen additive + dominance architecture

Description

Realized heterosis of a crossbred population: its mean genotypic value minus the mean of the pure breeds weighted by its expected breed composition (`breed_composition()`) – the mid-parent for an F1. Also returns, for every pair of breeds, the expected F1 heterosis: the expected mean of random matings between the two breed populations minus their mid-parent, computed from the breeds' own genotypes (this package's derivation; it uses only the breeds' gamete frequencies, so linkage does not enter). Per locus, with gamete frequencies p_A, p_B of the +a allele, the F1's allele frequency is the breeds' average, so the additive part cancels against the mid-parent and the heterosis is $d[h_{AB} - (h_A + h_B)/2]$, where $h_{AB} = p_A(1 - p_B) + (1 - p_A)p_B$ is the F1's heterozygote frequency and h_A, h_B the breeds' own. When each breed is in Hardy-Weinberg proportions ($h = 2p(1 - p)$) this is $\sum_j d_j (p_{Aj} - p_{Bj})^2$ – Falconer & Mackay's (1996) $H_{F1} = \sum dy^2$, with y the difference in allele frequency between the two populations – zero for a purely additive architecture; for inbred-line breeds ($h = 0$) it is $\sum_j d_j (p_{Aj} + p_{Bj} - 2p_{Aj}p_{Bj})$. The classical fractions of the F1 heterosis retained in later crosses (one half in an F2 or a backcross, two thirds in a two-breed rotation at equilibrium) follow from the probability that an individual's two alleles come from different breeds, a dominance (breed-origin) model, and are exact **only under conditions on the breeds' Hardy-Weinberg deviations** $h - 2p(1 - p)$ at the locus (not, in general, on each breed separately). A single fixed inbred line has no deviation ($p = 0$ or 1 , so $h = 0 = 2p(1 - p)$), but a breed made of several fully inbred lines that differ at a locus does ($h = 0 < 2p(1 - p)$). Exactly, at one locus with non-zero F1 heterosis the retained fraction in a backcross to breed A is one half if and only if $h_A = 2p_A(1 - p_A)$, i.e. only the recurrent breed A need be in Hardy-Weinberg proportions (whatever breed B is), and in the F2 if and only if $[h_A - 2p_A(1 - p_A)] + [h_B - 2p_B(1 - p_B)] = 0$, i.e. the two deviations sum to zero (for example -0.12 and $+0.12$, with neither breed in Hardy-Weinberg proportions); for breeds that violate this the F2 and backcross retain a different fraction. (The two-thirds rotation figure is the classical value for breeds in Hardy-Weinberg proportions; its condition for breeds with deviations is not derived here.) For example, an equal mixture of AA and aa lines crossed to an aa line (pure dominance) has F1 heterozygosity $1/2$ and backcross heterozygosity $1/2$ against baselines of 0, so the retention is 1, not $1/2$. realized is computed from the actual genotypes, so it does not assume either fraction; epistasis is outside this function.

Usage

```
heterosis(pop, breeds, qtn, a, d = 0)
```

Arguments

pop	the crossbred Population (pedigree traced to the breed founders).
breeds	a named list of the pure-breed Populations, each name the founder pool its population traces to (as_population(pool =)); checked. Each must hold every founder of its pool that pop's pedigree traces to (the breed means are taken from these populations only), and share the marker map of pop.
qtn, a, d	frozen loci and effects, as for genotypic_value() ; d may be a single value.

Value

A list with realized (mean minus composition-weighted breed mean), mean, breed_means, composition (the population's mean expected breed fractions) and expected_f1 (breeds x breeds matrix of the expected F1 heterosis of each pair of distinct breeds; symmetric, NA on the diagonal, since a breed mated with itself is not a cross).

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow
– heterosis of a cross between two populations, $H_{F1} = \sum dy^2$.

See Also

[breed_composition\(\)](#), [crossbreed\(\)](#), [genotypic_value\(\)](#)

Examples

```
g <- data.frame(snp = paste0("m", 1:20), allele = "A/G",
               chr = rep(1:2, each = 10), pos = rep(1:10, 2),
               cm = rep(seq(0, 90, by = 10), 2))
set.seed(1)
A <- as_population(cbind(g, matrix(sample(c(-1L, 1L), 200, TRUE), 20,
                                     dimnames = list(NULL, paste0("A", 1:10)))), pool = "A")
B <- as_population(cbind(g, matrix(sample(c(-1L, 1L), 200, TRUE), 20,
                                     dimnames = list(NULL, paste0("B", 1:10)))), pool = "B")
f1 <- crossbreed(list(A = A, B = B), "two_way", n_progeny = 50, seed = 2)
heterosis(f1, list(A = A, B = B), qtn = 1:20, a = rep(0.2, 20), d = 0.3)$realized
```

liability_threshold *Categorical phenotypes from a liability-threshold model*

Description

Turns the simulated (continuous) phenotype into an ordered categorical one: the continuous value is the unobserved **liability** l , and the observed category is $y = 1$ if $l < t_1$, $y = 2$ if $t_1 \leq l < t_2$, ..., $y = C$ if $l \geq t_{(C-1)}$ (Wright 1934; Falconer 1965). The thresholds are set from the expected category proportions `prop` on the standardized liability: $t_k = qnorm(\text{sum}(\text{prop}[1:k]))$ for the liability standardized to mean 0 and variance 1 in the simulated population, so the realized proportions match `prop` up to sampling (exactly so only for a normal liability).

Usage

```
liability_threshold(sim, prop, trait = NULL)
```

Arguments

<code>sim</code>	a <code>phenotype_sim</code> .
<code>prop</code>	expected category proportions: a numeric vector of at least two positive values summing to 1 (2 = a binary disease trait, e.g. <code>c(0.9, 0.1)</code> for a prevalence of 10%), or a list with one such vector per trait in <code>trait</code> .
<code>trait</code>	the traits to categorize (default all).

Details

Every variance quantity of the simulation (layers, `var_budget`, `h2`, `genetic_values()`, `qtn_table()`) stays on the **liability scale**, where the genetic model is defined; only the phenotype is categorical. The liability is kept in `sim$liability` (same columns as `sim$pheno`). Selection on `on = "pheno"` ranks the categories (ties broken as `select_ind()` does). Applied to a population whose trait is reused (`simulate_phenotype(refit = FALSE)`, see `population_trait()`) the thresholds are the base population's, on its liability scale, so the category frequencies change as the liability mean moves under selection.

Value

the `phenotype_sim` with categorical phenotypes (values 1..C).

References

Falconer DS (1965) The inheritance of liability to certain diseases, estimated from the incidence among relatives. *Annals of Human Genetics* 29, 51–76. doi:[10.1111/j.14691809.1965.tb00500.x](https://doi.org/10.1111/j.14691809.1965.tb00500.x)

Wright S (1934) An analysis of variability in number of digits in an inbred strain of guinea pigs. *Genetics* 19(6), 506–536. doi:[10.1093/genetics/19.6.506](https://doi.org/10.1093/genetics/19.6.506)

See Also

[simulate_phenotype\(\)](#), [coheritability\(\)](#)

Examples

```
data("SNP55K_maize282_maf04")
# a binary trait with 20% prevalence and h2 = 0.5 on the liability scale
sim <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.5, n_qtn = 10,
                          seed = 1) |>
  liability_threshold(prop = c(0.8, 0.2))
table(sim$pheno$value)
# three ordered categories
sim3 <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.4, n_qtn = 10,
                           seed = 1) |>
  liability_threshold(prop = c(0.25, 0.5, 0.25))
table(sim3$pheno$value)
```

mabc_select

*Marker-assisted backcross selection***Description**

Selects backcross individuals in the three standard stages of marker-assisted introgression of a target gene (Frisch & Melchinger 2001): **foreground** selection for the donor allele at the target marker(s), applied as a hard feasibility filter; optional **recombinant** selection at flanking markers, which favours candidates that are already recurrent-homozygous next to the target and so carry less donor linkage drag; and **background** selection, ranking the remaining candidates by their recovery of the recurrent-parent genome ([recurrent_parent_recovery\(\)](#)).

Usage

```
mabc_select(
  pop,
  recurrent,
  donor,
  target_markers,
  target_requirement = c("donor_carrier", "donor_homozygote"),
  background_markers = NULL,
  exclude_interval = NULL,
  flanking_markers = NULL,
  n = 1L,
  marker_weights = NULL,
  seed = NULL
)
```

Arguments

pop the candidate Population (e.g. a BC1 family).

recurrent, donor single-individual Populations: the recurrent parent and the donor of the target allele, on the same marker map as pop.

target_markers target marker(s), by name or map index. All must be informative.

target_requirement "donor_carrier" (at least one donor allele at every target marker; backcross generations) or "donor_homozygote" (donor homozygous at every target marker; e.g. after the final self).

background_markers markers scored for recurrent-genome recovery, by name or index. NULL (default) uses every informative marker except the target and flanking markers and those in **exclude_interval**.

exclude_interval optional genomic interval(s) removed from the background, typically the target region, whose donor segment foreground selection deliberately keeps: a list or

data frame with chr, from, to in centiMorgans (the map's cm). Several rows give several intervals. The bounds are inclusive (a marker at exactly from or to is excluded). Markers inside are dropped, and with marker_weights = "interval" the interval's length is also removed from the genome the remaining markers represent; with the default equal weights only the markers are removed and the excluded region's genome share is *not* taken out of the denominator. Target and flanking markers leave the background too, but the genome around them stays represented by the neighbouring background markers unless an interval covers it.

flanking_markers	optional markers for recombinant selection, by name or index; all must be informative and on a target's chromosome, ideally one on each side of the target (a one-sided set warns).
n	number of individuals to select.
marker_weights	background marker weights: NULL (equal weights), "interval" (each marker weighted by the cM length of genome nearer to it than to any other background marker on its chromosome – half-way to each neighbour, with the end markers covering the chromosome out to its first and last mapped marker – minus any part inside exclude_interval; each chromosome thus counts its mapped length less the excluded part, and a dense marker cluster does not count as a disproportionate share of the genome), or a numeric vector over all map markers (named by marker, or in map order).
seed	optional seed for the random tie-break (the ambient RNG state is restored afterwards).

Details

Only markers at which the recurrent and donor founders are homozygous for alternate alleles are informative: there a candidate's genotype says which parent each allele came from (recurrent homozygote, heterozygote, donor homozygote). Every target and flanking marker must be informative – the function errors rather than guess the donor allele. Non-informative markers are dropped from the background.

Ranking is lexicographic among feasible candidates: most flanking markers in the recurrent-homozygous state first, then highest background recovery, then a seeded random tie-break. Recurrent genome recovery is **marker-observed** allele recovery: a Population carries no per-locus founder ancestry, so this is not identity by descent.

For an unselected cohort from t successive backcrosses to the recurrent parent after the F1 ($t = 0$ for the F1), the expected donor genome proportion in BC t is $1/2^{(t+1)}$ (Frisch & Melchinger 2005), so the expected recovery is $R_t = 1 - (1/2)^{(t+1)}$: 0.5, 0.75, 0.875, 0.9375 at F1, BC1, BC2, BC3. It follows from Mendelian transmission – each backcross joins one fully recurrent gamete to a gamete carrying, on average, the backcrossed parent's recurrent fraction, so $R_0 = 1/2$ and $R_t = 1/2 + R_{(t-1)}/2$. Background-selected individuals are meant to exceed their cohort's mean. The staged foreground / recombinant / background procedure follows Frisch & Melchinger (2001); the weighted marker estimator of recovery is this package's definition. Plain truncation selection with `select_ind()` on a phenotype is not marker-assisted backcrossing.

recurrent, donor and pop must share one allele coding: a dosage of +1 must mean the same allele in all three at every marker. That holds by construction when the candidates are bred from the

founders with `cross()`, `selfcross()` and the other crossing functions. It does not hold automatically for genotyped material converted separately: `as_numeric()`'s default method = "frequency" codes each dataset's own major allele as +1, so separately converted founders and candidates can be reverse-coded at some markers. Convert them together in one `as_numeric()` call, or separately with method = "reference" and the same `ref_allele`. The coding is **not checked** – it cannot be inferred from the dosages, since every candidate genotype is possible at an informative marker – and a reverse-coded marker swaps the recurrent- and donor-homozygote calls there.

Value

The selected individuals as a crossable Population, with attribute "mabc": a data frame over all candidates with id, target (per-target genotype, R/H/D for recurrent homozygote / heterozygote / donor homozygote, ;-separated in target_markers order), feasible, flank_recurrent (flanking markers in the recurrent-homozygous state), background_recovery, tiebreak, rank (among feasible candidates) and selected; and attribute "mabc_info" recording the target, flanking and background marker sets and settings.

References

- Frisch M, Melchinger AE (2001) Marker-assisted backcrossing for introgression of a recessive gene. *Crop Science* 41:1485–1494. doi:10.2135/cropsci2001.4151485x
- Frisch M, Melchinger AE (2005) Selection theory for marker-assisted backcrossing. *Genetics* 170(2):909–917. doi:10.1534/genetics.104.035451
- Yadav AK, Kumar A, Grover N, et al. (2020) Marker aided introgression of 'Saltol', a major QTL for seedling stage salinity tolerance into an elite Basmati rice variety 'Pusa Basmati 1509'. *Scientific Reports* 10:13877. doi:10.1038/s41598020706640

See Also

`recurrent_parent_recovery()`, `cross()`, `selfcross()`, `select_ind()`.

Examples

```
g <- data.frame(snp = paste0("m", 1:40), allele = "A/G",
               chr = rep(1:2, each = 20), pos = rep(1:20, 2) * 1e6,
               cm = rep(seq(0, 95, by = 5), 2), P1 = 1L, P2 = -1L)
founders <- as_population(g)
recurrent <- founders[1]; donor <- founders[2]
f1 <- cross(recurrent, donor, seed = 1)
bc1 <- cross(f1, recurrent, n = 50, seed = 2)
best <- mabc_select(bc1, recurrent, donor, target_markers = "m5",
                  exclude_interval = list(chr = 1, from = 10, to = 30),
                  flanking_markers = c("m3", "m7"), n = 3, seed = 3)
attr(best, "mabc")[attr(best, "mabc")$selected, ]
```

marker_select

*Marker-assisted selection and gene pyramiding***Description**

Keeps the candidates carrying the favourable allele at target markers – marker-assisted selection (MAS) of major genes – and ranks the feasible ones. The favourable allele is given directly (favorable), so no founders are needed; for introgression from a donor with recovery of the recurrent genome, use `mabc_select()`, which this generalizes for the foreground step.

Usage

```
marker_select(
  pop,
  markers,
  favorable = 1L,
  requirement = c("carrier", "homozygote"),
  min_markers = NULL,
  n = NULL,
  prop = NULL,
  rank_on = NULL,
  direction = c("high", "low"),
  seed = NULL
)
```

Arguments

pop	the candidate Population.
markers	target markers, by name or map index.
favorable	which homozygote carries the favourable allele in pop's coding: 1 or -1, one value or one per marker.
requirement	"carrier" or "homozygote", one value or one per marker.
min_markers	feasible candidates meet the requirement at this many targets or more (default all; 0 makes every candidate feasible, e.g. for a pure index ranking).
n, prop	optional number, or proportion of the candidates, to keep; neither keeps every feasible candidate. An error if fewer are feasible.
rank_on	optional ranking score among feasible candidates: a numeric vector (named by id, or in id order) or a function of pop returning one.
direction	rank rank_on from "high" (default) or "low".
seed	optional seed for the tie-break (the ambient RNG state is restored).

Details

A candidate meets the requirement at a marker when it is a "carrier" (at least one favourable allele) or a "homozygote" for it. It is feasible when it meets the requirement at min_markers or more of the targets: the default, all of them, is a full pyramid; fewer is a **staged** pyramid (e.g. 3 of 4 genes this cycle), reported as such. Feasible candidates are ranked on rank_on (by default the number of targets met), then by a seeded random tie-break.

rank_on may be any score: a phenotype, predicted breeding values, or a marker index. `additive_value()` with marker names and weights is the marker index $\sum_j w_j x_{ij}$ of marker-assisted recurrent selection (MARS). With the simulation's own causal loci weighted by their **average effects** it is an **oracle** index of transmissible merit – the upper bound of an index on estimated additive marker effects (Lande & Thompson 1990), not a realistic one. For an additive architecture those weights are the additive effects a of `template_effects()`; with dominance they are $\alpha = a + d(q - p)$ at the candidates' allele frequencies, not the raw a (which ignores dominance's contribution to the breeding value; α is the average effect of Falconer & Mackay 1996). Ranked on such an oracle index, `marker_select()` keeps the individuals `select_ind(on = "bv")` keeps, except among tied scores at the cut-off: `marker_select()` breaks ties at random (see `seed`), `select_ind()` by input order.

The favourable allele is identified by the pop's -1/0/1 coding; as for `mabc_select()`, that coding cannot be verified from dosages, so favorable must refer to the same coding as pop.

Value

The selected individuals as a Population, with attribute "marker_select": a data frame over all candidates with id, state (per-target genotype, F / H / U for favourable homozygote / heterozygote / unfavourable homozygote, ; -separated), n_met, feasible, score, tiebreak, rank (among feasible) and selected; and attribute "marker_select_info" with the settings, including whether the pyramid was staged.

References

- Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow – the average effect $\alpha = a + d(q - p)$.
- Lande R, Thompson R (1990) Efficiency of marker-assisted selection in the improvement of quantitative traits. *Genetics* 124:743–756. doi:10.1093/genetics/124.3.743

See Also

`mabc_select()`, `additive_value()`, `select_ind()`

Examples

```
g <- data.frame(snp = paste0("m", 1:4), allele = "A/G", chr = 1:4, pos = 1,
               cm = 0, P1 = 1L, P2 = -1L)
pop <- as_population(g)
f2 <- selfcross(cross(pop[1], pop[2], seed = 1), n = 200, seed = 2)
pyr <- marker_select(f2, markers = c("m1", "m2"), requirement = "homozygote")
nrow(attr(pyr, "marker_select")[attr(pyr, "marker_select")$feasible, ]) # ~ 200/16
```

mate	<i>Execute a mating plan</i>
------	------------------------------

Description

Runs every row of a mating plan – mother, father, number of progeny n – through `cross()` (or `selfcross()` when a row names the same individual twice, or `double_haploid()` when `design = "dh"`), and pools the progeny into one Population with its pedigree recorded (see `parentage()`). Parents may come from several Populations sharing one marker map, e.g. two breeds or heterotic pools.

Usage

```
mate(plan, ..., seed = NULL, prefix = NULL, interference = NULL)
```

Arguments

<code>plan</code>	a data frame with columns <code>mother</code> , <code>father</code> (individual ids) and <code>n</code> (progeny per row, a positive whole number), as produced by <code>mating_design()</code> . With several pools, add <code>mother_pool</code> and <code>father_pool</code> naming the pool (an argument name in ...) of each parent. An optional <code>design</code> column (" <code>cross</code> ", " <code>self</code> ", " <code>dh</code> ") overrides the default, which is a cross, or a self when mother and father are the same individual; an NA entry takes the default for its row.
<code>...</code>	the parent Population(s). A single population may be unnamed; several must be named, and the names are the pools the plan refers to.
<code>seed</code>	optional RNG seed (see Details).
<code>prefix</code>	progeny id prefix; progeny are named <code><prefix>_1</code> , <code><prefix>_2</code> , ... in plan order. Default: the pool name(s) involved, joined by <code>x</code> (e.g. " <code>A</code> " or " <code>AxB</code> "), or " <code>prog</code> " for a single unnamed population.
<code>interference</code>	crossover interference model of <code>cross()</code> (see its section "Crossover interference"): <code>NULL</code> (default: the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$), " <code>poisson</code> " (Poisson crossovers, the <code>isqg</code> stream, the default before 2.0.0.9003) or <code>list($\nu =$, $p =$)</code> . Applied to every row.

Details

The names in ... only tell `mate()` where to look up each parent; they do not make individuals distinct. Individuals are identified by their pedigree: a founder by its `as_population()` pool label, id and haplotypes, a progeny by its mating. Import each breed with its own pool label, or two founders that share an id and genotype are one individual, and mating them is a self.

With a `seed`, the RNG is seeded once and the rows are run in plan order, so a one-row plan draws the same random numbers and produces the same progeny genotypes as the equivalent call with that seed – `cross()` for a cross row, `selfcross()` for a self, `double_haploid()` for a "`dh`" row; the returned object differs only in its ids, origin and plan attribute. The caller's RNG state is restored on exit.

The populations must share one marker map and the allele coded +1 (see `cross()` on allele orientation).

All rows are executed in one pass through the Rust core: the meioses of the rows are drawn one row after the other (so the random stream, and every progeny, pedigree key and id, are exactly those of calling `cross()`, `selfcross()` or `double_haploid()` row by row), and the progeny of every row are assembled in a single integer-in, integer-out call. This is the way to produce many families at once: a plan with `design = "dh"` and one row per parent gives the doubled haploids of all parents in one call. The per-call overhead of serialising the parents and pooling the results is paid once, not once per row, and the random draws – which must follow R's stream – are the main cost that remains (on a 14,000-marker, 20-chromosome map, 100 families of 100 doubled haploids take about 2 s, against about 14 s before the batched path).

Every row's meioses follow the same crossover interference model (the `interference` argument; by default the gamma model, see the section "Crossover interference" of `cross()`).

Value

A Population of all progeny, in plan order, with attribute `plan` (the plan with the progeny ids of each row in a list column `progeny`).

See Also

`mating_design()`, `cross()`, `parentage()`, `families()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:4)
plan <- mating_design(pop, design = "half_diallel", progeny_per_cross = 2)
prog <- mate(plan, pop, seed = 1)
table(families(prog, "full_sib"))
```

mating_design

Build a mating plan

Description

Writes the mother, father, n plan that `mate()` executes, for the standard mating designs.

Usage

```
mating_design(
  mothers,
  fathers = mothers,
  design = c("random", "factorial", "nested", "diallel", "half_diallel"),
  n_crosses = NULL,
  progeny_per_cross = 1L,
  mothers_per_father = NULL,
```

```

    allow_self = FALSE,
    seed = NULL
)

```

Arguments

mothers, fathers	Populations (or character vectors of ids). fathers defaults to mothers; the diallels use mothers only (fathers is ignored).
design	one of the designs above.
n_crosses	number of random pairs ("random" only).
progeny_per_cross	progeny per plan row (the plan's n).
mothers_per_father	mothers per father ("nested" only).
allow_self	allow a parent to be mated with itself.
seed	optional RNG seed for "random"; the caller's RNG state is restored on exit. The other designs are deterministic, draw nothing and leave the RNG untouched, so seed is ignored for them.

Details

- "random": n_crosses pairs drawn at random (a mother from mothers, a father from fathers; with one parent set, two distinct individuals). Without selfs, a mother is drawn among those with at least one father other than herself, then one of those fathers. The draw is uniform over mothers and then over each mother's admissible fathers, so it is uniform over the admissible *pairs* only when every mother has the same number of admissible fathers; pairs drawn with replacement, a parent can recur.
- "factorial": every mother with every father (North Carolina Design II).
- "nested": each father with its own set of mothers_per_father distinct mothers, no mother shared between fathers (North Carolina Design I).
- "diallel": every ordered pair of distinct parents, reciprocals included (Griffing's method 3; with allow_self = TRUE, method 1).
- "half_diallel": every unordered pair of distinct parents, once (Griffing's method 4; with allow_self = TRUE, method 2).

Designs pair individuals by id. A pair of the same **individual** is a self, excluded unless allow_self = TRUE ("diallel" and "half_diallel" then add each parent's self). Individuals are compared by their pedigree identity, not their id: two populations drawn from one (e.g. pop[1:3] and pop[2:4]) share individuals, while two pools that merely reuse ids (e.g. breeds whose animals are both named P1) do not, provided they were imported with their own `as_population()` pool labels (a founder is identified by that label, its id and its haplotypes). When either set is given as character ids, individuals are compared by id. Each individual enters a set once: the same individual under a second id (e.g. after `c(pop, pop)`) is dropped, keeping its first id, so no pair is listed twice.

Value

A data frame with columns mother, father, n, ready for `mate()` (add mother_pool / father_pool when mating across populations).

References

Griffing, B. (1956). Concept of general and specific combining ability in relation to diallel crossing systems. *Australian Journal of Biological Sciences* 9(4), 463–493. doi:10.1071/BI9560463

See Also

`mate()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:4)
mating_design(pop, design = "half_diallel")
mating_design(pop[1:2], pop[3:4], design = "factorial", progeny_per_cross = 5)
```

mediation_split	<i>Genetic-mediated vs environmental split of a derived transcriptome component</i>
-----------------	---

Description

For a phenotype built with a genome-**derived** transcriptome() layer (simulate_phenotype(transcriptome = ...)), the expression-mediated component splits into a genetic-mediated part (the share of expression that traces to the genome, which counts toward heritability and appears in `genetic_values()`) and an environmental part. This returns that realized split, per trait, as proportions of phenotypic variance: genetic_mediated, env_mediated, and their covariance term $2 \cdot \text{Cov}(\text{Tx}_g, \text{Tx}_e) / V_P$ (finite-sample, ~ 0 because the generator draws the genetic and non-genetic parts of expression independently). The three columns sum to the realized expression-mediated share of variance.

Usage

```
mediation_split(sim)
```

Arguments

sim a phenotype_sim.

Details

Returns NULL when the phenotype has no derived transcriptome layer: a **real** expression source (simulate_phenotype(expression = ...)) has no asserted genetic/environmental decomposition, so no split is reported.

Value

A data frame (trait, genetic_mediated, env_mediated, covariance), or NULL.

See Also

[genetic_values\(\)](#), [transcriptome\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
tx <- simulate_transcriptome(SNP55K_maize282_maf04, n_genes = 100, seed = 1)
ph <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.3, seed = 2,
                        transcriptome = tx) |>
  transcriptome(prop = 0.4, n_genes = 20)
mediation_split(ph)
```

n_individuals	<i>Number of individuals in a Population</i>
---------------	--

Description

Number of individuals in a Population

Usage

```
n_individuals(x)
```

Arguments

x a Population.

Value

An integer.

Examples

```
data("SNP55K_maize282_maf04")
n_individuals(as_population(SNP55K_maize282_maf04))
```

observe_counts

*Draw RNA-seq counts from a simulated transcriptome***Description**

Maps the normalized latent expression of a transcriptome_sim to integer counts through a negative-binomial observation model. For gene g and individual i , with z_{gi} the latent expression standardized per gene,

$$\log \mu_{gi} = \log L_i + \alpha_g + \sigma_g z_{gi}, \quad Y_{gi} \sim \text{NB}(\mu_{gi}, \phi_g),$$

where L_i is a per-individual library-size (sequencing-depth) factor, α_g a per-gene log baseline count, σ_g the latent-to-log-mean coupling, and ϕ_g the NB dispersion ($\text{Var} = \mu + \phi \mu^2$). Standardizing the latent expression makes σ_g a per-standard-deviation log-fold change, comparable across genes and unaffected by any mimic rescaling.

Usage

```
observe_counts(
  tx,
  library_size = 1,
  baseline = 3,
  coupling = 0.5,
  dispersion = 0.1,
  seed = NULL
)
```

Arguments

tx	a transcriptome_sim from simulate_transcriptome() .
library_size	per-individual depth factor L_i : one positive number, or one per individual. Default 1 (no depth variation).
baseline	per-gene log baseline count α_g : one number or one per gene. Default 3 (a mean of about $\exp(3)$ counts at $z = 0$).
coupling	latent-to-log-mean slope σ_g : one non-negative number or one per gene. Default 0.5.
dispersion	NB dispersion ϕ_g ($\text{Var} = \mu + \phi \mu^2$): one non-negative number or one per gene. 0 gives a Poisson limit. Default 0.1.
seed	optional seed; the RNG state is restored afterwards.

Details

Count-scale heritability differs from the latent heritability. The link is nonlinear and mean-dependent, so the fraction of count variance explained by the genome is not the latent h^2 ; treat counts as an observation layer for count-based methods, not as a re-parameterization of the genetic model.

Value

the transcriptome_sim with a counts matrix (genes x individuals, R storage type integer on both the Poisson and negative-binomial paths) and a count_model list (the per-gene/per-individual parameters and the realized mean matrix mu). Counts must fit R's integer range: a mean above 1e9, or a draw above .Machine\$integer.max, is an error rather than a silently huge double.

References

The $\text{Var} = \mu + \phi \mu^2$ negative-binomial parameterization and its use for RNA-seq counts follow the edgeR convention: McCarthy DJ, Chen Y, Smyth GK (2012) *Nucleic Acids Res.* 40(10):4288-4297, doi:10.1093/nar/gks042; Robinson MD, Smyth GK (2008) *Biostatistics* 9(2):321-332, doi:10.1093/biostatistics/kxm030 (page-level equation numbers not verified). The log-link to the latent expression and the library-size factor are this package's own design.

See Also

[simulate_transcriptome\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
tx <- simulate_transcriptome(SNP55K_maize282_maf04, n_genes = 50, seed = 1)
tx <- observe_counts(tx, seed = 2)
tx$counts[1:5, 1:5]
```

optimum_contribution *Optimum contribution selection*

Description

Chooses how much each candidate should contribute to the next generation to maximize genetic merit while holding down relatedness — the modern answer to "truncation selection erodes diversity and long-term gain" (Meuwissen 1997). It maximizes $c'g - \frac{\lambda}{2}c'Gc$ over contributions c on the simplex ($c \geq 0$, $\text{sum}(c) = 1$), where g is merit and G the genomic relationship matrix; the group coancestry of the chosen parents is $\frac{1}{2}c'Gc$. Give a λ to trace the gain–diversity frontier, or a target_coancestry / max_coancestry and the penalty is found by bisection.

Usage

```
optimum_contribution(
  x,
  merit = "bv",
  trait = 1L,
  direction = c("high", "low"),
  lambda = NULL,
  target_coancestry = NULL,
  max_coancestry = NULL,
```

```

G = NULL,
rep = 1L,
min_contribution = 1e-04,
max_iter = 10000L,
tol = 1e-10
)

```

Arguments

x	a phenotype_sim (supplies both merit and, via its genotypes, G) or a Population (then merit must be a numeric vector or G is built from it and merit supplied).
merit	selection criterion: "bv" (default), "gv", "pheno", a numeric vector (named by id or in order), or a function merit(sim) — the same hook as select_ind() , so genomic/phenomic EBVs plug straight in. Meuwissen's OCS maximizes the <i>transmissible</i> merit of the parents' progeny, so the default is the additive breeding value "bv", not the total genotypic value "gv": with "gv" a non-transmissible dominance/epistasis advantage (e.g. an overdominant heterozygote) would be chased even though it is not passed on. "bv" is the <i>transmitting</i> average effect, computed analytically from the simulation's own effects as $\alpha_j = a_j + d_j(q_j - p_j)$ per locus (DECISION-019; the same quantity as select_ind(on = "bv")) — not a least-squares (Fisher statistical) projection of the genotypic value, which differs off Hardy-Weinberg. It is refused (an error) for a model with an epistasis layer and for architecture = "complex", because those have no per-locus a/d ; pass a numeric merit there.
trait	trait index when merit is "gv"/"pheno" (default 1).
direction	"high" (default) maximizes merit, "low" minimizes it.
lambda	penalty on group coancestry (≥ 0). Larger spreads contributions and lowers coancestry. Give exactly one of lambda, target_coancestry, max_coancestry.
target_coancestry	desired group coancestry $\frac{1}{2}c'Gc$; the penalty is tuned to meet it (or the minimum attainable, with a warning). A target above the coancestry of the unconstrained (merit-only) optimum cannot be met by a non-negative penalty: a warning is issued and lambda = 0 is used.
max_coancestry	as target_coancestry, but only enforced if the unconstrained optimum exceeds it.
G	optional precomputed relationship matrix (from g_matrix()); built from x when omitted. A supplied G must be a symmetric positive semi-definite matrix with the individual ids as identical row and column names (an unnamed matrix is an error: merit is aligned to G by name).
rep	replication to read merit from (default 1).
min_contribution	contributions below this are treated as zero when listing selected parents (default 1e-4).
max_iter, tol	Iteration cap (default 10000; a whole number ≥ 1) and duality-gap tolerance (positive) for the away-step Frank-Wolfe optimizer. A warning is issued if it

does not reach `tol` within `max_iter`, meaning the contributions may be sub-optimal. `tol` is on the merit scale; when a coancestry target is tuned and the merit spread is extreme (outside $1e-6..1e6$) the search and the final solve use merit rescaled to unit spread, which leaves the contributions unchanged (the penalty scales linearly with merit).

Value

an `ocs` object: contributions (named, summing to 1), parents (ids with contribution above `min_contribution`), `n_parents`, merit (expected $c'g$, on the original merit scale), coancestry ($\frac{1}{2}c'Gc$), and `lambda`. The contributions do not depend on a constant added to every merit (the optimum lies on the simplex $\sum c = 1$); merit is centred internally before the penalty is tuned.

References

Meuwissen THE (1997) Maximizing the response of selection with a predefined rate of inbreeding. *Journal of Animal Science* 75(4):934–940. doi:10.2527/1997.754934x

See Also

`g_matrix()`, `select_ind()`, `recurrent_selection()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:40)
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
f2 <- selfcross(f1, n = 60, seed = 2)
ph <- simulate_phenotype(f2, h2 = 0.5, seed = 3) |> additive(n_qtn = 30)

# Trace the frontier: more penalty -> lower coancestry, lower gain.
hi <- optimum_contribution(ph, merit = "bv", lambda = 1)
lo <- optimum_contribution(ph, merit = "bv", lambda = 50)
c(gain_hi = hi$merit, coan_hi = hi$coancestry,
  gain_lo = lo$merit, coan_lo = lo$coancestry)
```

parentage

Parentage of a Population's individuals

Description

Every Population records its pedigree: `as_population()` makes founders, and `cross()`, `selfcross()` and `double_haploid()` add each progeny's parents. Subsetting with `[]` keeps the ancestors of the individuals kept, and `c()` pools pedigrees, so the parentage survives selection and pooling. Individuals are linked internally by a unique key rather than by display id, so pooling populations whose ids collide (and are renamed by `c()`) keeps the links intact.

Usage

```
parentage(x, ancestors = FALSE)
```

Arguments

<code>x</code>	a Population.
<code>ancestors</code>	FALSE (default): one row per individual of <code>x</code> , in order. TRUE: every individual in the recorded pedigree – <code>x</code> 's individuals and all their ancestors.

Value

A data frame with columns `id`, `mother`, `father` (parent display ids; NA for founders; both the parent for a self or doubled haploid), `generation` (0 for founders, else one more than the older parent's), `pool` (the founder pool label given to `as_population()`, NA for progeny), `design` ("founder", "cross", "self" or "dh"), and `key`, `mother_key`, `father_key`. Display ids are the names individuals were last given and need not be unique – two pools may both have a P1, and `c()` renames duplicates – so use the key columns to link individuals unambiguously (keys are stable identifiers, unchanged by renaming).

See Also

`families()`, `as_population()`, `cross()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:2)
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
f2 <- selfcross(f1, n = 3, seed = 2)
parentage(f2)
parentage(f2, ancestors = TRUE)
```

pedigree

Pedigree selection

Description

Selfs and selects each generation: the population is phenotyped, the best fraction is kept, and those are selfed to form the next generation (Bernardo 2020; Falconer & Mackay 1996).

Usage

```
pedigree(
  x,
  phenotype,
  generations = 5L,
  prop = 0.1,
```

```

    n_select = NULL,
    pop_size = NULL,
    on = "pheno",
    trait = 1L,
    direction = "high",
    seed = NULL,
    interference = NULL
)

```

Arguments

<code>x</code>	a Population, or a phenotype_sim built on one (its genotypes are used).
<code>phenotype</code>	a function mapping a Population to a realized phenotype_sim (e.g. <code>function(p) simulate_phenotype(p, ...) > additive(...)</code>). Called once per generation to score the current population, which it must build its simulation from. From the second generation on it reuses the trait the population inherited (see the section "Heritability across generations").
<code>generations</code>	number of selfing generations.
<code>prop, n_select</code>	proportion (or count) selected each generation; give one (supplying both is an error; the default <code>prop = 0.1</code> applies only when <code>n_select</code> is NULL).
<code>pop_size</code>	number of plants grown each generation (default: the founder size). Each selected line is selfed into an equal-sized family and the families are pooled to this size, so the population does not drift down as it inbreeds.
<code>on, trait, direction</code>	passed to <code>select_ind()</code> . <code>trait</code> may be a vector of traits, one per generation (recycled): tandem selection , improving one trait at a time; the history then records the trait selected on. Every requested trait must exist in the phenotype callback's simulation. <code>direction</code> is one "high" or "low" (a vector is an error). This scheme selects on the criterion alone (mass selection), so the family methods and <code>n_per_family</code> of <code>select_ind()</code> are not forwarded.
<code>seed</code>	optional RNG seed for the whole scheme: NULL (default; the ambient RNG stream is used) or one non-negative whole number. When given, the caller's RNG state is restored on exit, so a seeded scheme does not disturb the surrounding random stream.
<code>interference</code>	crossover interference model of <code>cross()</code> (see its section "Crossover interference"): NULL (default: the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with <code>nu = 2.6</code> , <code>p = 0</code>), "poisson" (Poisson crossovers, the <code>isqg</code> stream, the default before 2.0.0.9003) or <code>list(nu = , p =)</code> . It is applied to every meiosis the scheme runs, in every generation, so a whole scheme can use one meiosis model.

Value

a Population, carrying attribute history: a data frame with one row per generation and columns `generation`, `n_selected`, `differential` (the selection differential *S*) and `intensity` (the standardized selection intensity *i*), per DECISION-015.

Heritability across generations

The first generation's phenotype defines the trait: `select_ind()` stores its causal loci, effects and residual variance on the selected individuals (see `population_trait()`), and their progeny inherit it. In later generations the phenotype callback's `simulate_phenotype()` reuses that trait (`refit = FALSE`, its default for a population that carries one): the genetic value stays on the base population's scale and the residual variance is held fixed, so the heritability falls as selection and inbreeding exhaust genetic variance, as in AlphaSimR's `setPheno(varE = )` and the Bancic et al. scripts. The callback's `h2` and layer settings are then ignored (silently when they match the trait, as a callback written for the founders does).

Write `simulate_phenotype(p, ..., refit = TRUE)` in the callback for the pre-2.0.0.9003 rule: the genetic layers are re-standardised to their prop and the residual re-fitted to `h2` in **every** generation, so the genetic share of the phenotype does not decay: for a purely additive model the accuracy of mass selection, $\text{cor}(P, BV)$, stays near $\sqrt{h^2}$ (with dominance it declines as the additive part of the re-fitted genetic variance shrinks). A phenotype with a `vqtl()` or `transcriptome()` layer has no storable trait and is always re-fitted.

`history$differential` and `history$intensity` are measured on the criterion each generation was ranked on. To follow genetic gain on a frozen scale, measure it with `additive_value()` or `genotypic_value()` on the frozen effects (see the examples). The callback must build its simulation from the population it is given; one that returns a simulation backed by a different population (e.g. a closure over the base population) is an error.

References

Bernardo R (2020) *Breeding for Quantitative Traits in Plants*, 3rd ed. Stemma Press, Woodbury, Minnesota; Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow.

See Also

`select_ind()`, `recurrent_selection()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
f2 <- selfcross(f1, n = 40, seed = 9)
pheno <- function(p) {
  simulate_phenotype(p, h2 = 0.5, seed = 7) |> additive(n_qtn = 30)
}
# the trait is defined in f2; later generations reuse it (h2 can fall)
out <- pedigree(f2, pheno, generations = 3, prop = 0.2, seed = 2)
attr(out, "history")
population_trait(out)

# follow the genetic variance on the base scale
tmpl <- template_effects(pheno(f2))
var(genotypic_value(f2, tmpl$qtn, tmpl$a, tmpl$d))
var(genotypic_value(out, tmpl$qtn, tmpl$a, tmpl$d))
```

```
# the pre-2.0.0.9003 rule: re-fit to h2 = 0.5 in every generation
refit <- function(p) {
  simulate_phenotype(p, h2 = 0.5, seed = 7, refit = TRUE) |>
  additive(n_qtn = 30)
}
out2 <- pedigree(f2, refit, generations = 3, prop = 0.2, seed = 2)
attr(out2, "history")
```

phenotype_value

Phenotype on a fixed, cross-generational scale

Description

The phenotypic counterpart of [additive_value\(\)](#): each individual's phenotype is its **fixed** additive genetic value (frozen loci qtn and their effects, on the -1/0/1 dosage scale, with no per-population rescaling; with d, its total genotypic value $A + D$, see [genotypic_value\(\)](#)) plus an independent residual $e \sim N(0, \text{var_e})$ on a **fixed** residual-variance parameter `var_e`. Because neither the genetic scale nor `var_e` is re-fit to the scored population, the **parametric (population) heritability** $\text{Var}(g) / (\text{Var}(g) + \text{var_e})$ *declines* as selection exhausts genetic variance – which is exactly what a faithful cross-generation on = "pheno" selection driver needs, and what [genetic_values\(\)](#) / [simulate_phenotype\(\)](#) cannot express (they re-scale the genetic layer to its target prop on every population, so the genetic share is re-fixed each generation: with an additive-only genetic layer, selection accuracy stays near $\sqrt{h^2}$ under this re-standardization, and it declines only when dominance enters). (`var_e` is the residual *variance parameter*, not a sample-standardized value: e is a genuine normal draw, so its realized sample variance scatters around `var_e` and $\text{Cov}(g, e) \sim 0$ in expectation. The package's sample statistic $\text{Var}(g)/\text{Var}(y)$ therefore tracks the parametric heritability up to finite-sample scatter, and is not forced to it.)

Usage

```
phenotype_value(
  x,
  qtn,
  effect,
  h2 = NULL,
  var_e = NULL,
  ref = NULL,
  seed = NULL,
  d = NULL,
  gxe = NULL,
  gxe_intercept = 0,
  env = NULL,
  var_env = 0
)
```

Arguments

x	a Population, a Population-backed phenotype_sim, or a marker x individual dosage matrix coded -1/0/1 (marker names as row names when qtn is given by name; individual ids as column names).
qtn	the causal loci, as marker names (matched against the map) or integer marker (row) indices.
effect	a finite numeric vector of per-locus additive effects, one per entry of qtn and in the same order.
h2	target heritability in $(0, 1]$, used with ref to set a fixed residual variance: of the additive value (narrow-sense) by default, or of the total genotypic value $A + D$ when d is given (broad-sense, see d). Give exactly one of h2 or var_e.
var_e	fixed residual variance (a single non-negative number), on the same scale as <code>Var(additive_value(x, qtn, effect))</code> . Give exactly one of h2 or var_e.
ref	optional reference population/genotypes (same forms as x) whose genetic variance converts h2 to var_e; defaults to x. Ignored when var_e is given.
seed	optional seed for the residual draw – NULL or one non-negative whole number (the RNG state is restored afterwards), for reproducible phenotypes.
d	optional dominance deviations of the loci (one value, or one per locus): the genetic part is then the fixed total genotypic value $A + D$ of <code>genotypic_value()</code> (with effect as its a), and h2 is the heritability of that total value in ref – a broad-sense heritability. Default NULL: additive only.
gxe	optional per-locus genotype-by-environment (slope) effects, one per entry of qtn and in the same order (AlphaSimR <code>addTraitAG()</code> 's <code>gxeEff</code>). The phenotype then follows AlphaSimR's <code>setPheno()</code> for that trait:

$$y_i = g_i + s_i w + e_i, \quad w = \Phi^{-1}(\text{env}; 0, \sigma_w),$$

where s_i is `gxe_value()`(x, qtn, gxe, gxe_intercept) and $\sigma_w = \sqrt{\text{var_env}}$, or 1 when `var_env = 0`. One covariate w is shared by every individual scored in the call (one trial). The G x E term is not part of the genetic value: h2 converts to var_e from the genetic variance alone (as AlphaSimR's `setPheno(h2 =)` does from `varA` and `varG`), and the `genetic_value` attribute excludes it. Default NULL: no G x E, and the result and random stream are unchanged.

gxe_intercept	the slope intercept (AlphaSimR <code>gxeInt</code> ; see <code>gxe_value()</code>). Only with gxe.
env	the trial's environment as a probability in $(0, 1)$: the quantile of the covariate (AlphaSimR <code>setPheno(p =)</code>). NULL (default) draws it uniformly, before the residual, from the same seeded stream. Only with gxe.
var_env	the variance of the environmental covariate (AlphaSimR <code>varEnv</code>). With <code>var_env > 0</code> and a mean slope of 1 (<code>gxe_intercept</code> set as AlphaSimR does), w is also a main effect of the environment shared by all individuals; with 0 (the AlphaSimR default) the covariate has standard deviation 1. Only with gxe.

Details

Supply **exactly one** of h2 or var_e to set that fixed residual variance:

- `var_e` – the residual variance directly. This is the robust choice for multi-generation use: compute it once at the base generation and pass the same value every cycle so it is truly frozen. `pedigree()` and `recurrent_selection()` do this by default (`residual = "fixed"`).
- `h2` – a target heritability, converted to a residual variance $\text{var_e} = \text{Var}(g_{\text{ref}}) (1 - h2) / h2$ from a reference population's genetic variance. `ref` names that reference (default: `x` itself). For cross-generation use pass the **base** population as `ref` (or precompute `var_e`); `h2` with the default `ref = x` re-derives `var_e` from each scored population and so does *not* freeze it across generations.

Value

a named numeric vector of phenotypes, one per individual, with attributes `var_e` (the fixed residual variance used) and `genetic_value` (the fixed additive – or, with `d`, total genotypic – values). With `gxe`, also `gxe_value` (the slopes s_i), `env` (the quantile used) and `env_value` (the covariate w).

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow (heritability $h^2 = V_A / (V_A + V_E)$; with `d`, the broad-sense $H^2 = V_G / (V_G + V_E)$); Lynch M, Walsh B (1998) *Genetics and Analysis of Quantitative Traits*. Sinauer, Sunderland, MA. Gaynor RC, Gorjanc G, Hickey JM (2021) AlphaSimR: an R package for breeding program simulations. *G3* 11(2):jkaa017. doi:10.1093/g3journal/jkaa017 (the additive-by-environment trait; the exact formula of `gxe` follows the AlphaSimR 2.1.0 source of `SimParam$addTraitAG()` and `calcPheno()`).

See Also

[additive_value\(\)](#), [genetic_values\(\)](#), [select_ind\(\)](#), [simulate_phenotype\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
# Freeze the residual variance from the base population at h2 = 0.5, then reuse
# it so later generations' heritability can decline as variance is exhausted.
y0 <- phenotype_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
                      h2 = 0.5, seed = 1)
ve <- attr(y0, "var_e")
# a descendant population would then be scored with the same frozen var_e:
# phenotype_value(descendants, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
#               var_e = ve, seed = 2)
head(y0)

# The same trait with G x E slopes in a low (env = 0.1) and a high
# (env = 0.9) value of the environmental covariate. With the defaults the
# covariate has no main effect (mean slope near 0): only the ranking changes.
b <- c(0.2, -0.1, 0.3)
y_low <- phenotype_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
                        var_e = ve, gxe = b, env = 0.1, seed = 3)
y_high <- phenotype_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
                        var_e = ve, gxe = b, env = 0.9, seed = 3)
cor(y_low, y_high)
```

```
# A main effect of the environment as well (AlphaSimR varEnv > 0): set the
# mean slope of the base population to 1, so env = 0.9 is the better trial.
int <- 1 - mean(gxe_value(pop, qtn = c(1, 5, 9), effect = b))
y_poor <- phenotype_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
                          var_e = ve, gxe = b, gxe_intercept = int,
                          env = 0.1, var_env = 2, seed = 3)
y_good <- phenotype_value(pop, qtn = c(1, 5, 9), effect = c(0.5, -1, 2),
                          var_e = ve, gxe = b, gxe_intercept = int,
                          env = 0.9, var_env = 2, seed = 3)
mean(y_good) - mean(y_poor) # 2 * qnorm(0.9, sd = sqrt(2)) with equal residuals
```

phenotypes_long

Realized phenotypes in long format

Description

The canonical output of the v2 grammar: one row per individual x trait x rep with columns id, trait, rep, value.

Usage

```
phenotypes_long(sim)
```

Arguments

sim a phenotype_sim.

Value

a data frame in long format.

See Also

[phenotypes_wide\(\)](#), [write_phenotypes\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3)

long <- phenotypes_long(ph)
head(long)

# One row per individual x trait x rep, which is the shape most
# mixed-model and plotting tools expect.
table(long$trait)
```

phenotypes_wide	<i>Realized phenotypes in wide format</i>
-----------------	---

Description

One row per individual x rep; one column per trait.

Usage

```
phenotypes_wide(sim)
```

Arguments

sim a phenotype_sim.

Value

a data frame in wide format.

See Also

[phenotypes_long\(\)](#), [write_phenotypes\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3)

wide <- phenotypes_wide(ph)
head(wide)

# One column per trait, which makes cross-trait comparisons direct.
round(cor(wide$Trait_1, wide$Trait_2), 2)
```

plot.phenotype_sim	<i>Diagnostic plots for a simulated phenotype</i>
--------------------	---

Description

A quick visual summary of what a simulation produced, in base graphics (no extra dependencies). Four panels:

"variance" the variance partition – each layer's proportion plus the residual – as a stacked bar per trait.

"hist" the distribution of the realized phenotype (first trait).

"effects" the QTN effect sizes, from [qtn_table\(\)](#).

"cor" the genetic-value scatter between the first two traits (only when n_traits > 1); otherwise realized vs target heritability.

Usage

```
## S3 method for class 'phenotype_sim'
plot(x, which = c("variance", "hist", "effects", "cor"), ...)
```

Arguments

<code>x</code>	a <code>phenotype_sim</code> .
<code>which</code>	panels to draw; any of "variance", "hist", "effects", "cor". Defaults to all four in a 2x2 layout.
<code>...</code>	named graphical parameters accepted by <code>graphics::par()</code> .

Value

`x`, invisibly.

See Also

`genetic_values()`, `qtn_table()`.

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, h2 = 0.5,
                        seed = 1) |>
  additive(prop = 0.4, n_qtn = 8) |>
  dominance(prop = 0.1)
plot(ph)
```

population_from_haplotypes

Create a population from phased haplotypes

Description

`as_population()` has to guess the phase of every heterozygote, because a -1/0/1 dosage matrix does not say which allele sits on which homologous chromosome. When the phase is *known* (simulated founders from a coalescent or another engine, statistically phased data, a population exported from a breeding simulation) `population_from_haplotypes()` builds the Population directly from the two haplotype (strand) matrices, so nothing has to be guessed and the cis/trans slots of the object never have to be replaced by hand.

Usage

```
population_from_haplotypes(
  cis,
  trans,
  map,
  ids = NULL,
  pool = NA_character_,
  individuals_in_rows = FALSE
)
```

Arguments

- | | |
|---------------------|--|
| cis, trans | the two haplotype matrices, with the same dimensions: the first and second homologous strand of every individual. Entries 0/1 (numeric, integer or logical), no missing values, see the coding section. |
| map | a data frame describing the markers, one row per marker in the order of the haplotype matrices, with the columns snp (unique marker names), chr, pos (physical position, non-negative) and cm (genetic position in centiMorgans , non-decreasing within a chromosome), exactly as for as_population() (the same checks, including the centiMorgan-units warning, apply; see its section on the genetic map). Optional columns: allele (the "allele1/allele2" label of as_numeric()) and counted (character: the allele the haplotype value 1 stands for, NA where unknown; each entry a single non-empty allele symbol, one of the two alleles of allele when that label exists; "" and numeric values are rejected). The map of an existing population (pop\$map) is a valid argument. Other columns are ignored. |
| ids | optional character vector of individual ids, one per individual. Default: the individual names of the matrices (column names, or row names with <code>individuals_in_rows = TRUE</code>). They are required, non-missing, non-empty and unique, from one of the two sources. If cis and trans both carry individual names these must agree; ids takes precedence over them. |
| pool | optional label for the founder pool, as in as_population() . Default NA. |
| individuals_in_rows | logical. FALSE (default): the matrices are markers by individuals. TRUE: individuals by markers. |

Value

A Population whose individuals are recorded as pedigree founders, identical in structure to the result of [as_population\(\)](#) (so it works in [cross\(\)](#), [selfcross\(\)](#), [double_haploid\(\)](#), [g_matrix\(\)](#), [simulate_phenotype\(\)](#), [a_matrix\(\)](#), [c.Population\(\)](#), ...). Its `dosages()` equal `cis + trans - 1`.

Allele coding of the haplotype entries

An entry is 1 when the strand carries the allele that the package counts as +1, and 0 when it carries the other allele (the same convention as the numeric format of [as_numeric\(\)](#)). The dosage of an individual at a marker is therefore `cis + trans - 1`, in -1/0/1 (`dosages(population_from_haplotypes(cis,`

trans, map))), exactly the coding `as_population()` expects. Which physical allele (A, G, ...) the 1 stands for is recorded only if the map has a counted column (see map); without it the orientation guard of `cross()` and `c.Population()` falls back to the allele labels, as it does for numeric data without a record. Give a counted column whenever populations built from different sources will be crossed or pooled.

Which strand is which

cis and trans are the two homologous chromosomes of each individual. The package gives them no parental meaning (maternal/paternal) and meiosis treats them symmetrically, so either assignment is correct as long as the two strands of one individual stay in the same column of both matrices.

Layout

Following `dosages()`, `as_numeric()` and the Population object itself, the matrices are **markers by individuals** by default (one column per individual). Set `individuals_in_rows = TRUE` for the more common individuals-by-markers layout of phased-haplotype exports; the rows then name the individuals and the columns the markers.

See Also

`haplotypes()` (the inverse), `as_population()` (genotypes without phase), `cross()`, `dosages()`.

Examples

```
map <- data.frame(snp = paste0("m", 1:6), chr = rep(1:2, each = 3),
                  pos = rep(c(100, 200, 300), 2),
                  cm = rep(c(0, 10, 20), 2))
# two individuals; ind1 is heterozygous at m2 and m5, with known phase
cis  <- cbind(ind1 = c(1, 1, 0, 1, 0, 0), ind2 = c(0, 0, 0, 1, 1, 1))
trans <- cbind(ind1 = c(1, 0, 0, 1, 1, 0), ind2 = c(0, 1, 0, 1, 1, 1))
pop <- population_from_haplotypes(cis, trans, map)
dosages(pop)
haplotypes(pop)$cis
progeny <- cross(pop[1], pop[2], n = 3, seed = 1)
dosages(progeny)
```

population_trait

The trait a Population carries

Description

A trait is defined once, in a base population, and then travels with its descendants: `select_ind()` stores the trait of the phenotype it ranked on the Population it returns, and `cross()`, `selfcross()`, `double_haploid()`, `mate()` (and every function built on them) copy it to the progeny when all parents carry the same one. `simulate_phenotype()` on a Population that carries a trait reuses it (`refit = FALSE`, the default for such a population): the same causal loci and effects on the base

population's genetic scale, and the base population's residual variance. The genetic variance can then change from one generation to the next while the residual variance stays fixed, so the heritability changes too (AlphaSimR's `setPheno(varE = )`).

Usage

```
population_trait(x)
```

Arguments

`x` a Population.

Details

Additive, dominance and `epistasis()` traits are stored (each epistatic locus keeps its base-population centering, so an epistatic term is a fixed function of the genotype). A phenotype with a `vqtl()` or `transcriptome()` layer (with `prop > 0`), or `architecture = "complex"`, has no fixed form, so its selected population carries no trait and every generation is re-fitted.

Value

NULL when `x` carries no trait; otherwise a data frame with one row per trait: `trait`, `h2` (the genetic share defined in the base population) and `var_e` (the fixed single-record residual variance, on the base population's phenotypic scale, where the phenotypic variance was 1).

See Also

`simulate_phenotype()` (argument `refit`), `select_ind()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:20)
f2 <- selfcross(cross(pop[1], pop[2], n = 1, seed = 1), n = 60, seed = 2)
sim <- simulate_phenotype(f2, h2 = 0.4, n_qtn = 20, seed = 3)
top <- select_ind(sim, prop = 0.2)
population_trait(top)
# the progeny carry the trait; their phenotype reuses it
f3 <- selfcross(top[1], n = 30, seed = 4)
population_trait(f3)
sim3 <- simulate_phenotype(f3, seed = 5)
```

predict.transcriptome_sim

Apply a fixed transcriptome architecture to a new population

Description

Reuse the reference-calibrated architecture of a transcriptome_sim to generate expression for a **different** set of genotypes – descendants, a cross, or a selected subset – without re-estimating any centering or scaling constant. The genetic component is reconstructed from the stored effect tables applied to dosages centered on the **reference** allele frequencies (object\$reference\$marker_mean), so a given genotype maps to the same genetic expression regardless of the population it sits in; realized heritability in the new population is emergent, not re-forced (the fixed-scale principle of additive_value()). The non-genetic residual is not a function of genotype, so it is drawn afresh; set residual = FALSE for noiseless genetic expression.

Usage

```
## S3 method for class 'transcriptome_sim'
predict(object, geno, seed = NULL, residual = TRUE, ...)
```

Arguments

object	a transcriptome_sim from simulate_transcriptome() .
geno	new genotypes – a Population, a Population-backed phenotype_sim, a numeric-format genotype data frame, or an individuals-by-markers dosage matrix (as for simulate_transcriptome()) – carrying (at least) every marker named in the architecture’s eQTL tables, in the same coding and effect-allele orientation as the reference. Only marker names and the -1/0/1 dosage coding are checked; the architecture does not store allele labels, so a marker whose reference/alternate alleles are swapped (same name, flipped dosage) would pass the check yet produce different genetic values. When combining datasets, harmonize effect-allele orientation first.
seed	optional seed for the fresh residual draws (NULL or one non-negative whole number).
residual	add a freshly drawn non-genetic residual (default TRUE); FALSE returns noiseless genetic expression. Must be TRUE or FALSE.
...	unused.

Value

a new transcriptome_sim for the new individuals, carrying the same architecture (effect tables, per-gene targets, reference constants, and the calibration of a mimic fit) with expression / genetic_expression and a variance budget realized on the new population. At least two new individuals are needed: the realized variance budget (and h2_realized, h2_allocated, cis_fraction_realized) is undefined for one, so a single individual is rejected with an error.

predict_ebv

*Predicted breeding values by BLUP with known variance components***Description**

Best linear unbiased prediction (Henderson 1975) of every individual's breeding value from the observable phenotypes of a phenotyped subset and a relationship matrix: genomic ("gblup", $K = G$ from `g_matrix()`, VanRaden 2008) or pedigree ("pedigree", $K = A$ from `a_matrix()`). Individuals without a record – selection candidates – are predicted through their relationships to the phenotyped ones.

Usage

```
predict_ebv(
  x,
  pheno,
  method = c("gblup", "pedigree"),
  h2 = NULL,
  var_a = NULL,
  var_e = NULL,
  ref = NULL,
  K = NULL,
  base_freq = NULL,
  ridge = 0
)
```

Arguments

x	a Population holding every individual to predict (phenotyped and not).
pheno	a numeric vector of phenotypes named by id (a subset of x), or, for multi-trait BLUP, a numeric individuals x traits matrix (or data frame) with row names ids of x, column names the traits and NA for a missing record.
method	"gblup" or "pedigree".
h2, var_a, var_e	the variance components: either h2 (then $\sigma_A^2 = h^2\sigma_P^2$, σ_P^2 the variance of ref, default pheno) or both var_a (base additive variance) and var_e. For multi-trait BLUP, var_a and var_e are required: the traits x traits base additive genetic and residual covariance matrices (named by trait, or in pheno's column order), each positive semidefinite with positive variances; h2 and ref are not used.
ref	optional numeric vector (at least two finite values) of base-population phenotypes setting σ_P^2 for h2; it is an error without h2 (with var_a and var_e it would be silently unused). Not used by multi-trait BLUP.
K	optional precomputed relationship matrix over x's individuals overriding the one method builds: finite, symmetric and positive semidefinite (it is a covariance matrix up to σ_A^2 ; only floating-point rounding below zero is tolerated – $n\epsilon$ times the largest eigenvalue on the correlation scale – and $K + (\sigma_e^2/\sigma_A^2)I$ over the

phenotyped individuals must be positive definite), with row and column names both the ids of x, in the same order (it is reordered to x's order).

base_freq, ridge
passed to `g_matrix()` for "gblup".

Details

The model is $y = 1\mu + Zu + e$, $u \sim N(0, K\sigma_A^2)$, $e \sim N(0, I\sigma_e^2)$, with the variance components **known** – a simulation knows them – so this is a deterministic linear solve, with no variance-component estimation. It is computed in the equivalent generalized least-squares form $\hat{u} = KZ'(ZKZ' + \lambda I)^{-1}(y - 1\hat{\mu})$, $\lambda = \sigma_e^2/\sigma_A^2$, which needs no inverse of K (so a singular genomic matrix is fine). Reliability is $1 - PEV_i/(K_{ii}\sigma_A^2)$.

The result is an **estimate** from phenotypes: pass observable records (e.g. `phenotype_value()` or a simulation's phenotypes), never true genetic or breeding values, and compare it with the truth using `prediction_accuracy()`. Across selection cycles keep the variance components and, for "gblup", the base allele frequencies (base_freq) fixed at the base population's; per-cycle values inflate the apparent accuracy. Records of every individual that selection was based on should be included, or predictions are biased (Henderson 1975).

Scope (single trait). One trait, one record per individual (a pheno name must not be repeated, and two ids of x that are the same individual – e.g. after `c(pop, pop[1])` – may not both carry a record), an intercept as the only fixed effect, and no missing phenotypes (they are refused, not dropped); a single record gives every EBV zero and every reliability zero – NA for an individual whose prior variance $K_{ii}\sigma_A^2$ is zero, where reliability is undefined – since one record cannot separate the mean from the breeding value. Repeated-record and single-step models are not covered; for several traits see *Multi-trait BLUP* below. With h^2 , $\sigma_A^2 = h^2\sigma_P^2$ and $\sigma_e^2 = (1 - h^2)\sigma_P^2$, with σ_P^2 the sample variance of ref or, by default, of the **phenotyped subset** pheno. Because both components are proportional to σ_P^2 , the ratio $\lambda = (1 - h^2)/h^2$, the EBVs and the reliabilities do not depend on which σ_P^2 is used (multiplying it by any factor leaves them unchanged); only the reported absolute var_a and var_e attributes scale with it. So ref matters when those absolute variances do – e.g. after selection the phenotyped subset has a truncated variance, and the base population's phenotypes as ref (or var_a and var_e directly) keep the reported components at the base population's scale.

Multi-trait BLUP. Give pheno as an individuals x traits matrix (row names ids, column names traits, NA for a trait an individual was not recorded on) and var_a, var_e as the known traits x traits base additive genetic covariance matrix G_0 and residual covariance matrix R_0 . The model (Henderson & Quaas 1976) is $y_t = 1\mu_t + Z_t u_t + e_t$ for every trait t , with $Var(u) = G_0 \otimes K$ and residuals correlated only within an individual, $Cov(e_{it}, e_{js}) = R_{0,ts}$ if $i = j$ and 0 otherwise. It is solved in the same generalized least-squares form over the stacked records, $\hat{u} = Cov(u, y)V^{-1}(y - X\hat{\mu})$ with $V = Var(y)$, so K and G_0 may be singular (e.g. a genetic correlation of 1). A trait is predicted for every individual, including those not recorded on it, through the genetic covariances with the traits that were recorded; an individual's own record on a correlated trait also enters through the residual covariance. The records need not be complete, but every trait needs at least one. With uncorrelated traits (diagonal G_0 and R_0) the result is single-trait BLUP of each trait. The BLUP of an aggregate genotype $H = a'u$ is $a'\hat{u}$, so rank on ebv %*% a (e.g. `select_ind(method = "mass", on = ...)`), or pass the matrix to `select_ind(method = "culling", on = ...)`.

Value

A numeric vector of predicted breeding values named by x's ids, with attributes reliability, mu (the estimated mean), lambda, var_a, var_e, method and, for "gblup" with the genomic matrix built here (no K supplied) and ridge = 0, marker_effects (back-solved, on the $M - 2p$ gene-content scale of `g_matrix()`; a supplied K carries no marker scale, so none are returned). For multi-trait BLUP, an individuals x traits matrix, with reliability the matching matrix, mu a vector named by trait, var_a and var_e the covariance matrices used, method, and marker_effects (markers x traits) under the same condition; no lambda.

References

- Henderson CR, Quaas RL (1976) Multiple trait evaluation using relatives' records. *Journal of Animal Science* 43:1188–1197. doi:10.2527/jas1976.4361188x
- Henderson CR (1975) Best linear unbiased estimation and prediction under a selection model. *Biometrics* 31:423–447. doi:10.2307/2529430
- VanRaden PM (2008) Efficient methods to compute genomic predictions. *Journal of Dairy Science* 91:4414–4423. doi:10.3168/jds.20070980

See Also

`prediction_accuracy()`, `a_matrix()`, `g_matrix()`, `select_ind()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:60)
q <- c("ss196442916", "ss196439337", "ss196480535")
y <- phenotype_value(pop, q, c(1, 0.5, 0.25), h2 = 0.5, seed = 1)
ebv <- predict_ebv(pop, y[1:40], h2 = 0.5)      # 20 unphenotyped candidates
head(ebv)

# multi-trait (illustrative covariances): a second, correlated trait recorded
# on every individual, the first unrecorded on the 20 candidates
y2 <- phenotype_value(pop, q, c(0.8, 0.6, 0), h2 = 0.5, seed = 2)
Y <- cbind(t1 = y, t2 = y2)
Y[41:60, "t1"] <- NA
tr <- c("t1", "t2")
G0 <- matrix(c(1, 0.6, 0.6, 1), 2, dimnames = list(tr, tr))
R0 <- diag(2); dimnames(R0) <- list(tr, tr)
ebv2 <- predict_ebv(pop, Y, var_a = G0, var_e = R0)
head(ebv2)
```

Description

The realized accuracy $cor(\hat{u}, u)$ of predictions against the true breeding values a simulation knows, with the regression slope of the truth on the prediction (1 for an unbiased predictor, BLUP's $E[u | \hat{u}] = \hat{u}$ property; below 1 means over-dispersed predictions).

Usage

```
prediction_accuracy(ebv, truth)
```

Arguments

ebv	predicted values (named by id).
truth	true breeding values (named by id; e.g. from a simulation's on = "bv" criterion). Matched by name when both are named.

Value

A named numeric vector: accuracy, slope, n. Named vectors are matched by name (each name may occur once; a duplicated name in either vector is an error, even when the other vector is unnamed) and only the common, finite pairs are scored. If ebv or truth is constant the correlation is undefined: accuracy is NA (and slope is NA for a constant ebv), with a warning.

See Also

[predict_ebv\(\)](#)

Examples

```
prediction_accuracy(c(a = 1, b = 2, c = 3), c(a = 1.2, b = 1.9, c = 3.3))
```

progeny_test

Progeny test: score parents by the mean of their progeny

Description

Mates each parent to n_progeny distinct mates drawn at random, without replacement, from mates, one progeny per mating, so each parent gets a half-sib family: a parent is never its own mate, each individual of mates counts once (by pedigree identity, see [parentage\(\)](#)), and n_progeny must not exceed the distinct mates available to a parent. Individuals are identified by pedigree key, which includes the founders' pool label (the same genotypes under two different pool labels are two individuals). Scores every progeny on the frozen architecture (qtn, a, d), optionally adds an independent residual (exactly one of h2 / var_e, as in [phenotype_value\(\)](#)), and reports each parent's progeny mean. The progeny Population (pedigree recorded, see [families\(\)](#)) is returned for phenotyping or selection within and among families.

Usage

```
progeny_test(
  parents,
  mates,
  qtn,
  a,
  d = 0,
  n_progeny,
  h2 = NULL,
  var_e = NULL,
  ref = NULL,
  seed = NULL,
  interference = NULL
)
```

Arguments

parents	a Population of parents to test, each individual once (the same individual under a second id is an error).
mates	a Population to draw random mates from, on the same map.
qtn, a, d	frozen loci and effects, as for <code>genotypic_value()</code> ; d may be a single value (default 0).
n_progeny	progeny (and distinct mates) per parent.
h2, var_e, ref	optional residual: at most one of h2 / var_e; h2 is the heritability of the genotypic value in ref (default: the progeny), a <i>broad-sense</i> value when $d \neq 0$ (the accuracy formula above is for an additive trait, so with dominance the h2 given is not the h2 in it). ref is an error without h2.
seed	optional RNG seed (mates, meioses and residuals are drawn in that order from one stream); the caller's random-number stream is left as it was found.
interference	crossover interference model of <code>cross()</code> (see its section "Crossover interference"): NULL (default: the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the isqg stream, the default before 2.0.0.9003) or <code>list(nu = , p =)</code> . Used for the meioses that make the progeny.

Details

When every parent draws its mates from the same population – mates separate from parents – a parent's expected half-sib progeny mean (over the random choice of mates) is half its breeding value **in the mates' population** plus a constant common to all parents. (If parents are among the mates, each parent's own genotype is left out of its mate pool, so the reference frequencies, and the constant, differ slightly among parents – negligibly for a large mate population.) That breeding value uses the average effects $\alpha = a + d(q - p)$ at the mates' allele frequencies (Falconer & Mackay 1996), so dominance enters the *parent-dependent* part of the progeny mean – the part that ranks parents – through them; what does not enter is a separate dominance deviation of the parent itself. Dominance also remains in the constant common to all parents (under Hardy-Weinberg at the mates' frequencies, with p the frequency of the counted allele, that constant is $a(p - q) + 2dpq$ per locus).

For example, with $a = 0$ and $p = 1/2$, $\alpha = 0$ and every parent has the same expected progeny mean, which equals $d/2$ per locus, not zero. (With a fixed tester population this is combining ability toward that tester, see `combining_ability()`.)

For an additive trait with heritability h^2 the accuracy of a progeny test on n half-sib records is, under the classical half-sib assumptions – the mates a large random-mating, non-inbred population in Hardy-Weinberg and linkage equilibrium, each progeny by a different, independently drawn mate, so the only genetic covariance among a parent's progeny is the parent's own contribution – (this package's derivation): each progeny receives half the parent's breeding value, so $Cov(A, \bar{y}) = V_A/2$; half-sibs share a quarter of their additive variance, so their phenotypic correlation is $t = h^2/4$ and $Var(\bar{y}) = V_P[1 + (n-1)t]/n$; hence

$$r = \frac{V_A/2}{\sqrt{V_A Var(\bar{y})}} = \sqrt{nh^2/(4 + (n-1)h^2)},$$

which rises to 1 as n grows. It exceeds the accuracy of selection on own performance, $\sqrt{h^2}$, once $n > (4 - h^2)/(1 - h^2)$ (e.g. more than 4.75 progeny at $h^2 = 0.2$) – why progeny testing, given enough progeny, pays for low-heritability traits. With few, inbred or shared mates (e.g. every parent mated to the same small set) the mates' contribution is no longer independent noise and the realized accuracy departs from this value.

Value

A data frame with `id` (parent), `progeny_mean` and `n`, with attributes `progeny` (the progeny Population), `records` (each progeny's value, named by `id`) and `var_e` (the residual variance used, 0 for none).

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow – the average effect $\alpha = a + d(q - p)$ and the breeding value.

See Also

`combining_ability()`, `families()`, `select_ind()`

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:30)
q <- c("ss196442916", "ss196439337", "ss196480535")
pt <- progeny_test(pop[1:5], pop[6:30], qtn = q, a = c(1, 0.5, 0.25),
                  n_progeny = 10, h2 = 0.3, seed = 1)
pt
```

qtn_table

*The QTNs behind a simulation, with their effects***Description**

One row per QTN per trait per layer, naming the marker and the effect it was given. This is the v2 equivalent of the v1 Additive_QTNs.txt / QTN_effects_summary.txt files, assembled from the simulation object rather than written to disk.

Usage

```
qtn_table(sim, rep = 1L)
```

Arguments

sim	a phenotype_sim.
rep	replication whose QTN architecture to report (default 1). This matters when the simulation used vary_qtn = TRUE.

Details

Epistatic QTNs come in interacting sets, so each member of a set gets its own row, sharing the set's effect and identified by a common set number. For every other layer type set is NA.

Under architecture = "ld" the two traits have distinct causal loci that are in linkage disequilibrium. QTN_t1 and QTN_t2 name the linked pair's causal SNP for trait 1 and trait 2 (shown on both of the pair's rows so each row is self-contained), and ld_r2 is the squared correlation between them. All three are NA for the other architectures. (For ld_type = "indirect" the hidden cause-of-LD locus is not shown here – it is not a QTN – but is available programmatically on the layer.)

A transcriptome() layer's causal features are **genes**, not markers, so its rows carry the gene identifier in the snp column, the per-gene slope in effect, and the gene's marginal variance share in var_explained; the marker-only columns chr, pos and maf are NA. Filter with subset(qtn_table(ph), layer == "transcriptome") to isolate them.

Value

A data frame with columns trait, layer, set, snp, chr, pos, maf, effect, d, var_explained, QTN_t1, QTN_t2 and ld_r2, or a zero-row frame when no layers have been added. effect is the additive effect (a for an orthogonal layer); d is the per-locus dominance deviation of an orthogonal layer and NA for every other layer. For transcriptome layers snp holds the gene name and chr/pos/maf are NA.

See Also

[genetic_values\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.5, seed = 1) |>
  additive(prop = 0.3, n_qtn = 3) |>
  epistasis(prop = 0.2, n_pairs = 2)

qtn_table(ph)

# Which markers were used, and how large were the effects?
subset(qtn_table(ph), layer == "additive")
```

```
recurrent_parent_recovery
```

Marker-observed recovery of the recurrent-parent genome

Description

For each individual, the weighted mean over informative markers of its recurrent-parent allele fraction: 1 for a recurrent homozygote, 1/2 for a heterozygote, 0 for a donor homozygote. A marker is informative when the recurrent and donor founders are homozygous for alternate alleles; the others carry no information about which parent an allele came from and are dropped. A Population carries no per-locus founder ancestry, so this is recovery of recurrent-parent **marker alleles**, not identity by descent; the weighted mean is this package's estimator. Expected value in an unselected cohort after t backcrosses following the F1: $1 - (1/2)^{(t+1)}$ (Frisch & Melchinger 2005; see [mabc_select\(\)](#)). pop, recurrent and donor must share one allele coding (see [mabc_select\(\)](#)).

Usage

```
recurrent_parent_recovery(
  pop,
  recurrent,
  donor,
  markers = NULL,
  weights = NULL
)
```

Arguments

pop	the candidate Population (e.g. a BC1 family).
recurrent, donor	single-individual Populations: the recurrent parent and the donor of the target allele, on the same marker map as pop.
markers	markers to score, by name or index (default: all informative markers).
weights	NULL (equal), "interval" (the cM length of genome nearer to each marker than to any other scored marker on its chromosome, end markers covering out to the chromosome's first and last mapped marker; see marker_weights in mabc_select()), or a numeric vector over all map markers (named by marker, or in map order).

Value

A named numeric vector (one value per individual of pop) with attribute `n_markers`, the number of informative markers scored.

References

Frisch M, Melchinger AE (2005) Selection theory for marker-assisted backcrossing. *Genetics* 170(2):909–917. doi:10.1534/genetics.104.035451

See Also

`mabc_select()`.

Examples

```
g <- data.frame(snp = paste0("m", 1:40), allele = "A/G",
               chr = rep(1:2, each = 20), pos = rep(1:20, 2) * 1e6,
               cm = rep(seq(0, 95, by = 5), 2), P1 = 1L, P2 = -1L)
founders <- as_population(g)
f1 <- cross(founders[1], founders[2], seed = 1)
bc1 <- cross(f1, founders[1], n = 200, seed = 2)
mean(recurrent_parent_recovery(bc1, founders[1], founders[2])) # about 0.75
```

recurrent_selection	<i>Recurrent selection</i>
---------------------	----------------------------

Description

Cycles of select-then-intermate for population improvement (Bernardo 2020; Falconer & Mackay 1996): each cycle the population is phenotyped, the best parents are selected, and they are intercrossed to form the next cycle's population. As in `pedigree()`, the founders' phenotype defines the trait and later cycles reuse it, so heritability changes as selection changes the genetic variance (see the section "Heritability across generations" of `pedigree()`).

Usage

```
recurrent_selection(
  x,
  phenotype,
  cycles = 3L,
  n_parents = 10L,
  n_crosses = NULL,
  progeny_per_cross = 10L,
  on = "pheno",
  trait = 1L,
  direction = "high",
  seed = NULL,
  interference = NULL
)
```

Arguments

x	a Population, or a phenotype_sim built on one (its genotypes are used).
phenotype	a function mapping a Population to a realized phenotype_sim (e.g. function(p) simulate_phenotype(p, ...) > additive(...)). Called once per generation to score the current population, which it must build its simulation from. From the second generation on it reuses the trait the population inherited (see the section "Heritability across generations").
cycles	number of selection cycles.
n_parents	number of parents selected each cycle. It must be smaller than the number of plants phenotyped in the cycle (the founder population, then n_crosses * progeny_per_cross); otherwise no selection would take place and the call is an error.
n_crosses	number of crosses among the selected parents (default: n_parents). Each cross draws an independent random pair of selected parents (random mating with replacement across crosses), so with few crosses some selected parents may not be sampled.
progeny_per_cross	progeny produced per cross.
on, trait, direction	passed to select_ind() . trait may be a vector of traits, one per generation (recycled): tandem selection , improving one trait at a time; the history then records the trait selected on. Every requested trait must exist in the phenotype callback's simulation. direction is one "high" or "low" (a vector is an error). This scheme selects on the criterion alone (mass selection), so the family methods and n_per_family of select_ind() are not forwarded.
seed	optional RNG seed for the whole scheme: NULL (default; the ambient RNG stream is used) or one non-negative whole number. When given, the caller's RNG state is restored on exit, so a seeded scheme does not disturb the surrounding random stream.
interference	crossover interference model of cross() (see its section "Crossover interference"): NULL (default: the option simplePHENOTYPES.interference if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the isqg stream, the default before 2.0.0.9003) or list($\nu =$, $p =$). It is applied to every meiosis the scheme runs, in every generation, so a whole scheme can use one meiosis model.

Value

a Population (the final cycle), carrying attribute history.

References

Bernardo R (2020) *Breeding for Quantitative Traits in Plants*, 3rd ed. Stemma Press, Woodbury, Minnesota; Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow.

See Also

[select_ind\(\)](#), [pedigree\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:30)
pheno <- function(p) {
  simulate_phenotype(p, h2 = 0.5, seed = 7) |> additive(n_qtn = 50)
}
out <- recurrent_selection(pop, pheno, cycles = 3, n_parents = 10,
  progeny_per_cross = 10, seed = 2)
attr(out, "history")
```

sample_parents

Draw parents for mating in proportion to OCS contributions

Description

Turns continuous [optimum_contribution\(\)](#) contributions into a concrete set of n parent slots. Feed the result to the crossing primitives or [recurrent_selection\(\)](#).

Usage

```
sample_parents(ocs, pop, n, seed = NULL, method = c("allocate", "multinomial"))
```

Arguments

ocs	an ocs object from optimum_contribution() .
pop	the Population the contributions were computed on.
n	number of parent slots to fill.
seed	optional RNG seed: one non-negative whole number. It orders the slots ("allocate") or drives the draws ("multinomial"); the caller's RNG state is restored on exit.
method	"allocate" (default, largest-remainder allocation) or "multinomial" (independent weighted draws with replacement).

Details

Two methods are available. `method = "allocate"` (the default) **allocates** the slots so that the realized contributions match the optimized ones as closely as integers allow. This controls the *marginal* count error of each individual; it is an approximation, not a guarantee on the group coancestry $\frac{1}{2}c'Gc$, which is quadratic in the contributions and is not preserved exactly by integer counts. The gap between the realized and the optimized coancestry depends on n and on G and shrinks as n grows (for example with $G = I$ and $c = (0.5, 0.5)$ the optimum is 0.25, but n = 1 must pick one parent, whose realized coancestry is 0.5). Individual *i* receives $\lfloor nc_i \rfloor$ slots and the remaining slots go, one each, to the individuals with the largest fractional parts of nc_i (largest-remainder rule; ties

are broken by the larger contribution and then at random with seed), so every count is within one slot of nc_i . The counts are deterministic except for such exact ties. The order of the slots is shuffled (with seed), so repeated copies of a parent are not adjacent.

method = "multinomial" is the former behaviour: n independent draws with replacement, weighted by the contributions. Its expected number of slots is also nc_i , but the *realized* counts carry sampling noise (with $c = (0.5, 0.5)$ and $n = 10$ one parent's count ranges widely around 5), so the realized group coancestry is on average above the optimum, more so for small n. Use it only when that sampling variation is itself wanted.

Value

a Population of n parents. An individual can occupy several slots; because the crossing and phenotyping primitives require unique individual names, repeated copies of one parent are given disambiguated ids ("A", "A_1", "A_2", ...) while carrying identical genotypes. With "allocate", individuals whose n * contribution is below one may receive no slot when n is small; use a larger n if every contributor must be kept. The returned object carries attribute source, a data frame with one row per slot, in slot (draw) order: slot (1..n, the position in the returned Population), id (the parent's id in pop, before the copies are disambiguated), index (its position in pop) and name (the unique id the slot has in the returned Population). It is the same for both methods and is dropped by operations that rebuild the Population (c(), subsetting).

See Also

`optimum_contribution()`, `recurrent_selection()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:40)
f2 <- selfcross(cross(pop[1], pop[2], n = 1, seed = 1), n = 60, seed = 2)
ph <- simulate_phenotype(f2, h2 = 0.5, seed = 3) |> additive(n_qtn = 30)
oc <- optimum_contribution(ph, merit = "bv", lambda = 5)
mates <- sample_parents(oc, f2, n = 10, seed = 4)
n_individuals(mates)
```

select_ind

Select individuals from a simulated population

Description

Ranks the individuals of a realized `simulate_phenotype()` result on a chosen criterion and returns the selected ones as a **Population**, ready to cross forward. This is the truncation-selection primitive the breeding-scheme wrappers (`single_seed_descent()`, `bulk()`, `pedigree()`, `recurrent_selection()`) build on.

Usage

```

select_ind(
  sim,
  n = NULL,
  prop = NULL,
  intensity = NULL,
  on = "pheno",
  trait = 1L,
  direction = c("high", "low"),
  method = c("mass", "within_family", "among_family", "combined", "index",
    "quadratic_index", "random", "culling", "bqp"),
  family = NULL,
  weights = NULL,
  quad_weights = NULL,
  h2 = NULL,
  family_relationship = 0.25,
  rep = 1L,
  culling = NULL,
  sequential = FALSE,
  n_per_family = NULL,
  lambda = 1,
  min_gain = NULL
)

```

Arguments

sim	a realized phenotype_sim, ideally built on a Population so the selected individuals can be crossed on.
n	number of individuals to keep. Exactly one of n, prop, intensity. For method = "among_family" this is a floor: whole families are kept, so more than n individuals can be returned (see the Method section).
prop	proportion of individuals to keep (0-1); the count is round(prop * N) (half-to-even) and at least 1.
intensity	standardized selection intensity i; the number kept is the count whose asymptotic (infinite-population) truncation-selection intensity $i(p) = \phi(z_p)/p$ is closest to i. The finite-sample expectation of the realized intensity is somewhat lower than $i(p)$ (small N).
on	selection criterion (see the Criterion section).
trait	one trait index in 1..n_traits to select on (default 1); not used for method = "index"/"quadratic_index" (all traits) or for a numeric/function on, but it is still validated in every non-culling method. A vector is an error here (it is only meaningful for method = "culling", and as a tandem schedule in pedigree() / recurrent_selection()).
direction	"high" (default) keeps the largest scores, "low" the smallest; one value (a vector is an error, except per trait for method = "culling").
method	selection method (see the Method section).

family	optional grouping vector (length = individuals) for the family methods. Every individual needs a non-NA, non-empty label: NA and empty ("") labels are errors, because they would silently drop the individual or merge unrelated ones into one pseudo-family.
weights	economic weights (one per trait) for method = "index", or the linear weights w for method = "quadratic_index".
quad_weights	symmetric n_traits x n_traits matrix of quadratic (diagonal) and cross-product (off-diagonal) weights W for method = "quadratic_index" (default: zero, i.e. a linear index).
h2	narrow-sense heritability of the selection trait, required by method = "combined" to weight family versus individual information.
family_relationship	correlation of breeding values among family members for method = "combined". The default 0.25 is for half-sibs of non-inbred, unrelated parents; with such parents use 0.5 for full-sibs or doubled haploids and 2/3 for S1 sibs. Inbred or related parents change these – see Details.
rep	replication to select on when several were simulated (default 1).
culling	per-trait kept proportions in (0, 1] for method = "culling".
sequential	for method = "culling": FALSE (default) culls every trait on the whole population at once; TRUE culls the traits in order, each among the survivors of the previous ones.
n_per_family	for method = "within_family" only: the number of individuals to keep in each family , in place of the proportional allocation of a total n. One whole number ≥ 1 keeps that many from every family; a numeric vector keeps a family-specific number (whole numbers ≥ 0), either named by family label (every family must be named, none twice or unknown) or unnamed with one value per family in the order of the sorted family labels (sort(unique(as.character(family)))), the order of split()). n_per_family replaces n, prop and intensity: giving either with it is an error, as is using it with another method. A family with fewer individuals than requested is an error naming the family (the count is never capped silently). Within each family the top-scoring individuals on the criterion are kept; the selection differential S and the intensity $i = S / \text{sd}(\text{criterion})$ are the same quantities as for the other methods (the mean of the kept scores minus the mean of all candidates, over the sample SD of all candidates). The default NULL keeps the proportional allocation. The scheme wrappers pedigree() and recurrent_selection() use mass selection only and do not forward it.
lambda	relatedness penalty weight for method = "bqp" (default 1;= 0).
min_gain	for method = "bqp": NULL (default, no trait constraints), or the paper's minimum desired gain R_j in percent, in $[0, 100]$ (one value, or one per trait when weights is given); the constraint is that the selected set's summed standardized criterion is at least $R_j N / 100$.

Value

the selected individuals as a Population (when sim is Population-backed) or their ids, carrying attributes selected (ids), differential (selection differential S on the criterion), intensity (realized standardized $i = S / \text{sample SD of the criterion}$), criterion and method (plus n_per_family,

the realized named count per family, when that argument was used). A criterion whose scores, differential or SD overflow to a non-finite value is an error.

Criterion (on)

"pheno" the observed phenotype – realistic mass selection. For a purely additive model this is the classic $R = i h^2 \sigma_P$ (narrow-sense h^2 ; Falconer & Mackay). In general the expected response of the mean breeding value is $R = i \text{Cov}(A, P) / \sigma_P$ (the regression of breeding value on the criterion; Smith 1936, Hazel 1943) **provided** $E[A | P]$ is linear in P (as under joint normality of A and P); it is not an identity for an arbitrary joint distribution. A discrete counterexample: $A = (0, 1, 2)$ equiprobable with $P = A^2$, selecting the top third ($A = 2$), gives an actual response of 1 against a linear-regression value of about 1.077. Under that linearity assumption the expression equals $i h^2 \sigma_P$ with $h^2 = V_A / V_P$ only when $\text{Cov}(A, P - A) = 0$, i.e. when every non-additive component of the phenotype (dominance, epistasis and the residual) is uncorrelated with the breeding value. Under random mating / Hardy-Weinberg proportions with linkage equilibrium this holds for dominance, but it can fail for epistasis under linkage disequilibrium: the package's centred epistatic products need not be orthogonal to the additive values (SPEC S2), so $\text{Cov}(A, P)$ can differ from V_A (e.g. a two-locus population in Hardy-Weinberg proportions at each locus but with disequilibrium $D = 0.05$ gives $\text{Cov}(A, I) \neq 0$). When dominance/epistasis layers are present the response is therefore governed by the covariance of the breeding value with the phenotype, not by broad-sense H^2 , and an epistatic term can itself contribute transmissible (marginal) average effects, which is why "bv" is not offered for epistatic models. Off Hardy-Weinberg (selfed, inbred or selected populations) the covariance is likewise not V_A .

"gv" the true *total* genetic value (broad-sense: additive plus any dominance/epistasis). Idealized selection on genetic merit – an upper bound on selectable genetic value, but not on breeding-value response, since the non-additive part is not, in general, transmitted to progeny. For a derived transcriptome() layer this includes its genetic-mediated part, so "gv" equals `genetic_values()`.

"bv" the true *breeding* value – the classical transmissible merit, $A_i = \sum_j \alpha_j (x_{ij} - 2p_j)$, summing each causal locus's average effect of substitution $\alpha_j = a_j + d_j(q_j - p_j)$ (`.breeding_value_matrix()`). The per-locus effects are reconstructed from the simulation's own additive/dominance QTN effects (it is a simulation, so they are known exactly), making this the genetic value transmitted to random-mated progeny (twice the expected progeny deviation from the population mean, $A_i = 2 \times$ the progeny mean deviation) – robust to both linkage disequilibrium (exact for an F2) and departures from HWE (after inbreeding/selection). It captures the additive average effects dominance loci induce away from $p = 0.5$ and reduces to the additive value for a purely additive model. This is the merit that governs response to selection in the next generation. Not available when the model has an epistasis layer or `architecture = "complex"`: an epistatic term has no per-locus a/d , so its induced additive average effects cannot be reconstructed and the breeding value would be incomplete (see `.breeding_value_matrix()`); both cases error rather than return a partial value. A derived (genome-mediated) transcriptome() layer of `prop > 0` is likewise refused (its genetic value has no per-locus decomposition); use "gv" or a custom criterion. Supply your own predicted values via a numeric/function criterion there.

a numeric vector one score per individual (named by id or in population order) – the hook for **genomic selection**, **phenomic selection** and any predicted/estimated value you compute externally.

a function `on(sim)` returning such a vector – the same hook, resolved at selection time.

on applies to method = "mass", "within_family", "among_family" and "combined". The multi-trait index methods ("index", "quadratic_index") score on all traits' true breeding values and ignore on (a single per-individual score cannot supply per-trait breeding values); supplying one with an index method warns. To rank on externally predicted values, compute your index and pass it through on with method = "mass".

Method

"mass" truncates on the individual criterion. "within_family" keeps the top fraction inside each family, "among_family" keeps whole top-ranked families, and "combined" ranks on the Lush combined index that weights an individual's own record and its family mean to predict breeding value; the weights are the selection-index solution $b = V^{-1}c$ built from h^2 and family_relationship, following Falconer & Mackay (1996) and Lynch & Walsh (1998). h^2 is the candidates' own heritability, so family_relationship is the **correlation of breeding values** between two members of a family, $A_{ij}/\sqrt{A_{ii}A_{jj}}$ (A the additive relationship matrix), not A_{ij} alone; the index uses one such value for every family. For families of non-inbred, unrelated parents it is 1/4 for half-sibs, 1/2 for full-sibs, 1/2 for doubled haploids and 2/3 for S1 sibs (for which $A_{ij} = 1$, $A_{ii} = 1.5$). Inbred or related parents change these: this package's derivation from the tabular A gives $2(1+F)/(3+F)$ for S1 sibs and $(1+F)/2$ for doubled haploids of a parent with inbreeding F ; for other families compute $A_{ij}/\sqrt{A_{ii}A_{jj}}$ from the pedigree. The index uses one h^2 and one family_relationship for every family, so it assumes families of one type from comparable parents; mixing family types (e.g. S1 with full-sib families) is outside this model. The weights are optimal under an **additive** model, in which family members covary only through breeding values ($t = rh^2$). Dominance, epistasis or a shared family environment add to the covariance of full-sib or selfed family members (e.g. $V_D/4$ for full sibs), so with those in the phenotype the index is a reasonable but not an optimal predictor. Scores are predictions from deviations of the records from their mean (so families of different sizes are ranked on one scale); a family of one is scored by its own record, h^2 times its deviation. "index" is the Smith–Hazel multi-trait economic index (weights = economic weights, one per trait): $b = P^{-1}Ga$, selecting on $b'y$, with P the phenotypic covariance matrix and G the covariance matrix of the true breeding values ($G = \text{Cov}(A)$). Hazel's (1943) G is the covariance of each trait's phenotype with each trait's breeding value, $\text{Cov}(P, A)$; the two coincide only when $\text{Cov}(A, D) = 0$ (random mating / Hardy-Weinberg proportions, e.g. an F2 or a random-mated population). In selfed, inbred or previously selected populations with dominance $\text{Cov}(P, A) \neq \text{Cov}(A)$ and the index is then Smith–Hazel under that assumption, not the exact maximiser of the correlation with the aggregate breeding value (DECISION-015). Phenotype rows are matched to individuals by id, so the row order of `sim$pheno` is irrelevant. "quadratic_index" is the nonlinear genomic selection index of Ceron-Rojas et al. (2026), $\hat{I} = w'y + y'Wy$ (weights = linear w , quad_weights = the symmetric quadratic/cross-product matrix W), which captures trait interactions and intermediate optima. "random" draws at random (a drift control); its differential and intensity attributes are the *realized* values on the on criterion of the individuals drawn (expected 0, either sign), not those of the random draw score, and on is validated. "bqp" is the relatedness-penalized binary quadratic programming selection of Montesinos-Lopez et al. (2025): it chooses **exactly** N individuals maximizing $\sum_i \sum_j w_j s_{ij} x_i - \lambda \sum_i \sum_k G_{ik} x_i x_k$, with $x_i \in \{0, 1\}$, s_{ij} the standardized criterion of trait j , weights the economic weights w_j (omitted: the single criterion on/trait, weight 1; the paper's objective is the unweighted sum) and $G = WW'/p$ the paper's genomic relationship matrix of the individuals, from the column-standardized marker matrix W over the p polymorphic markers (not `g_matrix()`'s VanRaden method 1). λ is the paper's penalty weight k ($\lambda = 0$ reduces to truncation on the merit). `min_gain` adds the paper's per-trait constraints $\sum_i s_{ij} x_i \geq l_j = R_j N/100$ (Eq. 3-4), with R_j the minimum desired gain in percent, in $[0, 100]$; an unattainable constraint is an

error (with the heuristic solver, "no set found" does not prove that none exists). The paper solves the problem with CVXR; here it is solved without dependencies and deterministically: exact enumeration when $\text{choose}(n, N) \leq 2e5$, otherwise a greedy construction followed by best-improvement 1-swap local search (a heuristic with no optimality guarantee; the greedy start ignores `min_gain`, which the swap phase enforces by penalty). The attribute "bqp" of the result reports the objective, the solver used and the arguments (DECISION-046). "culling" is independent culling levels: culling gives one proportion per trait (traits in `trait`, default the first `length(culling)`), and an individual is kept iff it is in the top `culling[t]` fraction of every trait (simultaneous; sequential = TRUE culls trait 1, then trait 2 among the survivors, and so on). The number kept then follows from the proportions, so `n / prop / intensity` must be left NULL; direction may be one value per trait, and on may be an individuals x traits matrix of external per-trait predictions. Under Hazel & Lush's (1942) idealized conditions – T uncorrelated traits of equal variance and heritability, equal weights – this package's derivation gives the per-generation aggregate responses $i(p)\sqrt{T}h\sigma_A$ (index), $T i(p^{1/T})h\sigma_A$ (simultaneous culling at $p^{1/T}$ per trait) and $i(p)h\sigma_A$ (tandem: one trait per generation, via the scheme wrappers' `trait` vector), so `index` $\geq$ `culling` $\geq$ `tandem`; at $T = 2$, $p = 0.1$ culling and tandem reach 0.907 and 0.707 of the index. Family methods need a family grouping with no missing labels (NA is an error); "combined" additionally needs `h2`. "among_family" keeps **whole** families (highest family mean first) until at least `n` individuals are held, so it returns *at least* `n` – often more – and `intensity` describes that whole set; "within_family" allocates exactly `n` across families in proportion to family size by largest remainder (remainder ties follow the character sort order of the family labels), or, with `n_per_family`, keeps a stated number from every family (see that argument; this is the form for unequal families). The count from `prop` is `round(prop * N)` (R's half-to-even rounding) with a minimum of 1. The reported `intensity` is $S / \text{sd}(\text{criterion})$ with the sample $(n - 1)$ standard deviation.

References

Truncation response and selection intensity: Falconer DS, Mackay TFC (1996) Introduction to Quantitative Genetics, 4th ed. Longman, Harlow; Lynch M, Walsh B (1998) Genetics and Analysis of Quantitative Traits. Sinauer Associates, Sunderland, Massachusetts. Combined (family + individual) selection: Lush JL (1947) Family merit and individual merit as bases for selection. *The American Naturalist* 81:241–261 (Part I, doi:10.1086/281520) and 362–379 (Part II, doi:10.1086/281532). Multi-trait selection index $b = P^{-1}Ga$: Smith HF (1936) A discriminant function for plant selection. *Annals of Eugenics* 7(3):240–250. doi:10.1111/j.14691809.1936.tb02143.x; Hazel LN (1943) The genetic basis for constructing selection indexes. *Genetics* 28(6):476–490. doi:10.1093/genetics/28.6.476 Index vs independent culling vs tandem selection ("culling", tandem in `pedigree()` / `recurrent_selection()`): Hazel LN, Lush JL (1942) The efficiency of three methods of selection. *Journal of Heredity* 33:393–399. doi:10.1093/oxfordjournals.jhered.a105102 Quadratic (nonlinear) genomic selection index: Ceron-Rojas JJ, Montesinos-Lopez OA, Montesinos-Lopez A, et al. (2026) Nonlinear genomic selection index accelerates multi-trait crop improvement. *Nature Communications* 17:1991. doi:10.1038/s41467026698903 Additive average-effect (breeding value) decomposition used as the merit for the "index" / "quadratic_index" methods (per-locus average effect of substitution $\alpha = a + d(q - p)$): Fisher RA (1918) The correlation between relatives on the supposition of Mendelian inheritance. *Transactions of the Royal Society of Edinburgh* 52:399–433. doi:10.1017/S0080456800012163; Falconer DS, Mackay TFC (1996) Introduction to Quantitative Genetics, 4th ed. Longman; Lynch M, Walsh B (1998) Genetics and Analysis of Quantitative Traits. Sinauer. Relatedness-penalized selection by binary quadratic programming ("bqp"): Montesinos-Lopez OA, Montesinos-Lopez A, Hernandez-Suarez CM, Alemu A (2025) A selection index with minimal genetic relatedness for multi-trait data via binary quadratic program-

ming. *Plant Methods* 22:7. doi:[10.1186/s13007025014844](https://doi.org/10.1186/s13007025014844) Breeding schemes: Bernardo R (2020) Breeding for Quantitative Traits in Plants, 3rd ed. Stemma Press.

See Also

[single_seed_descent\(\)](#), [bulk\(\)](#), [pedigree\(\)](#), [recurrent_selection\(\)](#), [simulate_phenotype\(\)](#).

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:60)
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
f2 <- selfcross(f1, n = 50, seed = 2)
ph <- simulate_phenotype(f2, h2 = 0.5, seed = 3) |> additive(n_qtn = 20)

top <- select_ind(ph, prop = 0.2, on = "pheno") # keep the best 20%
n_individuals(top)
```

selection_methods	<i>The engine's selection operators, as a manifest</i>
-------------------	--

Description

A table of the selection and scoring operators the engine exports, with their parameters and sources, for front ends (e.g. breedingDesigner) that build their interface from it.

Usage

```
selection_methods()
```

Value

A data frame with columns id, label, fn (the function and method to call), params (main arguments of that function, comma-separated; | separates alternatives, + arguments given together) and source.

See Also

[select_ind\(\)](#), [predict_ebv\(\)](#), [combining_ability\(\)](#)

Examples

```
selection_methods()[, c("id", "fn")]
```

selfcross	<i>Self-pollinate an individual</i>
-----------	-------------------------------------

Description

Produces *n* progeny by selfing: both gametes come from independent meioses of the same individual. Selfing a heterozygous individual halves heterozygosity each generation, so repeated selfing drives a line toward homozygosity.

Usage

```
selfcross(parent, n = 1, seed = NULL, interference = NULL)
```

Arguments

parent	a single-individual Population.
n	number of progeny.
seed	optional RNG seed. All randomness is drawn in R, so <code>set.seed()</code> before the call works equally well. With a seed the caller's RNG state is restored on exit (a seeded call does not disturb the ambient stream); with <code>seed = NULL</code> the draws consume the ambient stream.
interference	crossover interference model (see the section "Crossover interference"). <code>NULL</code> (default): the option <code>simplePHENOTYPES.interference</code> if set, else the gamma model with $\nu = 2.6$, $p = 0$ (AlphaSimR's default). "poisson": Poisson crossovers, no interference, the isqg random stream (the default before 2.0.0.9003). <code>list(nu = , p =)</code> : the two-pathway gamma model with these values; $1 \leq \nu \leq 1e6$ is the interference strength ($1 = \text{none}$), p in $[0, 1]$ (default 0 when omitted) the share of chiasmata that do not interfere. The expected number of crossovers per Morgan is the same under every model.

Value

A Population of *n* progeny.

References

Toledo, F.H., Perez-Rodriguez, P., Crossa, J. and Burgueno, J. (2019). isqg: A Binary Framework for in Silico Quantitative Genetics. *G3 Genes|Genomes|Genetics* 9(8), 2425–2428. doi:10.1534/g3.119.400373

See Also

[cross\(\)](#), [double_haploid\(\)](#)

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = c("33-16", "38-11"))
f1 <- cross(pop[1], pop[2], n = 1, seed = 1)
f2 <- selfcross(f1, n = 10, seed = 2)
f2
```

simulate_phenotype	<i>Start a phenotype simulation (v2 grammar foundation)</i>
--------------------	---

Description

`simulate_phenotype()` is the entry point of the composable v2 grammar. It fixes the genetic architecture and residual variance and returns a realized `phenotype_sim` object. With no genetic layers the trait is pure noise (broad-sense heritability $h^2 = 0$); pipe it into `additive()`, `dominance()`, `epistasis()` to add mean-effect genetic components, and `vqtl()` to add a genotype-dependent residual-variance component. Each layer is expressed as a requested marginal proportion of phenotypic variance.

Usage

```
simulate_phenotype(
  geno = NULL,
  architecture = c("independent", "pleiotropy", "ld"),
  n_traits = 1,
  n_qtn = 0,
  n_reps = 1,
  vary_qtn = FALSE,
  seed = NULL,
  h2 = NULL,
  mean = NULL,
  individuals = NULL,
  model = "A",
  expression = NULL,
  transcriptome = NULL,
  reps = 1,
  resid_cor = NULL,
  refit = NULL,
  ...,
  residual_mode = c("fixed", "random")
)
```

Arguments

<code>geno</code>	genotype input: a simplePHENOTYPES numeric-format data frame (first five columns <code>c("snp", "allele", "chr", "pos", "cm")</code>), e.g. <code>SNP55K_maize282_maf04</code> ; the optional character counted column written by <code>as_numeric(counted_column</code>
-------------------	---

	= TRUE) is accepted and ignored), an individuals-by-markers numeric matrix coded -1/0/1, or a Population from cross(), selfcross() or double_haploid(). At least three individuals are required (with two, the exact-variance standardization forces the genetic value and residual to be collinear and the realized heritability is meaningless); duplicate chr/pos values are accepted (the map need not be unique); markers that are monomorphic, or heterozygous in every individual (constant dosage), can never be QTNs and are skipped. Optional when an expression basis is given: with <code>geno = NULL</code> and an expression matrix (or a <code>transcriptome_sim</code> in <code>transcriptome</code>), the phenotype is built from expression alone – individuals come from the expression source's columns, there are no markers, and only <code>transcriptome()</code> layers are valid (<code>h2</code>, <code>n_qtn</code>, and the marker layers all need <code>geno</code>).
architecture	one of "independent" (each trait its own QTNs), "pleiotropy" (shared QTNs with a controlled genetic correlation), or "ld" (two traits whose <i>distinct</i> causal loci are in linkage disequilibrium, so they covary through linkage rather than pleiotropy; requires <code>n_traits = 2</code>). Under "pleiotropy" the correlation is controlled in every mean-effect layer – additive(), dominance() and epistasis() each target <code>cor</code> (the realized correlation converges to it as the numbers of QTNs / sets and of individuals grow, for causal loci in approximate linkage equilibrium and no major QTN holding a fixed share of the variance; see <code>cor</code> below) – and the total genetic correlation targets <code>cor</code> when the layers' per-trait prop profiles are proportional, e.g. scalar prop (DECISION-023). Under "ld" the linked-loci design covers one additive() layer and dominance() on the same linked loci (<code>same_as_add = TRUE</code>, the default); epistasis(), a fresh dominance draw and a second additive layer are rejected there, since they could not keep every causal marker trait-specific and inside the <code>r2</code> window; a transcriptome() layer on a genome-derived source adds a shared genetic cause there (warned). Fixing loci with <code>qtn =</code> is rejected under "pleiotropy" and "ld", which draw their own loci.
n_traits	number of traits to simulate.
n_qtn	baseline QTN count; a per-layer <code>n_qtn</code> overrides it with a warning.
n_reps	number of replications.
vary_qtn	if TRUE, each replication (<code>n_reps</code>) draws an independent set of QTNs and effects, so replications are distinct genetic architectures rather than the same one with fresh residuals. Layers given an explicit <code>qtn</code> keep their fixed loci across replications.
seed	RNG seed stored on the object and threaded to every layer: each layer, replication and residual draws under a sub-seed derived from (<code>seed</code> , <code>layer type</code> , occurrence of that type) see the reproducibility paragraph above. The caller's RNG state is left untouched.
h2	optional requested genetic-variance share for one-call simulation. For a single mean-effect layer this is the simulated broad-sense heritability (the genetic component is scaled to <code>h2</code> exactly; only its sample covariance with the residual moves the realized ratio in fixed mode; random mode also allows residual sample variance to fluctuate). With multiple non-orthogonal layers – above all additive and dominance on shared loci – the realized value can differ structurally from the requested sum; see Details and the reported realized <code>H2 / \$ad_report</code> . <code>h2</code> governs the marker genetic budget (additive + dominance + epistasis prop

must sum to it). A `transcriptome()` layer's prop is a separate expression-mediated variance category and is **not** part of this budget, so a model may deliberately fill only part of h^2 with markers and leave the rest to expression; the marker-completeness check is then skipped and only the realized h^2 (which includes any genome-mediated expression variance) is reported. The genetic layers and the residual are fitted to h^2 in the population the simulation is built on. On a population whose heritability is already set (it carries a trait) h^2 is ignored with a warning unless `refit = TRUE`; see `refit`.

mean	optional per-trait intercept added to the phenotype (scalar or length <code>n_traits</code>). Genetic values stay centered; only the phenotype is shifted.
individuals	optional subset of individuals to simulate, given as IDs or indices (at least three). Marker minor-allele frequencies are recomputed on the subset, and the genotypes are never copied – only the selected rows are read.
model	one-call model string: "A" (additive, default), "AD" (additive + dominance), "AE" (additive + epistasis).
expression	optional real/observed expression as a genes-by-individuals numeric matrix (columns named by individual id, or in the individual order), used by a <code>transcriptome()</code> layer. Give at most one of <code>expression</code> / <code>transcriptome</code> .
transcriptome	optional genome-derived expression source for a <code>transcriptome()</code> layer: a <code>transcriptome_sim</code> from <code>simulate_transcriptome()</code> , or <code>TRUE</code> to derive one from <code>geno</code> with default settings. Give at most one of <code>expression</code> / <code>transcriptome</code> .
reps	number of independent records averaged into each entry's phenotype (default 1; entry-mean replication, the AlphaSimR <code>setPheno(varE, reps)</code> semantics): a positive whole number, or a vector of length <code>n_traits</code> for per-trait counts. The phenotype becomes the mean of <code>reps</code> independent records of the same genotype, so the residual (including the <code>vqtl()</code> heterogeneity component) is divided by <code>sqrt(reps)</code> ; the genetic value and any <code>transcriptome()</code> component are unchanged. h^2 and every layer prop stay on the single-record scale (shares of the unit record variance $V_G + V_E = 1$, the <code>var_budget</code> and the printed "residual" row).

Target versus realized. Two heritabilities exist. The **target** (expected-value) values come from the variance allocation: single-record $V_G / (V_G + V_E)$ (what h^2 requests) and entry-mean $V_G / (V_G + V_E / \text{reps})$, which is larger for `reps` > 1. The **realized** values are $\text{Var}(G) / \text{Var}(y)$ computed from the realized values, so they carry the sample covariance between the genetic value and the residual e : without a transcriptome layer the entry-mean variance is $\text{Var}(y_{\text{bar}}) = V_G + V_E / \text{reps} + 2 \text{Cov}(G, e)$ and the single-record variance is $\text{Var}(y) = V_G + V_E + 2 \text{Cov}(G, e)$, with V_E and e the realized single-record residual variance and values. The allocation formula is the realized one only when that sample covariance is zero. The realized H^2 printed by the object, and the shares reported against "V_P" (`$ad_report`, `mediation_split()`), are on the **entry-mean** scale (the stored phenotype); `print()` also shows the single-record realized value when `reps` > 1. The residual is drawn exactly as for `reps` = 1 (same RNG stream) and then scaled by $1 / \text{sqrt}(\text{reps})$, so `reps` = 1 is bit-identical to a call without `reps`.

Scope of the replication model. Only the phenotype residual is replicated: its realized variance is exactly the `reps` = 1 residual variance divided by `reps` (for

a `vqtl()` layer this is $[V_0 + V_v + 2 \text{Cov}(e_0, e_v)] / \text{reps}$, with V_0 the homoskedastic and V_v the heterogeneity part, not the nominal V_E / reps , because the two standardized components have a non-zero sample covariance). Without a `transcriptome()` layer the reps records are independent given the genotype; this does not model repeated measures with a shared permanent environment. With a derived `transcriptome()` layer the environmental transcriptome component is a persistent entry-level quantity that is **not** redrawn per record, so the records are independent only conditional on that fixed transcriptome covariate (and the genotype), and only the phenotype residual is rescaled (its value by $1/\sqrt{\text{reps}}$, its variance by $1/\text{reps}$). The realized denominator is then the variance of the full stored phenotype, which includes the unreplicated transcriptome component and its covariances.

`resid_cor` target **residual** (environmental) correlation between traits; `cor` is the *genetic* correlation. NULL (default) draws independent residuals, bit-identical to output before the argument existed. A single value in $[-1, 1]$ is the common pairwise correlation (for `n_traits` traits it must be at least $-1/(\text{n_traits} - 1)$); or give an `n_traits` x `n_traits` symmetric positive semi-definite matrix with unit diagonal and entries in $[-1, 1]$. Needs `n_traits` ≥ 2 . Each trait's residual is drawn exactly as in the independent case (same sub-seed), the traits are mixed through the Cholesky factor of the correlation matrix. In fixed mode each column is re-standardized to the same exact variance, so every trait's residual variance is still its h^2 -implied $1 - \text{sum}(\text{prop})$ target and only correlation is induced (trait 1's residual is unchanged up to rounding). The correlation is a target: the realized sample correlation matches it up to sampling error of order $1/\sqrt{n}$, and it is a statement about the homoskedastic residual: a `vqtl()` heterogeneity component is drawn independently per trait and dilutes it, and `reps` > 1 scales every trait's residual by its own $1/\sqrt{\text{reps}}$, which leaves the correlation unchanged. Because the realized-variance standardization is applied per trait in fixed mode, the residual sample variances and requested `var_budget` are unchanged by `resid_cor`; realized h^2 can change because genetic-residual covariance changes. Random mode skips standardization and treats these variances as distribution parameters. The residual has sample covariance with the genetic values (as without it) and, with `resid_cor`, the traits' residuals are correlated, so the phenotypic correlation is a mix of the genetic (`cor`) and residual (`resid_cor`) correlations, weighted by the variance shares.

`refit` whether to fit the genetic layers and the residual to h^2 in this population. NULL (default): FALSE when `geno` is a `Population` whose heritability is already set (it carries a trait, see `population_trait()`), else TRUE.

- TRUE – the usual behaviour, and the only one for marker data or a population without a trait: the layers you add are drawn and scaled, and the residual is fitted to the target h^2 , in the population given. This is the rule for a single population (e.g. simulating phenotypes for GWAS). On a population that carries a trait it defines a new one, which `select_ind()` then stores on the selected individuals.
- FALSE – reuse the population's trait: its causal loci and effects on its base population's genetic scale (layers centered and scaled as in the base population) and its residual variance, held fixed. The genetic variance of this population can differ from the base one, so the heritability is no longer the

base h2: it falls as selection and inbreeding exhaust genetic variance, as in AlphaSimR's `setPheno(varE = )`. `h2`, `n_qtn`, `n_traits`, `architecture`, `model`, `mean`, `resid_cor` and ... are taken from the trait; an h2 that differs from the population's warns that the population's heritability is used (the same h2 is silent). Layer verbs piped onto the result (`additive()`, `dominance()`, ...) are ignored: a layer type the trait already has silently, any other with a warning. `seed`, `n_reps`, `reps` and `individuals` apply as usual; with `reps` the fixed residual variance is divided by `reps`.

...

architecture-specific arguments (validated – an unknown name is an error, and an argument for a different architecture warns). For "pleiotropy": `cor`, `pi` (or the two-trait `pi_target / pi_secondary`), `n_pleio_major`, `prop_var_major`. For "ld" (two traits only): `ld_type` ("indirect"/"direct"), `r2_min`, `r2_max` (the `r2` window the linked causal pair must fall in; see `qtn_table()`). For "independent": `distinct_chr` (TRUE puts each trait's QTNs on disjoint chromosomes). `ld_type` defaults to "direct" (the two traits' causal SNPs are directly in LD). Under "ld", `partner` chooses the trait-2 locus for a "direct" pair (or the ordering of flanking candidates for "indirect"): "strongest" (default) takes the in-window partner with the **highest** `r2` with a randomly drawn anchor SNP (so realized `r2` skews toward `r2_max`), "random" takes a uniformly random in-window partner. Perfectly collinear markers (`r2` = 1) are never used as a partner, and the window must satisfy $0 < r2_max$ and $r2_min < 1$.

Under "ld", `ld_phase` = `c("coded", "coupling", "repulsion")` sets the **haplotype-derived phase** of each linked pair. The `r2` window ignores the sign of the pair's dosage correlation `r`, so with the default "coded" the sign of the linkage-induced covariance – `sign(e1 * e2 * r)` per pair, `e1/e2` the pair's trait-1/trait-2 additive effects – follows the arbitrary marker coding and the effect series. "coupling" flips trait 2's effect wherever needed so that every pair's sign is positive (the allele raising trait 1 travels with the allele raising trait 2; positive linkage-induced genetic correlation); "repulsion" makes every pair's sign negative (the allele raising trait 1 travels with the allele lowering trait 2). The sign is controlled pair by pair (each matched pair's contribution); the total genetic correlation also includes the cross-pair terms between loci of different pairs in LD, so with pairs close together on one chromosome it can differ in sign from the pairs' common sign. Only trait 2's additive effects change: the loci, trait 1's effects, `r` and `r2` are those of "coded". It acts on the `additive()` layer's effects (also per replication with `vary_qtn` = TRUE); a `dominance()` layer reusing the linked loci (`same_as_add` = TRUE) keeps its own deviations untouched. It is applied after the positional `additive(phase = )` alternation, which it overrides pair by pair on trait 2. The signed `r` of each pair is kept as the "r" attribute of the layer's \$ld pair frame.

`cor` is the target **genetic** correlation and works for any number of traits: a scalar applied to every trait pair, or a full `n_traits` × `n_traits` matrix (negative correlations allowed). It applies to every mean-effect layer (additive, dominance, epistasis), each targeting it: `cor` sets the cross-trait covariance of each layer's effect draw, and after the layer is scaled to its `prop` the realized correlation – a random ratio – converges to `cor` as the numbers of shared QTNs / sets and of individuals grow, **provided the causal loci are in approximate linkage equilibrium and no unit keeps a non-vanishing share of the variance**. It is a

sample correlation over the simulated individuals, so with a fixed sample its scatter levels off at that sampling spread however many QTNs there are (e.g. SD about 0.18 at $\text{cor} = 0.5$ with 20 individuals, even with 5 000 QTNs). With $\text{n_pleio_major} / \text{prop_var_major}$ the major QTNs keep prop_var_major of the shared variance however many QTNs there are, so the realized correlation stays as variable as a few-QTN draw and does not converge. With few shared units it is attenuated toward 0 on average by an amount that depends on cor , pi and the designs (about $0.82 \times \text{cor}$ with two shared units at $\text{cor} = 0.5$, $\text{pi} = 1$, unlinked loci); strong LD among the causal loci can prevent convergence (under complete LD every realized value is ± 1); and a single shared unit gives ± 1 or one noisy draw (warned, for any cor strictly between -1 and 1, 0 included). pi sets the shared vs trait-specific split in each layer. The **total** genetic correlation targets cor when every layer's per-trait prop profile is proportional (always so for scalar prop and for the one-call models); otherwise it is the layers' variance-weighted combination, attenuated toward 0, and a warning reports that large-sample target (when it falls more than 1% short of cor) (e.g. additive prop = $c(0.49, 0.01)$ with dominance prop = $c(0.01, 0.49)$ gives $0.28 \times \text{cor}$, i.e. 0.14 at $\text{cor} = 0.5$). A `transcriptome()` layer on a genome-derived source is outside cor : its genome-mediated signal adds to the genetic value with its own cross-trait correlation, so the total is then not targeted at cor (warned). $\text{n_pleio_major} / \text{prop_var_major}$ shape the additive layer only. The request must be attainable – the implied genetic covariance matrix has to be positive semi-definite, which for two traits is $\text{cor}^2 \leq \text{pi}_1 \times \text{pi}_2$ – otherwise an error is raised rather than an approximation returned. Using cor prints a citation notice once per session.

`residual_mode` "fixed" (default) centers and rescales residual draws to the requested sample variance, preserving existing seeded results. "random" draws normal residuals with the requested variance parameter, without centering or sample rescaling: sample mean, variance and covariance with genetic values fluctuate. Genetic layer scaling is unchanged. With `resid_cor`, random mode draws multivariate normal residuals with the target covariance; with `reps`, residuals are divided by $\sqrt{\text{reps}}$ in either mode. A `vqtl()` component in random mode uses genotype-dependent normal variances whose average over individuals equals its requested share; its realized variance fluctuates too. Neither mode forces genetic-residual covariance to zero. For a carried population trait (`refit = FALSE`), omission inherits its residual mode (older stored traits use "fixed"); an explicit value overrides the residual mode for this simulation.

Details

The pipe runs eagerly: every object already carries the realized phenotypes, so there is no terminal `simulate()` call.

One-call vs piped: if the call already carries a self-sufficient genetic specification (h2 together with $\text{n_qtn} > 0$), `simulate_phenotype()` builds the implied model and realizes a complete phenotype in a single call. Otherwise it returns the $\text{h2} = 0$ foundation, ready to be completed with layers.

Layer proportions are marginal sample variances after scaling. The simple -1/0/1 additive, heterozygote-indicator dominance, and centered-product epistatic designs are **not** a Fisher/NOIA-orthogonal decomposition, so the scaled components are generally correlated and the realized broad-sense heri-

tability $H^2 = \text{Var}(g) / \text{Var}(P)$ is not, in general, the sum of the requested proportions. The clearest case is `additive() + dominance()` on the **same loci** (the default `same_as_add = TRUE`, and the one-call `model = "AD"`): with one layer of each type $\text{Var}(g) = \text{prop}_A + \text{prop}_D + 2 \text{Cov}(c_A, c_D)$ (with several layers of a type $\text{Var}(c_A)$, $\text{Var}(c_D)$ also carry the covariance among those layers; the exact identity is $\text{Var}(g) = \text{Var}(c_A) + \text{Var}(c_D) + 2 \text{Cov}(c_A, c_D)$), and under Hardy-Weinberg $\text{Cov}(\text{dosage}, \text{heterozygote indicator}) = -(2p - 1) 2pq$ at every locus (p = the frequency of the counted +1 allele). With the default geometric effect series every locus has the same effect sign, so each locus's cross term has the sign of $1 - 2p$: positive where the counted allele is the minor allele, negative where it is the major allele. The terms therefore reinforce, rather than cancel, when the counted-allele frequencies lie on one side of 0.5 (they partly cancel when the frequencies straddle 0.5, and `phase = "repulsion"` alternates the additive signs). The cross term is a **structural** bias that depends on the allele frequencies **and on which allele is coded +1** (on the same loci and seeds a panel with minor-allele frequencies 0.10-0.20 gave a realized H^2 of about 0.63 with the minor allele coded +1 and about 0.25 with the major allele coded +1, for a requested 0.5), not a finite-sample fluctuation that shrinks with n . The realized H^2 printed by the object, and the `$ad_report` component (per trait: the requested share, the realized share, the realized $\text{Var}(A)$, $\text{Var}(D)$ and $2\text{Cov}(A,D)$ of the additive/dominance block as fractions of V_P , and the component variances $\text{Var}(c_A)$, $\text{Var}(c_D)$ and $2\text{Cov}(c_A, c_D)$, which also sum to the realized share; the gap to the request is `realized - requested/V_P`), report what was actually simulated whenever additive and dominance layers share loci; use `additive(orthogonal = TRUE, a =, d =)` when the additive/dominance partition must be Fisher-orthogonal. Epistatic products that share loci with other layers, and strong LD between causal loci, can likewise correlate components; no separate report is produced for those.

Reproducibility: with a non-NULL seed, every layer draws its QTNs and effects under a sub-seed derived from (`seed`, `layer type`, `occurrence`), where *occurrence* counts the earlier layers **of the same type** (0 for the first `additive()`, 1 for the second, ...); replication r of a `vary_qtn` layer and each trait's residual use the labelled variants "`<type>_rep<r>`" and "`residual_t<t>`". The label is hashed position-sensitively, so labels that differ only by a permutation of characters (replications 12 and 21, traits 12 and 21) get different sub-seeds, and distinct labels are collision-resistant over ordinary ranges of replications, traits and layers. A 31-bit sub-seed cannot be injective in general, so two very distant labels can in principle share one (found: with `seed = 123`, replication 106 of the first `transcriptome()` layer and the residual of trait 40160 in replication 4); this is far outside realistic use. Development versions before this rule summed character codes, which made e.g. replications 12 and 21 identical; every seeded value changed with the fix. Consequently adding, removing or reordering a layer of one type never changes the draws of layers of other types, but inserting another layer of the *same* type before an existing one shifts that layer's occurrence index and so changes its draws.

Value

a `phenotype_sim` object. Beyond the realized pheno table and the requested `var_budget`, it carries `ad_report` (a per-trait data frame of the realized additive/dominance partition when additive and dominance layers share loci in the variance-partition coding, else NULL).

Examples

```
data("SNP55K_maize282_maf04")
# piped form
ph <- simulate_phenotype(SNP55K_maize282_maf04, seed = 1)
```

```
ph <- additive(ph, prop = 0.5, n_qtn = 3)
# one-call form
ph2 <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.5, n_qtn = 3, seed = 1)
```

simulate_transcriptome

Simulate a genetically controlled transcriptome

Description

Generates a genes-by-individuals expression matrix as a hybrid latent-factor eQTL model, fully parametrically – no reference expression data or gene annotation is required. For gene g the normalized expression of individual i is $E_{gi} = \mu_g + G_{gi} + R_{gi}$, a genetic component G_{gi} (a cis score over markers near the gene plus a trans score mediated by latent regulatory factors, jointly scaled to a per-gene heritability h_g^2) and a residual component R_{gi} (shared non-genetic co-expression modules plus gene-specific noise). The cis fraction ω_g , residual module fraction κ_g , and h_g^2 are non-competing controls; because cis and trans covary (linkage disequilibrium, structure), the combined genetic score is scaled jointly and the cis/trans covariance is reported rather than assumed zero.

Usage

```
simulate_transcriptome(
  geno = NULL,
  n_genes = 1000,
  annotation = NULL,
  cis_window = 1e+06,
  h2 = "beta",
  cis_fraction = 0.25,
  n_factors = NULL,
  residual_module_fraction = 0.15,
  profile = "generic_bulk",
  seed = NULL,
  mimic = NULL,
  n_ind = NULL,
  epistasis = 0
)
```

Arguments

geno	a Population, a Population-backed phenotype_sim, a numeric-format genotype data frame, or an individuals-by-markers dosage matrix coded -1/0/1. May be NULL for a purely non-genetic transcriptome (co-expression modules and gene noise only, every gene $h2 = 0$, no eQTL tables); then n_ind sets the number of individuals and h2/cis_fraction/ mimic do not apply.
n_genes	number of genes to simulate (ignored when annotation is given). Synthetic gene ids are gene0001, gene0002, ... with a genome and synthetic coordinates, and gene1, gene2, ... in genotype-free and mimic mode (unless mimic

	carries row names). Only markers with $MAF \geq 0.05$ and a known chromosome/position are eligible as eQTL (a fixed threshold).
annotation	optional gene annotation, a data frame with columns gene_id, chr, tss (transcription start site, in the same physical units as the genotype pos). When NULL, synthetic coordinates are generated.
cis_window	cis window half-width in physical position units (default 1e6). A marker is cis to a gene if it is on the same chromosome within cis_window of the TSS.
h2	per-gene target expression heritability: "beta" (draw $Beta(1.5, 6)$, mean 0.20), a single number, or a length-n_genes vector.
cis_fraction	per-gene cis fraction of the marginal genetic variance omega: "beta" (draw $Beta(2, 6)$, mean 0.25), a single number in $[0, 1]$, or a length-n_genes vector. omega is the target of the <i>marginal</i> share $v_{cis} / (v_{cis} + v_{trans})$, which equals omega by construction (the standardized parts each have unit variance). The realized cis_fraction_realized is the share of the realized genetic variance, $v_{cis} / Var(G)$, which differs from omega by the cis-trans covariance (and is not bounded by 1 when cis and trans covary negatively; with epistasis > 0 compare it with $omega * (1 - epsilon)$). Not used when geno = NULL.
n_factors	number of latent regulatory factors Q; default $\max(1, \min(50, \max(5, \text{ceiling}(n_genes / 100)), n_ind - 2))$.
residual_module_fraction	the residual module fraction kappa, a single number in $[0, 1]$: the target/expected fraction of residual variance carried by the shared module factor before normalization. As with h2 and the cis fraction, the realized fraction scatters around it in finite samples (the scatter is large only at very small n).
profile	a named calibration profile (currently "generic_bulk").
seed	optional seed (NULL or one non-negative whole number); the RNG state is restored afterwards.
mimic	optional user expression matrix (genes x individuals, columns named/ordered to the genotypes) to calibrate the generator to. When supplied, the generator's <i>targets</i> are set from it: the per-gene mean and variance, met up to floating-point rounding (a gene whose mean is many orders of magnitude larger than its standard deviation is limited by the spacing of doubles near that mean, so its variance is met only to that resolution; the rescale is skipped only for a gene whose realized unit-scale variance is exactly zero or non-finite; a counted warning is issued when the rescale is ill-conditioned, i.e. the realized unit variance is below $1e-12$); a per-gene heritability from a GREML estimator (REML on the genotypes' GRM, single variance component; it replaces h2, which is ignored with a warning); the co-expression factor count n_factors (Marchenko-Pastur) and the residual module fraction kappa. n_genes is taken from mimic. The eQTL effects, loadings, and (downstream) phenotype slopes are still drawn de novo – mimic calibrates the <i>distribution</i> of expression, it does not fit individual effects: loadings and their signs are not retained (every generated gene loads +1 on one factor, so anti-correlated blocks of the input are regenerated as positively co-expressed) and only n_factors and kappa summarize the input's co-expression. kappa is the mean within-module correlation of the input after subtracting the genetic trans share implied by the GREML h2 and cis_fraction

	(a spiked-covariance estimator, see <code>.tx_estimate_kappa()</code>); pure noise gives kappa near 0. The per-gene GREML values are a <i>distribution</i> -level calibration: for sparse eQTL architectures they do not match the truth gene by gene (per-gene sd about 0.1 at $n = 280$), and a warning is issued below 100 individuals. The estimates are returned in <code>\$calibration</code> .
n_ind	number of individuals, required only when <code>geno = NULL</code> (otherwise taken from <code>geno</code>).
epistasis	per-gene epistatic fraction of the genetic variance <code>epsilon</code> : 0 (default, purely additive cis/trans), a single number in $[0, 1]$, "beta" (draw $\text{Beta}(1.5, 6)$, mean 0.20), or a length- <code>n_genes</code> vector. <code>epsilon</code> is a marginal share, the blend weight of the standardized epistatic score against the standardized additive score: $v_{\text{epi}} / (v_{\text{cis}} + v_{\text{trans}} + v_{\text{epi}})$ equals <code>epsilon</code> exactly only when the cis and trans scores are uncorrelated (or one of them is absent). In general, with $s_{\text{ct}}^2 = \text{Var}(\text{sqrt}(\omega) c + \text{sqrt}(1 - \omega) t)$ the pre-standardization variance of the combined standardized cis (<code>c</code>) and trans (<code>t</code>) scores ($s_{\text{ct}}^2 = 1 + 2 \text{sqrt}(\omega (1 - \omega)) \text{cor}(c, t)$), the marginal share is $\epsilon / (\epsilon + (1 - \epsilon) / s_{\text{ct}}^2)$, which can differ materially from <code>epsilon</code> : e.g. perfectly positively correlated cis and trans scores with $\omega = 0.5$ give $s_{\text{ct}}^2 = 2$, so $\epsilon = 0.4$ yields a marginal share of $0.4 / (0.4 + 0.6 / 2) = 0.5714$ (and strongly negative correlation pushes it further the other way). s_{ct}^2 is recoverable from the variance budget as $1 + \text{cis_trans_cov} / (v_{\text{cis}} + v_{\text{trans}})$. This formula holds for the non-degenerate blend only: when the cis and trans scores exactly (or numerically) cancel (s_{ct}^2 is 0, e.g. perfectly negatively correlated scores with $\omega = 0.5$), the generator falls back to the surviving component, the cis part is dropped, and the marginal share is the target <code>epsilon</code> itself (for $\epsilon = 0.4$ it is 0.4, not the formula value 0; the budget then describes the one-component architecture). The share of the realized genetic variance, $v_{\text{epi}} / \text{Var}(G)$, is reported as <code>epistasis_realized</code> and differs further by the covariance rows. When positive, each gene gains 1–2 marker pairs (drawn from all eligible markers, with no exclusion of the gene's own cis window or trans hub, and possibly the same pair twice) whose centered dosage product (an additive-by-additive interaction) contributes to its expression; the additive share is then $(1 - \epsilon)$ split by <code>cis_fraction</code> . Reported as an <code>epi_eqtl</code> truth table and $v_{\text{epi}} / \text{cis_epi_cov} / \text{trans_epi_cov}$ budget rows; reconstructable and reusable by <code>predict.transcriptome_sim()</code> . If fewer than two eligible markers exist a warning is issued and no epistasis is realized (<code>epistasis_realized = 0</code>). Not used when <code>geno = NULL</code> ; reported as 0 for genes with $h^2 = 0$.

Details

Per-marker reference means (for centering) and the drawn architecture (cis and hub effects, loadings, module assignment) are computed on the supplied (reference) individuals and returned, fixing the genetic scale in the spirit of `additive_value()` / `phenotype_value()`; re-scoring the frozen architecture on *new* individuals (descendants, a cross, a selected subset) is supported by `predict.transcriptome_sim()`. Defaults come from a named calibration profile, a transparent benchmarking compromise, **not** biological constants.

The genetic component G (scaled to $\text{Var}(G) = h^2$) and the residual R (scaled to $\text{Var}(R) = 1 - h^2$),

carrying the shared non-genetic module factor so genes co-express through kappa) are drawn **independently**, so $\text{Cov}(G, R) = 0$ in expectation. Per gene the table reports:

- $h2_realized = \text{Var}(G) / \text{Var}(P)$, the **realized heritability**: the variance of the realized genetic values divided by the variance of the realized expression phenotype $P = E = G + R$ (plus the constant location). It therefore includes the finite-sample covariance, $\text{Var}(E) = \text{Var}(G) + \text{Var}(R) + gr_cov$ with $var_budget\$gr_cov = 2 \text{ Cov}(G, R)$, so on the reference panel $h2_realized = \text{Var}(G) / (\text{Var}(G) + \text{Var}(R) + gr_cov)$ at any scale. Only when $\text{Var}(G) + \text{Var}(R) = 1$ (no mimic rescaling) does this reduce to $h2 / (1 + gr_cov)$; under mimic the variances carry the mimicked scale and that shortcut does not hold. It is **not bounded by 1** and is unreliable for small samples: fewer than 30 individuals ($n_ind < 30$) triggers a warning; fewer than 3 is an error. The covariance is reported rather than projected away – the package’s realized-not-asserted convention.
- $h2_var_ratio$, the same quantity under its original name (kept for backward compatibility; identical to $h2_realized$).
- $h2_allocated = \text{Var}(G) / (\text{Var}(G) + \text{Var}(R))$, always in $[0, 1]$. This is the **variance allocation**, not a heritability: it drops $2 \text{ Cov}(G, R)$ and, because G and R are each scaled exactly on the reference population, equals $h2_target$ there by construction (it is informative on a new population via `predict.transcriptome_sim()`, where R is redrawn). Do not read it as a realized heritability.

Degenerate genes realize $h2 = 0$: a gene whose cis window holds no eligible marker ($MAF \geq 0.05$ and known chromosome/position) **and** whose module factor is inert (which can happen on a small panel, where the union of the module’s genes’ cis windows covers every marker, leaving no distant hub) has no realizable genetic variance and gets $h2_realized = 0$ (and $h2_allocated = 0$).

Random draws are made gene by gene from one stream, so changing an option that alters one gene’s draws (e.g. its `cis_fraction`) re-flows the random numbers of every later gene; per-gene results are reproducible only for the same full call and seed.

Value

a transcriptome_sim: expression and genetic_expression (genes x individuals), genes (per-gene table: $h2_target$, $h2_realized = \text{Var}(G)/\text{Var}(P)$ (also $h2_var_ratio$; not bounded by 1, it includes $2 \text{ Cov}(G, R)$), the bounded allocation $h2_allocated = \text{Var}(G)/(\text{Var}(G)+\text{Var}(R))$; `cis_fraction_target` and the realized `cis_fraction_realized =  $v\_cis / \text{Var}(G)$` ; `epistasis_target`, `epistasis_realized =  $v\_epi / \text{Var}(G)$` , `n_cis`, `n_epi`, `module`, `coordinates`, and `trans_scale`), `cis_eqtl` (effective coefficient on centered dosage), `factor_eqtl` (raw hub effects), and – when `epistasis > 0` – `epi_eqtl` (interacting marker pairs: `snp1`, `snp2`, `effect`, `prod_mean`) truth tables, loadings, the reference constants (marker means), the profile, seed, and a per-gene variance budget. The budget closes as $v_cis + v_trans + v_epi + cis_trans_cov + cis_epi_cov + trans_epi_cov = \text{Var}(G)$ (the epistatic terms are zero without epistasis), plus `gr_cov`, the finite-sample genetic-residual covariance $2 \text{ Cov}(G, R)$. The genetic component is reconstructable from centered dosage: the cis part uses `cis_eqtl$effect` directly; the trans part of a gene is `trans_scale` times its module’s `factor_eqtl$hub_effect` (loading 1); the epistatic part sums `epi_eqtl$effect` times each pair’s centered dosage product ($Z_snp1 * Z_snp2 - prod_mean$). The truth tables are on the **unit scale**; when mimic is used the returned genetic_expression is affine-scaled to the mimicked moments, so this reconstruction must be multiplied by the per-gene reference\$gene_scale (which is 1 without mimic) to match it. mimic also adds a \$calibration element (the GREML $h2$, mean, var per gene, plus `n_factors` and `kappa`).

References

Falconer DS, Mackay TFC (1996) *Introduction to Quantitative Genetics*, 4th ed. Longman, Harlow – the additive dosage model and heritability used for each gene’s genetic component. The cis / trans / co-expression-module architecture and its default calibration follow reported eQTL structure: Albert et al. (2018) *eLife*, doi:10.7554/eLife.35471 (trans hotspots); GTEx Consortium (2020) *Science*, doi:10.1126/science.aaz1776 (cis-eQTL windows; multiple cis-eQTL per gene); Ouwers et al. (2020, published online 2019) *Eur. J. Hum. Genet.* 28(2):253-263, doi:10.1038/s41431-01905115 (cis/trans expression heritability). The mimic GREML step is an EMMA-style single-component REML: Kang et al. (2008) *Genetics* 178(3):1709-1723, doi:10.1534/genetics.107.080101. Falconer & Mackay is cited only for the general additive-dosage value and the definition of heritability; the cis / trans latent-factor construction and its finite-sample standardization and scaling are this package’s own composite design, not attributed to a single source.

See Also

`simulate_phenotype()`, `additive_value()`.

Examples

```
data("SNP55K_maize282_maf04")
tx <- simulate_transcriptome(SNP55K_maize282_maf04, n_genes = 200, seed = 1)
tx
dim(tx$expression)
```

single_seed_descent	<i>Single seed descent</i>
---------------------	----------------------------

Description

Advances every line to near-homozygosity by selfing one seed per line per generation, with no selection during inbreeding (Bernardo 2020). Line count is preserved: n lines in, n lines out.

Usage

```
single_seed_descent(x, generations = 5L, seed = NULL, interference = NULL)
```

Arguments

<code>x</code>	a Population, or a phenotype_sim built on one (its genotypes are used).
<code>generations</code>	number of selfing generations.
<code>seed</code>	optional RNG seed for the whole scheme: NULL (default; the ambient RNG stream is used) or one non-negative whole number. When given, the caller’s RNG state is restored on exit, so a seeded scheme does not disturb the surrounding random stream.

interference crossover interference model of `cross()` (see its section "Crossover interference"): NULL (default: the option `simplePHENOTYPES.interference` if set, else the gamma model with $\nu = 2.6$, $p = 0$), "poisson" (Poisson crossovers, the `isqg` stream, the default before 2.0.0.9003) or `list(nu = , p = )`. It is applied to every meiosis the scheme runs, in every generation, so a whole scheme can use one meiosis model.

Value

a Population of inbred lines.

References

Bernardo R (2020) *Breeding for Quantitative Traits in Plants*, 3rd ed. Stemma Press, Woodbury, Minnesota.

See Also

`bulk()`, `pedigree()`, `recurrent_selection()`, `select_ind()`.

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:10)
f1 <- cross(pop[1], pop[2], n = 20, seed = 1)
ril <- single_seed_descent(f1, generations = 5, seed = 2)
# Heterozygosity is driven down by repeated selfing:
mean(dosages(ril) == 0)
```

synthetic_map

Build a synthetic genetic map from physical marker positions

Description

Meiosis simulation (`cross()`, `selfcross()`, `double_haploid()`) needs recombination distances in Morgans, but marker data usually carries only physical positions in base pairs. `synthetic_map()` converts physical positions into a plausible centiMorgan map.

Usage

```
synthetic_map(
  chr,
  pos,
  total_cm = NULL,
  cm_per_mb = 0.73,
  centromere = NULL,
  suppression = 0.85,
  width = 0.15
)
```

Arguments

chr	chromosome identifier per marker.
pos	physical position per marker, in base pairs. Must be non-decreasing within each chromosome. Co-located markers (equal pos) are allowed and receive equal cM, which correctly implies no recombination between them.
total_cm	target genetic length per chromosome, in cM. Either a single value applied to every chromosome, a vector with one entry per chromosome, or NULL (default) to derive it from cm_per_mb.
cm_per_mb	average centiMorgans per megabase, used when total_cm is NULL. The default of 0.73 gives roughly 1,500 cM across a 2.05 Gb maize genome, the scale of published maize consensus maps.
centromere	centromere position per chromosome, in base pairs. Either a named vector (names matching chr values), an unnamed vector in the order chromosomes first appear, or NULL (default) to place each centromere at the midpoint of its chromosome's span.
suppression	strength of pericentromeric suppression, from 0 (a uniform map) up to, but not including, 1.
width	width of the suppressed region as a fraction of the chromosome span.

Value

A numeric vector of centiMorgan positions, one per marker, starting at 0 on each chromosome and non-decreasing within it.

This map is synthetic

The output is a *model*, not a measured linkage map. It is not derived from any published genetic map and its cM values should not be quoted as estimates of real recombination distances. Use it to give simulations a realistic recombination landscape; if you have a real map for your population, supply that instead.

Recombination model

Crossovers are not distributed uniformly along a chromosome. In maize and many other species recombination is strongly suppressed around the centromere and elevated toward the telomeres, so a large fraction of the physical genome contributes very little genetic distance. `synthetic_map()` reproduces that pattern by giving each position a local rate

$$r(p) = 1 - s \exp(-0.5((p - c)/(wL))^2)$$

where c is the centromere, L the physical span, s the suppression strength and w the relative width of the suppressed region. The rate is integrated across marker intervals and rescaled so each chromosome spans its target length, so cM is a monotone function of bp.

The Gaussian form here is a **phenomenological choice**, not a formula taken from a published model: it is a smooth, single-parameter way to reproduce the qualitative pattern of pericentromeric suppression. Bauer et al. (2013) document that recombination in maize is strongly reduced around

the centromere and report the genome-wide scale (~0.74 cM/Mb, ~1,508 cM mean map length) that the defaults approximate, but they do not prescribe this functional form or its parameters. Treat suppression and width as tuning knobs for a plausible landscape, not as estimated biological quantities.

Setting `suppression = 0` gives a uniform map, i.e. cM proportional to bp.

References

Bauer, E. *et al.* (2013). Intraspecific variation of recombination rate in maize. *Genome Biology* 14, R103. doi:10.1186/gb2013149r103 (for the pericentromeric-suppression pattern and genome-wide scale the defaults approximate; the Gaussian form itself is not from this source).

Examples

```
data("SNP55K_maize282_maf04")
geno <- SNP55K_maize282_maf04
cm <- synthetic_map(geno$chr, geno$pos)
# Genetic length of each chromosome:
tapply(cm, geno$chr, max)

# A uniform map instead, for comparison:
cm_flat <- synthetic_map(geno$chr, geno$pos, suppression = 0)
```

template_effects	<i>Realized-scale additive and dominance effects of a simulation</i>
------------------	--

Description

The per-locus additive effect a and dominance deviation d a `phenotype_sim` realizes, on the scale of its genetic values – the frozen architecture `genotypic_value()`, `combining_ability()` and `phenotype_value()` take. They are reconstructed from the simulation's own layers exactly as the breeding-value criterion does (each layer's stored effects times the `sqrt(prop)` / `sd` factor its realization applies), not estimated. So `genotypic_value(sim$geno, qtn, a, d)` reproduces the simulation's additive + dominance genetic value up to a constant (the realization centers each layer).

Usage

```
template_effects(sim, trait = 1L, rep = 1L)
```

Arguments

<code>sim</code>	a <code>phenotype_sim</code> with additive and/or dominance layers.
<code>trait</code>	trait index (default 1).
<code>rep</code>	replication (default 1).

Value

A data frame with columns qtn (marker index), snp, a, d, one row per causal locus. Refuses models with an epistasis layer or architecture = "complex", whose effects have no per-locus a / d form, and models with a *derived* transcriptome() layer that carries variance (prop > 0) for the trait: its heritable part is mediated by gene expression, not by per-locus effects, so a template would silently drop it (a layer with prop = 0, or one built on a real expression source, whose genetic content is not asserted, adds nothing to the genetic value and is allowed).

See Also

[combining_ability\(\)](#), [genotypic_value\(\)](#), [qtn_table\(\)](#)

Examples

```
data("SNP55K_maize282_maf04")
pop <- as_population(SNP55K_maize282_maf04, individuals = 1:40)
f2 <- selfcross(cross(pop[1], pop[2], n = 1, seed = 1), n = 60, seed = 2)
sim <- simulate_phenotype(f2, h2 = 0.6, seed = 3) |>
  additive(prop = 0.4, n_qtn = 5) |> dominance(prop = 0.2)
template_effects(sim)
```

transcriptome	<i>Transcriptome layer: a phenotype component driven by gene expression</i>
---------------	---

Description

Adds a phenotype component that is a **sparse linear function of standardized gene expression** – the continuous-predictor counterpart of [additive\(\)](#). It scores an expression source attached to the phenotype_sim (a real matrix via `simulate_phenotype(expression =)`, or a genome-derived transcriptome via `simulate_phenotype(transcriptome =)`): each causal gene's expression is z-scored on the scored individuals, multiplied by a slope, and summed; the sum is centered and scaled to the target variance share prop, exactly like [additive\(\)](#) (so only the relative slopes matter).

Usage

```
transcriptome(sim, prop = NULL, n_genes = NULL, genes = NULL, slopes = NULL)
```

Arguments

sim	a phenotype_sim carrying an expression source (see simulate_phenotype() 's expression / transcriptome arguments).
prop	required target share of phenotypic variance for the expression component (single number, or one per trait). Unlike additive() , it is <i>not</i> drawn from a remaining h2 budget – the expression component is a separate variance category, bounded (with all layers) to sum to at most 1.
n_genes	number of causal genes to draw (sparse); give this or genes.

genes	an explicit causal-gene set (names matching the expression source's row names, or integer gene indices); overrides <code>n_genes</code> . Must be non-empty and free of duplicates.
slopes	optional per-causal-gene slopes; a finite numeric vector, one per causal gene. When NULL, slopes are drawn $N(0, 1)$ (their absolute scale is irrelevant – the component is scaled to prop).

Details

The expression-mediated component is a **distinct variance category**, reported on its own budget row and scaled by prop outside the marker `h2` budget – because observed/derived expression is not purely additive-genetic. How much of it counts as heritable depends on the source:

- **Derived** (`transcriptome =`): the component splits into a genetically-mediated part `Tx_g` (the share of expression traced to the genome) and an environmentally-mediated part `Tx_e = Tx - Tx_g`. `Tx_g` is genetic, so it is added to `genetic_values()` and counts toward realized H2; `mediation_split()` reports the realized genetic / environmental / covariance shares of `V_P`.
- **Real** (`expression =`): the genetic content of an observed matrix is not asserted, so the whole component is treated as an environmental predictor and **excluded** from the genetic value and from H2.

Under `architecture = "pleiotropy"` or `"ld"`, the correlation design covers the marker layers only: a derived source's genome-mediated part `Tx_g` enters `genetic_values()` with a cross-trait correlation set by the per-trait genes and slopes, so the total genetic correlation is then not targeted at `cor ("pleiotropy")` or no longer comes solely from distinct linked loci (`"ld"`) (warned). A genotype-free basis (`simulate_phenotype(expression = ...)` with no `geno`), `qtn_table()` gene rows, and cross-population reuse of an architecture (`predict.transcriptome_sim()`) are all supported, as are mimic calibration of the generator to a user expression matrix (`simulate_transcriptome()`) and an RNA-seq count observation layer (`observe_counts()`).

With `simulate_phenotype(transcriptome = TRUE)` the derived transcriptome is built on the phenotype's whole genotype panel (before any individuals subsetting) and uses the phenotype's raw seed unchanged for `simulate_transcriptome()`.

Value

the `phenotype_sim` with the transcriptome layer added and the phenotype re-realized.

See Also

`simulate_phenotype()`, `additive()`, `mediation_split()`, `simulate_transcriptome()`.

Examples

```
data("SNP55K_maize282_maf04")
g <- SNP55K_maize282_maf04
tx <- simulate_transcriptome(g, n_genes = 200, seed = 1)
# phenotype driven by 20 causal genes of the derived transcriptome, plus a
# direct additive genome effect
ph <- simulate_phenotype(g, h2 = 0.5, seed = 2, transcriptome = tx) |>
  transcriptome(prop = 0.3, n_genes = 20) |>
```

```

    additive(prop = 0.2, n_qtn = 10)
  ph

```

vqtl

Add a variance-QTL (vQTL) component

Description

A variance QTL affects the *spread* of a trait rather than its mean: genotypes at a vQTL differ in how variable they are. Here `prop` is the marginal phenotypic-variance share assigned to a heterogeneous residual component; it is not genetic variance and is not included in broad-sense heritability. An explicit `prop` is therefore always required, even when the simulation has an `h2` budget.

Usage

```

vqtl(
  sim,
  prop = NULL,
  same_as_add = TRUE,
  n_qtn = NULL,
  qtn = NULL,
  dist = "geometric"
)

```

Arguments

<code>sim</code>	a <code>phenotype_sim</code> .
<code>prop</code>	proportion of total phenotypic variance (scalar or length <code>n_traits</code>). May be omitted when <code>h2</code> was set in <code>simulate_phenotype()</code> , in which case the layer takes whatever of that budget is unspent – so a single additive() layer gets <code>prop = h2</code> . When <code>prop</code> is given and <code>h2</code> is set, the layer <code>prop</code> values must sum to <code>h2</code> .
<code>same_as_add</code>	reuse the additive layer's QTNs (default <code>TRUE</code>). Supplying <code>qtn</code> fixes the loci instead, so <code>same_as_add</code> is then treated as <code>FALSE</code> .
<code>n_qtn</code>	number of vQTL loci for a fresh draw (<code>same_as_add = FALSE</code>); ignored, with a warning, when the loci are reused or fixed by <code>qtn</code> .
<code>qtn</code>	optional user-supplied QTNs for this layer, so you can fix the causal loci for one effect type while the others are drawn at random. Give marker names (matched against the map) or column indices; a vector is used for every trait, or a length- <code>n_traits</code> list sets each trait's loci separately (every element the same length). <code>n_qtn</code> is then taken from what you supply. Fixed loci are not redrawn by <code>vary_qtn</code> . (For <code>epistasis()</code> , <code>qtn</code> describes interacting sets; see there.) Every architecture accepts <code>qtn</code> , and each keeps its construction: under "independent" the loci carry the effect series; under "pleiotropy" (multi-trait) every locus affects every trait, so give a single vector (or the same loci for each trait in a list); loci that affect only some traits are partial pleiotropy, built

with `complex_phenotypes()`. If a correlation is controlled (`cor`, `pi`, ...) the correlated draw sets the effects and the genetic correlation is its target; otherwise the correlation is just an outcome of the shared loci, and an explicit effect sets the effects (without it the default draw does, with implicit `cor = 0`). `pi` only matters for controlling a correlation, and `pi < 1` cannot be used with fixed shared loci. Under "ld" (`ld_type = "direct"` only) give `qtn = list(trait1_loci, trait2_loci)` of equal length – the *i*-th locus of each is a linked pair on one chromosome, no marker may be causal for both traits, and each pair's `r2` is reported (by `qtn_table()`) and warned about when outside `[r2_min, r2_max]`. A marker that is monomorphic (or heterozygous in every individual) carries no variance: it is accepted with a warning, except where it makes the architecture undefined (an "ld" pair, or a pleiotropy layout with no varying locus), which is an error.

`dist` within-layer effect distribution (default "geometric").

Details

Conditional residual variance uses a log link, $\log \text{Var}(E_v \mid \text{genotype}) = \text{constant} + \text{loading}$, where loading is the sum of standardized vQTL genotype scores. This guarantees positive conditional variances. In fixed residual mode the heterogeneous residual is scaled to sample variance `prop`. In random mode independent normal draws use genotype-dependent variances whose average over individuals is `prop`, without sample centering or rescaling. Finite-sample covariance and, in random mode, residual variance fluctuations mean total realized phenotypic variance need not be exactly one.

The log-linear variance link is this package's simulation choice (a standard double-generalized-linear-model parameterization of the residual variance); it is **not** a reproduction of the specific phenotype generator or fitted variance model of any one reference. The references below are for the vQTL concept and its detection in experimental crosses and plants – the setting this layer creates data for – rather than the source of this exact equation. In particular, Murphy et al. simulate variance heterogeneity with a different generator and fit a DGLM to detect it; only the modeling idea is shared, not the formula used here.

Value

the updated `phenotype_sim`.

References

- Ronnegard, L. and Valdar, W. (2011). Detecting major genetic loci controlling phenotypic variability in experimental crosses. *Genetics* 188, 435–447. doi:10.1534/genetics.111.127068 (vQTL concept / detection.)
- Murphy, M.D., Fernandes, S.B., Morota, G. et al. (2022). Assessment of two statistical approaches for variance genome-wide association studies in plants. *Heredity* 129, 93–102. doi:10.1038/s41437-022005411 (plant vGWAS context; its simulation and DGLM differ from the log link used here.)

See Also

`additive()`, `dominance()`, `epistasis()`.

Examples

```
data("SNP55K_maize282_maf04")

# A vQTL on the same loci that carry the additive effects (the default).
ph <- simulate_phenotype(SNP55K_maize282_maf04, seed = 1) |>
  additive(prop = 0.3, n_qtn = 4) |>
  vqtl(prop = 0.2)
ph

# Independent variance-only loci: set same_as_add = FALSE and say how many.
simulate_phenotype(SNP55K_maize282_maf04, seed = 2) |>
  additive(prop = 0.3, n_qtn = 4) |>
  vqtl(prop = 0.2, same_as_add = FALSE, n_qtn = 2)
```

write_phenotypes	<i>Write realized phenotypes to disk</i>
------------------	--

Description

Writes the long (default) or wide table as a delimited text file or as JSON. Specialized exporters (gemma / plink / multi-file) are out of scope for the grammar core.

Usage

```
write_phenotypes(
  sim,
  file,
  format = c("long", "wide"),
  sep = "\t",
  file_type = c("text", "json"),
  qtn_file = NULL,
  split_markers = FALSE,
  markers_files = NULL,
  rep = 1L
)
```

Arguments

sim	a phenotype_sim.
file	output path.
format	"long" (default) or "wide".
sep	field separator (default tab); text files only.
file_type	"text" (default, a delimited file) or "json".
qtn_file	optional path; when given, the QTN table of qtn_table() is also written there with write_qtn_table() , in the same file_type and sep. Default NULL (not written, unless split_markers = TRUE).

split_markers	TRUE also writes the marker data as two files, causal and non-causal markers (see Details), plus the QTN table. Default FALSE. Needs genotypes: a phenotype built from expression alone has no markers.
markers_files	optional named character vector <code>c(causal = path, noncausal = path)</code> overriding the default marker file names used by <code>split_markers = TRUE</code> .
rep	replication(s) whose QTN table and causal set are written: a vector of replication numbers or "all". Default 1L. Only matters with <code>vary_qtn = TRUE</code> ; see Details. Does not affect the phenotype file.

Details

Delimited files are written with `data.table::fwrite()`'s default of up to 15 significant digits, so reading the file back reproduces the phenotypes to about 1e-14 (relative), not bit-for-bit.

JSON (`file_type = "json"`) is an array with one object per row of the chosen layout, e.g. `[{"id": "33-16", "trait": "Tr"}]` for "long", written as UTF-8 with 17 significant digits so every value reads back exactly (e.g. `jsonlite::read_json(file, simplifyVector = TRUE)`, or `pandas.read_json()` in Python). It needs the **jsonlite** package.

Value

file, invisibly, when no companion file is written; otherwise a named character vector of every path written (phenotypes, and whichever of `qtn_table`, `causal`, `noncausal` apply), invisibly.

Companion files (QTN table and split marker data)

The phenotype file alone does not say which markers produced it. Two options write that information next to it, in the same `file_type` (and sep):

- `qtn_file`: the QTN table of `qtn_table()`, every column, through `write_qtn_table()`.
- `split_markers = TRUE`: the marker data are split into two files, one with the **causal** markers and one with every **non-causal** marker, so a downstream analysis can be run on the non-causal markers alone (or on both) with the truth kept separately. The QTN table is written too (to `qtn_file`, or to a default name) because it is the companion of the causal file.

Default names are derived from file: `<stem>_qtn_table.<ext>`, `<stem>_qtn_markers.<ext>` and `<stem>_noncausal_markers.<ext>`, where `stem` is file without its extension (`ext`; `txt` or `json` when file has none). `markers_files = c(causal = ..., noncausal = ...)` overrides the two marker paths. All paths must be distinct.

Causal set. A marker is causal when any layer (`additive()`, `dominance()`, `epistasis()` – every member of an interacting set –, `vqtl()`, and under `architecture = "ld"` the loci of both traits) of any trait uses it in any of the replications selected by `rep`: the union of the `snps` values of `qtn_table()` over those replications, excluding the gene rows of `transcriptome()` layers (genes are not markers). With `vary_qtn = TRUE` the replications differ in their QTNs, so pass `rep = "all"` (or the vector of replications of interest) to make the causal file cover every replication of the phenotype file; a marker causal in one selected replication but not another is in the causal file, never in the non-causal one. `rep` only affects the QTN table and the causal set – the phenotype file always holds every replication, as before.

Text layout of the marker files (`file_type = "text"`): the package's numeric format – columns `snp`, `allele`, `chr`, `pos`, `cm` (then counted when the input carried it), then one column per simulated individual with the -1/0/1 dosage – so `as_numeric()` and `simulate_phenotype()` read each file back. `as_population()` reads a file too, provided it is not empty and its map is complete (`chr`, `pos` and `cm` without missing values): a matrix-origin simulation has no such map, and a file with no markers (no causal marker, or every marker causal) cannot found a population. Missing metadata (e.g. no allele label for a matrix input) is written as NA. Markers keep their map order. The causal text file has no embedded QTN table; the QTN table text file is its companion.

JSON layout of the marker files (`file_type = "json"`): one object `{"individuals": [...], "markers": [...]}`. `individuals` lists the simulated individuals' ids; each element of `markers` is an object with `snp`, `allele`, `chr`, `pos`, `cm`, `maf` (and counted when available) and `genotypes`, an array with one dosage per individual in `individuals` order (null for a missing value). The causal file adds, at the top level, `qtn_table` (the rows of `write_qtn_table()` for the selected replications, with a `rep` field when several), and in every marker object a `causal_for` array of `{trait, layer, set}` objects (plus `rep` when several replications) naming the layers that use the marker, so a reader can join without the table. Gene rows of `transcriptome()` layers appear in `qtn_table` but have no marker object. Numbers carry 17 significant digits, so `jsonlite::read_json(file, simplifyVector = TRUE)` returns the `individuals` vector, a `markers` data frame whose `genotypes` column is a list of vectors, and (causal file) the `qtn_table` data frame; in Python, `json.load()` then `pandas.DataFrame(obj["markers"])`.

Marker data can be large (tens of thousands of markers by hundreds of individuals); both layouts are written in chunks of markers. For a data-frame or matrix input the chunks stream from the user's object, so no second copy of the genotypes is made. A `Population` stores haplotypes, not dosages, so its one markers-by-individuals dosage matrix is built once and held in memory for the duration of the export.

Safety. Before anything is written, every path is checked: its directory must exist and be writable, it must not be a directory, and no two paths may resolve to the same file (relative and absolute spellings, symlinked directories and symlinked files – live or dangling – are resolved; on macOS the comparison also ignores case and Unicode normalisation form, on Windows case; two *hard links* to one file have unrelated names and are not detected). A file name longer than 255 bytes – including a default companion name derived from a long file – is rejected before anything is written. With companions, all files are first written into a private staging directory (`.<token>.stage`, created exclusively in each destination directory) under short fixed names that keep the destination's extension, so a `.gz` destination is still compressed by the text writer and no staged file can share a name with a requested output or with anything else in the directory. The set is then moved into place as a group: an existing file keeps its permission mode and is first moved into the staging directory as its backup, and if any step of that move fails the backups are put back. Should a backup itself fail to go back, its staging directory is kept and the error names it and the destination, so the previous content is never silently lost. A destination that is a symbolic link is written through (the link stays, its target is updated or created); a link loop or a chain of more than 40 links is an error. This is a best-effort guarantee built on `file.rename()` within one directory; it has been exercised on POSIX file systems, not on Windows, and a staging directory that cannot be removed (its parent made read-only meanwhile) is named in a warning rather than silently left. The plain one-file call writes directly, as earlier versions did. JSON numbers use a period as decimal mark whatever the session's `LC_NUMERIC`.

See Also

`phenotypes_long()`, `phenotypes_wide()`, `write_qtn_table()`, `qtn_table()`.

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, n_traits = 2, seed = 1) |>
  additive(prop = 0.5, n_qtn = 3)

# Written to a temporary directory here; use your own path in practice.
out <- file.path(tempdir(), "phenotypes.txt")
write_phenotypes(ph, file = out)
head(read.delim(out))

# Wide layout, comma separated.
out_wide <- file.path(tempdir(), "phenotypes_wide.csv")
write_phenotypes(ph, file = out_wide, format = "wide", sep = ",")
head(read.csv(out_wide))

# JSON, one object per row (needs the jsonlite package).
if (requireNamespace("jsonlite", quietly = TRUE)) {
  out_json <- file.path(tempdir(), "phenotypes.json")
  write_phenotypes(ph, file = out_json, file_type = "json")
  head(jsonlite::read_json(out_json, simplifyVector = TRUE))
  unlink(out_json)
}

# Phenotypes plus the QTN table that produced them.
out_qtn <- file.path(tempdir(), "phenotypes_qtn.txt")
write_phenotypes(ph, file = out, qtn_file = out_qtn)
head(read.delim(out_qtn))

# Phenotypes, QTN table, and the marker data split into causal and
# non-causal markers (default names derived from `file`).
files <- write_phenotypes(ph, file = out, split_markers = TRUE)
files
causal <- read.delim(files[["causal"]])
causal$snp # the QTNs, in map order
nrow(read.delim(files[["noncausal"]])) # every other marker

unlink(c(out, out_wide, out_qtn, files))
```

write_qtn_table

Write the QTN table of a simulation to disk

Description

Writes every column of `qtn_table()` – the v2 equivalent of the v1 `Additive_QTNs.txt` / `QTN_effects_summary.txt` files – as a delimited text file or as JSON. One replication is written by default; several (or `rep = "all"`) are stacked with a leading `rep` column, which is how the per-replication architectures of a `vary_qtn = TRUE` simulation are kept apart. Gene rows of `transcriptome()` layers are included (their `snp` is the gene identifier; see `qtn_table()`).

Usage

```
write_qtn_table(
  sim,
  file,
  rep = 1L,
  file_type = c("text", "json"),
  sep = "\t"
)
```

Arguments

sim	a phenotype_sim.
file	output path.
rep	replication(s) whose QTN architecture to write: a vector of replication numbers, or "all". Default 1L. With more than one, the rows are stacked and a leading rep column says which replication each row belongs to.
file_type	"text" (default, a delimited file) or "json".
sep	field separator (default tab); text files only.

Details

Text files are written with `data.table::fwrite()` (up to 15 significant digits, so effects read back to about $1e-14$ relative; missing values are empty fields). JSON follows the conventions of [write_phenotypes\(\)](#): an array with one object per row, UTF-8, 17 significant digits (every value reads back exactly), null for a missing value; it needs the **jsonlite** package. Read it back with `read.delim(file)` or `jsonlite::read_json(file, simplifyVector = TRUE)`.

Value

file, invisibly.

See Also

[qtn_table\(\)](#) for the columns, [write_phenotypes\(\)](#) which can call this and also split the marker data into causal and non-causal files.

Examples

```
data("SNP55K_maize282_maf04")
ph <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.5, seed = 1) |>
  additive(prop = 0.3, n_qtn = 3) |>
  epistasis(prop = 0.2, n_pairs = 2)

# Written to a temporary directory here; use your own path in practice.
out <- file.path(tempdir(), "qtn_table.txt")
write_qtn_table(ph, file = out)
read.delim(out)

# Every replication of a vary_qtn simulation, stacked with a `rep` column.
```

```
ph3 <- simulate_phenotype(SNP55K_maize282_maf04, h2 = 0.5, n_reps = 3,
                          vary_qtn = TRUE, seed = 1) |>
  additive(n_qtn = 2)
out3 <- file.path(tempdir(), "qtn_table_all_reps.csv")
write_qtn_table(ph3, file = out3, rep = "all", sep = ",")
read.csv(out3)

# JSON (needs the jsonlite package).
if (requireNamespace("jsonlite", quietly = TRUE)) {
  out_json <- file.path(tempdir(), "qtn_table.json")
  write_qtn_table(ph, file = out_json, file_type = "json")
  jsonlite::read_json(out_json, simplifyVector = TRUE)
  unlink(out_json)
}

unlink(c(out, out3))
```

Index

[.Population, 3

a_matrix, 4

a_matrix(), 87, 91, 93

additive, 5

additive(), 25, 43, 44, 47, 48, 110, 111, 114, 125, 126, 128

additive_value, 9

additive_value(), 57–60, 68, 80, 81, 83

as_numeric, 11

as_numeric(), 14, 16, 36, 50, 52, 66, 87, 88, 131

as_population, 14

as_population(), 12, 13, 16, 36, 37, 55, 69, 71, 77, 78, 86–88, 131

breed_composition, 16

breed_composition(), 15, 40, 41, 61, 62

bulk, 17

bulk(), 37, 102, 108, 122

c(), 77, 78

c.Population, 18

c.Population(), 12, 16, 50, 87, 88

coheritability, 19

coheritability(), 63

combining_ability, 20

combining_ability(), 37, 96, 108, 124, 125

complex_phenotypes, 23

complex_phenotypes(), 6, 25, 42, 47, 128

cor_ar1, 24

create_phenotypes, 25

cross, 35

cross(), 12–16, 18, 21, 37, 39–41, 46, 49, 53, 55, 66, 69, 70, 77–79, 87, 88, 95, 100, 109, 111, 122

cross_usefulness, 38

cross_usefulness(), 37

crossbreed, 40

crossbreed(), 17, 37, 62

dominance, 41

dominance(), 8, 9, 25, 47, 48, 52, 110, 111, 114, 128

dosages, 44

dosages(), 10, 56, 60, 88

double_haploid, 45

double_haploid(), 14, 16, 37, 49, 53, 69, 70, 77, 87, 88, 109, 111, 122

epistasis, 46

epistasis(), 6, 9, 25, 42, 44, 89, 110, 111, 127, 128

families, 48

families(), 70, 78, 94, 96

filter_geno, 49

filter_geno(), 12, 44

founders_coalescent, 53

g_matrix, 55

g_matrix(), 5, 76, 77, 87, 91–93, 106

genetic_values, 56

genetic_values(), 10, 19, 59, 63, 72, 73, 81, 83, 86, 97, 105, 126

genotypic_value, 57

genotypic_value(), 20–22, 62, 80–82, 95, 124, 125

graphics::par(), 86

gxe_value, 59

gxe_value(), 82

haplotypes, 60

haplotypes(), 88

heterosis, 61

heterosis(), 17, 41

liability_threshold, 62

liability_threshold(), 19

mabc_select, 64

mabc_select(), 67, 68, 98, 99

- marker_select, [67](#)
- mate, [69](#)
- mate(), [21](#), [22](#), [37](#), [40](#), [41](#), [45](#), [70–72](#), [88](#)
- mating_design, [70](#)
- mating_design(), [69](#), [70](#)
- mediation_split, [72](#)
- mediation_split(), [24](#), [112](#), [126](#)
- n_individuals, [73](#)
- observe_counts, [74](#)
- observe_counts(), [126](#)
- optimum_contribution, [75](#)
- optimum_contribution(), [8](#), [56](#), [101](#), [102](#)
- parentage, [77](#)
- parentage(), [4](#), [5](#), [15](#), [17](#), [35](#), [49](#), [69](#), [70](#), [94](#)
- pedigree, [78](#)
- pedigree(), [18](#), [37](#), [83](#), [99](#), [101–104](#), [107](#), [108](#), [122](#)
- phenotype_value, [81](#)
- phenotype_value(), [21](#), [59](#), [60](#), [92](#), [94](#), [124](#)
- phenotypes_long, [84](#)
- phenotypes_long(), [24](#), [57](#), [85](#), [131](#)
- phenotypes_wide, [85](#)
- phenotypes_wide(), [84](#), [131](#)
- plot.phenotype_sim, [85](#)
- Population, [49](#), [53](#), [102](#), [111](#)
- population_from_haplotypes, [86](#)
- population_from_haplotypes(), [54](#), [55](#), [60](#)
- population_trait, [88](#)
- population_trait(), [63](#), [80](#), [113](#)
- predict.transcriptome_sim, [90](#)
- predict.transcriptome_sim(), [119](#), [120](#), [126](#)
- predict_ebv, [91](#)
- predict_ebv(), [5](#), [94](#), [108](#)
- prediction_accuracy, [93](#)
- prediction_accuracy(), [92](#), [93](#)
- progeny_test, [94](#)
- progeny_test(), [37](#)
- qtn_table, [97](#)
- qtn_table(), [6](#), [10](#), [42](#), [57](#), [63](#), [85](#), [86](#), [114](#), [125](#), [128–133](#)
- recurrent_parent_recovery, [98](#)
- recurrent_parent_recovery(), [64](#), [66](#)
- recurrent_selection, [99](#)
- recurrent_selection(), [18](#), [37](#), [77](#), [80](#), [83](#), [101–104](#), [107](#), [108](#), [122](#)
- sample_parents, [101](#)
- select_ind, [102](#)
- select_ind(), [8](#), [10](#), [22](#), [40](#), [48](#), [49](#), [58](#), [59](#), [63](#), [65](#), [66](#), [68](#), [76](#), [77](#), [79](#), [80](#), [83](#), [88](#), [89](#), [93](#), [96](#), [100](#), [101](#), [108](#), [113](#), [122](#)
- selection_methods, [108](#)
- selfcross, [109](#)
- selfcross(), [14](#), [16](#), [35](#), [37](#), [39](#), [46](#), [49](#), [53](#), [66](#), [69](#), [70](#), [77](#), [87](#), [88](#), [111](#), [122](#)
- simulate_phenotype, [110](#)
- simulate_phenotype(), [6–8](#), [15](#), [23–25](#), [28](#), [33](#), [42](#), [43](#), [46](#), [49](#), [52](#), [63](#), [80](#), [81](#), [83](#), [87–89](#), [102](#), [108](#), [121](#), [125–127](#), [131](#)
- simulate_transcriptome, [117](#)
- simulate_transcriptome(), [74](#), [75](#), [90](#), [126](#)
- single_seed_descent, [121](#)
- single_seed_descent(), [18](#), [37](#), [40](#), [102](#), [108](#)
- SNP55K_maize282_maf04, [15](#), [49](#), [110](#)
- synthetic_map, [122](#)
- synthetic_map(), [13](#), [16](#), [36](#)
- template_effects, [124](#)
- template_effects(), [21](#), [22](#), [68](#)
- transcriptome, [125](#)
- transcriptome(), [73](#), [80](#), [89](#), [111–113](#), [115](#)
- vqtl, [127](#)
- vqtl(), [9](#), [25](#), [44](#), [48](#), [80](#), [89](#), [110](#), [112](#), [113](#), [115](#)
- write_phenotypes, [129](#)
- write_phenotypes(), [84](#), [85](#), [133](#)
- write_qtn_table, [132](#)
- write_qtn_table(), [129–131](#)