

Package ‘hsquared’

October 21, 2026

Title 'R' Interface for 'Julia'-Backed Quantitative-Genetic Models

Version 0.9.0

Description Provides an interface for quantitative-genetic models in 'R', backed by the 'HSquared.jl' engine for 'Julia'. Fits Gaussian pedigree animal models by restricted maximum likelihood, using average-information restricted maximum likelihood (AI-REML; Gilmour et al. (1995) <[doi:10.2307/2533274](https://doi.org/10.2307/2533274)>) for the univariate model. Returns heritability, variance components, breeding values, and covariance and correlation matrices for multivariate models. Formula and pedigree validation can run without 'Julia'. Model fitting requires a local 'Julia' and 'HSquared.jl'; use `hs_control(engine = ``validate")` to preview the model contract without fitting. What you may report is listed on the package article 'Can I fit and report this?', not in `validation_status()`, which is a developer evidence table. Model-specific restrictions are documented in that article. Uncertainty intervals are not coverage-calibrated.

License MIT + file LICENSE

URL <https://itchyshin.github.io/hsquared/>,
<https://github.com/itchyshin/hsquared>,
<https://github.com/itchyshin/HSquared.jl>

BugReports <https://github.com/itchyshin/hsquared/issues>

Encoding UTF-8

Language en-US

Imports ggplot2, lifecycle, Matrix, stats

Suggests enhancer, JuliaCall, knitr, lme4, nadiv, pedigreemm, rmarkdown, sommer, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Shinichi Nakagawa [aut, cre, cph] (ORCID:
[<https://orcid.org/0000-0002-7765-5182>](https://orcid.org/0000-0002-7765-5182)),
 Yefeng Yang [aut, cph] (ORCID: [<https://orcid.org/0000-0002-8610-4016>](https://orcid.org/0000-0002-8610-4016)),
 Szymon Drobnik [aut, cph] (ORCID:
[<https://orcid.org/0000-0001-8101-6247>](https://orcid.org/0000-0001-8101-6247))

Maintainer Shinichi Nakagawa <itchyshin@gmail.com>

Repository CRAN

Date/Publication 2026-10-21 13:10:26 UTC

Contents

animal	3
backend_info	4
breeding_values	5
common_env_effects	5
common_env_proportion	6
common_env_proportion_interval	7
covariance_standard_errors	8
covariance_structure_lrt	9
data_status	10
direct_maternal_extractors	11
factor_g_extractors	13
fit_diagnostics	14
fixef	15
formula_status	15
gamma_matrix	16
genomic_markers	16
gwas	18
g_matrix_geometry	20
heritability	22
heritability_interval	23
hsquared	24
hsquared-autoplot	25
hs_control	26
hs_data	32
hs_recovery_forest	33
inference_blocks	34
marker_extractors	35
maternal_effects	36
maternal_proportion	36
metafounder_effects	37
metafounder_groups	38
model_spec	38
multivariate_extractors	39
permanent_effects	40
plot.hsquared_fit	41
prediction_error_variance	42

qg_effect_markers	42
random_regression_extractors	44
ranef	46
reliability	47
repeatability	48
repeatability_interval	48
response_scale_methods	49
theme_hsqared	50
validation_status	50
variance_components	51
variance_component_standard_errors	52
Index	54

animal	<i>Animal-model formula marker</i>
--------	------------------------------------

Description

[Experimental]

Usage

animal(formula, pedigree = NULL, ...)

Arguments

- formula

A random-effect expression. The v0.1 parser accepts 1 | id.
- pedigree

A pedigree data frame with individual, sire, and dam columns. Optional only when the enclosing `hsquared()` or `model_spec()` call uses `data = hs_data(..., pedigree = ...)`. In v0.1 this route rejects any row with the same known sire and dam (selfing); no argument here reaches the engine’s existing `allow_selfing` flag. The current workaround for a selfing or hermaphroditic pedigree is `relmat(1 | id, K = A)` with a hand-built or `AGHmatrix`-built relationship matrix A (an experimental route; see `?hs_control`).
- ...

Reserved for future syntax such as `cov =`.

Details

`animal()` is the additive-genetic term in an `hsquared()` formula. Write `animal(1 | id, pedigree = ped)`, or `animal(1 | id)` when data is an `hs_data()` object with a pedigree component. Fitting happens in `hsquared()`, not here: calling `animal()` on its own is a syntax marker and returns `NULL`.

Value

`NULL`, invisibly. The call is interpreted by `hsquared()` when it appears inside a model formula.

Examples

```
# n = 4 is a syntax demo, not a number for a paper.
ped <- data.frame(
  id = c("sire", "dam", "off1", "off2"),
  sire = c(NA, NA, "sire", "sire"),
  dam = c(NA, NA, "dam", "dam")
)
dat <- data.frame(
  id = c("sire", "dam", "off1", "off2"),
  sex = c("m", "f", "m", "f"),
  weight = c(42, 38, 40, 37)
)
hsquared(
  weight ~ sex + animal(1 | id, pedigree = ped),
  data = dat,
  control = hs_control(engine = "validate")
)
model_spec(
  weight ~ sex + animal(1 | id, pedigree = ped),
  data = dat
)
```

backend_info

Inspect planned compute backends

Description

[Experimental]

Usage

```
backend_info(control = hs_control())
```

Arguments

control An object created by [hs_control\(\)](#).

Details

`backend_info()` reports which backend names are accepted by [hs_control\(\)](#) and whether any of them are execution-ready from the R package. In the current package state, backend names are control metadata only: they are selectable but not dispatched.

Value

A data frame of backend status records with class "hs_backend_info".

breeding_values	<i>Extract breeding values</i>
-----------------	--------------------------------

Description**[Experimental]****Usage**`breeding_values(object, ...)``EBV(object, ...)``BLUP(object, ...)`**Arguments**

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`breeding_values()` is part of the planned v0.1 fitted-object contract. It works for `hsquared_fit` objects that contain a Julia result. `EBV()` and `BLUP()` are aliases for applied quantitative-genetic workflows.

Value

Breeding value results for `hsquared_fit` objects.

common_env_effects	<i>Extract common-environment effects</i>
--------------------	---

Description**[Experimental]****Usage**`common_env_effects(object, ...)`**Arguments**

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`common_env_effects()` returns the predicted common-environment effects of the opt-in, experimental two-effect model (`target = "two_effect"`).

Value

Common-environment effect results for two-effect `hsquared_fit` objects.

`common_env_proportion` *Extract the common-environment variance ratio (c2)*

Description

[Experimental]

Usage

```
common_env_proportion(object, ...)
```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`common_env_proportion()` returns the estimated common-environment variance ratio $c2 = \sigma_{c2} / (\sigma_{a2} + \sigma_{c2} + \sigma_{e2})$ from the opt-in, experimental two-effect model (`target = "two_effect"` with a `common_env()` term).

Falconer fence: $c2$ is a **variance ratio** (the proportion of phenotypic variance from the shared common-environment effect), **not a heritability**. In the same fit, `heritability()` reports narrow-sense $h2 = \sigma_{a2} / (\sigma_{a2} + \sigma_{c2} + \sigma_{e2})$ *within this two-effect model*. The returned data frame carries this note as an "interpretation" attribute. For an interval on $c2$, see `common_env_proportion_interval()`.

Value

A one-row data frame with term and estimate, plus an "interpretation" attribute, for two-effect `hsquared_fit` objects.

```
common_env_proportion_interval
```

Extract an experimental common-environment / maternal variance-ratio interval

Description

[Experimental]

Usage

```
common_env_proportion_interval(object, ...)
```

```
maternal_proportion_interval(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

`common_env_proportion_interval()` and `maternal_proportion_interval()` return an **experimental** large-sample (logit delta-method) confidence interval for the second variance ratio ($c2 / m2$, `ratio2`) of the opt-in two-effect model, available only when the fit contains it. On the same two-effect fit, `heritability_interval()` returns the matching interval for the direct heritability ($h2$, `ratio1`).

This mirrors the engine row V3-TWOEFFECT-REML: the interval is the asymptotic delta-method CI built from the two-effect REML observed information (the finite-difference Hessian of the two-effect REML log-likelihood at the optimum). It is **asymptotic, delta-method, REML only, and NOT coverage-calibrated** – on small samples the REML surface is flat and the interval is unreliable (the parametric bootstrap is the only finite-sample- aware path). No calibrated coverage is claimed.

Boundary honesty: when the ratio's variance component sits on the boundary ($\sigma \rightarrow 0$) it is flagged (`boundary = TRUE`) and `lower/upper` are NA, never a spuriously tight CI. `lower_clamped/upper_clamped` flag when a bound reaches the numerical ($0, 1$) rails.

Falconer fence: $c2 / m2$ is a **variance ratio**, not a heritability (see `common_env_proportion()` / `maternal_proportion()`).

Value

A one-row data frame with `estimate` (the ratio), `lower`, `upper`, `level`, `se`, `lower_clamped`, `upper_clamped`, and `boundary`, plus an "interpretation" attribute, for two-effect `hsquared_fit` objects that contain it.

 covariance_standard_errors

Extract experimental multivariate covariance standard errors

Description

[Experimental]

Usage

```
covariance_standard_errors(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

covariance_standard_errors() returns **experimental** large-sample (delta-method) standard errors for the multivariate genetic/residual covariance and correlation matrices and per-trait h^2 , for an opt-in **unstructured** multivariate fit, when the engine returned them.

Heavy caveats (engine multivariate REML validation row, partial): the strict per-seed recovery gate is still a non-pass (7/12 unstructured seeds in the updated study), but a 12-seed bias/MCSE study (twin HSquared.jl#78) shows **no detectable bias** – all six covariance parameters have $|\text{bias}| \leq 2 \times \text{MCSE}$ (largest $0.84 \times \text{MCSE}$), a low-power non-rejection consistent with an unbiased estimator (not a proof), with EBV accuracy ~ 0.90 in both traits; a cold-start replication (#79) reaches the same optimum on all 12 seeds, so it is not a warm-start artifact. The per-seed gate failures reflect sampling variance of the estimated G at this design, not a detected bias. These SEs remain asymptotic, REML-only, **unstructured-only** (the engine refuses structured / factor-analytic fits, whose loadings are rotation-nonidentified), omitted at a flat/boundary optimum, not coverage-calibrated, with no external comparator, and not a validated capability.

The returned genetic_correlation SE is on the correlation (r) scale. A naive symmetric interval built as estimate $\pm 1.96 \times \text{se}$ on that scale can leave $(-1, 1)$ for strong correlations. The engine instead builds its own genetic-correlation interval on the **Fisher-z scale** ($z = \text{atanh}(r)$, $\text{se}_z = \text{se}_r / (1 - r^2)$, endpoints $\tanh(z \pm q \times \text{se}_z)$), which always lands inside $(-1, 1)$. That range-respecting endpoint is a separate property from calibration: the Fisher-z transform does not make these SEs coverage-calibrated (see above).

Value

A named list of standard-error matrices genetic_covariance, residual_covariance, genetic_correlation, residual_correlation, and a per-trait heritability SE vector, for hsquared_fit objects that contain them.

 covariance_structure_lrt

Likelihood-ratio test for genetic covariance structure

Description

[Experimental]

Usage

```
covariance_structure_lrt(constrained, full, ...)
```

Arguments

constrained, full	Two <code>hsquared_fit</code> objects from the multivariate model; full must nest constrained (more genetic covariance parameters).
...	Unused.

Details

`covariance_structure_lrt(constrained, full)` is an **experimental** nested likelihood-ratio test comparing two multivariate fits **on the same data**: a constrained genetic structure (currently `genetic_structure = "diagonal"`) against the full "unstructured" fit. The statistic is $2 * (\logLik(full) - \logLik(constrained))$ on $df = n_genetic_params(full) - n_genetic_params(constrained)$.

For diagonal-vs-unstructured the null (off-diagonal genetic covariances = 0) is interior, so the χ^2 reference is exact (`boundary = FALSE`). Structures whose null lies on a rank/PSD boundary (low-rank / factor-analytic) would need a χ^2 -mixture correction and are gated out of the R bridge for now.

It mirrors the engine multivariate REML validation row (partial): asymptotic, REML-only, dense validation-scale, with the multivariate recovery calibration not yet passed – a reported test, not a validated one. Both fits must be on the same response, fixed effects, and pedigree.

Value

A one-row data frame with `statistic`, `df`, `pvalue`, `boundary`, and the `constrained / full` genetic-structure labels.

data_status

Inspect hsquared data-container status

Description

[Experimental]

Usage

```
data_status(data)
```

Arguments

data An `hs_data()` object.

Details

`data_status()` gives a direct user-facing view of the checks stored in an `hs_data()` object. It reports component presence, ID overlap diagnostics, pedigree diagnostics, expression-feature diagnostics, genotype-column diagnostics, and marker-map/genotype-marker alignment diagnostics when those inputs are supplied. When `annotation_id` is supplied, it reports expression-feature annotation coverage diagnostics. When `environment_id` is supplied, it also reports environment metadata coverage diagnostics. It does not fit models, build genomic relationship matrices, add eQTL terms, or add environment-effect terms.

Value

An "hs_data_status" object.

Examples

```
ped <- data.frame(
  id = c("a", "b", "c", "d"),
  sire = c(NA, NA, "a", "a"),
  dam = c(NA, NA, "b", "c")
)
dat <- data.frame(
  y = c(1, 2, 3),
  sex = c("f", "m", "f"),
  id = c("a", "c", "d")
)
bundle <- hs_data(phenotypes = dat, pedigree = ped)
data_status(bundle)
```

direct_maternal_extractors

Direct-maternal correlated model extractors

Description

[Experimental]

Usage

```
direct_heritability(object, ...)

direct_variance(object, ...)

partner_variance(object, ...)

direct_maternal_covariance(object, ...)

total_heritability(object, ...)
```

Arguments

object	An <code>hsquared_fit</code> from <code>target = "direct_maternal"</code> .
...	Not used.

Details

These extractors summarize an opt-in, **covered at validation scale** direct-maternal correlated model (`target = "direct_maternal"`), fitted with `animal(1 | id, pedigree = ped) + maternal_genetic(1 | dam)`. Validation-scale coverage concerns point estimates within the documented scope; this is not a production-scale or calibrated-interval claim. The model estimates a 2x2 genetic covariance matrix G_{dm} between the direct additive and maternal additive effects, plus a residual variance.

Willham fence: the direct-maternal model (Willham 1963, 1972) distinguishes the **direct** narrow-sense heritability $h^2_d = \sigma_{ad} / \sigma^2_P$ from the maternal variance ratio $m^2 = \sigma_{am} / \sigma^2_P$, the Willham total (selection-response) heritability $h^2_T = (\sigma_{ad} + 1.5\sigma_{dm} + 0.5\sigma_{am}) / \sigma^2_P$, and the genetic covariance σ_{dm} . $\sigma^2_P = \sigma_{ad} + \sigma_{am} + \sigma_{dm} + \sigma_{e2} = \text{Var}(y_i)$ for a non-inbred base (coefficient 1 on σ_{dm} because $2 * A[i, dam] = 2 * (1/2) = 1$). **h^2 is denominator-dependent under maternal effects; compare (co)variance components, not h^2 values, across software (ASReml/BLUPF90/WOMBAT/sommer/MCMCglmm all leave σ^2_P to the user).** A **negative** genetic correlation r_{am} is real and biologically expected in many livestock traits; it reflects an antagonistic direct-maternal relationship and does not by itself indicate a model failure.

Note: the 2x2 G_{dm} formulation is due to Willham (1963, 1972), not Falconer (1965). Falconer's single-m model fixes $r_{am} = +/-1$ and is a special case (Bijma 2011).

`direct_heritability()` returns the direct narrow-sense heritability with an interpretation attribute. `total_heritability()` returns Willham's selection-response heritability h^2_T . `direct_variance()`,

`partner_variance()`, and `direct_maternal_covariance()` return the raw variance-component scalars. `genetic_correlation()` on a direct-maternal fit returns the between-effect correlation `r_am`.

Value

- `direct_heritability()`: a one-row data frame with term and estimate, plus an "interpretation" attribute.
- `total_heritability()`: a one-row data frame with term = "total_willham" and estimate, plus an "interpretation" attribute.
- `direct_variance()`, `partner_variance()`, `direct_maternal_covariance()`: single numerics (the raw variance component).

See Also

[variance_components\(\)](#), [genetic_correlation\(\)](#), [maternal_effects\(\)](#), [total_heritability\(\)](#)

Examples

```
# Synthetic result object: demonstrate extraction, not model fitting.
# Phenotypic variance is 0.30 + 0.15 - 0.10 + 0.55 = 0.90.
fit_dm <- structure(list(
  spec = list(target = "direct_maternal"),
  result = list(
    direct_variance = 0.30, partner_variance = 0.15, covariance = -0.10,
    variance_components = data.frame(
      component = c("direct", "maternal", "covariance", "residual"),
      estimate = c(0.30, 0.15, -0.10, 0.55)),
    heritability = data.frame(term = "direct", estimate = 0.30 / 0.90),
    genetic_correlation = data.frame(
      term_1 = "direct", term_2 = "maternal",
      estimate = -0.10 / sqrt(0.30 * 0.15)),
    maternal_effects = data.frame(id = "dam1", value = 0.1),
    converged = TRUE)
), class = "hsquared_fit")

direct_heritability(fit_dm)      # h2_d (direct narrow-sense)
total_heritability(fit_dm)      # h2_T (Willham selection-response)
genetic_correlation(fit_dm)     # r_am (may be negative)
direct_variance(fit_dm)         # sigma_ad
partner_variance(fit_dm)        # sigma_am
direct_maternal_covariance(fit_dm) # sigma_dm
variance_components(fit_dm)     # all four components
maternal_effects(fit_dm)        # maternal EBVs
```

factor_g_extractors *Reserved factor-analytic and G-matrix extractors*

Description

[Experimental]

Usage

genetic_loadings(object, ...)

specific_variance(object, ...)

latent_breeding_values(object, ...)

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

These extractor names are reserved for future factor-analytic G-matrix results. The current package can report invariant covariance and correlation matrices from multivariate fits, but it does not yet expose interpreted loadings, uniqueness/specific variance, or latent breeding values. Loading columns (`genetic_loadings()`) and latent breeding values (`latent_breeding_values()`) are rotation-nonunique until a rotation or constraint policy is validated. Future `hsquared_fit` methods reserve effect and rotation controls, but these controls currently error rather than implying that loading axes are interpretable. The **rotation-invariant** genetic eigenstructure and evolvability geometry are available now via [eigen_G\(\)](#) and the [g_matrix_geometry](#) family.

`specific_variance()` (Psi, the factor-analytic specific/unique variances) is a **different case**: for $G = \Lambda \Lambda' + \Psi$, rotating $\Lambda \rightarrow \Lambda Q$ with $Q Q' = I$ leaves $\Lambda Q Q' \Lambda' = \Lambda \Lambda'$ unchanged, so Psi is rotation-**invariant** and identified, not rotation-nonunique. The engine payload for structured multivariate fits carries Psi as `genetic_uniqueness`, explicitly marked identified, alongside the excluded, rotation-nonidentified loadings. This extractor still errors on the R surface, but for a different reason: `genetic_structure = "factor_analytic"` (and "lowrank") are planned on the R surface and are not yet activated on the R-to-Julia bridge, not because Psi is unidentified.

Value

These reserved extractors currently error for `hsquared_fit` objects.

fit_diagnostics	<i>Inspect fitted-model diagnostics</i>
-----------------	---

Description

[Experimental]

Usage

```
fit_diagnostics(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

fit_diagnostics() returns a compact diagnostics table for an hsquared_fit object. It is an inspection helper over the current result payload: it does not refit the model, rerun validation checks, or promote an experimental bridge target to production support.

Two rows share the word "boundary" but report unrelated things (hsquared#230):

- at_boundary / at_boundary_condition derive from the fitted **estimates**: a variance component at or near zero, or an inadmissible negative estimate (hs_fit_boundary_class()). Emitted for the Gaussian animal-model / genomic / single-step routes that report one primary variance-share layout; the _condition row is dropped (not NA) when the fit is interior, since there is nothing extra to say.
- search_boundary / search_boundary_condition mirror the non-Gaussian bridge's **optimizer** state: result\$boundary (documented as fit\$result\$boundary in ?hs_control) reports whether HSquared.jl's single-variance Brent search stopped on the rail of its log-scale bracket ($\log(\text{initial}\sigma_a^2) \pm 6$, HSquared.jl#327) or, under restart_check = TRUE, whether the two-start gap fence fired; either way the estimate is a function of the search, not the data. Only fits whose result carries a boundary field (the non-Gaussian route) gain this pair; a Gaussian engine = "fit" fit gains neither row (not even NA). Unlike at_boundary_condition, search_boundary_condition is always emitted alongside a present search_boundary flag, reading "interior" rather than disappearing when the flag is FALSE – the search-bracket rail is a binary optimizer state with no benign-vs- inadmissible split to only sometimes report. In practice a search_boundary = TRUE fit is refused before it reaches an hsquared_fit object at all, so a returned fit's search_boundary is always FALSE; TRUE is reachable only on a hand-built result object.

Value

A data frame with metric and value columns and class "hs_fit_diagnostics".

fixef	<i>Extract fixed effects</i>
-------	------------------------------

Description**[Experimental]****Usage**

```
fixef(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

fixef() is part of the planned v0.1 fitted-object contract for hsquared_fit objects.

Value

Fixed-effect estimates for hsquared_fit objects.

formula_status	<i>Inspect formula grammar status</i>
----------------	---------------------------------------

Description**[Experimental]****Usage**

```
formula_status()
```

Details

formula_status() reports which pieces of the planned hsquared() formula language are parsed today, reserved as syntax markers, or still roadmap-only. It is a status table, not a model-fitting helper.

The printed header is derived from the rows being printed, so it cannot lag the table. Covered-versus-experimental fences live in \$current_behavior and are not dumped into the default print.

Value

A data frame of formula grammar records with class "hs_formula_status".

Examples

```
formula_status()
```

gamma_matrix	<i>Extract a supplied metafounder Gamma matrix</i>
--------------	--

Description

[Experimental]

Usage

```
gamma_matrix(object, ...)
```

Arguments

- object A fitted model object.
- ... Reserved for future arguments.

Details

gamma_matrix() returns the supplied metafounder Gamma matrix carried by an experimental metafounder or H^Gamma single-step hsquared_fit object. It is provenance for the fitted relationship, not an estimated parameter.

Value

A numeric matrix with metafounder labels when the fitted object contains a supplied Gamma payload.

genomic_markers	<i>Genomic and QTL formula markers</i>
-----------------	--

Description

[Experimental]

Usage

```
genomic(formula, G = NULL, Ginv = NULL, markers = NULL, ...)
```

```
single_step(
  formula,
  H = NULL,
  Hinv = NULL,
  pedigree = NULL,
  markers = NULL,
  group = NULL,
  Gamma = NULL,
  tau = 1,
  omega = 1,
  blend_weight = 0,
  ridge = 0,
  ...
)
```

```
markers(M, model = c("random", "fixed", "scan"), ...)
```

```
marker_scan(M, map = NULL, ...)
```

```
qtl_scan(position, genotype_probs = NULL, ...)
```

Arguments

formula	A random-effect expression such as <code>1 id</code> .
G, Ginv, H, Hinv	Relationship or precision matrices for future genomic and single-step models.
markers	An individual-by-marker numeric dosage matrix with rows named by genotyped ID. Entries must be the biallelic allele count 0/1/2 (or an imputed dosage in the closed interval $[0, 2]$); a matrix in the centered $-1/0/1$ convention must be recoded (add 1) before it is passed in. For <code>genomic()</code> , the engine uses sample allele frequencies, VanRaden method 1, and ridge 0.01. For <code>single_step()</code> , this is the genotyped-subset marker matrix used to construct the genomic relationship.
...	Reserved for future syntax.
pedigree	A pedigree data frame (id, sire, dam) for the single-step H^{-1} <i>construction</i> path (<code>single_step(1 id, pedigree = ped, markers = M)</code>), in place of a precomputed Hinv.
group	Animal-to-metafounder group labels for the future H^{Gamma} single-step path; must be supplied together with Gamma.
Gamma	A supplied metafounder relationship matrix for the future H^{Gamma} single-step path; supplied, not estimated, and must be supplied together with group.
tau, omega, blend_weight, ridge	Single-step construction tuning knobs (Aguilar et al. 2010); defaults <code>tau = omega = 1, blend_weight = ridge = 0</code> .

M	A marker or dosage matrix for future marker-effect and scan models.
model	Planned marker-effect mode.
map	A marker map for future marker scans.
position	A chromosome-position table or variable for future QTL scans.
genotype_probs	Genotype probabilities for future interval/QTL scans.

Details

These functions provide readable formula vocabulary for genomic, single-step, marker-effect, GWAS, and QTL/eQTL models. Called directly they are inert (they return NULL); they take meaning only inside an `hsquared()` formula. `genomic()` fits a narrow Gaussian REML random-intercept model only through the explicit `engine = "julia"`, `target = "genomic"` path. `single_step()` also remains opt-in and experimental. The reported genomic coefficient is $\sigma_{g2} / (\sigma_{g2} + \sigma_{e2})$ on the declared relationship scale, not generally an average marginal phenotypic-variance fraction or pedigree/population heritability. Accepted genomic forms are `genomic(1 | id, Ginv = Ginv)` or `genomic(1 | id, markers = M)` (GREML, or SNP-BLUP via `target = "snp_blup"`), and `single_step(1 | id, Hinv = Hinv)` (a precomputed inverse) or `single_step(1 | id, pedigree = ped, markers = M)` (the engine constructs H^{-1} from the pedigree + genotyped-subset markers via `target = "single_step_construct"`). Adding group plus supplied Gamma fits a supplied-Gamma, validation-scale H^{Γ} path through `target = "metafounder_single_step"`; Gamma is supplied, not estimated. When data is an `hs_data()` container that bundles a pedigree and genotypes, `single_step(1 | id)` resolves both from the bundle (the `animal(1 | id)` precedent), so neither `pedigree =` nor `markers =` is required; explicit arguments override the bundle. The remaining markers (`markers()`, `marker_scan()`, `qtl_scan()`) are still inert syntax reservations that the parser rejects with a planned-not-implemented message.

Value

NULL, invisibly. Calls are interpreted by `hsquared()` when they appear inside model formulas.

gwas	<i>Post-fit relatedness-corrected marker scan (GWAS)</i>
------	--

Description

[Experimental]

Usage

```
gwas(  
  object,  
  markers,  
  marker_ids = NULL,  
  method = c("mixed", "single", "loco"),  
  marker_groups = NULL,  
  genome_wide = FALSE,
```

```

    n_permutations = 1000L,
    seed = 1L,
    ...
)

```

Arguments

object	A fitted Gaussian animal model (<code>hsquared_fit</code> from the default pedigree path); its variance components and pedigree relationship are reused so the scan is conditioned on the same covariance the model was fit under.
markers	A numeric matrix of marker dosages with one row per animal in the fit's pedigree (in pedigree order) and one column per marker.
marker_ids	Optional marker names; defaults to the <code>markers</code> column names, then to sequential ids.
method	"mixed" (default) for the relatedness-corrected mixed-model (GLS) scan with one whole-pedigree relationship correction across all markers; "single" for the relatedness- uncorrected single-marker (OLS) scan (a naive screen useful mainly as a contrast – it is more inflated by relatedness than the mixed scan); or "loco" for a leave-one-group-out scan with a per-group genomic relationship correction (requires <code>marker_groups</code>). The LOCO scan reuses the pedigree fit's variance components while correcting with a genomic relationship (a scale mismatch), so it is validation-scale and uncalibrated like the others.
marker_groups	Required for <code>method = "loco"</code> (and only then): a vector with one group label per marker column (for example a chromosome label). Markers in a group are tested with a genomic relationship built from all other groups. Needs at least two distinct, non-missing labels.
genome_wide	Logical; if TRUE (requires <code>method = "single"</code>), add a genome-wide-calibrated <code>genome_wide_p</code> column via the exact per-dataset add-one permutation rule (see Details). Defaults to FALSE (nominal p-values only).
n_permutations	Number of permutations for the genome-wide null when <code>genome_wide = TRUE</code> (default 1000). The add-one floor is $1/(n_permutations+1)$.
seed	Integer RNG seed for the genome-wide permutation null (default 1), so the <code>genome_wide_p</code> column is reproducible for a given Julia version.
...	Unused.

Details

`gwas()` runs a dense, supplied-variance, relatedness-corrected mixed-model (GLS) Wald marker scan on a fitted Gaussian animal model, reusing the fit's estimated variance components (σ^2_a , σ^2_e) and pedigree relationship. It is an experimental, validation-scale screen that surfaces the Julia-owned `HSquared.mixed_model_marker_scan()`.

The default `p_value/bonferroni_p/bh_qvalue` columns are NOT genome-wide calibrated. They are marker-by-marker Wald (nominal) p-values plus deterministic Bonferroni and Benjamini-Hochberg adjustments over the *supplied* marker set only. Do not report genome-wide significance from those columns.

Genome-wide calibration is available via `genome_wide = TRUE` (which requires `method = "single"`, the validated fixed-effect scope). It adds a `genome_wide_p` column: the exact per-dataset add-one permutation p-value (the marker is declared genome-wide significant when `genome_wide_p <= alpha`, `alpha = 0.05`). For each analysis the permutation null is rebuilt from the analysed phenotype (y permuted conditional on X , re-scanned `n_permutations` times), and the genome-wide p is $(1 + \#\{\text{null max} \geq \text{observed}\}) / (n_permutations + 1)$. This is the `HSquared.jl` `genome_wide_marker_scan` engine call, whose family-wise type-I control is validated at validation and production scale (the per-dataset add-one REBUILD gate; the anti-conservative $(1-\alpha)$ quantile rule is NOT used). The result carries a calibration attribute (`method permutation_addone`, `empirical_type1 = NA` because the per-dataset rule's validity is by construction and externally validated, named in `validation_reference`). SCOPE: fixed-effect / intercept-only; the relatedness-corrected mixed-model genome-wide null is NOT yet calibrated, so `genome_wide = TRUE` is rejected for `method = "mixed"/"loco"`.

A leave-one-group-out (LOCO) scan is available with `method = "loco"` and a `marker_groups` argument: when a marker is tested, the genomic relationship correction is built from the markers **not** in that marker's group (e.g. its chromosome), so the marker's own signal does not leak into the background relationship. The LOCO relationship is **genomic** (VanRaden) while the reused variance components are **pedigree**-estimated – a scale mismatch that keeps this validation-scale (see `method`).

Value

An `hs_gwas` data frame with one row per marker: `marker`, `effect`, `se`, `z`, `chisq`, `p_value`, `bonferroni_p`, `bh_qvalue`, `lod` (and, when `genome_wide = TRUE`, `genome_wide_p`), carrying a `scan_method` attribute (and, when `genome_wide = TRUE`, a calibration attribute). Its `print()` method restates the uncalibrated-significance caveat for the nominal columns (and, for `method = "single"`, the absence of any relatedness correction; for `method = "loco"`, the genomic-vs-pedigree scale mismatch).

`g_matrix_geometry`

G-matrix geometry and evolvability

Description

[Experimental]

Usage

`evolvability(object, direction, ...)`

`responsability(object, direction, ...)`

`conditional_evolvability(object, direction, ...)`

`autonomy(object, direction, ...)`

`mean_evolvability(object, ...)`

```

g_max(object, ...)

variance_along_gradient(object, direction, normalize = TRUE, ...)

eigen_G(object, ...)

```

Arguments

object	An <code>hsquared_fit</code> from the multivariate model.
direction	A numeric vector of selection gradients, one per trait. It is normalised to unit length internally; only its direction matters.
...	Unused.
normalize	For <code>variance_along_gradient()</code> , whether to scale direction to unit length first (TRUE, the default, which then equals <code>evolvability()</code>); FALSE uses direction as given, returning the genetic variance in the raw (un-normalised) selection gradient $\beta'G\beta$.

Details

These extractors summarise the genetic variance-covariance matrix G of an multivariate `hsquared_fit` (a `cbind()` response, which fits on the default path) through its **rotation-invariant** geometry, following Hansen & Houle (2008). They are defined on G itself (not on factor loadings), so they are well defined for any multivariate fit – unstructured, diagonal, or (when bridged) low-rank / factor-analytic – and do not depend on a loading rotation convention.

- `eigen_G()` returns the genetic eigenstructure: values (the variance along each genetic principal axis, descending) and vectors (the genetic principal components, sign-canonicalised so the largest-magnitude element of each is positive).
- `g_max()` returns the leading genetic axis: its eigenvalue and eigenvector (the direction of maximum evolvability).
- `mean_evolvability()` is the average evolvability over random selection directions, $\text{tr}(G) / t$.
- `evolvability(fit, direction)` is $e(\beta) = \beta'G\beta$ (unit β): the additive genetic variance available to directional selection along direction.
- `variance_along_gradient(fit, direction, normalize)` is $\beta'G\beta$ for a unit direction (`normalize = TRUE`, equal to `evolvability()`) or for the raw direction (`normalize = FALSE`).
- `responsability(fit, direction)` is $\|G\beta\|$: the length of the response.
- `conditional_evolvability(fit, direction)` is $1 / (\beta'G^{-1}\beta)$: evolvability when all other directions are held under stabilising selection (requires a positive-definite G).
- `autonomy(fit, direction)` is `conditional_evolvability / evolvability` in $[0, 1]$ (requires a positive-definite G).

These are computed in R from the fitted `genetic_covariance(fit)` and match the engine's `evolvability.jl` definitions (verified by a skip-guarded live parity test). They are descriptive geometry of the estimated G ; they carry the same experimental, REML-only, not-coverage-calibrated status as the multivariate fit itself and report no standard errors.

Value

A scalar for the directional metrics; a list for `eigen_G()` (values, vectors) and `g_max()` (eigenvalue, eigenvector).

References

Hansen, T. F., & Houle, D. (2008). Measuring and comparing evolvability and constraint in multi-variate characters. *Journal of Evolutionary Biology*, 21(5), 1201-1219.

heritability	<i>Extract heritability estimates</i>
--------------	---------------------------------------

Description

[Experimental]

Usage

```
heritability(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

`heritability()` is part of the planned v0.1 fitted-object contract. It works for `hsquared_fit` objects that contain a Julia result.

Falconer fence for the opt-in two-effect model (`target = "two_effect"`): the reported number is the **narrow-sense direct heritability** $h^2 = \sigma_{a^2} / (\sigma_{a^2} + \sigma_2 + \sigma_{e^2})$ *within that model* (the additive-genetic variance is divided by the total phenotypic variance, which now includes the second component σ_2). The second component's variance ratio (common-environment c^2 or maternal m^2) is **not** a heritability and is returned separately by `common_env_proportion()` / `maternal_proportion()`.

Willham fence for the opt-in direct-maternal correlated model (`target = "direct_maternal"`): `heritability()` returns the **labelled triple** – direct h^2_d , maternal m^2 , Willham total h^2_T , and r_{am} – as a data frame (Willham 1963, 1972). $\sigma^2_P = \sigma_{ad} + \sigma_{am} + \sigma_{dm} + \sigma_{e^2} = \text{Var}(y_i)$ (coefficient 1 on σ_{dm}). A warning is issued because h^2 is denominator-dependent under maternal effects and $h^2_T < h^2_d$ is expected when $r_{am} < 0$. Use `direct_heritability()` or `total_heritability()` for targeted accessors without the warning.

Reaction-norm fence for the opt-in random-regression model (`target = "random_regression"`): `heritability` is a **trajectory** $h^2(t)$ over the covariate, not a scalar, so `heritability()` **errors** on such a fit and names the implemented accessor. Use `rr_heritability()` for the $h^2(t)$ curve.

A non-converged fit still returns the engine number so you can inspect it, but **warns**: that number is not an estimate. A near-zero value from a failed fit is not evidence that heritability is zero. Use

`fit_diagnostics()` before reading any number. `logLik()` already refuses a non-converged fit; `heritability()` keeps the value and shouts instead.

Value

Heritability results for `hsquared_fit` objects.

`heritability_interval` *Extract an experimental heritability confidence interval*

Description

[Experimental]

Usage

```
heritability_interval(object, ...)
```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`heritability_interval()` returns an **experimental** large-sample confidence interval for h^2 . It is available only when an `hsquared_fit` object contains the interval field, which the default Gaussian animal-model fit (`engine = "fit"`) populates from the engine's `HSquared.heritability_interval()` when a local Julia engine is present and the estimate is interior to $(0, 1)$. On the opt-in two-effect fit it returns the direct-heritability ratio interval (`ratio1`), and on the opt-in multi-effect fit (`target = "multi_effect"`, $K \geq 3$ blocks) it returns the ANIMAL block's ratio interval (the animal additive variance over the total phenotypic variance); the other blocks' variance-ratio intervals are surfaced separately in `fit$result$variance_ratio_intervals`.

The interval leg is a REML-only, asymptotic (logit delta-method or profile) approximation returned by the engine. The 2000-rep C1 coverage confirm (HSquared.jl DRAC job **47925485**) places the univariate pedigree h^2 interval at the **directional-conservative** claim level under doc-34 §4: delta over-covers (worst $\hat{C}$ 0.969), profile is in-band (worst $\hat{C}$ 0.950), and bootstrap **mildly under-covers versus nominal 0.95** at the governing interior cell (worst $\hat{C}$ **0.924** at $h^2=0.5$, still at least 0.90). Those figures were measured on one **interpretable small design** ($q = 120$), at **interior** h^2 in $\{0.3, 0.5, 0.7\}$, at the **0.95** level, in DRAC job **47925485; unbalanced designs were not in the confirm grid**, so these numbers say nothing about coverage under imbalance. That is **not** coverage-calibrated at nominal and **not** "never under-covers." Profile and bootstrap h^2 interval legs were measured in the same confirm but are **not** separately surfaced by this extractor; only the engine-returned interval is shown. It is reported as a point estimate plus bounds, not a validated (coverage-calibrated) capability, and remains unreliable near the $h^2 \rightarrow 0$ boundary. Genomic fits still error (scale-labelled interval not validated). The underlying estimators V3-TWOEFFECT-REML / V3-NEFFECT-REML are covered, but this **interval** is not.

Value

A one-row data frame with estimate, lower, upper, level, se (NA for the profile method), and method, for `hsquared_fit` objects that contain it.

hsquared	<i>Fit a quantitative-genetic model</i>
----------	---

Description

The default path fits a univariate Gaussian animal model by REML: $y \sim \text{sex} + \text{age} + \text{animal}(1 \mid \text{id}, \text{pedigree} = \text{ped})$. Fitting needs a local Julia and HSquared.jl. Preview the same call with `control = hs_control(engine = "validate")`.

Usage

```
hsquared(
  formula,
  data,
  family = stats::gaussian(),
  REML = TRUE,
  control = hs_control(),
  ...
)
```

Arguments

formula	A model formula. The default path is $y \sim \text{fixed} + \text{animal}(1 \mid \text{id}, \text{pedigree} = \text{ped})$. <code>animal(1 id)</code> is also accepted when data is an <code>hs_data()</code> object with a pedigree.
data	A data frame of model variables, or an <code>hs_data()</code> object whose phenotypes component holds them. Formula arguments such as <code>pedigree = pedigree</code> can name components of the bundle.
family	A response family. The default path accepts <code>gaussian()</code> . Opt-in experimental paths accept <code>poisson()</code> and <code>binomial()</code> ; see <code>formula_status()</code> .
REML	Logical; REML estimation. The default path supports REML only (TRUE). REML = FALSE (ML) is rejected.
control	An object created by <code>hs_control()</code> .
...	Reserved for future arguments.

Details

What you may report is listed on [Can I fit and report this?](#) – not in `validation_status()`. Other routes live in `formula_status()` and on that limits page.

Value

A "hsquared_fit" object from a successful default-path fit. Missing Julia produces an install-guidance error. When engine = "validate", the validated model spec is returned invisibly as a named list (not the classed object from `model_spec()`).

hsquared-autoplot	<i>ggplot2 visualizations for hsquared results</i>
-------------------	--

Description

[Experimental]

Usage

```
## S3 method for class 'hsquared_fit'
autoplot(
  object,
  type = c("variance", "breeding_values", "g_matrix", "g_geometry", "reaction_norm",
    "rr_eigenfunctions", "rr_surface"),
  ...
)

## S3 method for class 'hs_gwas'
autoplot(object, type = c("manhattan", "qq"), ...)
```

Arguments

object	An hsquared_fit or hs_gwas object.
type	Which figure to draw (see Details).
...	Figure-specific options passed through: low_h2 (the genetic-correlation heatmap flags off-diagonal cells involving a trait with $h^2 < \text{low_h2}$ as imprecise; default 0.1), at/n (the reaction-norm covariate evaluation points), and correlation (rr_surface: draw the genetic-correlation surface instead of the covariance surface).

Details

autoplot() methods render the quantitative-genetic results an hsquared_fit (or a gwas() scan) carries as ggplot2 objects, in the style of the brms/bayesplot ecosystem and consistent with the sister packages drmTMB/gllvmTMB. They are **uncertainty-first**: where the fit carries the experimental standard errors / reliabilities, the figures show them (clearly labelled experimental and asymptotic).

Available types for autoplot.hsquared_fit():

- "variance" (default) – a horizontal forest of the variance components and per-trait h^2 , each with approximate 95% intervals ($\pm 1.96 * SE$) when the fit carries the experimental standard errors.

- "breeding_values" – a sorted caterpillar of the estimated breeding values, with $\pm 1.96 * \sqrt{\text{PEV}}$ bands when prediction error variances are available, faceted by trait for multivariate fits.
- "g_matrix" – a **rotation-invariant** genetic-correlation heatmap of the estimated G for multivariate fits (correlations are invariant to the factor rotation; raw loadings are never plotted – the ratified cross-lane convention). Off-diagonal cells involving a low- h^2 trait are flagged as imprecise (threshold low_h2, default 0.1).
- "g_geometry" – a scree of the **rotation-invariant** genetic eigenstructure (eigenvalues = variance per genetic axis, with percent variance explained) for multivariate fits. Axis directions / loadings are never drawn (rotation-arbitrary; span-ambiguous under repeated eigenvalues).
- "reaction_norm" – for random-regression fits, the genetic-variance and heritability trajectories across the covariate (faceted). The heritability trajectory carries the same caveat as [rr_heritability\(\)](#): with a homogeneous residual and no permanent-environment term it can overstate $h^2(t)$ for repeated-records designs.
- "rr_eigenfunctions" – for random-regression fits, the rotation-invariant eigenfunctions $\psi_j(t)$ of K_g as covariate functions (faceted by axis, labelled by percent genetic variance). Signs are arbitrary and the curves are span-ambiguous under repeated eigenvalues (do not over-read).
- "rr_surface" – for random-regression fits, the genetic covariance surface $S(s, t) = \phi(s)' K_g \phi(t)$ over the covariate grid as a heatmap (pass correlation = TRUE for the genetic-correlation surface). Supplied- K_g descriptive.

autoplot() on a gwas() scan (hs_gwas) draws type = "manhattan" (default) or type = "qq" (observed vs expected $-\log_{10}(p)$ with a $y = x$ null and the genomic-inflation λ_{GC} as a diagnostic). Both carry the EXPERIMENTAL, NOT-genome-wide-calibrated caveat. The Julia lane has a fixed-panel calibration smoke harness, but R significance thresholds are not activated.

The figure helpers are deliberately modular (each takes a tidy data frame and returns a ggplot) so they can be factored into a shared visualization package later.

Value

A ggplot object.

hs_control

Create hsquared control options

Description

hs_control() records execution and storage controls for hsquared model calls. The default engine = "fit" fits the validated v0.1 Gaussian animal model through the HSquared.jl engine (average-information REML); engine = "validate" parses and validates the contract without fitting, returning the validated spec invisibly; and engine = "julia" exposes advanced engine controls.

Usage

```

hs_control(
  engine = c("fit", "validate", "julia"),
  backend = c("auto", "cpu", "threads", "cuda", "amdgpu", "metal", "oneapi"),
  accelerator = c("auto", "none", "gpu", "cuda", "amdgpu", "metal", "oneapi"),
  precision = c("float64", "float32"),
  save = c("minimal", "full", "tiny"),
  engine_control = list()
)

```

Arguments

engine	Execution engine. "fit" (default) fits the v0.1 Gaussian animal model via the HSquared.jl engine; this requires a local Julia, the JuliaCall package, and an HSquared.jl checkout. "validate" validates the parser and bridge payload without fitting, then returns the validated spec invisibly (after a confirming message). "julia" exposes the advanced opt-in bridge with explicit target control.
backend	Planned compute backend. One of "auto", "cpu", "threads", "cuda", "amdgpu", "metal", or "oneapi".
accelerator	Planned accelerator preference. One of "auto", "none", "gpu", "cuda", "amdgpu", "metal", or "oneapi".
precision	Planned numeric precision. One of "float64" or "float32".
save	Planned fitted-object storage mode. One of "minimal", "full", or "tiny".
engine_control	A named list for engine-specific controls. The current experimental Julia bridge recognizes <code>julia_project</code>, <code>initial</code>, <code>iterations</code>, <code>em_warmup</code>, <code>target</code>, <code>variance_components</code>, <code>marginal</code>, and <code>max_dense_cells</code>. <code>julia_project</code> is honoured by every target; supplying a key a given target does not honour errors (hsquared#212) rather than being silently ignored. Per-target honoured keys (besides <code>julia_project</code>): • <code>fit_animal_model</code> (default): <code>initial</code>, <code>max_dense_cells</code>. • <code>henderson_mme</code>, <code>metafounder</code>, <code>snp_blup</code>: <code>variance_components</code>. • <code>ai_reml</code>: <code>initial</code>, <code>iterations</code>, <code>em_warmup</code>. • <code>sparse_reml</code>, <code>two_effect</code>, <code>direct_maternal</code>, <code>genomic</code>, <code>single_step</code>, <code>single_step_construct</code>, <code>metafounder_single_step</code>, <code>relmat</code>, <code>precision</code>: <code>initial</code>, <code>iterations</code>. • <code>repeatability</code>: <code>initial</code>, <code>iterations</code>, <code>max_dense_cells</code>, <code>scale_method</code>. • <code>multi_effect</code>: <code>initial</code>, <code>iterations</code>, <code>scale_method</code> – <code>initial/iterations</code> are honoured on the <code>scale_method = "dense"</code> (default) route only; the opt-in <code>scale_method = "auto"</code> route does not yet forward them (HSquared.jl#343, a known remaining gap). • <code>multivariate</code>: <code>initial</code>, <code>iterations</code>, <code>genetic_structure</code>, <code>rank</code>. • <code>random_regression</code>: <code>iterations</code> (no <code>initial</code>). • <code>nongaussian</code>: <code>marginal</code>, <code>iterations</code>, <code>initial</code> (a list with <code>sigma_a2</code>), <code>restart_check</code>. <code>max_dense_cells</code> bounds $\text{nobs}^2 + \text{nanimals}^2$ on the engine's dense-validation fitters (hsquared#214, #217): the default Julia target <code>target = "fit_animal_model"</code> (via <code>HSquared.fit_animal_model()</code>)

/ fit_variance_components()) and target = "repeatability". Both require engine = "julia". It has **no effect** under the default engine = "fit" path, which routes to the sparse-capable HSquared.fit_ai_reml() and enforces no dense-cell cap at all. It must be a single positive integer; the default, 1e6, mirrors the engine's own DEFAULT_MAX_DENSE_CELLS unchanged. Raise it to fit a larger dense problem at the cost of memory and time, or switch to a sparse route (target = "ai_reml"/"sparse_reml") instead of raising it indefinitely. Exceeding the cap now raises an hsquared_error naming the observed cell count and the effective cap, rather than a raw Julia trace. On target = "repeatability", max_dense_cells reaches the point fit only. The opt-in repeatability-coefficient interval is computed by a separate engine entry point that exposes no such control and runs at the engine default (1e6), so **raising the cap above 1e6 returns the point fit with the interval silently absent** (NULL) rather than with an error. Tracked on the engine side; not fixed here. This applies to scale_method = "dense" only: the sparse route forms its interval from the fitted components and never reaches that entry point, so it is unaffected by max_dense_cells.

target selects which Julia estimator the engine = "julia" bridge runs; it has no effect under the default engine = "fit" path. The supported targets are "fit_animal_model", "ai_reml", "sparse_reml", "henderson_mme", "repeatability", "two_effect", "multi_effect", "direct_maternal", "random_regression", "genomic", "single_step", "single_step_construct", "metafounder", "metafounder_single_step", "snp_blup", "relmat", "precision", "multivariate", and "nongaussian", described below. Covered opt-in routes (validation scale; not the default path) include "two_effect" (common_env()), "direct_maternal", "multi_effect", and "random_regression" at k = 2. marginal applies only to target = "nongaussian". With engine = "julia" and no target, the bridge defaults to target = "fit_animal_model": it surfaces the Julia-owned HSquared.fit_animal_model() dense NelderMead optimizer, honouring the REML flag. This is **not** the same estimator as the default engine = "fit" path, which runs the validated average-information REML estimator (HSquared.fit_ai_reml()), the same one reached by target = "ai_reml". The two paths target the same Gaussian animal model but differ at the optimizer level, so their estimates can differ slightly. Reach the validated estimator with engine = "fit" (the ordinary path) or with engine = "julia" and target = "ai_reml". target = "henderson_mme" is a supplied-variance validation path and requires variance_components with named sigma_a2 and sigma_e2 values. target = "metafounder" is an experimental supplied-variance validation path for metafounder(1 | id, pedigree = ped, group = mf_group, Gamma = Gamma). It builds the Julia-owned animal-only A^Gamma relationship; Gamma and the variance components are supplied, not estimated. target = "sparse_reml" is an experimental, opt-in validation path that surfaces the Julia-owned HSquared.fit_sparse_reml() REML-only sparse optimizer; it accepts initial (named sigma_a2/sigma_e2) and iterations. It is not the default, not production fitting, and not a variance-component estimation claim for the public R interface. target = "ai_reml" exposes the same average-information REML estimator (HSquared.fit_ai_reml()) that the default engine = "fit" path uses, with explicit initial and iterations control. This is the validated v0.1 estimator for the univariate Gaussian animal model; the engine = "fit" default is the ordinary way to reach it. em_warmup

(a non-negative integer, default 0) optionally runs that many EM-REML warm-start steps before the average-information iterations; it applies to this estimator on both the default engine = "fit" path and target = "ai_reml". em_warmup = 0 (the default) is byte-identical to the previous behaviour. A small positive value (e.g. 3) can rescue convergence from poor starting variances; on an identified fit it converges to the same optimum. target = "repeatability" is an experimental, opt-in path for the repeatability (permanent-environment) model. It requires animal(1 | id, pedigree = ped) + permanent(1 | id) in the formula and surfaces the Julia-owned HSquared.fit_repeatability_reml() REML-only optimizer (three-component initial with sigma_a2/sigma_pe2/sigma_e2). It is REML only, not the default, and the additive and permanent-environment variances are identifiable only with repeated records per individual. scale_method selects the estimator. "dense" (the default) is the covered validation-scale fit_repeatability_reml() above, which the engine refuses once nobs^2 + nanimals^2 exceeds max_dense_cells – a ceiling any field-scale repeated-measures pedigree passes easily. scale_method = "auto" fits the SAME model through the engine's sparse K-effect AI-REML (fit_multi_effect(method = :auto)), expressing it as the animal block plus an identity-relationship permanent-environment block, and so has no dense ceiling. Below the ceiling the two agree to REML tolerance. Two differences are deliberate on the sparse route: initial and iterations are NOT honoured (the engine picks its own; a supplied value warns), and loglik carries the REML normalising constant $-(n - p)/2 * \log(2 * \pi)$ that the dense route omits – so log-likelihoods, AIC, or likelihood-ratio tests must NOT be compared across scale_method on the raw loglik (HSquared.jl #365). Fit diagnostics expose loglik_convention, loglik_full_constant_offset (add to loglik for the full-constant scale), and loglik_comparable_across_routes when the linked engine is new enough; the engine helper comparable_loglik(fit) does the same conversion. repeatability_interval() IS available on both routes and agrees between them; the sparse one is formed from the fitted components (multi_effect_sum_ratio_interval()) rather than by refitting densely, and both are asymptotic logit-scale delta intervals that are NOT coverage-calibrated. target = "two_effect" is an experimental, opt-in path for two-effect models. It requires animal(1 | id, pedigree = ped) plus a second random effect – common_env(1 | group) (an IID common-environment effect) or maternal_genetic(1 | dam) (a maternal genetic effect carrying the pedigree relationship) – and surfaces the Julia-owned HSquared.fit_two_effect_reml() REML-only optimizer (three-component initial with sigma_a2/sigma_c2/sigma_e2). It is REML only and not the default. The common_env() leg is covered at validation scale; the maternal_genetic() leg on this same target stays experimental. The covered correlated sibling is target = "direct_maternal". target = "multi_effect" is an opt-in path for animal(1 | id, pedigree = ped) plus one or more bare (1 | group) IID intercepts. It is covered at validation scale (the independent generalization of two-effect; not the default path). Random slopes and correlated (x || group) terms remain rejected. The animal-block ratio is narrow-sense h2; other blocks are variance-explained proportions, not heritabilities. initial is a plain numeric vector of length K + 1 (one value per block, in formula order, plus the residual); it and iterations are honoured on the scale_method = "dense" route (see engine_control above for

the "auto" gap). `target = "direct_maternal"` is an opt-in path for `animal(1 | id, pedigree = ped) + maternal_genetic(1 | dam)`. It estimates the correlated 2x2 direct-maternal genetic covariance and is covered at validation scale (not the default path). `heritability()` returns the labelled Willham triple (direct h^2_d , maternal m^2 , total h^2_T , r_{am}), never a bare scalar. `initial` is a list with G_{dm} (a 2x2 matrix) and σ_{e2} (a positive scalar); it and iterations are honoured. `target = "random_regression"` is an opt-in reaction-norm path for `animal(rr(covariate, order = k) | id, pedigree = ped)`. It is covered at $k = 2$ (linear reaction norm). `rr_heritability()` returns $h^2(t)$ as a curve; `heritability()` errors on this result and names `rr_heritability()`. $k \geq 3$ stays experimental. `target = "genomic"` is the explicit experimental narrow Gaussian REML genomic route. It accepts `genomic(1 | id, markers = M)`, using sample allele frequencies, unweighted VanRaden method 1, and ridge 0.01, or `genomic(1 | id, Ginv = Ginv)`, whose construction method, allele-frequency source, ridge, and scale denominator remain unknown. `heritability()` returns the genomic variance-component ratio on the declared relationship scale, $\sigma_{g2} / (\sigma_{g2} + \sigma_{e2})$. Because relationship scaling enters through the kernel, this is not generally the fraction of average marginal phenotypic variance and is not pedigree-, founder-base-, population-, or universal narrow-sense heritability. `Genomic_heritability_interval()` and `heritability_standard_error()` are unavailable. The R capability remains covered at validation scale, experimental, and opt-in; the explicit route does not move `public_covered_count` from 7. A fail-closed boundary candidate matched an independent oracle on all 240 sealed holdouts, correcting 30 classifications with no losses, but one cell had a 5.99x p95 runtime ratio against the frozen 3x cap. The seeds are spent, the nine-cell campaign did not run, and the route must remain opt-in pending performance work and a new untouched holdout. `target = "single_step"` remains an experimental, opt-in path for `single_step(1 | id, Hinv = Hinv)` (a supplied single-step relationship inverse). Both targets surface `HSquared.fit_ai_reml()` and are REML-only; single-step is not a default route. `target = "single_step_construct"` fits `single_step(1 | id, pedigree = ped, markers = M)` after the engine builds H^{-1} from the pedigree and genotyped-subset markers. `target = "metafounder_single_step"` fits `single_step(1 | id, pedigree = ped, markers = M, group = mf_group, Gamma = Gamma)` through the Julia-owned supplied-Gamma H^{Gamma} path. Both are experimental, opt-in, dense/validation-scale, REML-only, and not comparator-validated; Gamma is supplied, not estimated. Both honour `initial` (named σ_{a2}/σ_{e2}) and iterations. `target = "snp_blup"` is an experimental, opt-in path for the SNP-BLUP / RR-BLUP marker-effect model. It requires `genomic(1 | id, markers = M)` (a raw marker matrix) and estimates per-marker effects (`marker_effects()`) and per-individual genomic breeding values. `variance_components` is **optional**: supply named σ_{g2} (genomic) and σ_{e2} (residual) for a supplied-variance solve (`HSquared.fit_snp_blup()`), or omit them to have `hsquared()` **estimate** σ_{g2}/σ_{e2} by REML from the markers (`HSquared.fit_snp_blup_reml()`). Not the default. `target = "relmat"` is an experimental, opt-in path for `relmat(1 | id, K = K)` (a supplied dense relationship matrix; the parser marshals the inverse). `target = "precision"` is the same experimental path for `precision(1 | id, Q = Q)` (a supplied precision/inverse). Neither is covered or the default; the supplied ma-

trix is provenance, not an estimate. The `animal(1 | id, pedigree = ped)` route rejects selfing (rows with the same known sire and dam) in v0.1, with no argument that reaches the engine's `allow_selfing` flag; `relmat(1 | id, K = A)` with a hand-built or AGHmatrix-built relationship matrix `A` is the current workaround for a selfing or hermaphroditic pedigree. `target = "multivariate"` names the experimental multivariate Gaussian animal model. Naming it is optional: a `cbind()` Gaussian response with an `animal()` term auto-routes to this target on the default path, and under engine = "julia" with no target. The `t = 2` unstructured capability is covered at validation scale, while remaining experimental. It requires a `cbind(trait1, trait2, ...)` response with `animal(1 | id, pedigree = ped)`, surfaces the Julia-owned `HSquared.fit_multivariate_reml()` REML-only optimizer, and returns G/R covariance matrices, genetic and residual correlations, per-trait heritability, and cross-trait breeding values. Three or more traits and structured covariance are experimental. The reserved `genetic_structure` control currently accepts "unstructured" and "diagonal" on the R bridge. "diagonal" is the rotation-free structured subset: off-diagonal genetic covariances are fixed at zero. "lowrank" remains planned. "factor_analytic" is planned on the R surface (Julia V4-FA is engine-covered at `HSquared.jl` 60895208 / #300) and is not activated on the R bridge; not covered. The future rank control is also reserved and currently errors instead of being ignored.

`target = "nongaussian"` is an experimental, opt-in conditional GLMM for `poisson(log)` or `binomial(logit)` (binary 0/1 or `cbind(successes, failures)` counts) with one intercept and `animal(1 | id, pedigree = ped)`. The marginal control selects a Laplace marginal likelihood approximation ("laplace", default) or a variational ELBO ("variational"; aliases "la"/"va"). It reports the ratified conditional three-field contract: Poisson latent and count-scale observation `h2`; logit latent, liability, and numerically integrated observation-scale `h2` for Bernoulli or common-trial Binomial input. Varying trials return literal NaN with "varying_trials_no_scalar_estimand", never a trial-count-averaged scalar. The ELBO is a lower bound on the marginal log-likelihood, so variational and Laplace `logLik/AIC` are **not** comparable. This path remains experimental and not coverage-calibrated. `initial` (`hsquared#225`) is a list with `sigma_a2`. The engine fits the single variance component with a **bracketed** Brent search over `log(sigma_a2)` on `log(initial$sigma_a2) +/- 6` – there is no start value, and `initial` sets the **centre of the bracket**, so supplying it moves the whole search window. Unsupplied, the engine's own hard-coded `sigma_a2 = 1.0` centres it, giving `[exp(-6), exp(6)]`; a true `sigma_a2` outside that window cannot be reached without an `initial` on the scale of the data, which is what makes `initial` the retry lever for a boundary refusal. `restart_check` (`hsquared#225`, logical, default FALSE) opts into the engine's two-start restart (`HSquared.jl#327`): it refits once from a bumped second start and flags `boundary = TRUE` when the estimate moves with the start, catching a boundary a single fit can otherwise miss. The result carries `boundary` (`hsquared#222`) next to `converged`: TRUE means the fitted `sigma_a2` is a function of the search start, not the data. In practice a `boundary = TRUE` fit is refused before it reaches the R result at all – the Julia payload builder raises, translated into a classed `hsquared_julia_error` naming `initial/restart_check` as the retry levers – so `boundary` on a returned fit is FALSE. Read it as `fit$result$boundary`, or

via `fit_diagnostics()`'s `search_boundary / search_boundary_condition` rows (`hsquared#230`); those rows are unrelated to that function's `at_boundary` rows, which flag a variance component at or near zero.

Value

An object of class "hs_control".

hs_data	Create an hsquared data container
---------	-----------------------------------

Description

[Experimental]

Usage

```
hs_data(  
  phenotypes,  
  pedigree = NULL,  
  genotypes = NULL,  
  markers = NULL,  
  expression = NULL,  
  annotation = NULL,  
  annotation_id = NULL,  
  environment = NULL,  
  environment_id = NULL,  
  id = "id"  
)
```

Arguments

phenotypes	A data frame of phenotypic records.
pedigree	Optional pedigree data frame.
genotypes	Optional genotype matrix or data frame. Matrix row names or data-frame ID values identify individuals. When <code>markers</code> is supplied, genotype marker column names must match marker-map IDs exactly.
markers	Optional marker map data frame. When supplied, it must contain marker ID, chromosome, and position columns. Recognized aliases include <code>marker</code> , <code>snp</code> , or <code>id</code> ; <code>chromosome</code> , <code>chr</code> , or <code>chrom</code> ; and <code>position</code> , <code>pos</code> , <code>bp</code> , or <code>base_pair</code> .
expression	Optional expression matrix or data frame.
annotation	Optional annotation data frame.
annotation_id	Optional column name used to match annotation rows to expression feature columns. When supplied, the column must exist in annotation.
environment	Optional environment/covariate data frame.

`environment_id` Optional column name used to match environment rows to phenotype records. When supplied, the column must exist in both phenotypes and environment.

`id` Name of the individual ID column in phenotypes.

Details

`hs_data()` collects phenotype, pedigree, genotype, marker, expression, annotation, and environment inputs into one checked container. It is a lightweight data-contract object for future genomic, QTL/eQTL, and multi-omics workflows. It does not fit models. The v0.1 parser can use an `hs_data` object directly as data, reading model variables from phenotypes and making named components such as pedigree available to formula terms.

`summary(hs_data(...))` reports ID overlap diagnostics, pedigree diagnostics, and, when expression, genotype, or marker components are supplied, expression-feature, genotype-column, marker-map, and genotype-marker alignment diagnostics. When `annotation_id` is supplied, it reports expression-feature annotation coverage diagnostics. When `environment_id` is supplied, it also reports environment metadata coverage diagnostics.

Value

An `hs_data` object.

Examples

```
ped <- data.frame(
  id = c("a", "b", "c", "d"),
  sire = c(NA, NA, "a", "a"),
  dam = c(NA, NA, "b", "c")
)
dat <- data.frame(
  y = c(1, 2, 3),
  sex = c("f", "m", "f"),
  id = c("a", "c", "d")
)
bundle <- hs_data(phenotypes = dat, pedigree = ped)
bundle
summary(bundle)
```

hs_recovery_forest	<i>Forest plot of a known-truth recovery study</i>
--------------------	--

Description

[Experimental]

Usage

```
hs_recovery_forest(data)
```

Arguments

`data` A data frame with `target`, `bias`, and `mcse` columns.

Details

Visualizes a bias $\pm 2MCSE$ recovery study (e.g. `data-raw/multivariate-recovery-study.R`): each target's bias with its $\pm 2MCSE$ interval and a zero-bias reference line. Targets whose interval covers zero show "no detectable bias".

Value

A ggplot object.

<code>inference_blocks</code>	<i>Block unsupported likelihood-inference helpers</i>
-------------------------------	---

Description

These methods intentionally fail with explicit scope messages. Validated `confint()` / `vcov()` / `profile()` / `anova()` surfaces are not implemented. Experimental interval extractors ([heritability_interval\(\)](#), [variance_component_standard_errors\(\)](#)) exist when the engine returned those fields; they are not coverage-calibrated. Point estimates remain [variance_components\(\)](#), [heritability\(\)](#), and [fit_diagnostics\(\)](#).

Usage

```
## S3 method for class 'hsquared_fit'
confint(object, parm, level = 0.95, ...)

## S3 method for class 'hsquared_fit'
vcov(object, ...)

## S3 method for class 'hsquared_fit'
profile(fitted, ...)

## S3 method for class 'hsquared_fit'
anova(object, ...)
```

Arguments

`object` An `hsquared_fit` object.

`parm, level` Included for compatibility with [stats::confint\(\)](#).

`...` Reserved for future arguments.

`fitted` An `hsquared_fit` object for [stats::profile\(\)](#).

Value

These functions always error.

marker_extractors	<i>Extract planned marker, QTL, GWAS, and eQTL results</i>
-------------------	--

Description

[Experimental]

Usage

```
marker_effects(object, ...)

marker_variance_explained(object, ...)

qtl_table(object, ...)

gwas_table(object, ...)

eqtl_table(object, ...)

lod_scores(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

These extractor names cover genomic, QTL, GWAS, and eQTL fitted results. They return values only when an `hsquared_fit` object contains the corresponding result field. `marker_effects()` and `marker_variance_explained()` are populated by the opt-in SNP-BLUP path (`target = "snp_blup"`). The variance-explained table is a descriptive fitted-marker share, computed as effect squared times centered marker variance and normalized across markers; it is not a marker-scan p-value, QTL statistic, or causal decomposition under linkage disequilibrium. The `gwas_table()` and `lod_scores()` methods for an already-computed `hs_gwas` object expose the current uncalibrated marker-scan table and marker-level LOD scores. Fit-level QTL/GWAS/eQTL result tables and map-annotated / calibrated scan outputs remain reserved for future results.

Value

The requested marker or scan result for `hsquared_fit` objects that contain the corresponding field.

maternal_effects	<i>Extract maternal genetic effects</i>
------------------	---

Description

[Experimental]

Usage

maternal_effects(object, ...)

Arguments

- object A fitted model object.
- ... Reserved for future arguments.

Details

maternal_effects() returns the predicted maternal genetic effects of the opt-in, experimental maternal two-effect model (target = "two_effect" with a maternal_genetic() term).

Value

Maternal genetic effect results for maternal hsquared_fit objects.

maternal_proportion	<i>Extract the maternal variance ratio (m2)</i>
---------------------	---

Description

[Experimental]

Usage

maternal_proportion(object, ...)

Arguments

- object A fitted model object.
- ... Reserved for future arguments.

Details

`maternal_proportion()` returns the estimated maternal variance ratio $m2 = \sigma_{m2} / (\sigma_{a2} + \sigma_{m2} + \sigma_{e2})$ from the opt-in, experimental maternal two-effect model (`target = "two_effect"` with a `maternal_genetic()` term).

Falconer fence: $m2$ is a **variance ratio** (the proportion of phenotypic variance from the maternal (dam) genetic effect), **not a heritability**. In the same fit, `heritability()` reports the narrow-sense *direct* $h2 = \sigma_{a2} / (\sigma_{a2} + \sigma_{m2} + \sigma_{e2})$ *within this two-effect model*; the direct and maternal genetic effects are modelled as uncorrelated here (no direct-maternal genetic covariance). The returned data frame carries this note as an "interpretation" attribute. For an interval on $m2$, see `maternal_proportion_interval()`.

Value

A one-row data frame with term and estimate, plus an "interpretation" attribute, for maternal two-effect `hsquared_fit` objects.

<code>metafounder_effects</code>	<i>Reserved metafounder effect extractor</i>
----------------------------------	--

Description

[Experimental]

Usage

```
metafounder_effects(object, ...)
```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`metafounder_effects()` reserves the extractor name for future fitted metafounder solutions. Current experimental metafounder and H^{Gamma} fits expose supplied Gamma and group-assignment provenance via `gamma_matrix()` and `metafounder_groups()`, but the engine does not yet return explicit combined-system metafounder effects for extraction.

Value

This extractor currently errors for all objects.

metafounder_groups	<i>Extract supplied metafounder group assignments</i>
--------------------	---

Description

[Experimental]

Usage

```
metafounder_groups(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

metafounder_groups() returns the ID-keyed metafounder group assignments carried by an experimental metafounder or H^{Gamma} single-step hsquared_fit object. These assignments are provenance for the supplied relationship, not estimated grouping parameters.

Value

A data frame with columns id, metafounder_group, and is_metafounder. Rows are **animals**, keyed by id; is_metafounder flags animals whose unknown-parent slot carries a supplied group label, not metafounders themselves. The metafounders are the pseudo-populations that are the rows and columns of [gamma_matrix\(\)](#).

model_spec	<i>Inspect a parsed hsquared model specification</i>
------------	--

Description

[Experimental]

Usage

```
model_spec(formula, data, family = stats::gaussian(), REML = TRUE, ...)
```

Arguments

formula	A model formula. The current implemented grammar is $y \sim \text{fixed} + \text{animal}(1 \mid \text{id}, \text{pedigree} = \text{ped})$, or $y \sim \text{fixed} + \text{animal}(1 \mid \text{id})$ when data is an <code>hs_data()</code> object with a pedigree component.
data	A data frame containing model variables, or an <code>hs_data()</code> object whose phenotypes component contains the model variables. When data is an <code>hs_data</code> object, formula arguments such as <code>pedigree = pedigree</code> can refer to named components in the bundle, and <code>animal(1 id)</code> uses the bundle pedigree by default.
family	A response family. The current parser accepts only <code>gaussian()</code> .
REML	Logical; whether the target Gaussian animal model would use REML.
...	Reserved for future arguments.

Details

`model_spec()` validates the narrow v0.1 animal-model grammar and returns the R-side model specification that would be used to build the Julia bridge payload. It is a preview and diagnostics helper: it does not fit a model.

Value

An "hs_model_spec" object.

`multivariate_extractors`

Extract multivariate covariance and correlation matrices

Description

[Experimental]

Usage

`genetic_covariance(object, ...)`

`G_matrix(object, ...)`

`residual_covariance(object, ...)`

`R_matrix(object, ...)`

`genetic_correlation(object, ...)`

`residual_correlation(object, ...)`

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

These extractors return the genetic (G) and residual (R) covariance or correlation matrices from multivariate `hsquared_fit` objects (a `cbind()` response, which fits on the default path). `G_matrix()` is an applied-workflow alias for `genetic_covariance()`, and `R_matrix()` is an alias for `residual_covariance()`. Use them after checking `fit_diagnostics()` because likelihood-based summaries are intentionally blocked when a multivariate fit has not converged.

Value

A numeric matrix for `hsquared_fit` objects that contain the requested multivariate result field.

Examples

```
# Synthetic result object: demonstrate extraction, not model fitting.
# No Julia installation is needed for this example.
G <- matrix(c(0.4, 0.1, 0.1, 0.3), 2,
            dimnames = list(c("weight", "length"), c("weight", "length")))
R <- diag(c(0.6, 0.7))
dimnames(R) <- dimnames(G)
fit_mv <- structure(list(
  spec = list(target = "multivariate"),
  result = list(genetic_covariance = G, residual_covariance = R,
               genetic_correlation = cov2cor(G),
               residual_correlation = cov2cor(R),
               heritability = data.frame(trait = colnames(G),
                                         estimate = diag(G) / diag(G + R)),
  converged = TRUE)
), class = "hsquared_fit")

genetic_covariance(fit_mv)
G_matrix(fit_mv)
residual_covariance(fit_mv)
R_matrix(fit_mv)
genetic_correlation(fit_mv)
residual_correlation(fit_mv)
heritability(fit_mv)
```

permanent_effects

Extract permanent-environment effects

Description

[Experimental]

Usage

```
permanent_effects(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

permanent_effects() returns the predicted permanent-environment effects of the opt-in, experimental repeatability model.

Separating sigma_a2 from sigma_pe2 needs repeated records per individual; see ?hs_control(target = "repeatability") for that identifiability requirement.

Value

Permanent-environment effect results for repeatability hsquared_fit objects.

plot.hsquared_fit	<i>Diagnostic plots for a fitted animal model</i>
-------------------	---

Description

A base-graphics diagnostic plot for hsquared_fit objects. Two panels are available:

Usage

```
## S3 method for class 'hsquared_fit'
plot(x, type = c("variance", "residuals"), ...)
```

Arguments

x	An hsquared_fit object.
type	Which panel to draw: "variance" or "residuals".
...	Passed to the underlying base-graphics call.

Details

- type = "variance" (default) plots the estimated variance components as points. When the fit carries the **experimental** variance-component standard errors (see [variance_component_standard_errors\(\)](#)), it adds approximate $\pm 1.96 \times \text{SE}$ whiskers and labels the panel experimental; those intervals are asymptotic, REML-only, and not coverage-calibrated. The governing coverage study found under-coverage for the σ^2_a delta/Wald interval (worst interior coverage 0.897 at $h^2 = 0.5$), so no conservative direction is claimed for the plotted variance-component SE.
- type = "residuals" plots residuals against fitted values (with a zero reference line), when the fit carries fitted values and a response.

No plotting dependency is added; this uses base graphics only.

Value

x, invisibly.

prediction_error_variance
<i>Extract prediction error variances</i>

Description

[Experimental]

Usage

prediction_error_variance(object, ...)

Arguments

- | | |
|--------|--------------------------------|
| object | A fitted model object. |
| ... | Reserved for future arguments. |

Details

prediction_error_variance() is part of the planned v0.1 fitted-object contract. It returns values only when an hsquared_fit object contains a Julia result field for prediction error variances.

The engine defines PEV as the animal-block diagonal of the inverse of the precision-scaled Henderson mixed-model-equation coefficient matrix, so `diag(C^-1)` is directly the prediction error variance in variance units, with no separate `sigma_e2` factor to apply.

Value

Prediction error variances for hsquared_fit objects.

qg_effect_markers	<i>Quantitative-genetic formula markers</i>
-------------------	---

Description

[Experimental]

Usage

```

permanent(formula, ...)

common_env(formula, ...)

maternal_genetic(formula, pedigree = NULL, ...)

maternal_env(formula, ...)

paternal_genetic(formula, pedigree = NULL, ...)

paternal_env(formula, ...)

cytoplasmic(formula, ...)

imprinting(formula, pedigree = NULL, parent = c("maternal", "paternal"), ...)

dominance(formula, pedigree = NULL, D = NULL, Dinv = NULL, ...)

epistasis(formula, pedigree = NULL, E = NULL, Einv = NULL, ...)

relmat(formula, K = NULL, Kinv = NULL, Q = NULL, ...)

precision(formula, Q = NULL, ...)

group(formula, ...)

unknown_parent_group(formula, ...)

metafounder(formula, pedigree = NULL, group = NULL, Gamma = NULL, ...)

inbreeding(formula, ...)

```

Arguments

formula	A random-effect expression such as <code>1 id</code> .
...	Reserved for future syntax.
pedigree	A pedigree data frame for future parental and relationship effects.
parent	Planned parent-of-origin side for imprinting effects.
D, Dinv	Dominance relationship or precision matrices.
E, Einv	Epistatic relationship or precision matrices.
K, Kinv, Q	User-supplied relationship or precision matrices.
group	Animal-to-metafounder group assignment, aligned to the normalized pedigree IDs for the opt-in supplied-variance metafounder() bridge.
Gamma	A supplied metafounder relationship matrix (an <i>m</i> -by- <i>m</i> covariance over the <i>m</i> metafounder pseudo-populations; Legarra et al. 2015). Supplied, not estimated; used by the opt-in supplied-variance metafounder() bridge.

Details

These functions provide readable formula vocabulary for standard quantitative-genetic, parental, inheritance, and custom-kernel models. Called directly they are inert (they return NULL); they take meaning only inside an `hsquared()` formula. `permanent()`, `common_env()`, and `maternal_genetic()` now fit through an opt-in, experimental engine path (`engine = "julia"`, REML-only, not the default, mirroring a partial validation gate) as the second random effect alongside `animal()`. `metafounder()` now fits through an opt-in, supplied-variance, experimental engine path (`engine = "julia"`, `target = "metafounder"`); Gamma and the variance components are supplied, not estimated. `relmat()` and `precision()` now fit through an opt-in, EXPERIMENTAL supplied-relationship primary engine path (`engine = "julia"`, REML-only, not the default, mirroring a partial validation gate): `relmat(1 | id, K = K)` supplies a dense symmetric positive-definite relationship/covariance matrix K (the parser marshals the inverse `Kinv = solve(K)`; `relmat()` also accepts a directly supplied inverse via `Kinv = Q`, no solve), and `precision(1 | id, Q = Q)` supplies the precision (inverse) directly (`Kinv =` also accepted); both fit through the SAME supplied-relationship-inverse path as `genomic()`'s `Ginv`, and the supplied matrix is provenance, not an estimate. The remaining markers (paternal/maternal-environment, dominance, epistasis, cytoplasmic, imprinting, genetic groups / unknown-parent-groups, and inbreeding) are still inert syntax reservations that the parser rejects with a planned-not-implemented message.

Some markers use generic names (e.g. `group()`, `inbreeding()`). They are formula-only tokens detected by call head and are not meant to be called directly, so attaching `hsquared` alongside a package that exports a same-named function (e.g. `pedigreemm::inbreeding()`) may print a masking message; this is expected and harmless.

Value

NULL, invisibly. Calls are interpreted by `hsquared()` when they appear inside model formulas.

random_regression_extractors

Random-regression (reaction-norm) extractors

Description

[Experimental]

Usage

```
rr_covariance(object, ...)

random_coefficients(object, ...)

rr_genetic_variance(object, at = NULL, n = 25L, ...)

rr_heritability(object, at = NULL, n = 25L, ...)

rr_correlation(object, at = NULL, n = 25L, ...)

rr_eigenfunctions(object, at = NULL, n = 25L, ...)
```

Arguments

<code>object</code>	A random-regression <code>hsquared_fit</code> object.
<code>...</code>	Reserved for future arguments.
<code>at</code>	Covariate values on the original scale at which to evaluate the trajectory. <code>NULL</code> (the default) uses an evenly spaced grid over the fitted covariate range. A supplied value outside the fitted covariate range errors (see the "Out-of-range at" section below).
<code>n</code>	Number of grid points used when <code>at = NULL</code> .

Details

These extractors summarize an opt-in, **experimental** random-regression (reaction-norm) fit (target = "random_regression"), fitted with an `animal(rr(covariate, order = k) | id, pedigree = ped)` term. The model estimates a $k \times k$ genetic covariance matrix K_g among an animal's normalized-Legendre random-regression coefficients plus a single homogeneous residual variance.

- `rr_covariance()` returns the estimated $k \times k$ coefficient genetic covariance matrix K_g .
- `random_coefficients()` returns the per-animal predicted Legendre coefficients (long format: `id, coefficient, value`).
- `rr_genetic_variance()` returns the additive genetic variance trajectory $v_g(t) = \phi(t)' K_g \phi(t)$ across covariate points.
- `rr_heritability()` returns the heritability trajectory $h^2(t) = v_g(t) / (v_g(t) + \sigma_e^2)$. Because the residual is homogeneous and there is no permanent-environment term yet, this can OVERSTATE $h^2(t)$ for repeated-records designs (test-day, growth curves).
- `rr_correlation()` returns the genetic correlation surface among the covariate points.
- `rr_eigenfunctions()` returns the eigen-decomposition of the coefficient covariance K_g evaluated as functions of the covariate: the eigenvalues (additive genetic variance carried by each axis), the proportion of genetic variance explained, the sign-canonicalized eigen-coefficients, and the eigenfunctions $\psi_j(t) = \phi(t)' v_j$ over the covariate grid. These are **rotation-invariant** functionals of K_g (mirroring the engine's `rr_eigenfunctions`); no raw, rotation-arbitrary loadings are returned.

The trajectories are computed in R from the estimated K_g and the recorded covariate standardization range; `at` is supplied on the ORIGINAL covariate scale (defaulting to a grid over the fitted range) and re-standardized to $[-1, 1]$ internally, matching the Julia engine's basis convention.

Value

`rr_covariance()` returns a numeric matrix; `random_coefficients()` returns a data frame; `rr_genetic_variance()` and `rr_heritability()` return data frames with `covariate` and `value` columns. `rr_correlation()` returns a numeric matrix. `rr_eigenfunctions()` returns a list containing eigenvalues, variance proportions, eigen-coefficients and evaluated eigenfunctions.

Out-of-range at

at must lie inside the fitted covariate range (object\$result\$random_regression\$lower to \$upper); a value outside that range errors with a message naming the fitted range, instead of silently returning the nearest endpoint's value. This matches the Julia engine (HSquared.legendre_basis), which refuses a standardized covariate outside $[-1, 1]$ rather than clamping it (#213).

Examples

```
# Synthetic result object: demonstrate extraction, not model fitting.
# The supplied covariance describes two Legendre coefficients.
fit_rr <- structure(list(
  spec = list(target = "random_regression"),
  result = list(coefficient_covariance = diag(c(1, 0.2)),
    residual_variance = 0.5,
    random_coefficients = data.frame(
      id = rep("animal1", 2),
      coefficient = c("legendre0", "legendre1"),
      value = c(0.1, 0.2)),
    random_regression = list(
      covariate = "age", lower = 1, upper = 5, order = 2),
    converged = TRUE)
), class = "hsquared_fit")

rr_covariance(fit_rr)
random_coefficients(fit_rr)
rr_genetic_variance(fit_rr)
rr_heritability(fit_rr)
rr_correlation(fit_rr, at = c(1, 3, 5))
rr_eigenfunctions(fit_rr)
```

ranef

Extract random effects

Description**[Experimental]****Usage**

```
ranef(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

ranef() is part of the planned v0.1 fitted-object contract for hsquared_fit objects.

Value

Random-effect estimates for `hsquared_fit` objects.

reliability	<i>Extract reliability and accuracy estimates</i>
-------------	---

Description

[Experimental]

Usage

```
reliability(object, ...)

accuracy(object, ...)
```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Details

`reliability()` is part of the planned v0.1 fitted-object contract. It returns values only when an `hsquared_fit` object contains a Julia result field for reliability estimates. `accuracy()` returns the square root of reliability for `hsquared_fit` objects.

The engine defines reliability as $1 - \text{PEV}_i / (\sigma_a^2 * A_{ii})$, using the dense relationship matrix $A = \text{inv}(A_{\text{inv}})$ implied by the supplied precision. For a genomic spec ($A_{\text{inv}} = G_{\text{inv}}$) this A_{ii} is `diag(inv(Ginv)) = diag(G)`

- `ridge`(the regularized genomic self-relationship, often not equal to 1), so the genomic ridge perturbation **rejects** a reliability value below 0 or above 1 rather than silently clipping it.

Value

Reliability estimates for `hsquared_fit` objects.

repeatability	<i>Extract repeatability estimates</i>
---------------	--

Description**[Experimental]****Usage**

```
repeatability(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

repeatability() reports the repeatability $R = (V_a + V_{pe}) / V_p$ of the opt-in, experimental repeatability (permanent-environment) model. It works for `hsquared_fit` objects fitted with `engine_control = list(target = "repeatability")`.

Separating `sigma_a2` from `sigma_pe2` needs repeated records per individual; see `?hs_control(target = "repeatability")` for that identifiability requirement.

Value

Repeatability results for repeatability `hsquared_fit` objects.

repeatability_interval	<i>Extract an experimental repeatability confidence interval</i>
------------------------	--

Description**[Experimental]****Usage**

```
repeatability_interval(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

`repeatability_interval()` returns an **experimental** large-sample (logit delta-method) confidence interval for the repeatability coefficient $t = (V_a + V_{pe}) / V_p$ of the opt-in repeatability (permanent-environment) model, available only when the fit contains it.

It mirrors the engine row V3-REPEAT-REML (partial): the engine's repeatability REML estimator and this interval are engine-internal self-consistency tested (recovery of t and interval bracketing / range / level-nesting / point-estimate match on seeded fixtures), but there is no external comparator, no h^2 interval, and no deep-pedigree validation. It is asymptotic, REML-only, unreliable at small n or near the (0, 1) boundary (where the engine throws and the field is omitted), and not a validated capability.

Value

A one-row data frame with `estimate` (the repeatability t), `lower`, `upper`, `level`, and `se`, for `hsquared_fit` objects that contain it.

response_scale_methods

Response-scale prediction helpers

Description

`predict()`, `fitted()`, and `residuals()` are part of the planned v0.1 fitted-object contract for univariate `hsquared_fit` objects. They are univariate-only: the multivariate model (a `cbind()` response) fits multiple traits jointly and is intentionally out of v0.1 response-scale scope, so these methods stop with a scope message pointing to `breeding_values()`, `genetic_covariance()`, and `residual_covariance()`.

Usage

```
## S3 method for class 'hsquared_fit'
predict(object, ...)

## S3 method for class 'hsquared_fit'
fitted(object, ...)

## S3 method for class 'hsquared_fit'
residuals(object, ...)
```

Arguments

<code>object</code>	A fitted model object.
<code>...</code>	Reserved for future arguments.

Value

Response-scale predictions, fitted values, or residuals for univariate `hsquared_fit` objects.

theme_hsqquared	<i>hsquared ggplot2 theme</i>
-----------------	-------------------------------

Description

[Experimental]

Usage

```
theme_hsqquared(base_size = 12)
```

Arguments

base_size	Base font size.
-----------	-----------------

Details

A light, publication-oriented ggplot2 theme shared by the `hsquared autoplot()` figures. Exported so users can restyle or extend the figures.

Value

A ggplot2 theme object.

validation_status	<i>Inspect validation evidence status</i>
-------------------	---

Description

[Experimental]

Usage

```
validation_status()
```

Details

`validation_status()` reports the current validation atoms and planned comparator lanes for `hsquared`. It is a **developer evidence table**, not the user-facing list of what you may report. For that list, read [Can I fit and report this?](#). This function does not run validation checks, fit models, or promote any capability to working status.

Value

A data frame of validation status records with class `"hs_validation_status"`.

This table is not the full list of covered routes

The rows here are validation *atoms*, grouped by evidence chain rather than by public modelling route, so the covered row count is not the number of covered routes. Two capabilities that count as publicly covered have no row of their own yet:

- the opt-in reaction-norm model, $k = 2$: `animal(rr(x, order = 2) | id, pedigree = ped) with engine_control = list(target = "random_regression");`
- the opt-in correlated direct-maternal 2x2 G model: `animal(1 | id, pedigree = ped) + maternal_genetic(1 | dam) with engine_control = list(target = "direct_maternal").`

Both are covered at validation scale, opt-in, REML-only, and dense ($n \leq 1000$ or so). Their scope and reporting limits are recorded in the repository capability ledger (`docs/design/capability-status.md`) and, for readers, in [Can I fit and report this?](#). Read this table together with that enumeration: an absent row is not an absent capability, and this table alone does not tell you everything you may report.

Bridge production fences (0.9)

The R-to-Julia bridge is **live at validation scale** for covered default and opt-in targets, but **not production-complete**. Infrastructure capabilities (R-to-Julia bridge payload, opt-in experimental Julia engine) stay **partial** even when individual targets are covered. Targets split into: default live (`engine = "fit"`), opt-in live (`engine_control$target`), **PATH_ONLY** smoke (C1-ext intervals — not a fit bridge), and **blocked** planned grammar (for example `genetic_structure = "factor_analytic"`). `payload_v2` routing is limited to `direct_maternal` and `multi_effect`. See `docs/design/45-bridge-production-fence`

Capability identifiers versus labels

`capability` is a **stable identifier**. Dated evidence records – comparator runs under `docs/dev-log/comparator-runs/` and check-log entries – cite these strings verbatim to name the row they report against, so an identifier is not rewritten once evidence points at it. Look rows up by `capability`.

`capability_label` is the **current wording**, and is what reader-facing surfaces print. The two differ only where an identifier's wording has been overtaken by a later change; today that is the multivariate row, whose `cbind()` route became default rather than opt-in.

Examples

```
validation_status()
```

variance_components	<i>Extract variance components</i>
---------------------	------------------------------------

Description

[Experimental]

Usage

```
variance_components(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

variance_components() is part of the planned v0.1 fitted-object contract. It works for `hsquared_fit` objects that contain a Julia result.

Value

Variance component results for `hsquared_fit` objects.

variance_component_standard_errors

Extract experimental variance-component and heritability standard errors

Description

[Experimental]

Usage

```
variance_component_standard_errors(object, ...)
```

```
heritability_standard_error(object, ...)
```

Arguments

object	A fitted model object.
...	Reserved for future arguments.

Details

variance_component_standard_errors() and heritability_standard_error() return **experimental** large-sample (delta-method) standard errors derived from the REML average-information matrix. They are available only when an `hsquared_fit` object contains them; the default Gaussian animal-model fit populates them from the engine when a local Julia engine is present and the AI matrix is invertible. When the engine could not compute them, `hsquared()` warns once at fit time and keeps the engine's reason in `attr(fit, "bridge_errors")`, a character vector named by the engine function that failed (`HSquared.jl#351`); the field is then absent, not NA.

These mirror the engine row V1-HERIT-CI (partial): asymptotic, REML-only, and unreliable at small n or near a variance-component boundary (where the AI matrix is ill-conditioned and the fields are omitted). The 2000-rep C1 coverage confirm (job **47925485**) measured the **additive-variance delta/Wald** interval implied by these SEs to **under-cover** (0.897 at nominal 0.95, $h^2=0.5$), placing

it at **experimental-only**: the SE is a point-estimate reference only, **not a calibrated and not a conservative interval**. (A profile additive-variance interval is directional-conservative by the same run — including the $h^2=0.3$ cell 0.963 — but is not surfaced by this delta-method extractor. Additive-variance bootstrap worst-cell $\hat{C}$ 0.918 is the same DC mild-under branch as h^2 bootstrap.) Not coverage-calibrated, not a validated capability.

Value

`variance_component_standard_errors()` returns a data frame with `component` and `se`. `heritability_standard_error` returns a one-row data frame with `term` and `se`, matching `heritability()` so the two merge on `term` (`hsquared#236`). Only for `hsquared_fit` objects that contain them. The fit object may still store `heritability_se` as a bare numeric internally; this extractor is the public shape.

Index

accuracy (reliability), 47
animal, 3
anova.hsquared_fit (inference_blocks), 34
autonomy (g_matrix_geometry), 20
autoplot.hs_gwas (hsquared-autoplot), 25
autoplot.hsquared_fit (hsquared-autoplot), 25

backend_info, 4
BLUP (breeding_values), 5
breeding_values, 5

common_env (qg_effect_markers), 42
common_env_effects, 5
common_env_proportion, 6
common_env_proportion(), 7, 22
common_env_proportion_interval, 7
common_env_proportion_interval(), 6
conditional_evolvability (g_matrix_geometry), 20
confint.hsquared_fit (inference_blocks), 34
covariance_standard_errors, 8
covariance_structure_lrt, 9
cytoplasmic (qg_effect_markers), 42

data_status, 10
direct_heritability (direct_maternal_extractors), 11
direct_heritability(), 22
direct_maternal_covariance (direct_maternal_extractors), 11
direct_maternal_extractors, 11
direct_variance (direct_maternal_extractors), 11
dominance (qg_effect_markers), 42

EBV (breeding_values), 5
eigen_G (g_matrix_geometry), 20
eigen_G(), 13
epistasis (qg_effect_markers), 42
eqtl_table (marker_extractors), 35
evolvability (g_matrix_geometry), 20

factor_g_extractors, 13
fit_diagnostics, 14
fit_diagnostics(), 23, 34, 40
fitted.hsquared_fit (response_scale_methods), 49
fixef, 15
formula_status, 15
formula_status(), 24

G_matrix (multivariate_extractors), 39
g_matrix_geometry, 13, 20
g_max (g_matrix_geometry), 20
gamma_matrix, 16
gamma_matrix(), 37, 38
genetic_correlation (multivariate_extractors), 39
genetic_correlation(), 12
genetic_covariance (multivariate_extractors), 39
genetic_loadings (factor_g_extractors), 13
genomic (genomic_markers), 16
genomic_markers, 16
group (qg_effect_markers), 42
gwas, 18
gwas_table (marker_extractors), 35

heritability, 22
heritability(), 6, 34, 37, 53
heritability_interval, 23
heritability_interval(), 7, 34
heritability_standard_error (variance_component_standard_errors), 52

- hs_control, 26
- hs_control(), 4, 24
- hs_data, 32
- hs_data(), 3, 10, 18, 24, 39
- hs_recovery_forest, 33
- hsquared, 24
- hsquared(), 3, 18, 44
- hsquared-autoplot, 25
- imprinting (qg_effect_markers), 42
- inbreeding (qg_effect_markers), 42
- inference_blocks, 34
- latent_breeding_values
 - (factor_g_extractors), 13
- lod_scores (marker_extractors), 35
- marker_effects (marker_extractors), 35
- marker_extractors, 35
- marker_scan (genomic_markers), 16
- marker_variance_explained
 - (marker_extractors), 35
- markers (genomic_markers), 16
- maternal_effects, 36
- maternal_effects(), 12
- maternal_env (qg_effect_markers), 42
- maternal_genetic (qg_effect_markers), 42
- maternal_proportion, 36
- maternal_proportion(), 7, 22
- maternal_proportion_interval
 - (common_env_proportion_interval), 7
- maternal_proportion_interval(), 37
- mean_evolvability (g_matrix_geometry), 20
- metafounder (qg_effect_markers), 42
- metafounder_effects, 37
- metafounder_groups, 38
- metafounder_groups(), 37
- model_spec, 38
- model_spec(), 3, 25
- multivariate_extractors, 39
- partner_variance
 - (direct_maternal_extractors), 11
- paternal_env (qg_effect_markers), 42
- paternal_genetic (qg_effect_markers), 42
- permanent (qg_effect_markers), 42
- permanent_effects, 40
- plot.hsquared_fit, 41
- precision (qg_effect_markers), 42
- predict.hsquared_fit
 - (response_scale_methods), 49
- prediction_error_variance, 42
- profile.hsquared_fit
 - (inference_blocks), 34
- qg_effect_markers, 42
- qtl_scan (genomic_markers), 16
- qtl_table (marker_extractors), 35
- R_matrix (multivariate_extractors), 39
- random_coefficients
 - (random_regression_extractors), 44
- random_regression_extractors, 44
- ranef, 46
- reliability, 47
- relmat (qg_effect_markers), 42
- repeatability, 48
- repeatability_interval, 48
- residual_correlation
 - (multivariate_extractors), 39
- residual_covariance
 - (multivariate_extractors), 39
- residuals.hsquared_fit
 - (response_scale_methods), 49
- responsability (g_matrix_geometry), 20
- response_scale_methods, 49
- rr_correlation
 - (random_regression_extractors), 44
- rr_covariance
 - (random_regression_extractors), 44
- rr_eigenfunctions
 - (random_regression_extractors), 44
- rr_genetic_variance
 - (random_regression_extractors), 44
- rr_heritability
 - (random_regression_extractors), 44
- rr_heritability(), 22, 26
- single_step (genomic_markers), 16

specific_variance
 (factor_g_extractors), 13
stats::confint(), 34
stats::profile(), 34

theme_hsqquared, 50
total_heritability
 (direct_maternal_extractors),
 11
total_heritability(), 12, 22

unknown_parent_group
 (qg_effect_markers), 42

validation_status, 50
validation_status(), 24
variance_along_gradient
 (g_matrix_geometry), 20
variance_component_standard_errors, 52
variance_component_standard_errors(),
 34, 41
variance_components, 51
variance_components(), 12, 34
vcov.hsqquared_fit(inference_blocks), 34