

Package: koine (via r-universe)

October 10, 2026

Title Synthesised Fourth Opinion for Bayesian Triangulation

Version 0.3.1

Description Synthesises one coherent, independent predictive opinion for a Bayesian model from a curated battery of known and underutilised inference methods, or abstains when no diversity gain exists. Built to serve as a fourth, synthesised witness alongside an independent panel of inference engines: it stacks over model-structure and method diversity in predictive space (leave-one-out stacking, PSIS-gated) on an axis orthogonal to the engine axis, never averaging the engines. Predictive opinions are exchanged through an ArviZ-style 'DataTree' 'NetCDF' interchange so no live 'Python' is required at read time. Abstention is a first-class outcome.

URL <https://github.com/max578/koine>

BugReports <https://github.com/max578/koine/issues>

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1)

Imports digest, loo, posterior, RNetCDF, stats, utils

Suggests BayesianTools, BayesLogit, bayesreg, bridgesampling, ggplot2, knitr, lme4, ncd4, nimble, rmarkdown, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libnetcdf-dev libudunits2-dev

Repository <https://max578.r-universe.dev>

Date/Publication 2026-10-09 22:32:14 UTC

RemoteUrl <https://github.com/max578/koine>

RemoteRef main

RemoteSha f55f0de989ef91485247bc215db28698617db16a

Contents

abstention	3
as_family_spec	4
assemble_loglik	4
evidence_axis	5
family_spec	6
fb_fit	8
fb_fit_diagnostics	9
fit_dram	10
fit_dream	10
fit_glmm_marginal	11
fit_glmm_robust_re	13
fit_horseshoe	14
fit_lasso	15
fit_nimble	16
fit_nimble_glmm	17
fit_pg_gibbs	18
fit_ridge	19
fit_robust_t	20
gate_corroboration	21
gate_psis	21
independence_report	22
is_abstention	23
koine_can_contribute	23
koine_lens_validation_setup	24
koine_opinion_draws	25
koine_pass	27
koine_pass_from_design	28
koine_stage1_registry	29
lens_effective_maturity	30
lens_spec	30
linear_pool_envelope	32
provenance_record	33
read_idata_nc	35
reference_posterior_check	35
resolve_battery	36
sbc_check	37
stack_opinion	39
synthesise	39
validate_fb_fit	40
validate_provenance	41
weight_concentration	42

abstention	<i>Construct a structured abstention</i>
------------	--

Description

An abstention records *why* koine declined to issue a fourth opinion, at what scope, and which lenses (if any) were dropped on the way. It is returned as a normal value and surfaces as "abstained" in the downstream triangulation.

Usage

```
abstention(  
  reason,  
  scope = c("synthesis", "lens"),  
  detail,  
  lenses_dropped = NULL  
)
```

Arguments

- reason

Single string from the closed taxonomy (Stage-0 SS8).
- scope

Either "synthesis" (the whole opinion) or "lens" (a single lens declined).
- detail

Human-readable explanation. Required — an abstention with no detail is itself misleading (I1).
- lenses_dropped

Optional list of list(id =, reason =) records for lenses dropped before the abstention.

Value

An object of class koine_abstention.

Examples

```
abstention("no_diverse_battery", detail = "only 1 opinion lens resolved")
```

as_family_spec	<i>Adapt a base-R family to an evaluable family_spec</i>
----------------	--

Description

Wraps a base-R / brms family object (which exposes linkinv but no pointwise log-likelihood) with the loglik_pointwise and rng functions koine needs. Supported families in this build are gaussian, binomial (and bernoulli, a binomial with one trial), and poisson. An unsupported family raises error.family_not_evaluable, never a silent approximation.

Usage

```
as_family_spec(family, trials = NULL)
```

Arguments

family	A base-R family object (e.g. gaussian(), binomial()), or a single string naming one.
trials	Optional length-N integer vector of binomial trial counts; defaults to 1 (the Bernoulli case).

Value

A koine_family_spec.

Examples

```
fs <- as_family_spec(gaussian())
fs$loglik_pointwise(y = c(0, 1), eta = c(0, 1), aux = list(sigma = 1))
```

assemble_loglik	<i>Assemble per-lens pointwise log-likelihood matrices (Stage-0 SS6.1)</i>
-----------------	--

Description

For each contributed opinion fit, takes the native pointwise log-likelihood if present, else reconstructs the S x N matrix by evaluating the family's loglik_pointwise at each draw's linear predictor. Columns are reindexed to the pass's canonical obs_id order (A3).

Usage

```
assemble_loglik(fb_fits, pass)
```

Arguments

fb_fits	A list of koine_fb_fit objects.
pass	The originating koine_pass.

Value

A named list (by lens method) of `list(ll =, chain_id =)`, where `ll` is an $S \times N$ matrix over the canonical observation axis.

Examples

```
set.seed(1)
d <- data.frame(y = rnorm(40), x = rnorm(40))
pass <- koine_pass(y ~ x, d, gaussian())
fit <- fit_dram(pass, n_iter = 600L, n_burn = 300L, n_chains = 2L)
ll <- assemble_loglik(list(fit), pass)
dim(ll[[1]]$ll)
```

evidence_axis	<i>Evidence-axis diagnostic (Stage-0 SS9.3)</i>
---------------	---

Description

Per-lens log marginal likelihood (model evidence) via bridge sampling (Meng & Wong 1996; Gronau et al. 2017), as a Bayes-factor cross-check on the battery. This axis has `role = "diagnostic"`: it is a scalar per lens and **never enters stacking** (the synthesis assembly excludes diagnostic-role output). The estimate is **prior-conditional** — it is computed under koine's canonical reference prior (coefficients $N(0, \text{beta_prior_sd})$; Gaussian dispersion via a log-sigma $N(0, \text{log_sigma_prior_sd})$ prior), and Bayes factors are sensitive to that prior, so the result is flagged accordingly. Lenses that fit an alternative model structure (a different likelihood, e.g. robust-t) are excluded from the cross-check: their evidence is not comparable on the shared prior.

Usage

```
evidence_axis(
  fb_fits,
  pass,
  beta_prior_sd = 5,
  log_sigma_prior_sd = 2,
  log_tau_prior_sd = 2
)
```

Arguments

<code>fb_fits</code>	A list of contributed <code>koine_fb_fit</code> objects.
<code>pass</code>	The originating <code>koine_pass</code> .
<code>beta_prior_sd</code>	Prior SD on the coefficients (default 5, matching the lenses' weakly-informative default).
<code>log_sigma_prior_sd</code>	Prior SD on log-sigma for Gaussian families (default 2).

log_tau_prior_sd

Prior SD on log-tau for mixed-model passes (default 2, matching the RE lenses' weakly-informative default).

Details

Requires the suggested **bridgesampling** package; absent it, returns a structured unavailable record rather than failing (11).

Value

A list with available (logical), prior_conditional (always TRUE), and, when available, evidence (a data frame of lens_id, log_ml, log_ml_se) plus note. role is documented as "diagnostic".

Examples

```
set.seed(1)
d <- data.frame(y = rnorm(40), x = rnorm(40))
pass <- koine_pass(y ~ x, d, gaussian())
fit <- fit_dram(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L)
evidence_axis(list(fit), pass)$evidence
```

family_spec

Construct an evaluable family specification

Description

A family_spec exposes, as pure functions, the pieces koine needs to compute a pointwise log-likelihood and a posterior-predictive draw for any lens that does not emit them natively. This low-level constructor is normally reached through [as_family_spec\(\)](#), which adapts a base-R family object.

Usage

```
family_spec(
  name,
  link,
  linkinv,
  loglik_pointwise,
  rng,
  support_check,
  aux_params = character(0),
  dlp_deta = NULL,
  d2lp_deta2 = NULL
)
```

Arguments

name	Family name, e.g. "gaussian", "binomial", "poisson".
link	Link name, e.g. "identity", "logit", "log".
linkinv	Function mapping the linear predictor eta to the response-scale mean mu.
loglik_pointwise	Function (y, eta, aux) returning a length-N vector of $\log p(y_i \eta_i, \text{aux})$. The contract koine relies on.
rng	Function (eta, aux) returning one posterior-predictive draw y_{tilde} (used only when posterior-predictive output is requested).
support_check	Function (y) returning TRUE if y lies in the family support.
aux_params	Character vector naming the auxiliary parameters (e.g. "sigma" for Gaussian, none for Binomial).
d1p_deta	Optional function (y, eta, aux) returning the first derivative of the pointwise log-likelihood with respect to the linear predictor (the canonical-link score), used by adaptive Gauss-Hermite quadrature to locate the random-effect integrand's mode.
d2lp_deta2	Optional function (y, eta, aux) returning the second derivative (the negative observed information), used for the quadrature's curvature scaling.

Value

An object of class `koine_family_spec`.

See Also

[as_family_spec\(\)](#)

Examples

```
# normally reached through as_family_spec(); the low-level constructor:
fs <- family_spec(
  name = "gaussian", link = "identity",
  linkinv = function(eta) eta,
  loglik_pointwise = function(y, eta, aux) {
    stats::dnorm(y, eta, aux$sigma, log = TRUE)
  },
  rng = function(eta, aux) stats::rnorm(length(eta), eta, aux$sigma),
  support_check = function(y) all(is.finite(y)),
  aux_params = "sigma")
fs$loglik_pointwise(c(0, 1), c(0, 1), list(sigma = 1))
```

fb_fit	<i>Construct an fb_fit</i>
--------	----------------------------

Description

Construct an fb_fit

Usage

```
fb_fit(
  draws,
  provenance,
  exactness = c("exact", "approximate", "resampled"),
  sample_stats = NULL,
  log_likelihood = NULL,
  posterior_predictive = NULL,
  diagnostics = NULL,
  status = c("contributed", "abstained"),
  abstention = NULL
)
```

Arguments

draws	A posterior::draws_array of equal-weight posterior draws (F1). Weighted samplers must resample to equal weight before constructing the fit and set exactness = "resampled".
provenance	A koine_provenance record (mandatory, F4).
exactness	"exact", "approximate", or "resampled".
sample_stats	Optional posterior::draws_array of HMC-style sample statistics; NULL for non-HMC lenses (F2).
log_likelihood	Optional array[chain, draw, obs_id]; usually NULL at construction and filled by assemble_loglik() (F3).
posterior_predictive	Optional predictive draws array.
diagnostics	Optional named list of convergence diagnostics; if NULL, rhat/ess are recovered from draws via posterior on demand.
status	"contributed" or "abstained".
abstention	An koine_abstention iff status == "abstained".

Value

A validated object of class koine_fb_fit.

Examples

```

set.seed(1)
draws <- posterior::as_draws_array(
  array(rnorm(8L), dim = c(2L, 2L, 2L),
    dimnames = list(NULL, NULL, c("b_(Intercept)", "b_x"))))
prov <- provenance_record(
  method = "demo", method_version = "0", impl_language = "R",
  seed = 1L, derived_seed = 1L, ir_hash = "h",
  maturity_tier = "reference-validated",
  independence_key = list(family = "random_walk", kernel = "demo"),
  role = "opinion", exactness = "exact")
fb_fit(draws = draws, provenance = prov, exactness = "exact",
  status = "contributed")

```

fb_fit_diagnostics	<i>Recover convergence diagnostics from an fb_fit's draws</i>
--------------------	---

Description

Computes split-Rhat and bulk/tail ESS from the posterior draws via `posterior::`, engine-agnostically. Never fabricates an HMC-only statistic (I1, F2): for non-HMC lenses `sample_stats` stays `NULL` and only the draws-derived diagnostics are returned.

Usage

```
fb_fit_diagnostics(x)
```

Arguments

`x` A contributed `koine_fb_fit`.

Value

A named list with `rhat`, `ess_bulk`, `ess_tail` (per variable).

Examples

```

set.seed(1)
draws <- posterior::as_draws_array(
  array(rnorm(80L), dim = c(20L, 2L, 2L),
    dimnames = list(NULL, NULL, c("b_(Intercept)", "b_x"))))
prov <- provenance_record(
  method = "demo", method_version = "0", impl_language = "R",
  seed = 1L, derived_seed = 1L, ir_hash = "h",
  maturity_tier = "reference-validated",
  independence_key = list(family = "random_walk", kernel = "demo"),
  role = "opinion", exactness = "exact")
fit <- fb_fit(draws = draws, provenance = prov, exactness = "exact",
  status = "contributed")
fb_fit_diagnostics(fit)

```

fit_dram

*Fit the DRAM lens***Description**

Runs delayed-rejection adaptive Metropolis on the pass's GLM and returns an `fb_fit()`. The fit carries equal-weight posterior draws and a complete provenance record; the pointwise log-likelihood is left NULL for `assemble_loglik()` to compute via the family spec (F3).

Usage

```
fit_dram(pass, lens_spec = NULL, n_iter = 4000L, n_burn = 2000L, n_chains = 4L)
```

Arguments

<code>pass</code>	A <code>koine_pass</code> .
<code>lens_spec</code>	The lens's registry record (for provenance + the derived seed); defaults to the registry's dram record.
<code>n_iter</code>	Iterations per chain.
<code>n_burn</code>	Burn-in iterations discarded per chain.
<code>n_chains</code>	Number of chains.

Value

A `koine_fb_fit`.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(40))
d$y <- 1 + 0.8 * d$x + rnorm(40)
pass <- koine_pass(y ~ x, d, gaussian())
fit_dram(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L)
```

fit_dream

*Fit the DREAM_ZS lens***Description**

Runs BayesianTools' DREAMzs sampler on the pass's GLM and returns an `fb_fit()`. Supports the Gaussian, Binomial (logit), and Poisson (log) families; for any other family/link it returns a lens-scope `abstention()`.

Usage

```
fit_dream(pass, lens_spec = NULL, n_iter = 15000L, burn_frac = 0.5)
```

Arguments

pass	A koine_pass.
lens_spec	The lens's registry record; defaults to the dream_zs record.
n_iter	Total DREAM iterations (model evaluations across the ensemble).
burn_frac	Fraction of each recorded chain discarded as burn-in.

Value

A koine_fb_fit, or a koine_abstention.

References

ter Braak & Vrugt (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. *Stat. Comput.* 18:435-446.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(40))
d$y <- 1 + 0.8 * d$x + rnorm(40)
pass <- koine_pass(y ~ x, d, gaussian())
fit_dream(pass, n_iter = 6000L)
```

fit_glmm_marginal	<i>Fit the marginal mixed-model lens</i>
-------------------	--

Description

Runs the koine-authored No-U-Turn Sampler on the random-effects *marginal* model — the random intercepts integrated out, exactly for the Gaussian LMM and by adaptive Gauss-Hermite quadrature for a GLMM — and returns an `fb_fit()` of (beta, sigma, tau) draws. For the Poisson marginal — whose variance-component funnel a single static-metric kernel cannot traverse — a funnel-safe blocked sampler is used: log tau is slice-sampled, and the coefficients are slice-sampled along their tau-resolved principal axes (a Gibbs block that needs no step size and cannot stall); the Gaussian and binomial marginals keep the joint NUTS kernel.

Usage

```
fit_glmm_marginal(
  pass,
  lens_spec = NULL,
  n_iter = 1500L,
  n_burn = 750L,
  n_chains = 4L,
  n_nodes = 20L,
  max_treedepth = 9L
)
```

Arguments

pass	A koine_pass carrying a single random intercept (1 g), or the MET structure of two crossed/nested random intercepts (1 variety) + (1 trial).
lens_spec	The lens's registry record; defaults to the registry's glmm_marginal record.
n_iter	Iterations per chain (including warm-up). The Hamiltonian kernel needs far fewer than a random walk for the same effective sample size, so the default is modest.
n_burn	Warm-up iterations per chain (step-size + mass-matrix adaptation), discarded from the returned draws.
n_chains	Number of chains.
n_nodes	Gauss-Hermite nodes for the GLMM marginal (ignored for the exact Gaussian LMM path).
max_treedepth	Maximum NUTS tree depth (the joint-NUTS Gaussian / binomial path only), capping the leapfrog budget per iteration at $2^{\text{max_treedepth}}$; saturation of the cap is surfaced in the provenance trace, never silently absorbed. The Poisson blocked slice kernel has no tree depth.

Details

For the breeder-standard multi-environment-trial (MET) structure — two crossed or nested random intercepts (1 | variety) + (1 | trial) — the Gaussian marginal stays exact (a dense Cholesky of $V = \sigma^2 I + \sum_r \tau_r^2 Z_r Z_r'$) and is sampled by a coordinate-wise slice kernel over (beta, sigma, tau_variety, tau_trial); the draws then carry one tau_<grouping> per random factor. A non-Gaussian MET (a dense crossed-GLMM marginal, which has no exact quadrature) returns a lens-scope [abstention\(\)](#). The pointwise log-likelihood is left NULL for [assemble_loglik\(\)](#) to compute as the per-observation integrated predictive (F3). If the pass carries no supported random intercept, returns a lens-scope [abstention\(\)](#) rather than a wrong fit (I1).

Value

A koine_fb_fit, or a koine_abstention for an unsupported pass.

Examples

```
set.seed(1)
g <- factor(rep(1:8, each = 5))
d <- data.frame(x = rnorm(40), g = g)
d$y <- 1 + 0.5 * d$x + rnorm(8)[g] + rnorm(40, 0, 0.7)
pass <- koine_pass(y ~ x + (1 | g), d, gaussian())
fit_glmm_marginal(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L)
```

fit_glmm_robust_re	<i>Fit the heavy-tailed random-effect lens (Student-t random intercepts)</i>
--------------------	--

Description

Fits a Gaussian linear mixed model whose random intercepts are Student-t ($b_g \sim t_{\nu}(\theta, \tau)$) by delayed-rejection adaptive Metropolis on the t-RE marginal, and returns an `fb_fit()` carrying its own t-RE integrated predictive. Resolves only for a Gaussian (identity) response with a supported single random intercept; otherwise returns a lens-scope `abstention()`.

Usage

```
fit_glmm_robust_re(
  pass,
  lens_spec = NULL,
  nu = 4,
  n_iter = 4000L,
  n_burn = 2000L,
  n_chains = 4L,
  n_nodes = 32L
)
```

Arguments

<code>pass</code>	A <code>koine_pass</code> with a Gaussian family and a single $(1 g)$.
<code>lens_spec</code>	The lens's registry record; defaults to the <code>glmm_robust_re</code> record.
<code>nu</code>	Degrees of freedom of the Student-t random effect (fixed; lower = heavier-tailed group effects). Default 4.
<code>n_iter, n_burn, n_chains</code>	Sampler controls.
<code>n_nodes</code>	Gauss-Laguerre nodes for the scale-mixture integral.

Value

A `koine_fb_fit`, or a `koine_abstention`.

Examples

```
set.seed(1)
g <- factor(rep(1:12, each = 8))
bt <- stats::rt(12, df = 4) * 1.2
d <- data.frame(x = rnorm(96), g = g)
d$y <- 1 + 0.5 * d$x + bt[g] + rnorm(96, 0, 0.7)
pass <- koine_pass(y ~ x + (1 | g), d, gaussian())
fit_glm_robust_re(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L)
```

fit_horseshoe

Fit the horseshoe (sparsity) lens

Description

Gaussian linear model with a horseshoe prior (Carvalho, Polson & Scott 2010) via bayesreg. Resolves only for a continuous (Gaussian, identity) response with at least one predictor; otherwise a lens-scope [abstention\(\)](#).

Usage

```
fit_horseshoe(
  pass,
  lens_spec = NULL,
  n_iter = 2000L,
  n_burn = 1000L,
  n_chains = 4L
)
```

Arguments

pass	A koine_pass with a Gaussian (identity) family.
lens_spec	The lens's registry record; defaults to the horseshoe record.
n_iter, n_burn, n_chains	Sampler controls.

Value

A koine_fb_fit, or a koine_abstention.

References

Carvalho, Polson & Scott (2010). The horseshoe estimator for sparse signals. *Biometrika* 97(2):465-480.

Examples

```
set.seed(1)
d <- data.frame(x1 = rnorm(60), x2 = rnorm(60))
d$y <- 1 + 0.8 * d$x1 + rnorm(60)
pass <- koine_pass(y ~ x1 + x2, d, gaussian())
fit_horseshoe(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_lasso

Fit the Bayesian lasso lens

Description

Gaussian linear model with a Laplace (Bayesian lasso) prior (Park & Casella 2008) via `bayesreg` — a moderate, soft-thresholding shrinkage between ridge and horseshoe. Same resolution rules as `fit_horseshoe()`.

Usage

```
fit_lasso(
  pass,
  lens_spec = NULL,
  n_iter = 2000L,
  n_burn = 1000L,
  n_chains = 4L
)
```

Arguments

`pass` A `koine_pass` with a Gaussian (identity) family.

`lens_spec` The lens's registry record; defaults to the lasso record.

`n_iter`, `n_burn`, `n_chains` Sampler controls.

Value

A `koine_fb_fit`, or a `koine_abstention`.

References

Park & Casella (2008). The Bayesian lasso. *JASA* 103(482):681-686.

Examples

```
set.seed(1)
d <- data.frame(x1 = rnorm(60), x2 = rnorm(60))
d$y <- 1 + 0.8 * d$x1 + rnorm(60)
pass <- koine_pass(y ~ x1 + x2, d, gaussian())
fit_lasso(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_nimble	<i>Fit the NIMBLE-Gibbs lens</i>
------------	----------------------------------

Description

Runs NIMBLE's conjugate / Metropolis-within-Gibbs sampler on the pass's GLM and returns an `fb_fit()` of equal-weight posterior draws with a complete provenance record. If the family/link pair is unsupported, returns a lens-scope `abstention()` rather than a wrong fit (I1).

Usage

```
fit_nimble(
  pass,
  lens_spec = NULL,
  n_iter = 4000L,
  n_burn = 2000L,
  n_chains = 4L
)
```

Arguments

<code>pass</code>	A <code>koine_pass</code> .
<code>lens_spec</code>	The lens's registry record; defaults to the registry's <code>nimble_gibbs</code> record.
<code>n_iter</code>	Iterations per chain (including burn-in).
<code>n_burn</code>	Burn-in iterations discarded per chain.
<code>n_chains</code>	Number of chains.

Value

A `koine_fb_fit`, or a `koine_abstention` for an unsupported link.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(40))
d$y <- 1 + 0.8 * d$x + rnorm(40)
pass <- koine_pass(y ~ x, d, gaussian())
fit_nimble(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_nimble_glmm	<i>Fit the NIMBLE GLMM lens</i>
-----------------	---------------------------------

Description

Runs NIMBLE's conjugate / Metropolis-within-Gibbs sampler on a mixed model with a single random intercept and returns an `fb_fit()` of (beta, sigma, tau) draws — the random effects are sampled but not retained, since the opinion is stacked on the integrated predictive. If the family/link pair is unsupported, or the pass carries no supported random intercept, returns a lens-scope `abstention()` rather than a wrong fit (II).

Usage

```
fit_nimble_glmm(
  pass,
  lens_spec = NULL,
  n_iter = 4000L,
  n_burn = 2000L,
  n_chains = 4L
)
```

Arguments

pass	A koine_pass carrying a single random intercept (1 g), or the MET structure of two crossed/nested random intercepts (1 variety) + (1 trial).
lens_spec	The lens's registry record; defaults to the registry's nimble_glmm record.
n_iter	Iterations per chain (including burn-in).
n_burn	Burn-in iterations discarded per chain.
n_chains	Number of chains.

Value

A koine_fb_fit, or a koine_abstention for an unsupported pass.

Examples

```
set.seed(1)
g <- factor(rep(1:8, each = 5))
d <- data.frame(x = rnorm(40), g = g)
d$y <- 1 + 0.5 * d$x + rnorm(8)[g] + rnorm(40, 0, 0.7)
pass <- koine_pass(y ~ x + (1 | g), d, gaussian())
fit_nimble_glmm(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_pg_gibbs

Fit the Polya-Gamma Gibbs lens (logistic regression)

Description

Runs the exact Polya-Gamma Gibbs sampler for a Bernoulli-logit model and returns an [fb_fit\(\)](#). Resolves only for the binomial (logit) family; for any other family it returns a lens-scope [abstention\(\)](#) rather than a wrong fit.

Usage

```
fit_pg_gibbs(
  pass,
  lens_spec = NULL,
  n_iter = 3000L,
  n_burn = 1500L,
  n_chains = 4L
)
```

Arguments

pass	A koine_pass with a binomial (logit) family.
lens_spec	The lens's registry record; defaults to the pg_gibbs record.
n_iter	Iterations per chain.
n_burn	Burn-in iterations discarded per chain.
n_chains	Number of chains.

Value

A koine_fb_fit, or a koine_abstention.

References

Polson, Scott & Windle (2013). Bayesian inference for logistic models using Polya-Gamma latent variables. *JASA* 108(504):1339-1349.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(80))
d$y <- rbinom(80, 1, plogis(0.5 * d$x))
pass <- koine_pass(y ~ x, d, binomial())
fit_pg_gibbs(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_ridge

Fit the Bayesian ridge lens

Description

Gaussian linear model with a Gaussian (ridge) prior of estimated global scale via bayesreg — mild, near-flat shrinkage anchoring the low-sparsity end of the shrinkage spectrum. Same resolution rules as [fit_horseshoe\(\)](#).

Usage

```
fit_ridge(
  pass,
  lens_spec = NULL,
  n_iter = 2000L,
  n_burn = 1000L,
  n_chains = 4L
)
```

Arguments

pass A koine_pass with a Gaussian (identity) family.

lens_spec The lens's registry record; defaults to the ridge record.

n_iter, n_burn, n_chains Sampler controls.

Value

A koine_fb_fit, or a koine_abstention.

Examples

```
set.seed(1)
d <- data.frame(x1 = rnorm(60), x2 = rnorm(60))
d$y <- 1 + 0.8 * d$x1 + rnorm(60)
pass <- koine_pass(y ~ x1 + x2, d, gaussian())
fit_ridge(pass, n_iter = 1000L, n_burn = 500L, n_chains = 2L)
```

fit_robust_t

Fit the robust-t lens (Student-t linear model)

Description

Fits $y \sim \text{Student-t}(\text{nu}, X \text{ beta}, \text{sigma})$ by delayed-rejection adaptive Metropolis (the DRAM kernel) and returns an `fb_fit()` carrying its own Student-t pointwise log-likelihood. Resolves only for a continuous (Gaussian-declared) response; otherwise returns a lens-scope `abstention()`.

Usage

```
fit_robust_t(
  pass,
  lens_spec = NULL,
  nu = 4,
  n_iter = 4000L,
  n_burn = 2000L,
  n_chains = 4L
)
```

Arguments

<code>pass</code>	A <code>koine_pass</code> with a Gaussian (identity) family.
<code>lens_spec</code>	The lens's registry record; defaults to the <code>robust_t</code> record.
<code>nu</code>	Degrees of freedom of the Student-t likelihood (fixed; lower = of heavier tails). Default 4.
<code>n_iter, n_burn, n_chains</code>	Sampler controls.

Value

A `koine_fb_fit`, or a `koine_abstention`.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(50))
d$y <- 1 + 0.8 * d$x + stats::rt(50, df = 4) * 0.6
pass <- koine_pass(y ~ x, d, gaussian())
fit_robust_t(pass, nu = 4, n_iter = 800L, n_burn = 400L, n_chains = 2L)
```

gate_corroboration	<i>The corroboration / maturity gate (Stage-0 SS11)</i>
--------------------	---

Description

Decides whether a lens of a given maturity tier may corroborate the fourth opinion, and at what weight discount. A koine-first lens (koine-implemented-validated) may corroborate only when validation evidence is on record (validation_ref); without it the lens is treated as experimental and may not corroborate — an unvalidated first-mover is never silently trusted (open-arms correctness, SS6/SS11).

Usage

```
gate_corroboration(maturity_tier, validation_ref = NULL)
```

Arguments

maturity_tier One of "reference-validated", "koine-implemented-validated", "experimental".
 validation_ref Optional identifier of the validation evidence (e.g. a ledger cairn id). Required to back the koine-implemented-validated tier.

Value

A list with effective_tier, may_corroborate (logical), and discount (1 reference-validated, 0.5 koine-validated, 0 experimental).

Examples

```
gate_corroboration("reference-validated")
gate_corroboration("koine-implemented-validated", validation_ref = "cairn-1")
gate_corroboration("koine-implemented-validated") # unbacked -> experimental
```

gate_psis	<i>The two-tier gate (Stage-0 SS7) — tier 1 (PSIS reliability)</i>
-----------	--

Description

Computes per-lens PSIS-LOO with a sample-size-dependent Pareto-k threshold $\min(1 - 1 / \log_{10}(S), 0.7)$. Lenses whose worst k exceeds the threshold are dropped with a recorded reason (never silently). This build implements the drop action; K-fold refit is a later stage.

Usage

```
gate_psis(loglik_list)
```

Arguments

loglik_list Output of `assemble_loglik()`.

Value

A list with loo (named list of loo objects for retained lenses), kept (ids), dropped (list(id =, reason =, k =)), and threshold.

References

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., & Gabry, J. (2024). Pareto smoothed importance sampling. *Journal of Machine Learning Research*, 25(72), 1-58.

Examples

```
set.seed(1)
d <- data.frame(y = rnorm(40), x = rnorm(40))
pass <- koine_pass(y ~ x, d, gaussian())
fit <- fit_dram(pass, n_iter = 600L, n_burn = 300L, n_chains = 2L)
gate_psis(assemble_loglik(list(fit), pass))
```

independence_report	<i>Build the independence report for the contributing lenses (Stage-0 SS10)</i>
---------------------	---

Description

One row per contributing opinion lens, carrying its two-level key, maturity tier, and a corroboration discount. The discount reflects: koine-first maturity (SS11), within-family kernel redundancy, and membership of a collapse grouping. This feeds the downstream fourth-vs-three grading.

Usage

```
independence_report(fb_fits)
```

Arguments

fb_fits A list of contributed koine_fb_fit objects.

Value

A data frame with columns lens_id, family, kernel, maturity_tier, discount, and collapsed_witness.

Examples

```
set.seed(1)
d <- data.frame(y = rnorm(40), x = rnorm(40))
pass <- koine_pass(y ~ x, d, gaussian())
fit <- fit_dram(pass, n_iter = 600L, n_burn = 300L, n_chains = 2L)
independence_report(list(fit))
```

is_abstention	<i>Is this object an abstention?</i>
---------------	--------------------------------------

Description

Is this object an abstention?

Usage

```
is_abstention(x)
```

Arguments

x Any object.

Value

TRUE if x is a koine_abstention.

Examples

```
out <- abstention("no_diverse_battery", scope = "synthesis",
                  detail = "only one opinion lens resolved.")
is_abstention(out)
is_abstention(42)
```

koine_can_contribute	<i>Can koine contribute a diverse opinion for this model? (pre-flight)</i>
----------------------	--

Description

A cheap capability predicate — it resolves the battery but fits nothing — for a parent panel's backend-availability gate. Returns TRUE when at least two independent opinion lenses would enrol, or a single reason string from the abstention taxonomy otherwise ("feature_unsupported", "no_diverse_battery", or "pass_invalid" for a malformed model). Because the PSIS reliability gate and weight-degeneracy check can only be evaluated after fitting, a TRUE here is necessarily optimistic: koine may still abstain at `synthesise()` time. The predicate is a *floor*, not a guarantee.

Usage

```
koine_can_contribute(
  formula,
  data,
  family,
  resources = list(allowed_tiers = 1L),
  registry = koine_stage1_registry()
)
```

Arguments

formula, data, family
 As for [koine_pass\(\)](#).

resources A resources list (see [koine_pass\(\)](#)); defaults to tier 1.

registry A registry list; defaults to [koine_stage1_registry\(\)](#).

Value

TRUE, or a single reason-code string.

See Also

[koine_opinion_draws\(\)](#), [synthesise\(\)](#)

Examples

```
d <- data.frame(y = rnorm(40), x = rnorm(40))
koine_can_contribute(y ~ x, d, gaussian())
```

koine_lens_validation_setup

Build the validation setup for a koine-authored lens

Description

Returns the prior draw, the model simulator, the sampler call, the parameters to rank, and a grid-exact reference-posterior builder for one of koine's koine-authored lenses ("dram", "pg_gibbs", "robust_t"). This is the shared harness behind [sbc_check\(\)](#) and [reference_posterior_check\(\)](#) for the recorded validation study; SBC simulates from the model returned here, so the calibration test sees the lens's own {prior, model, sampler} triple.

Usage

```
koine_lens_validation_setup(
  lens_id,
  n_obs = 40L,
  n_pred = 1L,
  nu = 4,
  fit_control = list(n_iter = 1500L, n_burn = 750L, n_chains = 2L),
  family = c("gaussian", "binomial", "poisson"),
  n_groups = 12L,
  n_per_group = 6L
)
```

Arguments

<code>lens_id</code>	One of "dram", "pg_gibbs", "robust_t" (population) or "glmm_marginal", "nimble_glmm" (mixed-model lenses).
<code>n_obs</code>	Observations per simulated dataset (population lenses).
<code>n_pred</code>	Number of predictors beyond the intercept.
<code>nu</code>	Student-t degrees of freedom for "robust_t" (the generative model and the sampler share this value).
<code>fit_control</code>	A list of sampler controls passed to the lens fit.
<code>family</code>	For the mixed-model lenses, the response family ("gaussian", "binomial", or "poisson"); ignored by the population lenses, which carry their own family.
<code>n_groups</code>	Number of random-effect groups (mixed-model lenses).
<code>n_per_group</code>	Observations per group (mixed-model lenses).

Value

A list with `generate`, `fit_fn`, `param_names`, and `reference`.

Examples

```
setup <- koine_lens_validation_setup("dram")
setup$param_names
```

<code>koine_opinion_draws</code>	<i>koine's opinion as a posterior draws array (the stacking mixture)</i>
----------------------------------	--

Description

Materialises `koine`'s single synthesised opinion as an equal-weight posterior `::draws_array` over the parameters its contributing lenses share, by drawing from each retained lens in proportion to its LOO-stacking weight. This is the parameter-space representation a parent panel's parameter-marginal triangulation consumes. The stacking weights, the stacked predictive elpd, any per-lens parameters dropped from the shared set, and any scale parameters whose lens set was restricted ride along as attributes (`koine_weights`, `koine_elpd_stacked`, `koine_dropped_vars`, `koine_scale_restricted`).

Usage

```
koine_opinion_draws(result, n_draws = 4000L, seed = NULL)
```

Arguments

<code>result</code>	A <code>koine_result</code> produced with <code>synthesise(pass, keep_fits = TRUE)</code> . A <code>koine_abstention</code> is returned unchanged (koine contributed no opinion to represent).
<code>n_draws</code>	Total draws in the returned mixture. Default 4000.
<code>seed</code>	Optional integer seed for the mixture sampling; defaults to a seed derived from the pass's content hash, so the representation is reproducible.

Details

The mixture is over koine's own battery (a legitimate single opinion), never a pool across the parent's independent engines (charter SS4). Each parameter is mixed only over the lenses for which it is comparable: the linear-predictor coefficients (`b_<col>`) over every contributing lens, but a scale parameter (`sigma`, `tau`) only over the family-likelihood lenses — a likelihood-variant lens (the robust-t lens, which carries its own Student-t likelihood) reports its scale on a different metric, so it would bias the synthesised value if averaged in. The returned draws are therefore a per-parameter marginal representation (each column is its comparable-lens stacking mixture); the joint is approximate, which is exactly what a parameter-marginal triangulation needs.

Value

A one-chain `posterior::draws_array`, or the input `abstention`.

See Also

[synthesise\(\)](#), [koine_can_contribute\(\)](#)

Examples

```
set.seed(1)
d <- data.frame(y = 1 + rnorm(80), x = rnorm(80))
pass <- koine_pass(y ~ x, d, gaussian())
res <- synthesise(pass, keep_fits = TRUE)
if (!is_abstention(res)) {
  dr <- koine_opinion_draws(res)
  posterior::summarise_draws(dr, "mean", "sd")
}
```

koine_pass	<i>Construct a koine pass</i>
------------	-------------------------------

Description

Construct a koine pass

Usage

```
koine_pass(
  formula,
  data,
  family,
  required_outputs = c("posterior", "log_likelihood"),
  seed = 1L,
  obs_id = NULL,
  resources = list(allowed_tiers = 1L),
  prior = NULL
)
```

Arguments

formula	A model formula. Population-level terms, optionally with lme4-style random <i>intercepts</i> : a single (1 g), or the breeder-standard multi-environment-trial (MET) structure of two crossed or nested random intercepts (1 variety) + (1 trial). Richer random-effects structure (random slopes, correlated effects, more than two groupings) is parsed and flagged unsupported so the battery abstains non-silently rather than fitting the wrong model. Whether two groupings are crossed or nested is resolved from the data, never from the formula.
data	A data frame holding every column referenced in formula.
family	A base-R family object or a known family name; wrapped into an evaluable family_spec() .
required_outputs	Character subset of c("posterior", "log_likelihood", "posterior_predictive", "sample_stats").
seed	Integer master seed; per-lens seeds are derived from it.
obs_id	Optional character vector of observation ids (defaults to obs_0001-style labels of length nrow(data)).
resources	A list with allowed_tiers (subset of 1:3) and optional budgets; defaults to tier 1 only.
prior	Optional structured prior table (unused by the DRAM lens, which uses its own weakly-informative default; reserved for later lenses).

Value

A validated object of class koine_pass.

Examples

```
d <- data.frame(y = rnorm(20), x = rnorm(20))
koine_pass(y ~ x, d, gaussian())
```

```
koine_pass_from_design
```

Construct a koine pass from design arrays (the IR-targetable builder)

Description

The programmatic, re-entrant, deparse-free entry point a parent triangulation panel drives (flexy-Bayes backend-contract clause 1): build a pass directly from a named design matrix, a response, a family, and an optional grouping vector — no model formula, no `match.call()` capture, no session-global state. Two passes can be built and synthesised in the same process without interference. The stored design is used verbatim downstream, so non-syntactic canonical column names (e.g. (Intercept)) pass through untouched.

Usage

```
koine_pass_from_design(
  x,
  y,
  family,
  group = NULL,
  groups = NULL,
  coef_names = colnames(x),
  response_name = "y",
  grouping_name = "g",
  required_outputs = c("posterior", "log_likelihood"),
  seed = 1L,
  obs_id = NULL,
  resources = list(allowed_tiers = 1L),
  prior = NULL
)
```

Arguments

<code>x</code>	A numeric design matrix with named columns (the fixed-effects design; include an (Intercept) column if wanted). Column names become the coefficient names (<code>b_<col></code> in the draws).
<code>y</code>	The response vector (length <code>nrow(x)</code>).
<code>family</code>	A base-R family object or a known family name.
<code>group</code>	Optional grouping vector (length <code>nrow(x)</code>) for a single random intercept (<code>1 group</code>); <code>NULL</code> for a population model. For the MET structure of two crossed/nested random intercepts pass <code>groups</code> instead.

groups	Optional named list of two grouping vectors (each length <code>nrow(x)</code>) for two crossed/nested random intercepts — the breeder-standard MET structure, e.g. <code>list(variety = ., trial = .)</code> . The list names become the grouping-factor names; mutually exclusive with <code>group</code> .
coef_names	Optional column names for <code>x</code> (defaults to <code>colnames(x)</code>).
response_name	Name recorded for the response (default <code>"y"</code>).
grouping_name	Name recorded for the (single) grouping factor when <code>group</code> is supplied (default <code>"g"</code>); ignored when <code>groups</code> is used.
required_outputs	Character subset of <code>c("posterior", "log_likelihood", "posterior_predictive", "sample_stats")</code> .
seed	Integer master seed; per-lens seeds are derived from it.
obs_id	Optional character vector of observation ids (defaults to <code>obs_0001</code> -style labels of length <code>nrow(data)</code>).
resources	A list with <code>allowed_tiers</code> (subset of <code>1:3</code>) and optional budgets; defaults to tier 1 only.
prior	Optional structured prior table (unused by the DRAM lens, which uses its own weakly-informative default; reserved for later lenses).

Value

A validated object of class `koine_pass`.

See Also

[koine_pass\(\)](#), [koine_opinion_draws\(\)](#)

Examples

```
set.seed(1)
x <- cbind(`(Intercept)` = 1, z = rnorm(40))
y <- 1 + 0.5 * x[, "z"] + rnorm(40)
pass <- koine_pass_from_design(x, y, gaussian())
pass$has_re           # FALSE -- a population model, built without a formula
```

`koine_stage1_registry` *The Stage-1 enrolment registry (leanest cut)*

Description

The Stage-1 enrolment registry (leanest cut)

Usage

```
koine_stage1_registry()
```

Value

A list of koine_lens_spec records.

Examples

```
reg <- koine_stage1_registry()
vapply(reg, function(L) L$id, character(1))
```

lens_effective_maturity

Effective maturity tier of a lens after the validation gate

Description

Resolves a lens spec's declared maturity against its validation_ref: a koine-first lens without recorded validation evidence is demoted to experimental (it may not corroborate). Reference-validated lenses and validation-backed koine lenses keep their declared tier.

Usage

```
lens_effective_maturity(lens_spec)
```

Arguments

lens_spec A koine_lens_spec.

Value

The effective maturity tier (a string).

Examples

```
lens_effective_maturity(koine_stage1_registry()[[1]])
```

lens_spec

Construct a lens specification (a registry record)

Description

Construct a lens specification (a registry record)

Usage

```
lens_spec(
  id,
  label,
  tier,
  independence_key,
  role,
  exactness,
  maturity_tier,
  family_support,
  feature_support,
  emits_loglik,
  fit_fn = NULL,
  implemented = FALSE,
  emits_evidence = FALSE,
  structure_variant = FALSE,
  validation_ref = NULL
)
```

Arguments

<code>id</code>	Stable key, e.g. "dram".
<code>label</code>	Human-readable label.
<code>tier</code>	1 (R/C++ native), 2 (opt-in C++), or 3 (opt-in Python).
<code>independence_key</code>	<code>list(family =, kernel =)</code> (Stage-0 SS10).
<code>role</code>	"opinion" (enters stacking) or "diagnostic" (never stacked).
<code>exactness</code>	"exact", "approximate", or "resampled".
<code>maturity_tier</code>	One of the maturity tiers (Stage-0 SS11).
<code>family_support</code>	Predicate (<code>family_spec</code>) -> logical.
<code>feature_support</code>	A <code>list(random_effects, distributional, nonlinear, met)</code> . <code>met</code> (default FALSE) flags a mixed-model lens that also covers the MET structure of two crossed/nested random intercepts.
<code>emits_loglik</code>	TRUE if the lens emits a pointwise log-likelihood natively; FALSE means koine computes it via the family spec.
<code>fit_fn</code>	The adapter function(<code>pass, lens_spec</code>) -> <code>koine_fb_fit</code> , or NULL if not yet implemented in this build.
<code>implemented</code>	TRUE if <code>fit_fn</code> is a working adapter.
<code>emits_evidence</code>	TRUE if the lens yields a marginal likelihood.
<code>structure_variant</code>	TRUE if the lens fits an alternative model structure (e.g. a robust-t likelihood) rather than a different inference method on the same model — the source of genuine stacking diversity.

`validation_ref` Optional identifier (e.g. a ledger cairn id) recording the SBC / reference-posterior evidence behind a koine-implemented-validated maturity tier (Stage-0 SS11). NULL means no validation evidence is on record — a koine-first lens without it is treated as experimental by the maturity gate, never silently trusted (open-arms correctness, SS11).

Value

A `koine_lens_spec`.

Examples

```
lens_spec(
  id = "demo", label = "Demonstration lens", tier = 1L,
  independence_key = list(family = "random_walk", kernel = "demo"),
  role = "opinion", exactness = "exact",
  maturity_tier = "reference-validated",
  family_support = function(fs) fs$name == "gaussian",
  feature_support = list(random_effects = FALSE, distributional = FALSE,
                        nonlinear = FALSE, met = FALSE),
  emits_loglik = FALSE)
```

`linear_pool_envelope` *Linear-pool envelope diagnostic (Stage-0 SS9.2)*

Description

An equal-weight mixture of the contributing lenses' conditional-mean predictives, deliberately **over-dispersed** to serve as a reference bracket rather than a confident prediction. For each observation the envelope half-width combines the average within-lens posterior uncertainty with the between-lens disagreement, the latter inflated by `inflate` (default 2). The reported envelope is the full equal-weight pool; flagging is leave-one-out — a lens is flagged at an observation when its posterior-mean response falls outside the over-dispersed bracket formed by the *other* lenses, so a disagreement carrier is caught against the consensus rather than inflating its own bracket. The envelope is a diagnostic surface only and is never shipped as the opinion (I2).

Usage

```
linear_pool_envelope(fb_fits, pass, inflate = 2, level = 0.95)
```

Arguments

<code>fb_fits</code>	A list of contributed <code>koine_fb_fit</code> objects (opinion lenses).
<code>pass</code>	The originating <code>koine_pass</code> .
<code>inflate</code>	Over-dispersion multiplier on the between-lens disagreement variance (default 2, per SS9.2's "inflated approximately 2x disagreement").
<code>level</code>	Central interval width for the bracket (default 0.95).

Details

Operates on the conditional-mean scale $\mu_i = \text{linkinv}(\eta_i)$, which is comparable across every lens regardless of likelihood-tail differences, so a robust-t lens and a Gaussian lens are bracketed on the same footing.

Value

A list with `obs_id`, `pool_mean`, `lower`, `upper` (length-N vectors), `flags` (a data frame of `lens_id` x out-of-envelope counts), `n_components`, and `inflate`; or `NULL` when fewer than two lenses expose a comparable mean-response (no envelope is meaningful).

Examples

```
set.seed(1)
d <- data.frame(y = 1 + rnorm(50), x = rnorm(50))
pass <- koine_pass(y ~ x, d, gaussian())
fits <- list(
  fit_dram(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L),
  fit_robust_t(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L))
env <- linear_pool_envelope(fits, pass)
env$flags
```

provenance_record	<i>Build a provenance record</i>
-------------------	----------------------------------

Description

Build a provenance record

Usage

```
provenance_record(
  method,
  method_version,
  impl_language,
  seed,
  derived_seed,
  ir_hash,
  maturity_tier,
  independence_key,
  role,
  exactness,
  timestamp_iso8601 = NULL,
  lib_versions = NULL,
  device = NULL,
  py_pins = NULL,
  trace = NULL,
```

```

    structure_variant = FALSE
)

```

Arguments

<code>method</code>	Lens method id (e.g. "dram").
<code>method_version</code>	Version string of the method implementation.
<code>impl_language</code>	"R", "C++", or "Python".
<code>seed</code>	Master seed from the pass.
<code>derived_seed</code>	Per-lens derived seed.
<code>ir_hash</code>	Content hash of the originating pass.
<code>maturity_tier</code>	One of the maturity tiers (Stage-0 SS11).
<code>independence_key</code>	<code>list(family =, kernel =)</code> (Stage-0 SS10).
<code>role</code>	"opinion" or "diagnostic".
<code>exactness</code>	"exact", "approximate", or "resampled".
<code>timestamp_iso8601</code>	ISO-8601 timestamp; defaults to now (UTC).
<code>lib_versions</code>	Optional named list of library versions.
<code>device</code>	Optional device string.
<code>py_pins</code>	Optional named list of Python pins (Tier-3 only).
<code>trace</code>	Optional list of per-stage outcomes.
<code>structure_variant</code>	TRUE if the lens fits an alternative model structure (a different likelihood or prior) rather than the canonical reference posterior. Carried so the diagnostic layer (the shared-prior evidence axis, SS9.3) and the independence registry's specification axis (SS10.1) can tell structure variants from reference-posterior lenses.

Value

A `koine_provenance` object.

Examples

```

provenance_record(
  method = "dram", method_version = "0.1.0", impl_language = "R",
  seed = 1L, derived_seed = 42L, ir_hash = "abc123",
  maturity_tier = "reference-validated",
  independence_key = list(family = "random_walk", kernel = "dram"),
  role = "opinion", exactness = "exact")

```

read_idata_nc	<i>Read an ArviZ DataTree NetCDF-4 file into an fb_fit</i>
---------------	--

Description

Read an ArviZ DataTree NetCDF-4 file into an fb_fit

Usage

```
read_idata_nc(path, pass = NULL, reader = c("auto", "rnetcdf", "ncdf4"))
```

Arguments

path	Path to the .nc file.
pass	Optional koine_pass; when supplied, the log-likelihood observation axis is asserted equal to pass\$obs_dim_names\$obs_id in the same order (A3).
reader	One of "auto" (RNetCDF if available, else ncdf4), "rnetcdf", or "ncdf4".

Value

A koine_fb_fit.

Examples

```
# read_idata_nc() resolves an ArviZ DataTree NetCDF written by a Tier-3
# (Python) lens into the uniform fb_fit contract; supply the .nc path and
# the originating pass:
## Not run:
fit <- read_idata_nc("inference.nc", pass)

## End(Not run)
```

reference_posterior_check	<i>Exact grid-based reference-posterior check (Stage-0 SS11)</i>
---------------------------	--

Description

Compares a lens's posterior to an exact reference computed by dense grid integration of the same unnormalised log-posterior — a reference that is independent of any sampler and needs no external engine, so it is runnable in this standalone package. Feasible only for low-dimensional models (the grid is the product over parameters); for larger models it returns a structured "not applicable" record rather than a misleading number (I1).

Usage

```
reference_posterior_check(
  fit,
  log_post_unnorm,
  grid,
  tol = 0.15,
  max_cells = 2e+05
)
```

Arguments

<code>fit</code>	A contributed <code>koine_fb_fit</code> from the lens under test.
<code>log_post_unnorm</code>	A function <code>(theta) -> numeric</code> : the lens's unnormalised log-posterior (likelihood + prior) at a named parameter vector on the draw scale (the same variable names the fit's draws carry).
<code>grid</code>	A named list of numeric grid vectors, one per parameter, naming the parameters in draw-scale order.
<code>tol</code>	Maximum tolerated absolute standardised discrepancy between the lens and reference posterior mean (in reference SDs) for a pass.
<code>max_cells</code>	Refuse (return not-applicable) above this many grid cells.

Value

A list with applicable (logical), and when applicable reference and lens summaries (per-parameter mean and SD), discrepancy (per parameter, in reference SDs), and passed.

Examples

```
setup <- koine_lens_validation_setup(
  "dram", n_obs = 30L,
  fit_control = list(n_iter = 1500L, n_burn = 750L, n_chains = 2L))
g <- setup$generate(1L)
fit <- setup$fit_fn(g$pass)
ref <- setup$reference(g, n_pts = 21L)
reference_posterior_check(fit, ref$log_post_unnorm, ref$grid)$passed
```

resolve_battery	<i>Resolve the battery for a pass (family-conditional enrolment, Stage-0 SS3.3)</i>
-----------------	---

Description

Enrols a lens iff: its tier is allowed; its family predicate passes; its feature support covers the formula; the log-likelihood is producible (native or via the family spec); the adapter is implemented in this build; and, for a koine-authored lens, its validation evidence is on record (the open-arms correctness gate, SS11 — an unvalidated first-mover abstains). A lens that fails any test is excluded *with a recorded reason* — a logged absence, not an error (I1).

Usage

```
resolve_battery(pass, registry = koine_stage1_registry())
```

Arguments

`pass` A `koine_pass`.
`registry` A registry list; defaults to `koine_stage1_registry()`.

Value

A list with enrolled (lens specs) and excluded (`list(id =, reason =)` records).

Examples

```
set.seed(1)
d <- data.frame(y = rnorm(30), x = rnorm(30))
pass <- koine_pass(y ~ x, d, gaussian())
resolved <- resolve_battery(pass)
vapply(resolved$enrolled, function(L) L$id, character(1))
```

sbc_check	<i>Simulation-based calibration of a lens (Stage-0 SS11; Talts et al. 2018)</i>
-----------	---

Description

Repeatedly draws true parameters from the prior, simulates a dataset from the lens's own generative model, fits the lens, and records the rank of each true parameter among the (thinned) posterior draws. If the lens samples its posterior correctly, the ranks are uniform on $0:n_keep$; systematic departures (a chi-squared test per parameter) reveal a miscalibrated sampler.

Usage

```
sbc_check(
  generate,
  fit_fn,
  param_names,
  n_sims = 128L,
  n_keep = 64L,
```

```

    n_bins = 8L,
    alpha = 0.05,
    seed = 1L
  )

```

Arguments

<code>generate</code>	A function (seed) -> list(theta = named-numeric, pass = koine_pass): draws true parameters from the prior and builds a pass whose data is simulated from the model at theta. theta's names must match the posterior draw variables to be ranked.
<code>fit_fn</code>	A function (pass) -> koine_fb_fit (the lens, with its sampler controls already set).
<code>param_names</code>	Character vector of parameters to rank (a subset of the names of theta and of the fit's draw variables).
<code>n_sims</code>	Number of SBC replications.
<code>n_keep</code>	Posterior draws retained per fit (the ranks lie in 0:n_keep); draws are thinned to this many to keep the rank statistic near-independent.
<code>n_bins</code>	Number of histogram bins for the chi-squared uniformity test.
<code>alpha</code>	Family-wise significance level (Bonferroni-adjusted across parameters) below which calibration is judged to have failed.
<code>seed</code>	Master seed; per-replication seeds are derived from it.

Details

SBC validates the {prior, model, sampler} triple as a whole, so the generate callback must simulate data from exactly the model the lens assumes (e.g. a Student-t likelihood for the robust-t lens), using the exact prior the lens places.

Value

A list with ranks (an `n_sims` x `length(param_names)` matrix), `n_keep`, `uniformity` (a data frame of `param`, `chisq`, `df`, `p_value`), `alpha_adj`, and `passed` (logical: every parameter's p-value exceeds the adjusted alpha).

Examples

```

setup <- koine_lens_validation_setup(
  "dram", n_obs = 30L,
  fit_control = list(n_iter = 800L, n_burn = 400L, n_chains = 2L))
res <- sbc_check(setup$generate, setup$fit_fn, setup$param_names,
  n_sims = 16L, n_keep = 16L, n_bins = 4L)
res$passed

```

stack_opinion	<i>LOO-stacking of opinion lenses (Stage-0 SS6.2)</i>
---------------	---

Description

Computes stacking weights over the retained lenses and the stacked leave-one-out predictive density (the stacking objective evaluated at the optimum). Weights are retained for the diagnostic surface; they are never shipped as a pooled posterior (I2).

Usage

```
stack_opinion(loos_list)
```

Arguments

`loos_list` A named list of loos objects (the gate's loos).

Value

A list with weights (named `simplex`), `elpd_stacked`, and `n_components`.

References

Yao, Y., Vehtari, A., Simpson, D., & Gelman, A. (2018). Using stacking to average Bayesian predictive distributions. *Bayesian Analysis*, 13(3), 917-1007.

Examples

```
set.seed(1)
d <- data.frame(y = 1 + rnorm(60), x = rnorm(60))
pass <- koine_pass(y ~ x, d, gaussian())
fits <- list(
  fit_dram(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L),
  fit_robust_t(pass, n_iter = 800L, n_burn = 400L, n_chains = 2L))
gate <- gate_psis(assemble_loglik(fits, pass))
if (length(gate$kept) >= 2L) stack_opinion(gate$loos)
```

synthesise	<i>Synthesise the fourth opinion for a pass (Stage-0 SS1 flow)</i>
------------	--

Description

Resolves the battery, fits each opinion lens, assembles pointwise log-likelihoods, applies the PSIS gate, stacks the survivors, and reports a weight-concentration diagnostic — or returns a first-class `abstention()` when fewer than two independent opinion lenses survive. The result is either a `koine_result` or a `koine_abstention`, never a silent partial.

Usage

```
synthesise(
  pass,
  registry = koine_stage1_registry(),
  on_degenerate = c("flag", "abstain"),
  keep_fits = FALSE
)
```

Arguments

<code>pass</code>	A <code>koine_pass</code> .
<code>registry</code>	A registry list; defaults to <code>koine_stage1_registry()</code> .
<code>on_degenerate</code>	What to do when the stacking weights are degenerate (the battery corroborated rather than diversified): "flag" (default) returns the opinion with the degeneracy recorded in its diagnostics; "abstain" returns a <code>weight_degenerate</code> abstention (SS8).
<code>keep_fits</code>	If TRUE, attach the contributing per-lens fits to the result (its <code>fits</code> field), so <code>koine_opinion_draws()</code> can materialise the stacking-mixture posterior without re-fitting. Default FALSE (the diagnostic-only contract is unchanged).

Value

A `koine_result` or a `koine_abstention`.

Examples

```
d <- data.frame(y = rnorm(30), x = rnorm(30))
pass <- koine_pass(y ~ x, d, gaussian())
# with a single-lens battery, koine abstains rather than guessing
res <- synthesise(pass, registry = list(koine_stage1_registry()[[1]]))
is_abstention(res)
```

`validate_fb_fit`
Validate an `fb_fit` against the field rules

Description

Enforces: a contributed fit carries draws + a complete provenance tuple (F4); exactness agrees between the fit and its provenance; an abstained fit carries an `koine_abstention` and no draws; `sample_stats/log_likelihood` are either NULL or arrays (no fabricated values, I1).

Usage

```
validate_fb_fit(x)
```

Arguments

`x` A koine_fb_fit.

Value

`x`, invisibly, if valid; otherwise stops.

Examples

```
set.seed(1)
draws <- posterior::as_draws_array(
  array(rnorm(8L), dim = c(2L, 2L, 2L),
    dimnames = list(NULL, NULL, c("b_(Intercept)", "b_x"))))
prov <- provenance_record(
  method = "demo", method_version = "0", impl_language = "R",
  seed = 1L, derived_seed = 1L, ir_hash = "h",
  maturity_tier = "reference-validated",
  independence_key = list(family = "random_walk", kernel = "demo"),
  role = "opinion", exactness = "exact")
validate_fb_fit(fb_fit(draws, prov, "exact", status = "contributed"))
```

validate_provenance	<i>Validate a provenance record's mandatory tuple</i>
---------------------	---

Description

Validate a provenance record's mandatory tuple

Usage

```
validate_provenance(x)
```

Arguments

`x` A koine_provenance object.

Value

`x`, invisibly, if every mandatory field is present and non-NULL; otherwise raises `error.provenance_incomplete`.

Examples

```
prov <- provenance_record(
  method = "dram", method_version = "0.1.0", impl_language = "R",
  seed = 1L, derived_seed = 42L, ir_hash = "abc123",
  maturity_tier = "reference-validated",
  independence_key = list(family = "random_walk", kernel = "dram"),
  role = "opinion", exactness = "exact")
validate_provenance(prov)
```

weight_concentration *Weight-concentration diagnostic (Stage-0 SS9)*

Description

Summarises how concentrated the stacking weights are. Concentration on one component means the battery added no predictive diversity (the lenses corroborate rather than diversify); spread weights mean genuine model-structure diversity was realised. Reports the effective number of components $n_{\text{eff}} = 1 / \sum(w^2)$, the maximum weight, the entropy, and a degenerate flag.

Usage

```
weight_concentration(weights, n_eff_threshold = 1.5)
```

Arguments

weights	A named numeric vector of stacking weights.
n_eff_threshold	Below this effective component count the weights are flagged degenerate. Default 1.5.

Value

A list with weights, max_weight, n_eff, entropy, and degenerate.

Examples

```
# spread weights: genuine diversity; concentrated weights: corroboration
weight_concentration(c(dram = 0.55, robust_t = 0.45))
weight_concentration(c(dram = 0.97, robust_t = 0.03))
```

Index

abstention, 3
abstention(), 10, 12–14, 16–18, 20, 39
as_family_spec, 4
as_family_spec(), 6, 7
assemble_loglik, 4
assemble_loglik(), 8, 10, 12, 22

evidence_axis, 5

family_spec, 6
family_spec(), 27
fb_fit, 8
fb_fit(), 10, 11, 13, 16–18, 20
fb_fit_diagnostics, 9
fit_dram, 10
fit_dream, 10
fit_glmm_marginal, 11
fit_glmm_robust_re, 13
fit_horseshoe, 14
fit_horseshoe(), 15, 19
fit_lasso, 15
fit_nimble, 16
fit_nimble_glmm, 17
fit_pg_gibbs, 18
fit_ridge, 19
fit_robust_t, 20

gate_corroboration, 21
gate_psis, 21

independence_report, 22
is_abstention, 23

koine_can_contribute, 23
koine_can_contribute(), 26
koine_lens_validation_setup, 24
koine_opinion_draws, 25
koine_opinion_draws(), 24, 29, 40
koine_pass, 27
koine_pass(), 24, 29
koine_pass_from_design, 28

koine_stage1_registry, 29
koine_stage1_registry(), 24, 37, 40

lens_effective_maturity, 30
lens_spec, 30
linear_pool_envelope, 32

provenance_record, 33

read_idata_nc, 35
reference_posterior_check, 35
reference_posterior_check(), 24
resolve_battery, 36

sbc_check, 37
sbc_check(), 24
stack_opinion, 39
synthesise, 39
synthesise(), 23, 24, 26

validate_fb_fit, 40
validate_provenance, 41

weight_concentration, 42