# Package: quorum (via r-universe)

November 27, 2026

**Title** The Orchestra's Consensus Layer: Lenses in, a Typed Verdict Out

**Version** 0.1.0.9000

**Description** The consensus layer of the Crop Analytics Orchestra. Registers any member's estimate of a shared estimand as a lens with declared independence, expressiveness and scope; compares a quorum of lenses with the hub's agreement semantics; resolves the comparison through a declared policy into a typed verdict and an orchestra manifest. A quorum of independent lenses must agree before an answer is carried, and no quorum means abstain. Carries the on-ramps by which non-hub estimates become lenses: ensemble-source ingestion, surrogate emulators of expensive fits, ensemble-derived priors, posterior predictive checks for surrogates, and the koine fourth-opinion backend. Depends on 'flexyBayes' for the core model representation and emits results in the same shapes.

**License** MIT + file LICENSE

**URL** https://github.com/max578/quorum, https://max578.github.io/quorum/

**BugReports** https://github.com/max578/quorum/issues

**Depends** R (>= 4.1.0)

**Imports** flexyBayes, orchestraManifest, S7, stats, utils

**Suggests** ggplot2, kernR, PESTO, glmmTMB (>= 1.1.9), tmbstan, rstan, INLA, sn, gretaR (>= 0.4.0), koine (>= 0.3.0), sommer, posterior, knitr, rmarkdown, testthat (>= 3.0.0), withr

**Remotes** max578/flexyBayes, max578/orchestraManifest, max578/kernR, max578/PESTO, max578/koine, max578/gretaR

**Additional_repositories** https://max578.r-universe.dev, https://inla.r-inla-download.org/R/stable

**Encoding** UTF-8

**Language** en-AU

**Roxygen** list(markdown = TRUE, roclets = c(``rd", ``namespace", ``collate"))

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**Config/roxygen2/version** 8.1.0

**Repository** https://max578.r-universe.dev

**Date/Publication** 2026-10-17 11:15:29 UTC

**RemoteUrl** https://github.com/max578/quorum

**RemoteRef** main

**RemoteSha** 4cd2711aebb8779d342bee4516c9c601586f2db5

# Contents

agreement_matrix . . . 3
agreement_pairwise . . . 4
agreement_status . . . 5
agreement_thresholds . . . 6
as_fb_ensemble . . . 8
canonical_names.flexybayes_koine . . . 9
coef.flexybayes_koine . . . 9
contributions . . . 10
emit_koine . . . 10
fb_as_draws_simple.flexybayes_koine . . . 12
fb_cov_slice . . . 13
fb_ensemble . . . 13
fb_pesto . . . 14
fb_prior_from_ensemble . . . 15
fb_surrogate_ies . . . 16
fb_surrogate_ppc . . . 18
fit_surrogate . . . 21
koine_genomic_lens . . . 23
koine_status . . . 24
lens_contribution . . . 25
lens_from_draws . . . 26
lens_from_fit . . . 27
lens_koine . . . 28
lens_mechanism . . . 29
lens_status_quo . . . 31
lens_tmb . . . 32
predict.fb_surrogate . . . 34
print.flexybayes_koine . . . 35
print.quorum::quorum_verdict . . . 36
print.quorum_independence . . . 36
quorum . . . 37
quorum_abstention . . . 38
quorum_compare_policies . . . 39

quorum_draws . . . . . 40
quorum_independence . . . . . 41
quorum_lens . . . . . 42
quorum_manifest . . . . . 43
quorum_policy . . . . . 44
quorum_policy_committed . . . . . 45
quorum_resolve . . . . . 46
quorum_verdict . . . . . 47
read_fb_ensemble . . . . . 48
register_ensemble_source . . . . . 49
register_kernr_surrogates . . . . . 49
register_lens . . . . . 50
register_pesto_surrogates . . . . . 52
register_surrogate . . . . . 52
verify_fb_ensemble . . . . . 53

**Index** **54**

---

`agreement_matrix` *Pairwise agreement across a quorum of lenses*

---

## Description

Run `agreement_pairwise()` over every unordered pair of a quorum's lenses and collect the results: a square matrix of pairwise statuses, and the long table of every component comparison behind it.

## Usage

```
agreement_matrix(draws, thresholds = agreement_thresholds())
```

## Arguments

| | |
|---|---|
| `draws` | A named list of `quorum_draws()` objects of kind `"parameters"`, or of named lists of numeric vectors, one per lens. Names are the lens names and label the matrix and the table. |
| `thresholds` | A list from `agreement_thresholds()`. |

## Details

A pair that shares no component is not a disagreement. Its status is `"inconclusive"` and it enters the long table as a single row whose reason is `no_common_parameters`, so the matrix and the table always tell the same story and no pair goes missing from either.

The matrix is symmetric and its diagonal is `NA`: a lens is not compared with itself, and a diagonal of `"concordant"` would inflate any count taken over the matrix.

## Value

A list of class `quorum_agreement` with elements `status` (an `N` by `N` character matrix of pairwise statuses, diagonal `NA`), `table` (the long data.frame: `lens_a`, `lens_b`, then the columns of `agreement_pairwise()`) and `thresholds` (the cut-offs used); or a `quorum_abstention()` of scope `"comparison"` when fewer than two lenses were given.

## See Also

`agreement_pairwise()`, `agreement_status()`.

## Examples

```
set.seed(2)
lenses <- list(
  inla = list(mu = rnorm(400L)),
  brms = list(mu = rnorm(400L)),
  koine = list(mu = rnorm(400L, 3))
)
agreement_matrix(lenses)$status
```

---

# agreement_pairwise *Per-component agreement between two lenses*

---

## Description

Compare two lenses' parameter draws component by component, on the hub's terms. For every component the two lenses share, the table reports each lens's posterior mean and SD, the ratio of the SDs, the drift at the two reported quantiles, the Wasserstein-1 distance between the two marginals, the mean shift and the Wasserstein-1 distance expressed in reference-SD units, and the verdict those three scaled quantities imply.

## Usage

```
agreement_pairwise(
  a,
  b,
  names = NULL,
  thresholds = agreement_thresholds(),
  sd_ref = c("max", "pooled", "a", "b")
)
```

## Arguments

| | |
|---|---|
| a, b | A `quorum_draws()` of kind `"parameters"`, or a named list of numeric vectors in that shape. |
| names | Optional character vector naming the components to compare. The default compares every component the two lenses share, in `a`'s order. Every name asked for must be present on both sides. |

| | |
|---|---|
| thresholds | A list from agreement_thresholds(). |
| sd_ref | The reference SD the mean shift and the Wasserstein-1 distance are scaled by: "max" (the hub's larger-of-the-two, the default), "pooled" (the root-mean-square of the two), "a" or "b". |

## Details

The reference SD is the larger of the two posterior SDs, which is the scale the hub uses (.triangulate_verdicts, flexyBayes 0.10.0 R/triangulate.R:809: s_ref <- max(metrics$sd_a[[i]], metrics$sd_b[[i]], na.rm = TRUE)), and it is the default here so that a quorum comparison returns the same numbers the hub's comparison returned on the same draws. sd_ref offers the pooled and the one-sided scales for callers who want the comparison read against a fixed lens instead, at the cost of leaving the hub's scale.

A component with no posterior spread on either side cannot be scaled against anything, so it is reported with its raw distances, a not_compared verdict and the hub's reason string rather than a verdict the scale cannot support.

Agreement is not validation. Two lenses that share a sampler, an objective, a prior or a data-preparation step can agree because they share it; the independence ledger, not this table, is what says whether their agreement carries evidential weight.

## Value

A data.frame with one row per compared component and the columns param, mean_a, mean_b, sd_a, sd_b, sd_ratio, q025_diff, q975_diff, wasserstein_1, mean_shift_sd, w1_sd, verdict and reason; or a quorum_abstention() of scope "comparison" when the two lenses share no component, when a requested component is absent, or when either input is a response-curve rather than a parameter lens.

## See Also

agreement_status() for the overall verdict, agreement_matrix() for a quorum of more than two lenses.

## Examples

```
set.seed(1)
a <- list(intercept = rnorm(500L, 1, 0.5), slope = rnorm(500L, 2, 0.3))
b <- list(intercept = rnorm(500L, 1, 0.5), slope = rnorm(500L, 2.9, 0.3))
agreement_pairwise(a, b)[, c("param", "w1_sd", "verdict")]
```

---

# agreement_status *The overall status of an agreement table*

---

## Description

Reduce a pairwise agreement table to one of the hub's three statuses. A table with at least one discordant component is "discordant"; a table whose compared components are all concordant is "concordant"; a table in which nothing was compared – every row not_compared, or no rows at all – is "inconclusive".

## Usage

```
agreement_status(table, thresholds = agreement_thresholds())
```

## Arguments

| | |
|---|---|
| table | A data.frame from agreement_pairwise(), or a quorum_abstention(), which is returned unchanged so a declined comparison propagates instead of being scored. |
| thresholds | A list from agreement_thresholds(). The status rule reads verdicts only – the cut-offs were already applied, per component, when the table was built – and the argument is validated and carried so one thresholds object passes unchanged through every verb of this layer. |

## Details

This is the verdict-derived half of the hub's rule (.triangulate_status, flexyBayes 0.10.0 R/triangulate.R:910-927). The other half of that rule turns a failed comparability check or a fit's own failed diagnostics into "inconclusive" before the verdicts are read at all. Those two gates take a model fingerprint and a sampler's diagnostics, which a lens does not carry, so in quorum they sit with the lens contract and the policy rather than here: a caller that has gate findings of its own reports "inconclusive" without consulting this function.

## Value

A single string, one of "concordant", "discordant" or "inconclusive"; or the abstention it was given.

## See Also

agreement_pairwise().

## Examples

```
tbl <- data.frame(
  param = c("a", "b"),
  verdict = c("concordant", "discordant"),
  stringsAsFactors = FALSE
)
agreement_status(tbl)
```

---

# agreement_thresholds *The hub's agreement cut-offs*

---

## Description

The cut-offs the hub's cross-engine comparison applied when it called a parameter concordant, with the hub's own defaults. That comparison, `triangulate()`, was removed from flexyBayes in version 0.14.0; the source lines cited below are from flexyBayes 0.10.0. A parameter is concordant when the absolute mean difference is at most `mean_shift_sd_max` reference SDs, the Wasserstein-1 distance is at most `w1_sd_max` reference SDs, and the ratio of the two posterior SDs lies inside `sd_ratio_band`; any breach makes it discordant.

## Usage

```
agreement_thresholds(
  mean_shift_sd_max = 0.1,
  w1_sd_max = 0.1,
  sd_ratio_band = c(0.9, 1.1),
  quantile_probs = c(0.025, 0.975)
)
```

## Arguments

`mean_shift_sd_max`
: Single positive number: the largest absolute difference in posterior means, in reference-SD units, a concordant parameter may show. Hub default 0.1, from `.FB_TRIANGULATE_MEAN_SHIFT_MAX` at `flexyBayes` 0.10.0 `R/triangulate.R:757`.

`w1_sd_max`
: Single positive number: the largest Wasserstein-1 distance, in reference-SD units, a concordant parameter may show. Hub default 0.1, from `.FB_TRIANGULATE_W1_MAX` at `R/triangulate.R:758`.

`sd_ratio_band`
: Numeric vector of length two, increasing: the band the ratio of the two posterior SDs must lie in. Hub default `c(0.9, 1.1)`, from `.FB_TRIANGULATE_SD_RATIO_BAND` at `R/triangulate.R:759`.

`quantile_probs`
: Numeric vector of length two, increasing, inside `(0, 1)`: the quantiles whose difference is reported. Hub default `c(0.025, 0.975)`, from the `stats::quantile()` calls in `.triangulate_one` at `R/triangulate.R:495` and `R/triangulate.R:497`.

## Details

The cut-offs are heuristics in the sense the Bayesian-workflow literature uses for cross-implementation checks: they are set to flag a difference a reader should look at, they are not derived from a sampling distribution, and they carry no error rate. The hub documents them that way and quorum inherits both the numbers and the reading.

`quantile_probs` names the two quantiles whose difference the table reports as `q025_diff` and `q975_diff`. The hub reports that tail drift without ever thresholding it, so there is no quantile tolerance in the verdict rule; the argument fixes which quantiles are reported, not a cut-off they must clear.

## Value

A named list with the four elements above.

## See Also

agreement_pairwise(), agreement_status().

## Examples

```
agreement_thresholds()
agreement_thresholds(w1_sd_max = 0.05)
```

---

# as_fb_ensemble *Coerce a conforming producer to a canonical* fb_ensemble

---

## Description

The single ingest entry point for contract C5. Dispatches on the producer: an fb_ensemble is returned unchanged; a pesto_ensemble_manifest (PESTO's S7 calibration manifest) is mapped via the C2 -> C5 bridge; a list carrying params / outputs is wrapped; a length-1 character path is read from disk. Any other class is looked up in the producer registry (register_ensemble_source()) before erroring.

## Usage

```
as_fb_ensemble(x, ...)
```

## Arguments

| | |
|---|---|
| x | A conforming producer object (see Details / the contract). |
| ... | Passed to the matched coercer. |

## Value

An fb_ensemble.

## See Also

inst/ensemble_source_contract.md.

## Examples

```
# an fb_ensemble round-trips unchanged
ens <- fb_ensemble(
  params = data.frame(p1 = runif(10)),
  outputs = data.frame(o1 = runif(10)),
  provenance = list(source = "raw")
)
identical(as_fb_ensemble(ens), ens)

# a bare list carrying params / outputs is wrapped the same way
as_fb_ensemble(list(params = data.frame(p1 = runif(5)),
                    outputs = data.frame(o1 = runif(5))))
```

## canonical_names.flexybayes_koine *Canonical names for a koine fit (self-naming)*

### Description

S3 method on flexyBayes's exported `canonical_names()` generic. koine is self-naming – `emit_koine()` canonicalises the draws – so no relabel map is needed; a comparison aligns on the already-canonical names.

### Usage

```
## S3 method for class 'flexybayes_koine'
canonical_names(fit, drop = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `fit` | A `flexybayes_koine` object. |
| `drop` | Unused; kept for generic signature compatibility. |
| `...` | Unused. |

### Value

A list with an empty `map`, empty `transform`, and `source = "self_naming"`.

## coef.flexybayes_koine *Posterior-mean coefficients for a koine fit*

### Description

Posterior-mean coefficients for a koine fit

### Usage

```
## S3 method for class 'flexybayes_koine'
coef(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | A `flexybayes_koine` object. |
| `...` | Unused. |

### Value

A named numeric vector of koine's synthesised posterior means.

## contributions *Bind lens contributions into one table*

### Description

Row-binds the rows produced by `lens_contribution()` into the contributions table a resolution reads. Parts may carry extra columns beyond the seven of the seam – a `cell` identifier when the table spans many decision cells, or the per-cell facts a comparison policy needs – and columns absent from one part are filled with `NA` rather than dropped, so lenses that declare different extras still bind.

### Usage

```
contributions(...)
```

### Arguments

| | |
|---|---|
| `...` | Data frames of contributions, each with at least the columns `lens_contribution()` produces. |

### Value

One data frame holding every row, with the union of the columns.

### See Also

`lens_contribution()`, `quorum_resolve()`.

### Examples

```
contributions(
  lens_contribution("mechanism", TRUE, FALSE, action = 120),
  lens_contribution("statistical", TRUE, TRUE, reason = "insufficient")
)
```

## emit_koine *Emit a koine fourth-opinion fit*

### Description

Activates the koine lens-axis backend from the companion. Given a flexyBayes `fb_terms` intermediate representation (the `$fb` element of any core fit) and the model frame, `emit_koine()` synthesises koine's fourth opinion and returns a `flexybayes_koine` object. `lens_from_fit()` turns it into a lens that `agreement_pairwise()` and `quorum()` compare with an engine-axis fit (greta / INLA / brms / gretaR).

## Usage

```
emit_koine(
  fb,
  data,
  n_samples = 4000L,
  seed = 1L,
  the_call = NULL,
  standardise = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| fb | An fb_terms object (the $fb element of a core flexyBayes fit). |
| data | The model frame the fit was built on. |
| n_samples | Integer count of opinion draws to synthesise. |
| seed | Integer RNG seed, pinned for reproducibility regardless of caller RNG state. |
| the_call | Optional originating call, carried into the fit's extras. |
| standardise | Logical. koine's random-effects lenses carry a fixed N(0, 5) prior on every coefficient and lognormal(0, 2) priors on the residual and group SDs, which the pass-level prior does not reach. On a response measured in the hundreds that prior dominates the opinion: the intercept is pulled toward zero and the group SD absorbs the offset. With standardise = TRUE (the default) the Gaussian response is centred and scaled and every non-intercept covariate is scaled to unit SD before koine fits, and the draws are mapped back to the original scale afterwards, so the same prior becomes a scale-aware weakly-informative one; the fit declares prior = "koine_scaled" in its extras. With FALSE koine fits the data as given and declares "koine_default". |
| ... | Reserved for future use. |

## Details

koine's corner is a population GLM for the gaussian, binomial, and poisson families, with either a single random intercept (1 \| g) or – from koine 0.3.0 – the breeder-standard multi-environment-trial structure of two crossed or nested random intercepts, (1 \| variety) + (1 \| trial). A single grouping is routed through koine_pass_from_design(group = ); two groupings are routed through koine_pass_from_design(groups = ), koine's crossed / nested entry, and whether they are crossed or nested is resolved by koine from the data. Outside that corner – more than two groupings, a random slope, a correlated random structure, koine below 0.3.0 on a two-grouping model – or whenever koine's own convergence gate fires, the returned object carries status = "abstained", the typed fourth column. The synthesised opinion is a stacking mixture over koine's battery, an independent witness on the orthogonal lens axis, never a pool across the engine-axis backends.

Activation is gated by koine_status(): koine installed at or above 0.2.0 and options(quorum.koine_activated = TRUE) (the default). The draws are canonicalised to flexyBayes's parameter tokens, so the fit is self-naming and needs no name_map when it is compared: the group SDs koine reports as tau / tau_<grouping> become sd_<grouping>.

## Value

A `flexybayes_koine` object: a list with `status` (`"contributed"` or `"abstained"`), the canonicalised `draws`, the koine `koine` result, the `fb` IR, and an `extras` record (stacking weights, stacked ELPD, independence and diagnostics on the contributed path; the refusal reason and detail on the abstained path). On the abstained path the object's class additionally carries `"quorum_abstention"` and `"orchestra_refusal"` prepended ahead of `"flexybayes_koine"`, so a producer-agnostic decline predicate (any class ending in `_refusal` / `_abstention`, or the `orchestra_refusal` marker) recognises the outer object without inspecting `$koine`; `inherits(x, "flexybayes_koine")` and every `*.flexybayes_koine` method still dispatch unchanged.

## See Also

`koine_status()` for the activation state; `lens_from_fit()` and `agreement_pairwise()` for comparing the result with another fit.

---

# fb_as_draws_simple.flexybayes_koine

*Extract simple draws from a koine fit*

---

## Description

S3 method on flexyBayes's exported `fb_as_draws_simple()` generic. Returns a named list of numeric draw vectors over the canonical parameters, the shape `lens_from_fit()` reads.

## Usage

```
## S3 method for class 'flexybayes_koine'
fb_as_draws_simple(fit, n_samples = 1000L, ...)
```

## Arguments

| | |
|---|---|
| `fit` | A `flexybayes_koine` object. |
| `n_samples` | Unused (koine draws are pre-synthesised); kept for generic signature compatibility. |
| `...` | Unused. |

## Value

A named list of numeric vectors, one per canonical parameter.

## fb_cov_slice *Reconstruct a per-row covariance slice from a surrogate prediction*

### Description

Returns the k x k cross-output predictive covariance at one row of the newdata passed to predict.fb_surrogate(), from either the dense (cov = TRUE) or the compact factored (cov = "lowrank") form – so a consumer can work slice-by-slice without materialising the full dense array for a large output count.

### Usage

```
fb_cov_slice(prediction, i = 1L)
```

### Arguments

| | |
|---|---|
| prediction | An fb_surrogate_prediction from predict(<fb_surrogate>, newdata, cov = TRUE) or cov = "lowrank". |
| i | Integer row index (a row of newdata; default 1). |

### Value

The k x k cross-output predictive covariance at row i, with output names on both dimensions.

### See Also

predict.fb_surrogate().

## fb_ensemble *Construct a canonical simulation-ensemble object*

### Description

Builds an fb_ensemble – quorum's canonical container for a parameter-to-output simulation ensemble (contract C5, inst/ensemble_source_contract.md). params and outputs are the (X, Y) an emulator or surrogate trains on; output_dims and observed add the calibration framing when present. quorum never executes the simulator (APSIM is run upstream by apsimx); this is the ingest floor the methods-intense consumers share.

### Usage

```
fb_ensemble(
  params,
  outputs,
  observed = NULL,
  output_dims = NULL,
  provenance = list()
)
```

### Arguments

| | |
|---|---|
| params | A data.frame, one row per parameter set, columns the simulator's input parameters. Row order is the ensemble key. |
| outputs | A data.frame row-aligned to params (nrow(outputs) == nrow(params)), columns the output dimensions. |
| observed | Optional named numeric of target observations; names a subset of colnames(outputs). NULL for a pure forward ensemble. |
| output_dims | Optional data.frame, one row per outputs column, labelling it (e.g. site, season, day, variable). NULL when outputs are unlabelled. |
| provenance | A named list of provenance fields; source is filled with "raw" when absent. |

### Value

An object of class c("fb_ensemble", "list").

### See Also

as_fb_ensemble(), verify_fb_ensemble(), read_fb_ensemble().

### Examples

```
e <- fb_ensemble(
  params = data.frame(k = c(0.1, 0.2, 0.3)),
  outputs = data.frame(yield = c(4.1, 4.6, 5.0))
)
verify_fb_ensemble(e)$ok
```

---

## fb_pesto *Carry a PESTO calibration into flexyBayes*

---

### Description

Ingests a calibrated simulation ensemble (a PESTO pesto_ensemble_manifest, or any C5-conforming source via as_fb_ensemble()) and returns the flexyBayes-ready artefacts for the "generalise" step of the calibration -> validation -> generalisation pipeline: the canonical fb_ensemble(), an informative prior derived from the calibrated parameters (fb_prior_from_ensemble()), and the per-parameter moment summary. The resulting $prior is passed straight to flexybayes(..., prior = ).

### Usage

```
fb_pesto(x, method = c("moments", "pesto", "kde"), parameters = NULL)
```

### Arguments

| | |
|---|---|
| x | A C5 ensemble source (PESTO manifest, fb_ensemble, list, or path). |
| method | Prior-construction method passed to fb_prior_from_ensemble(); "moments" (default), "pesto", "kde". |
| parameters | Optional subset of parameter columns for the prior. |

### Details

quorum never executes the simulator (APSIM is run upstream by apsimx); this verb consumes the calibration result through the contract.

### Value

An object of class c("fb_pesto", "list") with components ensemble (the fb_ensemble()), prior (the derived fb_prior()), moments (a data.frame of parameter, mean, sd), method, and provenance.

### See Also

fb_prior_from_ensemble(), as_fb_ensemble().

### Examples

```
e <- fb_ensemble(
  params = data.frame(k = stats::rnorm(50, 2, 0.3)),
  outputs = data.frame(yield = stats::rnorm(50))
)
res <- fb_pesto(e)
res$prior
```

---

## fb_prior_from_ensemble

*Derive an informative flexyBayes prior from a simulation ensemble*

---

### Description

Constructs an fb_prior from a calibrated parameter ensemble – the "generalise" step that carries a process-model calibration forward into a flexyBayes fit. The ensemble is ingested through the C5 contract (as_fb_ensemble()), so any conforming producer works (a PESTO manifest, a raw apsimx ensemble, a file).

### Usage

```
fb_prior_from_ensemble(
  ensemble,
  method = c("moments", "pesto", "kde"),
  parameters = NULL
)
```

### Arguments

| | |
|---|---|
| ensemble | A C5 ensemble source: an fb_ensemble, a PESTO pesto_ensemble_manifest, a list(params, outputs, ...), or a file path. |
| method | One of "moments" (default), "pesto", "kde". |
| parameters | Optional character vector selecting a subset of the ensemble's parameter columns. NULL (default) uses all of them. |

### Details

The "moments" method (the default, self-contained) fits an independent normal(mean, sd) prior to each parameter from its ensemble marginal mean and standard deviation, emitted as b("<param>") ~ normal(mean, sd) fixed-effect priors. **Cross-parameter correlation is discarded** – the moment-only path keeps the marginals, not the joint. The richer joint method ("pesto", the PESTO-native prior preserving covariance + positive-orthant structure) and the kernel-density method ("kde", via kernR) are specified but not yet wired; they raise a structured error naming their gate and the available fallback.

### Value

An fb_prior object (see fb_prior()), ready to pass to flexybayes(..., prior = ).

### See Also

fb_pesto(), as_fb_ensemble(), fb_prior().

### Examples

```
e <- fb_ensemble(
  params = data.frame(k = stats::rnorm(50, 2, 0.3),
                      r = stats::rnorm(50, 0.5, 0.1)),
  outputs = data.frame(yield = stats::rnorm(50))
)
p <- fb_prior_from_ensemble(e)
p
```

---

## fb_surrogate_ies *Surrogate-accelerated ensemble inversion (ES-MDA)*

---

### Description

Calibrates a parameter ensemble against observed data using a distribution-preserving surrogate (contract C7) in place of the forward model. Runs Ensemble Smoother with Multiple Data Assimilation (ES-MDA; Emerick and Reynolds, 2013): each of n_iter iterations applies a Kalman update with the data covariance inflated by n_iter, so the combined update equals a single tempered assimilation. No simulator is run – every forward evaluation is a surrogate predict().

## Usage

```
fb_surrogate_ies(
  surrogate,
  observed,
  obs_sd,
  prior,
  n_iter = 4L,
  use_surrogate_sd = TRUE,
  use_surrogate_cov = TRUE,
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| surrogate | An fb_surrogate from fit_surrogate(). |
| observed | A named numeric vector of target observations; names must be a subset of the surrogate's emulated outputs. |
| obs_sd | Observation-error standard deviation: a single number (shared) or a named numeric vector aligned with observed. |
| prior | A data.frame (or matrix) of the prior parameter ensemble – one row per realisation; must contain the surrogate's parameter columns. |
| n_iter | Number of ES-MDA iterations (equal inflation alpha = n_iter). |
| use_surrogate_sd | Add the surrogate's mean predictive uncertainty to the data covariance (default TRUE). |
| use_surrogate_cov | When the surrogate is joint, fold its full cross-output predictive covariance (not just the per-output variance) into the data covariance (default TRUE). Ignored when use_surrogate_sd is FALSE or the surrogate reports only marginals (the covariance is then diagonal and the two paths coincide). |
| seed | Optional integer RNG seed for the observation perturbations (reproducibility); the caller's RNG stream is restored on exit. |

## Details

When use_surrogate_sd = TRUE (default) the surrogate's predictive uncertainty is added to the observation-error covariance, so emulator uncertainty inflates the assimilation noise – the inversion trusts the surrogate less where it is less certain. This is the consumer that the distribution-preserving surrogate exists to serve. When the surrogate is joint (its predict(cov = TRUE) yields a cross-output covariance) and use_surrogate_cov = TRUE (default), the **full** mean predictive covariance enters the data covariance, so correlated emulator errors are assimilated coherently rather than as independent per-output noise; otherwise only the per-output predictive variance (the diagonal) is used.

## Value

An object of class `c("fb_surrogate_ies", "list")` with components `posterior` (the updated ensemble `data.frame`), `prior`, `observed`, `misfit` (mean data-misfit before each iteration and after the last), `n_iter`, `use_surrogate_sd`, `surrogate_method`, `param_names`, and `output_names`.

## References

Emerick, A. A. and Reynolds, A. C. (2013). Ensemble smoother with multiple data assimilation. *Computers & Geosciences*, 55, 3-15.

## See Also

`fit_surrogate()`, `fb_surrogate_ppc()`.

## Examples

```
set.seed(1)
# forward model y = G theta; build a training ensemble and an emulator
g <- c(1.5, -0.8)
tr <- data.frame(p1 = runif(150, -1, 3), p2 = runif(150, -2, 2))
e <- fb_ensemble(
  params = tr,
  outputs = data.frame(y = g[1] * tr$p1 + g[2] * tr$p2)
)
s <- fit_surrogate(e)
prior <- data.frame(p1 = rnorm(200, 1, 1), p2 = rnorm(200, 0, 1))
fit <- fb_surrogate_ies(s, c(y = 2.3), obs_sd = 0.1, prior = prior)
colMeans(fit$posterior)
```

---

# fb_surrogate_ppc *Posterior-predictive check of a surrogate against held-out outputs*

---

## Description

Tests whether a surrogate's predictive *distribution* matches the simulator's at held-out parameter sets, using kernR's maximum-mean-discrepancy posterior-predictive check (`kernR::mmd_ppc()`). The surrogate's predictive distribution at `newdata` is sampled into a predictive ensemble, which is compared against `observed` – the true simulator outputs at those parameter sets. A rejection means the emulator distorted the output distribution (typically by being over- or under-dispersed, or by getting the cross-output correlation wrong), not merely that its mean is off.

## Usage

```
fb_surrogate_ppc(
  surrogate,
  newdata,
  observed,
  n_draws = 1L,
```

```
  joint = TRUE,
  project = FALSE,
  whiten = FALSE,
  aggregate = FALSE,
  agg_weights = "uniform",
  seed = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| surrogate | An fb_surrogate from fit_surrogate(). |
| newdata | A data.frame of held-out parameter sets (must contain the surrogate's parameter columns). |
| observed | A matrix or data.frame of the true simulator outputs at held-out parameter sets, with columns named for the emulated outputs. |
| n_draws | Predictive draws per row of newdata (default 1); the predictive ensemble has n_draws * nrow(newdata) rows. |
| joint | Logical; sample from the joint cross-output covariance (default TRUE) rather than independently per output. Has no effect for a single output, or for a surrogate that reports only marginals (its covariance is diagonal). When the surrogate stores a low-rank covariance (cov_rank), the joint draws are sampled from the factored form directly, so the k x k slice is never formed (the native low-rank PPC consumer). |
| project | Logical; for a surrogate that stores a low-rank covariance (cov_rank), project both the predictive draws and observed onto the surrogate's q factor directions before the MMD (default FALSE). It then runs in q dimensions, not k – cheaper and higher-powered on the correlated cross-output structure for a large output count (the MMD loses power as the dimension grows). It is, deliberately, **blind to distortions orthogonal to the factor space** (purely idiosyncratic per-output errors, which the marginal <o>_sd and a joint = FALSE check cover), so it complements rather than replaces the full-space check. |
| whiten | Logical; Mahalanobis-whiten both the predictive draws and observed by their **pooled covariance** before the MMD (default FALSE). This puts the outputs on a common footing, removing the cross-output scale heterogeneity that makes a single median bandwidth too coarse to resolve a distortion living in a low-variance direction. The map depends only on the pooled cloud (not the draws-vs-observed labels), so the permutation test stays valid; the same map is applied to both clouds, so a relative scale, shape, or mean difference between them is preserved. Empirically it lifts power substantially on localised / heterogeneous-scale distortions a raw-output check misses, with no blind spot found across a mean / variance / shape battery. A graduated relative ridge guards a near-singular pooled covariance (highly collinear outputs); it remains opt-in for that numerical reason. Composes with project (the projected coordinates are whitened) and aggregate. |
| aggregate | Logical; run an **aggregated multi-kernel** MMD instead of a single-kernel (default FALSE). An MMDAgg test (Schrab et al. 2023) over an RBF bandwidth |

| | |
|---|---|
| | grid around the median heuristic, sharing the permutations across bandwidths and aggregating by the weighted-quantile statistic `min_g p_g / w_g` – it removes the bandwidth choice and is robust when the median heuristic is misscaled, at a valid level. Composes with `project`. Reads `n_permutations` and `alpha` from `...`. |
| `agg_weights` | Kernel weights for `aggregate = TRUE`: a scheme (`"uniform"` default, or `"increasing"` / `"decreasing"` / `"centred"`, ordered smallest-to-largest bandwidth) or a numeric vector over the bandwidth grid. A larger weight loosens that kernel's threshold, so up-weighting the scale a distortion is expected to live at recovers power; `"uniform"` is the safe default (it recovers the min p-value aggregation). |
| `seed` | Optional integer RNG seed (the caller's stream is restored). |
| `...` | Passed to `kernR::mmd_ppc()` (e.g. `n_permutations`); for `aggregate = TRUE`, `n_permutations` and `alpha` are read from here. |

## Details

This is the distribution-preserving counterpart to a mean-only accuracy check: it exercises the predictive uncertainty it reports, not just the mean. With `joint = TRUE` (default) the draws are sampled from the **full cross-output predictive covariance** (`predict.fb_surrogate()` with `cov = TRUE`), so the emulator's cross-output correlation is tested; this is the second consumer of the joint covariance after `fb_surrogate_ies()`. With `joint = FALSE` each output is sampled independently from its `<o>_sd`. On a simulator whose outputs are correlated, marginal draws discard that correlation, so the joint path is the faithful check (a marginals-only surrogate yields a diagonal covariance, and the two paths then coincide).

## Value

For the single-kernel, the `kernR::mmd_ppc` object (`$statistic`, `$p_value`, `$reject`, `$surprise_bits`); for `aggregate = TRUE` an `fb_mmd_agg` object (`$p_value`, `$reject`, `$statistic`, `$bandwidths`, `$p_per_bandwidth`).

## See Also

`fit_surrogate()`, `predict.fb_surrogate()`, `fb_surrogate_ies()`.

## Examples

```
g <- rbind(c(1.5, -0.8), c(0.4, 1.1))
theta <- cbind(p1 = runif(80, -2, 2), p2 = runif(80, -2, 2))
y <- theta %*% t(g) + matrix(rnorm(160, 0, 0.1), 80, 2L)
colnames(y) <- c("o1", "o2")
ens <- fb_ensemble(as.data.frame(theta), as.data.frame(y),
                   provenance = list(source = "raw"))
sur <- fit_surrogate(ens)

held <- data.frame(p1 = runif(30, -2, 2), p2 = runif(30, -2, 2))
truth_held <- held$p1 %o% g[, 1] + held$p2 %o% g[, 2] +
  matrix(rnorm(60, 0, 0.1), 30, 2L)
colnames(truth_held) <- c("o1", "o2")
check <- fb_surrogate_ppc(sur, held, as.data.frame(truth_held),
```

```
                          n_draws = 3L, seed = 1L, n_permutations = 99L)
check$p_value
```

---

`fit_surrogate` *Fit a distribution-preserving surrogate to a simulation ensemble*

---

## Description

Trains an emulator mapping the ensemble's parameter sets (`ensemble$params`, the design `X`) to its outputs (`ensemble$outputs`, the responses `Y`), so an expensive forward model can be replaced by a fast surrogate that still reports predictive **uncertainty** (mean + sd per output) – the property a calibration / inverse step needs preserved (contract C7, `inst/surrogate_contract.md`). The ensemble is ingested through C5 (`as_fb_ensemble()`), so any conforming producer works.

## Usage

```
fit_surrogate(
  ensemble,
  method = NULL,
  outputs = NULL,
  n_features = 256L,
  ard = TRUE,
  ard_refit = FALSE,
  joint = TRUE,
  cov_rank = NULL,
  cov_method = "eigen",
  seed = 1L,
  ...
)
```

## Arguments

| | |
|---|---|
| `ensemble` | A C5 ensemble source (an `fb_ensemble`, a PESTO manifest, a `list(params, outputs)`, or a path). |
| `method` | One of the registered methods; `"rff"` (default) is always available. See `register_surrogate()`. |
| `outputs` | Optional character vector selecting output columns to emulate; `NULL` (default) emulates all of them. |
| `n_features` | Number of random Fourier features for `method = "rff"`. |
| `ard` | Logical; for `method = "rff"`, learn per-dimension (automatic relevance determination) lengthscales by maximising the model evidence (default `TRUE`). `FALSE` uses a single isotropic median-heuristic lengthscale. Vacuous with one parameter. |

`ard_refit` When `ard = TRUE`, controls the relevance objective. `FALSE` (default) holds the prior/noise precisions `(a, t)` at their isotropic-fit values. `TRUE` re-estimates `(a, t)` by evidence at each candidate lengthscale (the evidence *profiled* over `(a, t)`) – a fuller, never-lower marginal likelihood, but in-sample, so it can over-shorten lengthscales. `"cv"` instead maximises the **held-out** predictive log-density over a k-fold split, the regularised fuller-ARD: it keeps the relevance sharpening without the in-sample over-shortening.

`joint` Logical; for `method = "rff"` with two or more outputs, fit a joint multi-output model preserving the cross-output predictive covariance (default `TRUE`). `FALSE` fits one independent model per output (marginals only). The per-output `<o>_sd` contract is unchanged either way; only the off-diagonal covariance (exposed by `predict(..., cov = TRUE)`) differs.

`cov_rank` Optional integer; for a joint `rff` fit with `k` outputs, store the cross-output covariance in a rank-`cov_rank` factor-analysis form (`U U' + diag`) instead of the dense `k x k`, for large output counts. The per-output marginal sd and the covariance diagonal are reproduced exactly; only the off-diagonal covariance is approximated. `NULL` (default) or `cov_rank >= k` keeps the full covariance. See `predict(..., cov = "lowrank")` for the compact predictive return.

`cov_method` How the rank-`cov_rank` factor is fitted (only when `cov_rank` is set): `"eigen"` (default) truncates the covariance's eigendecomposition; `"fa"` learns the loadings by factor-analysis EM, which recovers them more faithfully when the residual diagonal is heteroscedastic (the eigenvectors of the full covariance are otherwise contaminated by the uneven diagonal); `"auto"` picks `"fa"` or `"eigen"` from the heteroscedasticity of the eigen-residual diagonal. The covariance diagonal is reproduced exactly in every case. The resolved method is recorded in the fitted object.

`seed` Integer RNG seed for the random features (reproducibility).

`...` Passed to a registered method's fit function.

## Details

The built-in `method = "rff"` (default) is a self-contained random-Fourier- features Bayesian linear surrogate with evidence-optimised predictive variance. By default it uses **ARD** lengthscales (per-input relevance learned from the evidence) and a **joint** multi-output fit that preserves the predictive covariance across outputs (not only the marginals) – the distribution-preserving property the calibration / inverse consumers rely on. Richer methods (kernR's CME, PESTO's GP) register via `register_surrogate()`.

## Value

An object of class `c("fb_surrogate", "list")` with a `predict.fb_surrogate()` method.

## See Also

`predict.fb_surrogate()`, `register_surrogate()`, `as_fb_ensemble()`.

## Examples

```
set.seed(1)
x <- data.frame(a = runif(120, -2, 2), b = runif(120, -2, 2))
e <- fb_ensemble(
  params = x,
  outputs = data.frame(y = sin(x$a) + 0.5 * x$b^2 + rnorm(120, 0, 0.05))
)
s <- fit_surrogate(e)
predict(s, data.frame(a = 0, b = 1))
```

---

# koine_genomic_lens *Koine genomic fourth opinion (REML lens via sommer)*

---

## Description

Fit the established REML genomic BLUP with **sommer** on the same data and return a genomic *lens*: the field-standard fourth opinion to set beside the Bayesian multi-backend answer. (`triangulate_genomic()`, which compared the two, was removed from flexyBayes in version 0.14.0.) The lens carries the REML heritability, variance components (with standard errors), and breeding values (named by genotype, aligning to the flexyBayes GEBVs), so divergence between the Bayesian and REML answers is surfaced rather than hidden.

## Usage

```
koine_genomic_lens(fixed, data, K, geno = "geno")
```

## Arguments

| | |
|---|---|
| `fixed` | A fixed-effect formula, e.g. `y ~ 1` or `y ~ env`. |
| `data` | A data frame with the response, fixed effects, and the genotype factor; one or more records per genotype. |
| `K` | The genomic (or pedigree) relationship matrix; its row / column names are the genotype levels and must cover the genotypes in `data`. |
| `geno` | The genotype grouping-factor column name (default `"geno"`). |

## Value

A genomic lens: `list(h2, var_g, var_e, gebv, label)`, where `h2` / `var_g` / `var_e` are `list(estimate, se)` (the heritability SE is a delta-method approximation) and `gebv` is a named numeric vector of REML breeding values.

## Examples

```
## Not run:
bayes_fit <- flexyBayes::flexybayes(
  y ~ 1, random = ~ vm(geno, G), data = dat,
  known_matrices = list(G = G), backend = "greta"
)
koine <- koine_genomic_lens(y ~ 1, dat, G, geno = "geno")
koine$h2
head(koine$gebv)

## End(Not run)
```

---

`koine_status` *Introspect the koine backend slot*

---

## Description

Returns the activation state of the koine backend slot plus the procedure to activate it. koine is the synthesised fourth opinion on flexyBayes's orthogonal lens axis (the constellation's C1 + C3 contracts). The slot has been **active by default since 2026-06-08** on an installed koine at or above the version floor, gated only by `options(quorum.koine_activated)`; it falls back to dormant (a structured `koine_dormant_refusal`) when koine is absent, below floor, or the option is set `FALSE`. This helper is the canonical discoverability surface – call `koine_status()` to check whether the slot is currently active on the installed koine version and, if not, why and how to activate it.

## Usage

```
koine_status()
```

## Value

A list with components:

`activated` Logical: `TRUE` when koine is installed at or above the activation floor and `options(quorum.koine_activated)` is not set `FALSE` (the slot is live as of 2026-06-08).

`koine_installed` Logical: result of `nzchar(system.file(package = "koine"))`.

`builder_confirmed` Logical: whether koine's `koine_pass_from_design` builder is installed at or above the floor (the ported `emit_koine()` confirms it against the koine_pass/koine_result contract).

`dormancy_reason` Character: one of `"koine_not_installed"`, `"koine_below_floor"`, `"slot_provisioned_not_activ` or `"koine_dispatch_eligible"` (the last meaning the slot is fully active).

`activation_procedure` Character vector: numbered steps to activate the slot, in order.

## See Also

`gretaR_status()` (withdrawn from the hub in 0.10.0) for the engine-axis sibling slot.

## Examples

```
ks <- koine_status()
ks$activated         # TRUE when koine (>= 0.2.0) is installed
ks$dormancy_reason   # "koine_dispatch_eligible" when active
cat(ks$activation_procedure, sep = "\n")
```

---

`lens_contribution` *One lens's contribution to a resolution*

---

## Description

Builds the single row a lens adds to a contributions table: which lens it is, whether its estimate is grounded, whether it declined, the action it stands behind, and the reason and grounding token it declared. The row is the whole of what `quorum_resolve()` reads, so a lens that has abstained cannot smuggle a number into a resolution.

## Usage

```
lens_contribution(
  lens,
  grounded,
  abstained,
  action = NA_real_,
  reason = NA_character_,
  grounding = NA_character_,
  decisive = !abstained
)
```

## Arguments

| | |
|---|---|
| `lens` | Single string naming the lens. |
| `grounded` | Single logical: is the lens's estimate grounded? |
| `abstained` | Single logical: did the lens decline to stand behind a number? |
| `action` | The action the lens stands behind, as a single number, or `NA_real_` when it offers none. |
| `reason` | Single string naming why the lens declined, or `NA_character_`. |
| `grounding` | The grounding token the lens declared, conventionally one of `orchestraManifest::GROUNDING_GROUNDI` or its unverified complement, or `NA_character_`. |
| `decisive` | Single logical: is this lens's answer usable as the carried answer? Defaults to `!abstained`. |

## Details

`decisive` defaults to the complement of `abstained` and is separate from it because the two can come apart: a lens may fit, decline to abstain, and still have no action to offer, and a caller that knows this records `decisive = FALSE` rather than overloading the abstention flag.

## Value

A one-row data frame with the seven columns named above.

## See Also

`contributions()` to bind rows into a table, `quorum_resolve()` to resolve one.

## Examples

```
lens_contribution("mechanism", grounded = TRUE, abstained = FALSE,
                  action = 120, grounding = "grounded")
```

---

# lens_from_draws *Wrap draws a member has already produced as a lens*

---

## Description

The shortest path from a posterior to a lens. When a member has already produced its draws – a saved posterior, an ensemble read from disk, a constant a practitioner is defending – there is nothing left to fit, so the lens's `fit()` ignores the specification and the data and hands back the draws it was built from. Every built-in lens whose estimate is settled at construction time is built this way.

## Usage

```
lens_from_draws(
  draws,
  name,
  member,
  shares,
  expresses = "linear",
  scope = function(spec) TRUE
)
```

## Arguments

| | |
|---|---|
| `draws` | A `quorum_draws()` object the lens returns unchanged. |
| `name` | Single non-empty string, unique within a quorum. |
| `member` | Single non-empty string naming the package the estimate comes from. |
| `shares` | Named character vector over `sampler`, `objective`, `priors` and `data_prep`, the four components the independence ledger counts. |
| `expresses` | Character subset of `linear`, `plateau`, `threshold` and `nonmonotone`; `character(0)` for a lens that expresses no shape at all, such as a constant. |
| `scope` | A function `(spec)` returning `TRUE` inside the lens's scope and a `quorum_abstention()` outside it. The default accepts everything, which is right for draws that are already in hand. |

### Details

The kind is taken from the draws rather than declared again, because a lens that disagreed with its own draws about whether they are parameters or a response curve would be caught only at the comparison.

### Value

A quorum_lens() whose fit() returns draws.

### Examples

```
d <- quorum_draws(list(beta = rnorm(100L)), source = "saved")
l <- lens_from_draws(
  d, name = "saved", member = "quorum",
  shares = c(sampler = "none", objective = "saved", priors = "none",
             data_prep = "none")
)
identical(S7::prop(l, "fit")(NULL, NULL), d)
```

---

## lens_from_fit *Wrap a fitted hub or koine model as a lens*

---

### Description

Turns a model somebody has already fitted into a lens the comparison can read. The draws are taken once, at wrap time, and canonicalised: the fit's native parameter names become the hub's canonical tokens – (Intercept), one per fixed-effect term, sigma, and sd_<group> for each random-effect group – with the variance components on the standard-deviation scale whatever the engine stored. That is what lets an INLA lens, a brms lens and the koine lens be compared component by component instead of by position.

### Usage

```
lens_from_fit(fit, name = NULL, n_samples = 1000L, expresses = "linear")
```

### Arguments

| | |
|---|---|
| fit | A fitted model: a flexybayes_inla fit, a flexybayes_brms fit, or the flexybayes_koine wrapper emit_koine() returns. |
| name | Single non-empty string naming the lens, or NULL (the default) to name it after the objective – "inla", "brms" or "koine". |
| n_samples | Number of posterior draws to take. It reaches the engine's sampler on the INLA path, where the draws are generated on demand, and is ignored by backends that already carry every draw they kept. |
| expresses | Character subset of linear, plateau, threshold and nonmonotone naming the response shapes this parameterisation can express. The default is linear because a fit's expressiveness is a property of the model formula, which this function does not read. |

## Details

The independence declaration is read off the fit's class rather than asked for, because the caller holding a fitted object cannot be expected to restate what the engine that produced it already determines. An INLA fit declares the Laplace sampler and the INLA objective, a brms fit declares Stan's NUTS sampler and the brms objective, and the koine wrapper declares koine's stacked battery and koine's own objective; all three declare the hub's formula interlingua and the hub's data preparation, which they genuinely do share.

A class no adapter covers, an engine that cannot hand over its draws, and a koine wrapper that already stood down all return a `quorum_abstention()` rather than a lens. None of the three is a programmer error: they are facts about the fit, and a comparison that receives them can carry on with the lenses it does have.

## Value

A `quorum_lens()` whose `fit()` returns the canonicalised draws, or a `quorum_abstention()` with reason `unsupported_fit` (no adapter for the class), `lens_abstained` (koine stood down, with the nested reason attached), `draws_unavailable` (the engine could not hand over draws) or `draws_malformed` (the draws did not form a posterior).

## See Also

`lens_koine()` to fit koine rather than wrap an existing fit, `lens_from_draws()` for draws that need no adapter at all.

## Examples

```
# An object no adapter covers is declined by name, never guessed at.
a <- lens_from_fit(structure(list(), class = "lm"))
is_quorum_abstention(a)
a$reason
```

---

# lens_koine *The koine fourth opinion as a lens*

---

## Description

Declares koine's synthesised opinion as a lens without fitting anything yet. The lens's `fit()` calls `emit_koine()` on the specification and data it is handed and canonicalises the result. The specification is the hub's `fb_terms` intermediate representation; a fitted hub model is accepted in its place and the representation is taken from it, because different hub fits carry it at `$fb` and at `$extras$fb_terms` and a lens that read only one of the two would decline half the hub's fits for a reason that has nothing to do with koine. koine is already self-naming, so the canonicalisation is a pass-through there; it is the same code path a hub fit takes, which is what keeps the two lenses' parameter tokens identical by construction rather than by coincidence.

## Usage

```
lens_koine(name = "koine", ...)
```

### Arguments

| | |
|---|---|
| name | Single non-empty string naming the lens. |
| ... | Further arguments passed to emit_koine() at fit time, such as n_samples (the count of opinion draws to synthesise) and seed. |

### Details

koine is the independent witness on the methodological axis: a stacked battery of independent lenses synthesised into one opinion, not another sampler over the hub's objective. Its corner is a population GLM with at most two crossed or nested random intercepts, and outside that corner – or whenever koine's own convergence gate fires – it stands down. The lens re-types that as a lens_abstained decline carrying koine's own reason, so an abstaining lens contributes no number to the quorum and the reader still learns why.

### Value

A quorum_lens() of kind parameters whose fit(spec, data) returns a quorum_draws() over the canonical parameters, or a quorum_abstention() with reason engine_missing (koine is not installed), spec_missing (the fitted model handed in carries no parsed model), lens_abstained (koine stood down on this model, with the nested reason attached) or draws_unavailable.

### See Also

emit_koine() for the opinion itself, lens_from_fit() to wrap a koine fit already in hand.

### Examples

```
l <- lens_koine()
S7::prop(l, "shares")
```

---

## lens_mechanism *A simulation ensemble as a mechanism lens*

---

### Description

Reads a simulation ensemble as a lens on the response over a candidate grid: one row of the ensemble's outputs per ensemble member, one column per grid point. The simulator is the most independent witness the orchestra has – it shares no sampler, no objective, no priors and no data preparation with any statistical lens – and it is the only lens that can express a plateau, a threshold or a non-monotone response without being told to look for one, so it declares all four shapes.

### Usage

```
lens_mechanism(ensemble, grid, output = NULL, name = "mechanism")
```

## Arguments

| | |
|---|---|
| ensemble | A simulation ensemble: an fb_ensemble(), a PESTO calibration manifest, a saved .rds path, or a list carrying params and outputs. It reaches the canonical shape through as_fb_ensemble(), so an object no coercer covers is a programmer error rather than a decline. |
| grid | Non-empty numeric vector of candidate values the response is read over. |
| output | Which output columns carry the response, when the ensemble holds more than one output set: a character vector of column names, or a single string naming a value of the ensemble's output_dims$variable. NULL (the default) takes every output column, which is right for a single-output ensemble. |
| name | Single non-empty string naming the lens. |

## Details

quorum never runs the simulator. The ensemble is produced upstream and reaches this function through the C5 ingest contract, so a pesto_ensemble_manifest, a saved ensemble and a bare list carrying params and outputs are all accepted on the same terms.

The grid check is the substance of this adapter. An ensemble whose outputs do not cover the grid – the wrong number of columns, or numerically-named columns naming different values from the grid – is a mismatch between two descriptions of the same axis, and reading it anyway would silently attach the simulator's response to the wrong candidates. It returns a typed grid_mismatch decline instead.

## Value

A quorum_lens() of kind response_curve whose fit() returns a quorum_draws() matrix with one row per ensemble member and one column per grid point, or a quorum_abstention() with reason grid_mismatch (the selected outputs do not cover the grid) or outputs_not_numeric (the selected columns do not hold a numeric response).

## See Also

fb_ensemble() and as_fb_ensemble() for the ingest contract.

## Examples

```
ens <- fb_ensemble(
  params = data.frame(k = c(0.1, 0.2, 0.3)),
  outputs = data.frame(a = c(4.1, 4.6, 5.0), b = c(5.2, 5.9, 6.1))
)
l <- lens_mechanism(ens, grid = c(0, 50))
dim(S7::prop(S7::prop(l, "fit")(NULL, NULL), "draws"))

# A grid the outputs do not cover is declined, not stretched to fit.
is_quorum_abstention(lens_mechanism(ens, grid = c(0, 50, 100)))
```

`lens_status_quo` *The status quo as a lens*

## Description

A constant over the candidate grid: the rate a grower already applies, the variety already sown, the decision the analysis is being asked to improve on. It is the lens every comparison needs and no member emits, and it is the fallback a resolution policy carries when no quorum forms.

## Usage

```
lens_status_quo(value, grid, name = "status_quo")
```

## Arguments

| | |
|---|---|
| `value` | Single finite number, the status-quo response held across the whole grid. |
| `grid` | Non-empty numeric vector of candidate values – nitrogen rates, sowing dates, whatever the decision ranges over. |
| `name` | Single non-empty string naming the lens. |

## Details

It declares `"none"` for all four independence components, so it shares nothing with any lens that came from an engine and the ledger gives every such pair the full evidential weight. It expresses no response shape at all, which is the point: a constant cannot show a plateau or a threshold, so the shape-finding rule must never read one into it.

## Value

A `quorum_lens()` of kind `response_curve` whose `fit()` returns a one-row `quorum_draws()` matrix of `value` over `grid`.

## Examples

```
l <- lens_status_quo(value = 3.2, grid = c(0, 40, 80, 120))
d <- S7::prop(l, "fit")(NULL, NULL)
dim(S7::prop(d, "draws"))
```

lens_tmb *The TMB lens: a second Laplace implementation, sampled with NUTS*

## Description

Registers glmmTMB's Laplace-approximated marginal likelihood as a lens, and samples the very TMB objective that fit produced with `tmbstan`'s NUTS. The lens therefore contributes two things the hub's own engines do not: a Laplace implementation written independently of INLA's, and one objective sampled two ways, so a disagreement between the Laplace point and the Hamiltonian posterior is attributable to the approximation rather than to a difference of model.

## Usage

```
lens_tmb(
  family = "gaussian",
  laplace = TRUE,
  n_draws = 2000L,
  chains = 2L,
  iter = NULL,
  warmup = NULL,
  seed = 1L,
  priors = c("hub", "flat"),
  name = NULL
)
```

## Arguments

| | |
|---|---|
| `family` | One of `"gaussian"`, `"poisson"`, `"binomial"` or `"nbinom2"`, used when the spec is a formula. Any other family returns a `family_unsupported` abstention at fit time. |
| `laplace` | Logical: when `TRUE` (the default) the random effects stay integrated out by the Laplace approximation and NUTS samples the remaining parameters; when `FALSE` NUTS samples the joint vector. |
| `n_draws` | Target number of post-warmup draws in total across chains; sets the default `iter` when `iter` is `NULL`. |
| `chains` | Number of NUTS chains. |
| `iter` | Iterations per chain, warmup included. `NULL` derives it from `n_draws` and `chains` with a half-warmup split. |
| `warmup` | Warmup iterations per chain. `NULL` uses half of `iter`. |
| `seed` | Integer RNG seed passed to the sampler. |
| `priors` | `"hub"` to match the hub's default priors as far as glmmTMB's prior syntax allows, or `"flat"` to fit under glmmTMB's own defaults. See the Priors section. |
| `name` | Lens name; defaults to `"tmb"`. |

## Details

The `fit` function accepts the model either as an lme4-style formula or as the hub's `fb_terms` intermediate representation, from which the formula is rebuilt the way `emit_koine()` rebuilds it. The lens fields a plain random intercept on one grouping variable; a random slope, a correlated structure, a nested or interaction grouping, or a second grouping term returns an `out_of_scope` abstention. When the spec is an `fb_terms` object its own family and link are authoritative, so a hub fit and this lens are never compared across two different likelihoods; a formula spec takes the `family` argument instead.

## Value

A `quorum_lens()` with member `"glmmTMB"` and kind `"parameters"`. Its `fit(spec, data, ...)` returns a `quorum_draws()` whose components are named with the hub's canonical parameters – `"(Intercept)"`, the model-matrix column names, and `sd_<grouping>` – and whose `diagnostics` carry `laplace_point` (the glmmTMB point estimates on those same names), `rhat` (the worst-case R-hat), `n_divergent`, the `laplace` flag, and the prior plan actually used. Further arguments in `...` reach `tmbstan::tmbstan()`. Where glmmTMB or tmbstan is not installed, the model is outside the lens's shape, or the family is unsupported, `fit()` returns a `quorum_abstention()` and never a number.

## Priors

The hub's default is `uniform(0, U)` on every standard deviation, with `U = 5 * sd(y)` on an identity link, `3` on a log link and `5` on a logit link. glmmTMB parameterises the random-effect SD by its logarithm, so fitting with no prior at all is flat on `log(SD)` and thus proportional to `1/SD` on the SD scale, which is not integrable at zero: for a weakly identified variance component the posterior is improper and the sampler does not settle. With `priors = "hub"` (the default) the lens writes a shape-2 gamma prior on the random-intercept SD. glmmTMB adds a gamma row to the negative log density as `-log dgamma(exp(theta), shape, scale)` with no Jacobian term, so the implied density on the SD scale is proportional to `SD^(shape - 2)`, and shape 2 is flat; the scale is set far above the hub's bound so it stays flat across the whole interval the hub's uniform prior covers. Two parts of the hub's default have no counterpart in glmmTMB's prior syntax – the hard upper bound itself, and the uniform prior on the residual SD, since the dispersion parameter admits only log-scale families – and both are named in the fitted object's `diagnostics$priors$unmatched` rather than assumed away. `shares["priors"]` reads `"hub_matched"` when every hub prior on a component the lens reports (the fixed effects, which the hub leaves to the engine by default, and the random-intercept SD) was carried. With `priors = "flat"` the fit runs under glmmTMB's own defaults and the lens declares `"flat"`.

## See Also

`quorum_lens()` for the contract, `emit_koine()` for the formula construction and the canonical naming this lens follows.

## Examples

```
lens <- lens_tmb()
lens@shares
# Outside the lens's shape, the scope gate declines rather than fits.
```

```
is_quorum_abstention(lens@scope(y ~ x + (1 + x | g)))
```

---

`predict.fb_surrogate` *Predict from a quorum surrogate (mean + sd per output)*

---

## Description

Returns the surrogate's predictive distribution at new parameter sets. By default a `data.frame` carrying, for each emulated output `<o>`, a `<o>_mean` and a `<o>_sd` column. Reporting the predictive sd – not only the mean – is the contract-C7 distribution-preserving requirement.

## Usage

```
## S3 method for class 'fb_surrogate'
predict(object, newdata, cov = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `object` | An `fb_surrogate` from `fit_surrogate()`. |
| `newdata` | A `data.frame` of new parameter sets; must contain the surrogate's `param_names` columns. |
| `cov` | `FALSE` (default) for the marginal `data.frame`; `TRUE` for an `fb_surrogate_prediction` with the dense per-row cross-output covariance array; `"lowrank"` for the compact factored covariance (a joint `rff` fit only). |
| `...` | Unused. |

## Details

With `cov = TRUE` it returns the **joint** predictive distribution: an `fb_surrogate_prediction` carrying the predictive mean matrix and a per-row cross-output covariance array (contract C7 clause 4). A joint `rff` surrogate supplies the full covariance; a marginals-only surrogate (independent `rff`, or a conformer reporting only `<o>_sd`) returns a diagonal covariance built from the marginal sds, so the call always states its basis.

With `cov = "lowrank"` a joint `rff` surrogate returns the same joint predictive in a **compact factored** form for large output counts: an `fb_surrogate_prediction` carrying `cov_lowrank = list(u, d, infl)` such that per-row covariance is `infl[i] * (u u' + diag(d))` – `O(k q + n)` storage instead of the dense `O(k^2 n)` array. The factor `u` has the surrogate's `cov_rank` columns (the full `k` if it was fit without `cov_rank`). Reconstruct a slice with `fb_cov_slice()`. The diagonal still equals `<o>_sd^2` exactly.

For the reference `rff` method, writing $\phi(x)$ for the random Fourier feature map of a new parameter set $x$, the reported mean and variance are the Bayesian-linear-regression predictive moments

$$E[y \mid x] = \phi(x)^\top \mu_N, \qquad Var[y \mid x] = 1/t + \phi(x)^\top S_N \phi(x),$$

with posterior weight mean $\mu_N$, posterior weight covariance $S_N$, and noise precision $t$ set by MacKay evidence maximisation at fit time (Bishop, 2006, sec. 3.3, 3.5).

## Value

With cov = FALSE (default) a data.frame with <output>_mean and <output>_sd columns, one row per row of newdata. With cov = TRUE an fb_surrogate_prediction (mean, cov, output_names). With cov = "lowrank" an fb_surrogate_prediction (mean, cov_lowrank, output_names).

## References

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, secs. 3.3 and 3.5.

## Examples

```
theta <- cbind(p1 = runif(60, -2, 2), p2 = runif(60, -2, 2))
y <- theta %*% rbind(c(1.5, -0.8), c(0.4, 1.1)) + rnorm(120, 0, 0.1)
ens <- fb_ensemble(as.data.frame(theta), as.data.frame(y),
                   provenance = list(source = "raw"))
sur <- fit_surrogate(ens)
predict(sur, data.frame(p1 = 1, p2 = -1))
```

---

# print.flexybayes_koine

*Print a koine fourth-opinion fit*

---

## Description

Print a koine fourth-opinion fit

## Usage

```
## S3 method for class 'flexybayes_koine'
print(x, ...)
```

## Arguments

| | |
|---|---|
| x | A flexybayes_koine object. |
| ... | Unused. |

## Value

Invisibly returns x.

## print.quorum::quorum_verdict

*Print a quorum verdict*

### Description

Shows the status, how many lenses contributed and how many declined, the lens whose answer is carried and the number it carries, the count of shape findings, the independence caveat when there is one, and the typed decline when the verdict carries no number.

### Usage

```
## S3 method for class '`quorum::quorum_verdict`'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | A quorum_verdict(). |
| ... | Unused. |

### Details

The method is registered against the package-qualified S7 class name, which is what S7 puts in the class vector.

### Value

x, invisibly.

## print.quorum_independence

*Print an independence ledger*

### Description

Shows the evidential weights rounded to two decimal places, and the caveat whenever the data behind the lenses has not been established as independently collected.

### Usage

```
## S3 method for class 'quorum_independence'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | A quorum_independence object. |
| ... | Unused. |

### Value

Invisibly returns x.

---

quorum *Compare a quorum of lenses and resolve them into one verdict*

---

### Description

The front door. Each lens is gated on its own declared scope, then asked to fit; a lens that declines either is recorded as having declined and contributes no number. The lenses that returned draws are compared with the agreement semantics, weighed by the independence ledger, and resolved through the declared policy, and the whole comparison is emitted as a manifest the rest of the orchestra can read.

### Usage

```
quorum(
  lenses,
  spec = NULL,
  data = NULL,
  policy,
  grid = NULL,
  decide = NULL,
  data_independence = NA
)
```

### Arguments

| | |
|---|---|
| lenses | A quorum_lens(), a list of them, or their registered names. |
| spec | The estimand specification each lens's scope and fit receive. |
| data | The data each lens's fit receives. |
| policy | A quorum_policy() declaring the fallback order and quorum. |
| grid | The candidate grid a decision is taken over, or NULL. |
| decide | An optional function (draws, grid) returning a list with action, abstained, reason and grounding, supplied by the caller – a decision layer's wrapper, for instance. Without one, a contributing lens is decisive and its action is NA: quorum compares lenses and never prices a decision itself. |
| data_independence | Single logical: do the lenses read independent upstream data? FALSE blocks the manifest's grounded token, NA records that it was not declared. |

## Details

A disagreement between two lenses that cannot express the same response shapes is recorded as a finding rather than counted as discordance: the more expressive lens seeing a plateau the other cannot represent is information about the pair, not evidence that either is wrong.

The status is `"no_quorum"` when fewer lenses contribute than the policy requires, `"not_computed"` when there is nothing to compare or the metric layer is unavailable, `"concordant"` when every counted pair agrees, and `"discordant"` otherwise. A pair the metric layer could not compare is not concordant, so it makes the status `"discordant"`; the agreement matrix names it as inconclusive rather than as disagreement.

No quorum means no number. When fewer lenses contribute than the policy requires, the resolution abstains whatever the contributing lenses said, and the verdict carries the typed decline that says so.

## Value

A `quorum_verdict()`.

## See Also

`quorum_policy()`, `quorum_resolve()`, `quorum_manifest()`.

## Examples

```
fit_fn <- function(spec, data, ...) {
  quorum_draws(list(`(Intercept)` = stats::rnorm(200L)),
               source = "example")
}
one <- quorum_lens("one", "example", fit_fn,
                   c(sampler = "a", objective = "b", priors = "c",
                     data_prep = "d"))
two <- quorum_lens("two", "example", fit_fn,
                   c(sampler = "w", objective = "x", priors = "y",
                     data_prep = "z"))
quorum(list(one, two), policy = quorum_policy(c("one", "two")))
```

---

# quorum_abstention *Construct a typed quorum abstention*

---

## Description

The single constructor for every decline the package issues. A lens that is out of scope, an engine that is not installed, a comparison with fewer contributing lenses than the policy's quorum, or a resolution with nothing grounded left all return one of these rather than a number.

## Usage

```
quorum_abstention(reason, detail = NULL, scope = "quorum", ...)

is_quorum_abstention(x)
```

### Arguments

| | |
|---|---|
| reason | A single string naming the reason in snake_case, for example "engine_missing", "out_of_scope", "no_quorum", "not_grounded". |
| detail | Optional single string with the specifics a reader needs. |
| scope | Single string naming the part of the package that declined ("lens", "comparison", "resolution", "manifest"). |
| ... | Further named fields carried on the object. |
| x | Any object. |

### Value

A list of class c("quorum_abstention", "orchestra_refusal") with abstained = TRUE, reason, detail, scope and any extra fields.

### Examples

```
a <- quorum_abstention("engine_missing", "tmbstan is not installed", "lens")
is_quorum_abstention(a)
```

---

## quorum_compare_policies

*Price several fallback conventions in one table*

---

### Description

Resolves the same contributions under every policy handed in and returns one table carrying each policy's verdict and action side by side, so the cost of a fallback convention is read off a column rather than argued.

### Usage

```
quorum_compare_policies(contrib, policies)
```

### Arguments

| | |
|---|---|
| contrib | A contributions data frame from contributions(). |
| policies | A named list of quorum_policy() and policy_committed() objects. The names become the column suffixes. |

### Value

A data frame with one row per cell: cell (when the input had one), then verdict_<name> and action_<name> for each policy.

### See Also

quorum_resolve().

## Examples

```
contrib <- contributions(
  lens_contribution(
    "mechanism", grounded = FALSE, abstained = TRUE,
    reason = "input_ungrounded",
    grounding = orchestraManifest::GROUNDING_UNVERIFIED
  ),
  lens_contribution("statistical", grounded = TRUE, abstained = FALSE,
                    action = 30)
)
quorum_compare_policies(contrib, list(
  cascade = quorum_policy(c("mechanism", "statistical"), status_quo = 60),
  committed = policy_committed("zero", status_quo = 60, safe_action = 0)
))
```

---

# quorum_draws *The draws every lens returns*

---

## Description

A `quorum_draws` object is the one shape a lens hands to the comparison. For `kind = "parameters"` the draws are a named list of numeric vectors, one per estimand component, named with the hub's canonical parameter names. For `kind = "response_curve"` the draws are a numeric matrix with one row per draw and one column per grid point, and `grid` carries the candidate grid (for example nitrogen rates).

## Usage

```
quorum_draws(
  draws,
  kind = "parameters",
  grid = NULL,
  source = "unknown",
  diagnostics = list()
)
```

## Arguments

| | |
|---|---|
| `draws` | Named list of numeric vectors, or a numeric matrix. |
| `kind` | One of `"parameters"` or `"response_curve"`. |
| `grid` | Numeric grid for a response curve, else `NULL`. |
| `source` | Single string naming where the draws came from. |
| `diagnostics` | List of the producing lens's own diagnostics. |

## Value

A `quorum_draws` S7 object.

`quorum_independence` *The independence ledger over a set of lenses*

## Description

Counts what every pair of lenses shares and converts it into the evidential weight the quorum rule reads. Two lenses declare four components – `sampler`, `objective`, `priors` and `data_prep` – and the weight for a pair is `1 - shared / 4`, so a pair sharing nothing carries full weight, a pair sharing all four carries none, and a pair sharing one component carries three quarters.

## Usage

```
quorum_independence(lenses, data_independence = NA)
```

## Arguments

| | |
|---|---|
| `lenses` | A list of `quorum_lens()` objects. An empty list is allowed and yields empty matrices; anything in the list that is not a lens is a programmer error. |
| `data_independence` | Single logical. `TRUE` when the lenses were fitted to independently-collected data, `FALSE` when they share one dataset, and `NA` (the default) when the caller has not established which. Anything other than `TRUE` attaches the caveat. |

## Details

The declarations are the lenses' own. The ledger does not inspect an engine to discover what it shares with another, because the point of the declaration is that a member states its own dependencies where a reader can see and dispute them.

Data independence is a separate question the declarations cannot answer, so the caller supplies it. Left at `NA` – or set to `FALSE` – the result carries a caveat saying what agreement between lenses on one dataset does and does not establish. The caveat is empty only when the caller has positively established that the lenses saw independently-collected data.

## Value

A list of class `quorum_independence` with `components` (the four component names), `shared` (an integer matrix of the number of components each pair holds in common, `NA` on the diagonal), `weight` (the numeric matrix `1 - shared / 4`, `NA` on the diagonal, rows and columns named for the lenses), `data_independence` as supplied, and `caveat` (the sentence to carry alongside the weights, or `""`).

## See Also

`quorum_lens()` for the declarations the ledger reads.

## Examples

```
d <- quorum_draws(list(beta = rnorm(20L)))
mk <- function(nm, shares) {
  lens_from_draws(d, name = nm, member = "quorum", shares = shares)
}
a <- mk("a", c(sampler = "nuts", objective = "brms", priors = "hub",
               data_prep = "hub"))
b <- mk("b", c(sampler = "ies", objective = "simulator",
               priors = "ensemble", data_prep = "pesto"))
quorum_independence(list(a, b))
```

---

`quorum_lens` *A lens: a member's estimate of a shared estimand, with its declarations*

---

## Description

A lens: a member's estimate of a shared estimand, with its declarations

## Usage

```
quorum_lens(
  name,
  member,
  fit,
  shares,
  expresses = "linear",
  kind = "parameters",
  scope = function(spec) TRUE
)

lens(...)
```

## Arguments

| | |
|---|---|
| `name` | Single string, unique within a quorum. |
| `member` | Single string naming the package the estimate comes from. |
| `fit` | A function `(spec, data, ...)` returning a `quorum_draws()` or a `quorum_abstention()`. |
| `shares` | Named character vector over the components `sampler`, `objective`, `priors` and `data_prep`; two lenses that name the same value for a component share it, and the independence ledger counts it. |
| `expresses` | Character subset of `linear`, `plateau`, `threshold`, `nonmonotone`: the response shapes the lens can express. |
| `kind` | One of `"parameters"` or `"response_curve"`. |
| `scope` | A function `(spec)` returning `TRUE` inside the lens's scope and a `quorum_abstention()` outside it. |
| `...` | Passed to the `quorum_lens` constructor. |

### Value

A quorum_lens S7 object.

---

quorum_manifest *Emit a verdict as an orchestra manifest*

---

### Description

Builds the orchestra_manifest the rest of the orchestra reads from a quorum_verdict(): the contributions table as the payload, the status, the carried lens, the action, the independence weights and the independence caveat in the typed summary, and the grounding token the consensus earned.

### Usage

```
quorum_manifest(verdict)
```

### Arguments

| | |
|---|---|
| verdict | A quorum_verdict(). |

### Details

The grounding rule is stricter than the lenses'. The consensus is stamped grounded only when at least one contributing lens is grounded *and* data_independence was not declared FALSE; lenses that agree because they read the same upstream fact corroborate the reading, not the world, and the manifest says so by carrying the unverified token instead.

### Value

An orchestraManifest::orchestra_manifest, or a quorum_abstention() when the installed contract has no "consensus" target to emit into.

### See Also

quorum(), quorum_verdict().

### Examples

```
fit_fn <- function(spec, data, ...) {
  quorum_draws(list(`(Intercept)` = stats::rnorm(200L)),
               source = "example")
}
one <- quorum_lens("one", "example", fit_fn,
                   c(sampler = "a", objective = "b", priors = "c",
                     data_prep = "d"))
two <- quorum_lens("two", "example", fit_fn,
                   c(sampler = "w", objective = "x", priors = "y",
                     data_prep = "z"))
v <- quorum(list(one, two), policy = quorum_policy(c("one", "two")))
quorum_manifest(v)
```

quorum_policy *A resolution policy*

## Description

Declares how a comparison is resolved into one carried answer: the fallback priority of the lenses, how many must contribute before a verdict is issued, the agreement thresholds to compare them with, the evidential weight below which a pair is not counted, and what a cell falls back to when no lens stands behind a number.

## Usage

```
quorum_policy(
  order,
  min_lenses = 2L,
  thresholds = NULL,
  independence_min = 0,
  status_quo = NULL,
  safe_action = NULL
)
```

## Arguments

| | |
|---|---|
| order | Character vector of lens names in fallback priority: the first name that is present, grounded and decisive is the one carried. |
| min_lenses | Single whole number: how many lenses must contribute before a comparison is a quorum at all. |
| thresholds | A list of agreement thresholds, or NULL to read agreement_thresholds() at the point of use. |
| independence_min | Single number: the evidential weight at or above which a pair counts towards concordance. |
| status_quo | A single number to fall back on, or NULL to abstain instead of defaulting. |
| safe_action | The action a single lens falls back to inside its own decision. It is carried for the caller and is what policy_committed(fixed = "zero") resolves an abstention to; the cascade never mints a rate from it. |

## Details

status_quo is the whole of the difference between a cell that defaults and a cell that abstains. Left NULL, a resolution with no decisive lens abstains and issues no number at all; set to a value, that value is carried and the cell is marked as having defaulted rather than decided.

thresholds left NULL means the agreement thresholds in force at the time of use, read from agreement_thresholds() when the metric layer is present, so a policy written today does not pin yesterday's cut-offs.

## Value

A quorum_policy S7 object.

## See Also

quorum_resolve(), policy_committed().

## Examples

```
quorum_policy(order = c("mechanism", "statistical"), status_quo = 60)
```

---

# quorum_policy_committed

*The committed convention the cascade replaced*

---

## Description

Builds the policy Study A0 committed to before the cascade: one decision lens decides, a declared disagreement between the lenses demotes the cell to the status quo, a decision declined for insufficient evidence does the same, and an ungrounded input or an abstaining producer resolves to a fixed rate. It is kept as an object rather than as prose so that quorum_compare_policies() can price it against the cascade in one table.

## Usage

```
quorum_policy_committed(
  fixed = c("zero", "status_quo"),
  lens = "mechanism",
  status_quo = NULL,
  safe_action = NULL
)

policy_committed(...)
```

## Arguments

| | |
|---|---|
| fixed | Which fixed rate an abstention resolves to: "zero" for safe_action, "status_quo" for status_quo. |
| lens | Single string naming the deciding lens whose contribution rows the convention reads. |
| status_quo | A single number: the rate a demoted cell holds, or NULL to leave it unstated (the action is then NA). |
| safe_action | A single number: the no-input floor, or NULL. |
| ... | Passed to the quorum_policy_committed constructor. |

## Details

The convention's defining property is that an abstention still mints a number. `fixed = "zero"` resolves it to `safe_action`, the no-input floor; `fixed = "status_quo"` resolves it to `status_quo`. Withholding a rate is not among its outcomes, which is precisely what the cascade changed.

## Value

A `quorum_policy_committed` S7 object, which is a `quorum_policy()` carrying the two extra declarations.

## See Also

`quorum_resolve()`, `quorum_compare_policies()`.

## Examples

```
policy_committed("zero", lens = "mechanism", status_quo = 60,
                 safe_action = 0)
```

---

# quorum_resolve *Resolve a comparison into one carried answer*

---

## Description

Walks the policy's declared order and carries the first lens that is present, grounded and decisive for that cell. When no lens is, but at least one is present and grounded and simply would not stand behind a number, the policy's `status_quo` is carried and the cell is recorded as having defaulted. When nothing grounded remains, the cell abstains and carries no number at all.

## Usage

```
quorum_resolve(contrib, policy)
```

## Arguments

| | |
|---|---|
| `contrib` | A contributions data frame from `contributions()`. A `cell` column makes the resolution per cell; without one the whole table is treated as a single cell. |
| `policy` | A `quorum_policy()` or a `policy_committed()`. |

## Details

The three outcomes are deliberately distinct. A defaulting cell has a grounded lens that declined; an abstaining cell has none, and the difference is the whole reason an abstention is not simply resolved to a fixed rate. A lens that abstained never contributes its number to either.

Handed a `policy_committed()`, the function applies that convention instead of the cascade and returns the same columns, so the two are comparable cell by cell.

Only lenses named in `policy@order` take part: a contribution from a lens the policy did not declare is carried in the table but not in the resolution.

## Value

A data frame with one row per cell: `cell` (when the input had one), a `tier_<lens>` logical for each lens in the policy's order, `tier_status_quo`, `tier_abstain`, `carried_lens`, `verdict` (`"decide"`, `"default"` or `"abstain"`) and `action`.

## See Also

`contributions()`, `quorum_compare_policies()`, `quorum()`.

## Examples

```
contrib <- contributions(
  lens_contribution("mechanism", grounded = TRUE, abstained = TRUE),
  lens_contribution("statistical", grounded = TRUE, abstained = FALSE,
                    action = 30)
)
quorum_resolve(contrib, quorum_policy(c("mechanism", "statistical"),
                                      status_quo = 60))
```

---

# quorum_verdict *The verdict a quorum returns*

---

## Description

Everything the comparison established, in one object: which lenses were asked, what each contributed, the pairwise agreement, the independence weights behind it, the shape findings that explain a disagreement rather than counting it, the overall status, the resolution the policy reached, and the manifest the rest of the orchestra reads.

## Usage

```
quorum_verdict(
  lenses,
  contributions,
  status,
  resolution,
  policy,
  agreement = NULL,
  independence = NULL,
  findings = .empty_findings(),
  manifest = NULL,
  abstention = NULL,
  carried_draws = NULL
)
```

### Arguments

| | |
|---|---|
| lenses | Character vector of the lens names that were asked. |
| contributions | The contributions data frame, one row per lens. |
| status | One of "concordant", "discordant", "no_quorum" or "not_computed". |
| resolution | The resolution data frame from quorum_resolve(). |
| policy | The quorum_policy() the resolution used. |
| agreement | The agreement result, or NULL when the metric layer was not available. |
| independence | The independence result, or NULL. |
| findings | A data frame of shape findings. |
| manifest | The emitted manifest, or the typed decline explaining why none was emitted, or NULL. |
| abstention | A quorum_abstention() when the verdict carries no number, else NULL. |
| carried_draws | A numeric matrix of the carried lens's draws, one column per quantity, when the status is "concordant" over a single resolved cell; else NULL. |

### Value

A quorum_verdict S7 object.

### See Also

quorum(), quorum_manifest().

---

## read_fb_ensemble *Read a simulation ensemble from a file*

---

### Description

Reads a saved fb_ensemble (or a list(params, outputs, ...)) from an .rds file and returns a verified fb_ensemble. The file path's provenance is recorded. (Delimited / .fst readers can be added as the methods work needs them; .rds is the lossless round-trip used now.)

### Usage

```
read_fb_ensemble(path, ...)
```

### Arguments

| | |
|---|---|
| path | A single .rds file path. |
| ... | Unused; for generic compatibility. |

### Value

An fb_ensemble.

`register_ensemble_source` *Register an ensemble-source coercer for a producer class*

## Description

Extends `as_fb_ensemble()` so a new producer of simulation ensembles becomes ingestible without modifying quorum. The coercer takes a producer object and returns an `fb_ensemble`. This is the constellation extensibility seam for contract C5: a member registers its coercer rather than quorum hard-coding the member.

## Usage

```
register_ensemble_source(producer_class, coercer)
```

## Arguments

| | |
|---|---|
| `producer_class` | A single string – the S3 class of the producer object. |
| `coercer` | A function `function(x, ...) -> fb_ensemble`. |

## Value

Invisibly `NULL`.

`register_kernr_surrogates` *Register kernR's conditional mean embedding as a flexyBayes C7 conformer*

## Description

Makes `method = "cme"` available to `fit_surrogate()` by registering an adapter around kernR's `fit_cme()`. kernR's CME is a kernel linear smoother that yields a conditional *mean* but no predictive variance; the adapter supplies the standard linear-smoother predictive variance `sigma^2 (1 + ||w||^2)` (with `sigma^2 = RSS / (n - tr(S))` from the in-sample smoother), so the emulator honours the C7 distribution-preserving `<o>_sd` requirement. The conditional-mean map stays kernR's; only the uncertainty quantification is added here – not a reimplementation.

## Usage

```
register_kernr_surrogates()
```

## Value

Invisibly, the registered method name (`"cme"`).

### Out-of-support gate

The smoother weights `w(x*)` are an approximate partition of unity wherever `x*` is supported by the design: `sum(w)` is close to 1 across the training design. With a bounded, decaying kernel every `k(x*, x_i)` tends to zero as `x*` leaves the design, so `sum(w(x*))` tends to 0, the mean `w(x*)'y` decays to zero, and `||w(x*)||^2` decays to zero – which drives the reported sd *down* to its in-design floor `sqrt(sigma^2)`. Ungated, the conformer would report an in-design-sized uncertainty on a fabricated mean at exactly the point where it knows least.

The conformer therefore records the weight mass across the training design at fit time and refuses to predict where the mass falls below `weight_mass_frac` (default 0.5) times the least-supported design point's mass, raising a classed `fb_surrogate_cme_abstention` condition that names the offending rows, their mass, and the floor. Pass `weight_mass_frac` (or an absolute `weight_mass_floor`) through `fit_surrogate()` to retune it, or `weight_mass_floor` to `predict()` to override it for one call; set the floor to 0 to disable the gate. The reference `rff` surrogate widens as declared off-design and is the named fallback for extrapolation.

Called opportunistically from `.onLoad()` when kernR is installed.

### See Also

`register_surrogate()`, `fit_surrogate()`, `register_pesto_surrogates()`.

### Examples

```
if (requireNamespace("kernR", quietly = TRUE)) {
  register_kernr_surrogates()
  set.seed(1)
  a <- runif(80, -2, 2)
  e <- fb_ensemble(
    params = data.frame(a = a),
    outputs = data.frame(y = sin(a) + rnorm(80, 0, 0.05)),
    provenance = list(source = "raw")
  )
  s <- fit_surrogate(e, method = "cme", lambda = 1e-2)
  predict(s, data.frame(a = 0.5))
  # far outside the design the conformer declines rather than guessing:
  tryCatch(
    predict(s, data.frame(a = 40)),
    fb_surrogate_cme_abstention = function(cnd) conditionMessage(cnd)
  )
}
```

---

## register_lens *Register, retrieve, list and clear lenses*

---

### Description

The lens registry is the roster a quorum draws from. A member declares its estimate once, as a `quorum_lens()`, and registers it under a name; the comparison then reaches the lens by that name rather than by holding the object. Registration replaces by name, so re-registering after an engine upgrade updates the roster in place rather than leaving two lenses answering to one name.

## Usage

```
register_lens(lens)

get_lens(name)

list_lenses()

clear_lenses()
```

## Arguments

| | |
|---|---|
| lens | A quorum_lens() to register. Anything else is a programmer error and raises a condition rather than returning an abstention: a registry that accepted non-lenses would defer the failure to the comparison, where the caller can no longer see which registration was wrong. |
| name | Single non-empty string naming a registered lens. |

## Details

The registry holds declarations, not results. Registering a lens neither fits nor ingests anything, because a lens is a fitter behind an adapter and the fitting happens when the comparison calls its fit(). Clearing the registry therefore costs nothing but the declarations themselves.

## Value

register_lens() returns lens invisibly. get_lens() returns the registered quorum_lens(), or NULL when nothing is registered under name; a lookup miss is a fact about the roster, not a failure. list_lenses() returns a data.frame with one row per registered lens ordered by name, carrying name, member, kind, expresses (the declared shapes collapsed with "/", empty for a lens that expresses none) and one column per independence component (sampler, objective, priors, data_prep); it has zero rows when the registry is empty. clear_lenses() returns NULL invisibly.

## Examples

```
d <- quorum_draws(list(beta = rnorm(50L)), source = "demo")
l <- lens_from_draws(
  d, name = "demo", member = "quorum",
  shares = c(sampler = "none", objective = "demo", priors = "none",
             data_prep = "none")
)
register_lens(l)
list_lenses()
identical(get_lens("demo"), l)
get_lens("never_registered")
clear_lenses()
```

## register_pesto_surrogates *Register PESTO's GP surrogate as a flexyBayes C7 conformer*

### Description

Makes `method = "gp"` available to `fit_surrogate()` by registering an adapter around PESTO's `train_gp_surrogate()` / `predict_gp_surrogate()`. PESTO's GP already reports a per-output predictive mean and variance, so the adapter only reshapes its output into the contract-C7 `<o>_mean` / `<o>_sd` data.frame (`inst/surrogate_contract.md`) – PESTO's surrogate is consumed through the registry, never copied into flexyBayes.

### Usage

```
register_pesto_surrogates()
```

### Details

Called opportunistically from `.onLoad()` when PESTO is installed, so the method is usually already available; call it explicitly to (re-)register.

### Value

Invisibly, the registered method name (`"gp"`).

### See Also

`register_surrogate()`, `fit_surrogate()`, `register_kernr_surrogates()`.

## register_surrogate *Register a surrogate-fitting method*

### Description

Extends `fit_surrogate()` with a new conforming surrogate (contract C7, `inst/surrogate_contract.md`) without modifying quorum. The fit function takes the standardised design `x` (parameter sets), the response matrix `y` (outputs), and `...`, and returns an object with a `predict()` method yielding a per-output predictive mean and sd. This is how kernR's CME / dist_regression and PESTO's GP surrogates plug in.

### Usage

```
register_surrogate(method, fit_fn)
```

### Arguments

| | |
|---|---|
| method | A single string naming the method. |
| fit_fn | A function function(x, y, ...) -> <surrogate model>. |

### Value

Invisibly NULL.

---

## verify_fb_ensemble *Verify a simulation ensemble against the C5 contract*

---

### Description

Checks that e honours the ensemble-source contract (inst/ensemble_source_contract.md): row-aligned positive-length params / outputs, any observed names a subset of the outputs, any output_dims one row per output column, and a provenance$source. Returns a verdict rather than erroring, so callers can branch (the fb_ensemble() constructor turns a failure into an error; an interactive check can inspect the message).

### Usage

```
verify_fb_ensemble(e)
```

### Arguments

| | |
|---|---|
| e | An object to check (an fb_ensemble, or any list with the same components). |

### Value

A list list(ok = <logical>, message = <character>). ok = TRUE and an empty message on conformance; ok = FALSE and the first violation otherwise.

# Index


agreement_matrix, 3
agreement_matrix(), *5*
agreement_pairwise, 4
agreement_pairwise(), *3, 4*, *6*, *8*, *10*, *12*
agreement_status, 5
agreement_status(), *4, 5*, *8*
agreement_thresholds, 6
agreement_thresholds(), *3*, *5, 6*
as_fb_ensemble, 8
as_fb_ensemble(), *14–16*, *21, 22*, *30*, *49*

canonical_names.flexybayes_koine, 9
clear_lenses *(*register_lens*)*, 50
coef.flexybayes_koine, 9
contributions, 10
contributions(), *26*, *39*, *46, 47*

emit_koine, 10
emit_koine(), *27–29*, *33*

fb_as_draws_simple.flexybayes_koine, 12
fb_cov_slice, 13
fb_cov_slice(), *34*
fb_ensemble, 13
fb_ensemble(), *14, 15*, *30*
fb_pesto, 14
fb_pesto(), *16*
fb_prior(), *15, 16*
fb_prior_from_ensemble, 15
fb_prior_from_ensemble(), *14, 15*
fb_surrogate_ies, 16
fb_surrogate_ies(), *20*
fb_surrogate_ppc, 18
fb_surrogate_ppc(), *18*
fit_surrogate, 21
fit_surrogate(), *17–20*, *34*, *49, 50*, *52*

get_lens *(*register_lens*)*, 50

is_quorum_abstention *(*quorum_abstention*)*, 38

kernR::mmd_ppc(), *18*, *20*
koine_genomic_lens, 23
koine_status, 24
koine_status(), *11, 12*

lens *(*quorum_lens*)*, 42
lens_contribution, 25
lens_contribution(), *10*
lens_from_draws, 26
lens_from_draws(), *28*
lens_from_fit, 27
lens_from_fit(), *10*, *12*, *29*
lens_koine, 28
lens_koine(), *28*
lens_mechanism, 29
lens_status_quo, 31
lens_tmb, 32
list_lenses *(*register_lens*)*, 50

policy_committed *(*quorum_policy_committed*)*, 45
policy_committed(), *39*, *45, 46*
predict.fb_surrogate, 34
predict.fb_surrogate(), *13*, *20*, *22*
print.flexybayes_koine, 35
print.quorum::quorum_verdict, 36
print.quorum_independence, 36

quorum, 37
quorum(), *10*, *43*, *47, 48*
quorum_abstention, 38
quorum_abstention(), *4–6*, *26*, *28–30*, *33*, *42, 43*, *48*
quorum_compare_policies, 39
quorum_compare_policies(), *45–47*
quorum_draws, 40
quorum_draws(), *3, 4*, *26*, *29–31*, *33*, *42*

quorum_independence, 41
quorum_lens, 42
quorum_lens(), *27–31*, *33*, *37*, *41*, *50, 51*
quorum_manifest, 43
quorum_manifest(), *38*, *48*
quorum_policy, 44
quorum_policy(), *37–39*, *46*, *48*
quorum_policy_committed, 45
quorum_resolve, 46
quorum_resolve(), *10*, *25, 26*, *38, 39*, *45, 46*, *48*
quorum_verdict, 47
quorum_verdict(), *36*, *38*, *43*

read_fb_ensemble, 48
read_fb_ensemble(), *14*
register_ensemble_source, 49
register_ensemble_source(), *8*
register_kernr_surrogates, 49
register_kernr_surrogates(), *52*
register_lens, 50
register_pesto_surrogates, 52
register_pesto_surrogates(), *50*
register_surrogate, 52
register_surrogate(), *21, 22*, *50*, *52*

tmbstan::tmbstan(), *33*

verify_fb_ensemble, 53
verify_fb_ensemble(), *14*