Package: flexyBayes (via r-universe)

November 27, 2026

Title Bayesian Mixed Models with 'ASReml' Syntax via 'INLA' and 'brms'

Version 0.19.0

Description Specify a wide class of linear and generalized linear mixed models using familiar 'ASReml' or 'brms'-style formula syntax, and estimate them via 'INLA' or 'brms' ('Stan'). Returns results in formats compatible with the standard R modelling ecosystem including 'summary()', 'emmeans' and 'marginaleffects'. The package supports a range of responses from the exponential family (and a few beyond this), along with a few random effects covariance/residual structures including genotype-by-environment variances, per-level residual variances, and known-covariance genomic and pedigree matrices among others.

License MIT + file LICENSE

URL https://aagi-aus.github.io/flexyBayes/, https://github.com/AAGI-AUS/flexyBayes

BugReports https://github.com/AAGI-AUS/flexyBayes/issues

Depends R (>= 4.1.0)

Imports cli, data.table, generics, stats, methods, parallel, graphics, mgcv, Matrix, utils

Suggests brms, INLA, fst (>= 0.9.0), benchmarkme, coda, memuse, bayesplot, posterior, rstantools, emmeans, estimability, marginaleffects, insight, loo, testthat (>= 3.0.0), knitr, rmarkdown, agridat, lme4, pkgbuild, nlme, MASS, splines, sommer, sn, vdiffr, withr, callr, processx, ps

Additional_repositories 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

LazyData true

Config/testthat/edition 3

Config/testthat/parallel false

Config/rpkg/validationTier V2

Config/rpkg/fittingBackends INLA, brms

Config/roxygen2/version 8.1.0

X-schema.org-isPartOf
https://grdc.com.au/research/partnerships-and-initiatives/strategic-partnerships/aagi

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

Date/Publication 2026-11-03 21:02:37 UTC

RemoteUrl https://github.com/max578/flexyBayes

RemoteRef main

RemoteSha 2944c62c86f1756537d7e5499db809f0bf835b9a

Contents

anova.flexybayes . . . 4
as.data.frame.fb_plan . . . 5
as_draws_df.flexybayes . . . 5
augment.flexybayes . . . 7
backend_decision . . . 8
canonical_names . . . 9
cat_code . . . 10
coef.flexybayes . . . 11
coef.flexybayes_inla . . . 12
confint.flexybayes . . . 13
confint.flexybayes_brms . . . 14
confint.flexybayes_glm . . . 15
confint.flexybayes_inla . . . 15
emm_basis.flexybayes . . . 16
family.flexybayes . . . 17
family.flexybayes_inla . . . 17
fb_approx . . . 18
fb_backend_status . . . 19
fb_brms . . . 20
fb_complete_grid . . . 21
fb_cov . . . 22
fb_engine . . . 24
fb_from_asreml . . . 25
fb_from_brms . . . 27
fb_gblup_cv . . . 28
fb_inla . . . 29
fb_isolated_lapply . . . 31
fb_kfold . . . 32

fb_marginal_cov . . . 33
fb_plan . . . 35
fb_prior . . . 37
fb_refit . . . 40
fb_refusals . . . 42
fb_refuse . . . 43
fb_register_refusals . . . 44
fb_variance_draws . . . 45
fitted.flexybayes . . . 46
fitted.flexybayes_aggregated . . . 47
fitted.flexybayes_inla . . . 48
flexybayes . . . 48
flexybayes_stream . . . 61
formula.flexybayes . . . 63
formula.flexybayes_inla . . . 64
genomic_summary . . . 64
get_coef.flexybayes . . . 65
get_data.flexybayes . . . 66
get_predict.flexybayes . . . 67
get_vcov.flexybayes . . . 67
glance.flexybayes . . . 68
is_fb_approx . . . 69
is_fb_cov . . . 70
is_fb_engine . . . 70
logLik.flexybayes . . . 71
logLik.flexybayes_brms . . . 72
logLik.flexybayes_inla . . . 72
loo.flexybayes . . . 73
met_example . . . 74
model.matrix.flexybayes . . . 75
nobs.flexybayes . . . 76
ordinal_probabilities . . . 77
plot.flexybayes . . . 78
pp_check.flexybayes . . . 80
predict.flexybayes_brms . . . 81
predict.flexybayes_inla . . . 83
print.fb_approx . . . 85
print.fb_cov . . . 86
print.fb_engine . . . 86
print.fb_plan . . . 87
print.fb_predict_classify . . . 87
print.flexybayes . . . 88
print.flexybayes_aggregated . . . 89
print.flexybayes_brms . . . 89
print.summary.flexybayes . . . 90
prior_summary . . . 91
proceed . . . 92
ranef . . . 93

recover_data.flexybayes 94
residuals.flexybayes 95
residuals.flexybayes_aggregated 95
residuals.flexybayes_inla 96
set_coef.flexybayes 96
summary.fb_plan 97
summary.flexybayes 97
summary.flexybayes_aggregated 99
summary.flexybayes_glm 100
tidy.flexybayes 101
tidy.flexybayes_inla 102
update.flexybayes 104
validate_approximation 106
vcov.flexybayes 107
vcov.flexybayes_inla 108
vpredict 109

Index **111**

`anova.flexybayes` *Compare flexyBayes models on a plug-in information criterion*

Description

Ranks two or more fits by a DIC-shaped criterion built from the plug-in conditional log-likelihood of `logLik.flexybayes()` and the recorded parameter count. The criterion is approximate and the method says so on every print: it penalises the nominal parameter count, not an effective one, so it is a coarse ordering, not a model-selection procedure.

Usage

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

Arguments

`object`	A `flexybayes` fit supplying a conditional log-likelihood and a recorded parameter count.
`...`	Further `flexybayes` fits to compare against `object`.

Details

Every fit compared must supply both ingredients. A fit whose log-likelihood is `NA`, an INLA fit reports a marginal likelihood, not a conditional one, or which records no parameter count is refused by name, because ranking on an `NA` produces an ordering that looks computed and is not.

Value

Invisibly, a data frame with one row per model carrying the log-likelihood, parameter count, criterion value, and the difference from the best model. Printed as a side effect.

---

as.data.frame.fb_plan *Coerce an* <fb_plan> *to data.frame — one row, stable columns*

Description

Stable column ordering by the internal vector `.FB_PLAN_DF_COLS`. New fields are appended, and existing columns keep their order.

Usage

```
## S3 method for class 'fb_plan'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	An <fb_plan> object as returned by `fb_plan()`.
`row.names`	Ignored. Present for compatibility with the generic.
`optional`	Ignored. Present for compatibility with the generic.
`...`	Ignored. Present for compatibility with the generic.

Value

A one-row data.frame whose columns follow the internal `.FB_PLAN_DF_COLS` order, with one column per recorded plan field.

---

as_draws_df.flexybayes *Posterior draws from a flexyBayes fit*

Description

Methods for the **posterior** package's `as_draws()`, `as_draws_df()` and `as_draws_matrix()` generics, so a fitted model hands its posterior to the Bayesian workflow ecosystem, **posterior** summaries, **bayesplot** displays, anything reading a `draws` object, in the shape those tools expect.

Usage

```
## S3 method for class 'flexybayes'
as_draws_df(x, n_draws = 1000L, ...)

## S3 method for class 'flexybayes'
as_draws(x, n_draws = 1000L, ...)

## S3 method for class 'flexybayes'
as_draws_matrix(x, n_draws = 1000L, ...)
```

Arguments

x	A fitted flexybayes object from either active engine.
n_draws	Number of posterior draws to return, matching the default of fb_as_draws_simple(). Used on the INLA path, where the draws are sampled from the fitted approximation. Ignored on the brms path, which returns the draws the sampler kept.
...	Passed to the underlying draws extraction.

Details

The parameter names are canonical and the same on every engine: (Intercept), one entry per fixed-effect term, sigma for the residual standard deviation, and sd_<group> for the standard deviation of each random-effect group. Variance components are on the **standard-deviation scale** whatever the engine stored: INLA's precision hyperparameters are transformed on the way out, so sd_g is a draw of a standard deviation and not of a precision.

Value

A **posterior** draws object holding one column per canonical parameter: a draws_df from as_draws_df() and from as_draws(), a draws_matrix from as_draws_matrix().

Where the draws come from

On a **brms** fit the draws are the sampler's own: the kept post-warmup iterations, renamed. n_draws is ignored, because the number of draws was fixed when the model was sampled.

On an **INLA** fit there is no sampler. The fit is a nested Laplace approximation, and the draws are sampled *from that fitted approximation*, from the same distribution INLA::inla.posterior.sample() samples, reading the configuration store every flexyBayes INLA fit is built with (control.compute = list(config = TRUE)). Two consequences follow:

- The **fitting** is deterministic. The Laplace approximation draws no random numbers, so a seed passed to flexybayes() changes nothing about the fit itself.
- The **sampling** step draws from R's random number generator, so set.seed() before the call makes the draws repeatable. Monte-Carlo error in anything computed from them shrinks with n_draws, so raise n_draws before reading a small difference between two sets of draws as a difference between two posteriors.

See Also

fb_as_draws_simple() for the same draws as a plain named list, with no **posterior** dependency. canonical_names() for the name map itself. prior_summary() for the priors those draws were taken under. summary.flexybayes(), whose $converge slot carries the engine's own convergence diagnostics.

---

augment.flexybayes *Augment a flexyBayes fit with fitted values and residuals*

Description

Returns the model frame with two observation-level columns added: the posterior-mean fitted value and the response residual.

Usage

```
## S3 method for class 'flexybayes'
augment(x, data = NULL, ...)

## S3 method for class 'flexybayes_inla'
augment(x, data = NULL, ...)
```

Arguments

x	A flexyBayes fit (flexybayes or flexybayes_brms).
data	Optional data.frame to augment. Defaults to the data the model was fitted to.
...	Currently unused. Present for generic compatibility.

Value

The supplied (or original) data.frame with .fitted and .resid columns appended.

See Also

tidy.flexybayes()

Examples

```
# Not run: augment() appends fitted values and residuals to the data
# a sampled fit was built from, so the example needs a Stan compile.
# The fragment is complete and runs as written with a C++ toolchain.
## Not run:
set.seed(1)
dat <- data.frame(yield = rnorm(40), env = factor(rep(1:4, each = 10)))
fit <- flexybayes(yield ~ env, data = dat, backend = "brms",
                  chains = 1L, n_samples = 200L, warmup = 100L)
augment(fit)
```

```
## End(Not run)
```

---

backend_decision *Backend dispatch trace for a flexyBayes fit*

Description

Returns the dispatch trace recorded at fit time: which backend was selected, which gate checks ran, and why. Engine-agnostic, works on an `flexybayes_inla` or `flexybayes_brms` fit alike, reading `fit$extras$backend_decision` as populated by `.build_routing_decision()` during dispatch (see the top of this file).

Usage

```
backend_decision(fit)
```

Arguments

`fit`	A `flexybayes` object.

Value

A list holding the recorded dispatch trace. The first four components are present on every fit. The four routing-trace fields are present on fits from v0.3.6 onwards and `NULL` on earlier ones, for backward compatibility.

`backend` Character, one of `"inla"`, `"brms"`.

`path` Character: the dispatch path token.

`gate_checks` List or NULL: the `lgm_gate()` check trail (failures on refusal, capabilities on accept).

`reason` Character: the dispatch-decision rationale.

`preflight_summary` An <fb_preflight> object or NULL. Populated when `.fb_preflight()` ran (>1e5-row path). NULL on the small-data fast path.

`representation_plan` Named list of slim per-term entries (`term_id, representation_class, justification`) derived from `preflight_summary`. NULL when no preflight.

`rejected_routes` List of (`backend, reason`) pairs for the backends considered but not chosen. Empty for explicit user requests (the routing policy is bypassed when the backend is named directly).

`routing_policy_version` Character, e.g., `"stage5a_v1"`. Recorded for reproducibility: a policy change bumps this string.

Examples

```
# backend_decision() reads a slot recorded on an already-fitted
# object, so (unlike fb_plan()) an engine has to run first. Wrapped
# in \donttest{} and guarded on INLA so this does not fire on a
# machine without it.

if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))
  fit <- flexybayes(y ~ x + (1 | g), data = d, backend = "inla",
                    verbose = FALSE)
  bd <- backend_decision(fit)
  bd$backend   # "inla"
  bd$path      # the dispatch path token
  bd$reason    # why this backend was chosen
}
```

`canonical_names` *Canonical parameter-name view for a flexyBayes fit*

Description

Returns the backend-native -> canonical parameter-name map for a fit, plus per-parameter value transforms where applicable (e.g., the INLA precision-to-SD `sqrt(1/prec)` transform applied to hyperparameter draws before they are renamed). The canonical convention follows brms (`(Intercept)`, `<term>`, `sd_<group>`, `sigma`, `r_<group>[<level>]`).

Usage

```
canonical_names(fit, drop = FALSE, ...)

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

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

Arguments

`fit`	A `flexybayes_inla` or `flexybayes_brms` object.
`drop`	Logical: if `TRUE` (default `FALSE`), drop backend-native names that are not in the registered map (e.g., INLA's `Predictor.<i>` latent-predictor draws). When `FALSE`, un-mapped names appear in the returned `$unmapped` element.
`...`	Additional arguments (ignored by current methods).

Details

On `flexybayes_inla` and `flexybayes_brms` fits, the per-backend mapper registered at package load (`inla` or `brms`) drives the resolution, the returned map is cached on `fit$extras$canonical_map` for fast repeated access.

Value

A list holding the canonical-name map and its provenance.

`map` Named character vector keyed by backend-native parameter name with canonical name as the value.

`transform` Named list of `function(x) -> x'` transforms keyed by canonical name. Empty list when no transforms apply.

`source` Character: `"registry"`, `"user"`, `"registry_fallback_verbatim"`, or `"legacy_inferred"`.

`unmapped` Character vector of backend-native names not in the map (when `drop = FALSE`).

`prior_parametrization` Character, present only on aggregated fits, naming which prior the fit ran under. `"per_row_equivalent"` is the legacy scalar bridge, whose precision prior makes the aggregated posterior match the per-row posterior to numerical precision. `"package_default"` is the automatic half-normal-on-SD prior, which claims no such equivalence on this route. `"custom"` is an explicit prior from the caller.

Examples

```
# The map is engine-specific, so it needs a fit to read.

if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))
  fit <- flexybayes(y ~ x + (1 | g), data = d, backend = "inla",
                    verbose = FALSE)
  cn <- canonical_names(fit)
  head(cn$map)
}
```

`cat_code` *Emit the generated backend code for a deferred review object*

Description

Writes the backend code carried by a `<flexybayes_review>` object (Stan code via `brms::stancode()` for `flexybayes()` / `fb_brms()`) to a connection.

Usage

```
cat_code(x, ...)

## S3 method for class 'flexybayes_review'
cat_code(x, file = stdout(), ...)
```

Arguments

x	A <flexybayes_review> object carrying the generated backend code.
...	Method-specific arguments. The flexybayes_review method accepts file, a connection defaulting to stdout().
file	A connection to write the code to, defaulting to stdout().

Value

Invisibly, the code as a character vector. Called for the code it writes to file.

Deprecated

cat_code() is deprecated in flexyBayes 0.13.0 and will be removed in a later release. The generated code is the code element of the review object, and flexybayes(..., return_code = TRUE) returns it directly. A call warns with a condition of class flexybayes_deprecated.

Examples

```
# Generating the code goes through brms::make_stancode(), so brms must be
# present.
if (requireNamespace("brms", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))
  code <- flexybayes(y ~ x + (1 | g), data = d, backend = "brms",
                     return_code = TRUE)
  cat(code)
}
```

---

coef.flexybayes *Extract coefficients from a flexyBayes fit*

Description

Returns the fixed effects by default, and the random-effect predictions or the unobserved design cells on request. An ASReml user reaches for coef(fit)$random. The equivalent here is coef(fit, what = "random"), which is what ranef() calls.

Usage

```
## S3 method for class 'flexybayes'
coef(object, what = c("fixed", "random", "missing", "all"), ...)

## S3 method for class 'flexybayes_ordinal'
coef(object, what = c("fixed", "thresholds", "random", "missing", "all"), ...)
```

Arguments

`object`	A fitted `flexybayes` object of any backend.
`what`	Which part of the fit to return. `"fixed"` (the default) is the fixed-effect posterior means. `"random"` is one data frame per grouping factor, with the columns `group`, `level`, `estimate`, `std.error`, `conf.low` and `conf.high`. A grouping factor carrying more than one effect names the effect in `level`. `"missing"` is the table of unobserved design cells, in the columns `row`, `estimate`, `std.error`, `conf.low`, `conf.high` plus any design index variables the fit recorded. `"all"` is the named list of all three. On a fit with `family = "cumulative"` or `"cratio"`, `"thresholds"` is the posterior means of the thresholds, named `threshold[<lower>|<upper>]`, and `"all"` carries them as a fourth element.
`...`	Ignored, present for compatibility with the generic.

Details

The default is the historical return, a named numeric vector of the fixed effects, so every caller that treats `coef(fit)` as a numeric vector keeps working, including the **emmeans** and **marginaleffects** seams, `predict()`, `vcov()` and the tidiers.

Value

For `what = "fixed"`, a named numeric vector of the fixed effects' posterior means on the treatment-contrast basis, named for the design-matrix columns, empty when the model carries no fixed effects. For `"random"`, a named list of data frames, empty when the model carries no random terms. For `"missing"`, a data frame with zero rows when every response was observed. For `"all"`, a list with the elements `fixed`, `random` and `missing`.

See Also

`ranef()` for the random-effect table on its own, `summary.flexybayes()` whose `random` and `missing` slots are the same two objects, and `nobs()` for the design and observed counts.

`coef.flexybayes_inla` *Coefficients of a per-row INLA fit*

Description

Posterior means of the fixed effects, read from the INLA fit's `summary.fixed` slot (treatment-contrast basis). These are the coefficients consumed by `emmeans::emmeans()` and `marginaleffects::predictions()` via the flexyBayes support methods.

Usage

```
## S3 method for class 'flexybayes_inla'
coef(object, what = c("fixed", "random", "missing", "all"), ...)
```

Arguments

object	A flexybayes_inla fit.
what	Which part of the fit to return: "fixed" (the default), "random", "missing" or "all". See coef.flexybayes() for the shape of each.
...	Ignored. Present for compatibility with the generic.

Details

An INLA fit carries no $glm slot, so the fixed vector is read here and not inherited. Every other value of what is resolved by the shared body coef.flexybayes() uses, so the two engines answer the same question the same way.

A factor with a non-syntactic level (for example a level containing a space, "low N") is legalised internally (make.names()) before the INLA emit, so the fit itself never dies on it. This method restores the user's own, unlegalised label in the names it returns, as do ranef(), summary() and predict(classify =) on the same fit.

Value

For the default what = "fixed", a named numeric vector of fixed-effect posterior means. Otherwise as documented for coef.flexybayes().

---

confint.flexybayes *Credible intervals for the fixed effects of a flexyBayes fit*

Description

Returns posterior quantile-based credible intervals, not frequentist confidence intervals: the bounds would be empirical quantiles of the fixed-effect posterior draws the fit carries.

Usage

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

Arguments

object	A flexybayes fit.
parm	Character vector of parameter names to return, or NULL (the default) for every fixed effect.
level	Credible level for the interval, as a proportion. The default 0.95 returns the 2.5th and 97.5th posterior percentiles.
...	Ignored, present for compatibility with the generic.

Details

No active engine stores its posterior in the shape this bare fallback method reads. A fit from an active engine reaches its own method, `confint.flexybayes_inla()` for INLA, `confint.flexybayes_brms()` for brms: so this method always refuses by name instead of returning an empty interval matrix, which would read as "no fixed effects" and not "this fit cannot answer the question".

Value

Does not return: raises the classed `fit_lacks_posterior_draws` refusal.

---

confint.flexybayes_brms

Credible intervals on the brms path

Description

Uses the brms posterior draws directly (the parent `confint.flexybayes` refuses unconditionally with `fit_lacks_posterior_draws`, since no active engine reaches it without its own override). Returns quantile-based credible bounds over the `b_<term>` rows. Row names are stripped of the brms `b_` prefix to align with `coef()`.

Usage

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

Arguments

`object`	A `flexybayes_brms` object.
`parm`	Subset of fixed-effect names to return (NULL = all).
`level`	Credible level (default 0.95).
`...`	Ignored. Present for compatibility with the generic.

Value

A numeric matrix with one row per fixed-effect term and two columns holding the lower and upper credible bounds at `level`. Row names are the term names with the brms `b_` prefix removed.

confint.flexybayes_glm

Credible intervals for flexybayes_glm

Description

Posterior quantile credible intervals, computed from the fixed-effect draws.

Usage

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

Arguments

object	A flexybayes_glm object, reached as fit$glm on any fitted flexybayes object.
parm	A character vector naming the parameters to report, or NULL (the default) for every fixed effect.
level	A single numeric giving the credible level, defaulting to 0.95.
...	Ignored. Present for compatibility with the generic.

Details

Where the draws are unavailable the method falls back to the normal-approximation interval $\hat{\beta} \pm z_{\alpha/2}\mathrm{sd}$, and marks the returned matrix with `attr(, "interval_basis") == "normal_approximation"`. The two agree only for a symmetric posterior. On a skewed one they can differ materially, which is why the basis is reported instead of assumed. Earlier versions documented this method as quantile-based while always returning the approximation.

Value

A two-column matrix of interval bounds, carrying an `interval_basis` attribute of `"posterior_quantile"` or `"normal_approximation"`.

confint.flexybayes_inla

Credible intervals for the fixed effects of a per-row INLA fit

Description

Quantiles of INLA's own posterior marginals for the fixed effects, not frequentist confidence intervals and not a normal approximation to them. The bounds come from `INLA::inla.qmarginal()` applied to the marginal densities the fit stores, so any credible level is available and not only the 0.95 that `summary.fixed` tabulates.

Usage

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

Arguments

object	A flexybayes_inla fit carrying fixed-effect marginals.
parm	Character vector of coefficient names to return, or NULL (the default) for every fixed effect.
level	Credible level for the interval, as a proportion. The default 0.95 returns the 2.5th and 97.5th posterior percentiles.
...	Ignored, present for compatibility with the generic.

Details

Before 0.9.0 an INLA fit did not inherit the flexybayes parent class, so confint() on one reached stats::confint.default and failed on a missing vcov contract. This method is the INLA sibling of confint.flexybayes().

Value

A matrix with one row per fixed effect and two columns holding the lower and upper credible bounds, named for the percentiles used.

---

emm_basis.flexybayes *emmeans support: estimation basis (default method)*

Description

The bare flexybayes method: reached by any fit whose engine has no more specific override, which today is the brms backend.

Usage

```
## S3 method for class 'flexybayes'
emm_basis(object, trms, xlev, grid, ...)

## S3 method for class 'flexybayes_inla'
emm_basis(object, trms, xlev, grid, ...)
```

Arguments

object	A flexybayes_inla fit.
trms	Fixed-effect terms supplied by emmeans.
xlev	Factor levels supplied by emmeans.
grid	Reference grid supplied by emmeans.
...	Ignored. Present for compatibility with the generic.

Value

A list with `X`, `bhat`, `nbasis`, `V`, `dffun`, `dfargs`.

`family.flexybayes` *Extract model family*

Description

Extract model family

Usage

```
## S3 method for class 'flexybayes'
family(object, ...)
```

Arguments

`object`	A fitted `flexybayes` object of any backend.
`...`	Ignored, present for compatibility with the generic.

Value

The response family the model was fitted under, as the `family` object the fit recorded at emit time, so it names the family and link the engine actually used, not the string the caller passed.

`family.flexybayes_inla`
Response family of a per-row INLA fit

Description

Response family of a per-row INLA fit

Usage

```
## S3 method for class 'flexybayes_inla'
family(object, ...)
```

Arguments

`object`	A `flexybayes_inla` fit.
`...`	Ignored. Present for compatibility with the generic.

Value

A list with `family` and `link` entries.

`fb_approx` *Construct an approximation-scheme specification*

Description

Names an approximation scheme and its tuning, validated against the package's locked approximation registry. The result is consumed where an approximate representation is accepted, most directly in a smooth term, `s(x, representation = fb_approx("low_rank_smooth", rank = 5L))`.

Usage

```
fb_approx(scheme, ...)
```

Arguments

`scheme`	Character(1): the approximation-scheme name. Must be a scheme registered in the approximation registry. An unregistered name is refused with the supported vocabulary.
`...`	Scheme-specific tuning carried on the object, for example `rank` for `"low_rank_smooth"`.

Details

The only scheme with a smooth-basis fitting path in this release is `"low_rank_smooth"`, a rank-`K` principal-component truncation of the smooth basis. Its bias is the relative squared Frobenius error of the truncation (Wood, 2017, chapter 5). `validate_approximation()` reports the realised capture against the threshold for a fitted model.

Value

An `fb_approx` object: a classed list with the `scheme` and the tuning kwargs as elements, and a `bias_bound_promise` attribute describing the scheme's bias bound.

See Also

`validate_approximation()`, `fb_engine()`

Examples

```
a <- fb_approx("low_rank_smooth", rank = 5L)
a$scheme
a$rank
```

`fb_backend_status` *Report inference-backend readiness*

Description

Checks which inference backends flexyBayes can route to are installed and usable in the current session, returning a small table you can inspect before fitting. The check is read-only: it probes package availability without building a model or starting a fit.

Usage

```
fb_backend_status(deep = TRUE)
```

Arguments

`deep`	Logical. Currently unused (reserved for a future backend whose readiness probe is expensive enough to need an opt-out). Default `TRUE`.

Details

`installed` records whether the backend's R package is present. `usable` records whether the backend can actually run a fit now.

Value

A data frame of class `fb_backend_status` with one row per backend and the columns `backend`, `installed` (logical), `usable` (logical), and `note` (a human-readable status, including the install command when a backend is absent). A `print` method renders it as a readiness table.

See Also

`flexybayes()` for the universal entry, and the `fb_inla()` / `fb_brms()` single-engine pins.

Examples

```
fb_backend_status()
```

fb_brms *Fit a flexyBayes model via the brms (Stan) engine*

Description

Engine pin: fits the model with Stan through brms only. This is sugar for `flexybayes(..., backend = "brms")` and accepts the same arguments and grammars, an ASReml `fixed` / `random` / `residual` specification or a brms-style bar-grouped formula (see `flexybayes()` for the full argument list). flexyBayes builds the intermediate representation, translates the prior, calls `brms::brm()`, and wraps the result. The live `brmsfit` is available on the `$brms` slot for brms's own posterior tooling (`loo()`, `posterior_predict()`, `bayes_factor()`).

Usage

```
fb_brms(...)
```

Arguments

`...`	Arguments passed to `flexybayes()` (e.g. `formula` / `fixed`, `random`, `residual`, `data`, `family`, `prior`, `syntax`). The `backend` argument is pinned to `"brms"`. A conflicting `backend` value raises a structured refusal (the redundant `backend = "brms"` is accepted). The pre-v0.5.0 `formula =` argument is remapped to the universal entry's model-spec slot for call-compatibility.

Details

flexyBayes has no brms lowering for some ASReml structures: `fa()`, the autoregressive fields, a blocks or low-rank `vm()` / `ped()` carrier, and a `low_rank` smooth approximation. Such a model raises a structured refusal naming the term. Several of these fit on INLA through `fb_inla()`. The capability table in the README lists what each engine fits.

Value

An object of class `"flexybayes_brms"` (a subclass of `"flexybayes"`) carrying the live `brmsfit` on `$brms`. See `flexybayes()` for the shared structure.

Deprecated

`fb_brms()` is deprecated as of 0.13.0. Use `flexybayes(..., backend = "brms")`.

See Also

`flexybayes()` and `fb()` for the universal entry that picks a backend. `fb_inla()` for the other active engine pin and `fb_from_brms()` for building a brms-grammar IR.

Other flexyBayes engine pins: `fb_inla()`

Examples

```
# Held back from the default example run because the brms route compiles
# a Stan program before it samples, and the compile alone takes far
# longer than an example should. The sampler settings are the smallest
# that reach an effective sample size the fit does not warn about, and
# the seed makes that reproducible. On a single chain they cost about
# half a second next to the compile.

if (requireNamespace("brms", quietly = TRUE) &&
    requireNamespace("lme4", quietly = TRUE)) {
  data(sleepstudy, package = "lme4")
  fit <- flexybayes(
    Reaction ~ Days + (1 | Subject), data = sleepstudy,
    backend = "brms", chains = 1, n_samples = 4000, warmup = 1000,
    seed = 1
  )
  coef(fit)
}
```

---

fb_complete_grid *Complete a design grid before fitting*

Description

Reinstates the design cells that are absent from a data frame, returning the Cartesian product of the index variables with the response set to NA on every cell that was added. A field book already records a lost plot this way (one row per sown plot, with the design columns filled in and the yield missing) so this helper is for the trials where the plot is missing from the file altogether.

Usage

```
fb_complete_grid(data, index = ~row * col, response, unused_level = NULL)
```

Arguments

data	A data frame holding one row per recorded design cell.
index	A one-sided formula naming the design index variables, for example ~ row * col. The operators are ignored. What is read is the set of variables, whose observed levels are crossed.
response	Name of the response column, as a single string. The reinstated cells carry NA there.
unused_level	A single string, or NULL (the default). When supplied, design factors that vary across the trial are filled with this level on the reinstated cells and a warning names every column so filled. NULL refuses instead.

Details

The completed frame is what `na_action = "augment"` (the default) then carries through the fit, keeping the index set a structured covariance is built over intact. A separable `ar1(row):ar1(col)` field needs one observation per node of the array, which is the same requirement ASReml states for `residual = ~ ar1:ar1`.

A cell can only be reinstated when every other model column at it is determined: it is one of the index variables, or it takes one value across the whole trial. A column that varies (a variety label, a block) is a real quantity nobody recorded at the absent cell, and the default is to refuse instead of write one in. `unused_level` opens that door explicitly, filling the varying design factors with a level of your naming and warning with every column it wrote to. It is ASReml's nin89 LANCER coding, made a decision with a name on it, not a default. A varying *numeric* column is refused whatever `unused_level` says: a measured covariate has no level to name.

Value

A data frame: `data` with one row appended per absent combination of the index variables, the response `NA` on each, and the original row order preserved ahead of them. Returned unchanged when the grid is already complete.

See Also

`flexybayes()`, whose `na_action` argument carries the completed grid into the fit.

Examples

```
set.seed(1L)
trial <- expand.grid(row = factor(1:4), col = factor(1:3))
trial$yield <- rnorm(nrow(trial))
holed <- trial[-c(2L, 7L), ]
padded <- fb_complete_grid(holed, ~ row * col, response = "yield")
nrow(padded)
sum(is.na(padded$yield))
```

---

fb_cov *Construct a structured-covariance carrier*

Description

Wraps a known covariance, precision, Cholesky factor, block-diagonal list, or low-rank factor in a classed carrier object that the structured random-effect terms consume, most directly `vm(geno, cov = fb_cov(G, type = "chol"))`. It is the v0.4.0 constructor-noun replacement for the legacy `vm()` keyword forms (`chol = `, `precision = `, `blocks = `, `low_rank_factor = `), which deprecate at this release and are removed at v0.5.0.

Usage

```
fb_cov(M, type = "dense", levels = NULL, scheme = NULL, ...)
```

Arguments

M	The covariance carrier. For type = "blocks" a base-R list of K square covariance matrices. Otherwise a square numeric matrix (base-R or **Matrix**). Inside a formula this argument names a matrix resolved through known_matrices.
type	Character(1): the carrier type, one of "dense", "chol", "precision", "blocks", "low_rank". Defaults to "dense".
levels	Optional character vector of grouping-factor level labels the carrier's rows / columns align to. Carried as metadata. the fit-time validator checks alignment against the fitted factor.
scheme	Character(1), required when type = "low_rank": the name of a registered approximation scheme (see validate_approximation()). Ignored for the exact types.
...	Reserved for forward carrier-type options. Currently carried verbatim on the object.

Details

The carrier's type selects how the downstream emit path derives a covariance square root. "dense" takes the lower Cholesky of the supplied covariance. "chol" uses the supplied factor directly. "precision" inverts via the Cholesky of the precision matrix. "blocks" assembles a block-diagonal covariance from a list of per-block matrices. "low_rank" pairs a rank-K factor with a registered approximation scheme (reserved at v0.4.0, the carrier vocabulary is active and validated, while the approximate-covariance fit route activates in a subsequent release).

A "blocks" carrier is one covariance with one standard deviation, and flexyBayes refuses it by name on both engines: its brms route has no blocks carrier, and its INLA route would fit each block with its own standard deviation. The precision carrier of Matrix::bdiag(lapply(blocks, solve)) states the same covariance on INLA, and the dense carrier of the block-diagonal matrix states it on brms.

Inside a model formula the carrier is written inline, the matrix argument names a matrix passed through known_matrices, exactly as the legacy keyword forms did. The construction-time check is a light structural probe (shape, lower-triangular pattern, symmetry, block count). The full level-aware validation runs at fit time against the grouping factor.

Value

An fb_cov object: a classed list with the matrix M, the type, optional levels, and (for low-rank) scheme as elements, plus representation_class and validation_summary attributes.

See Also

fb_approx(), fb_engine(), fb_prior(), validate_approximation()

Examples

```
set.seed(1L)
G <- crossprod(matrix(rnorm(9L), 3L, 3L))
fb_cov(G, type = "dense")
```

```
L <- t(chol(G))
fb_cov(L, type = "chol")
```

`fb_engine` *Construct an inference-engine specification*

Description

Names a concrete inference engine and its tuning options. The result is passed as the `backend` argument of the fitting verbs: `flexybayes(..., backend = fb_engine("brms", chains = 4L))`. The bare string form (`backend = "brms"`) remains valid and is equivalent to the default `fb_engine()` for that engine.

Usage

```
fb_engine(name, opts = list(), ...)
```

Arguments

`name`	Character(1): the engine, one of `"inla"`, `"brms"`.
`opts`	Named list of tuning options. Recognised names are `n_samples`, `warmup`, `chains` and `adapt_delta`, described under "Recognised options" below. An unrecognised name is an error.
`...`	Tuning options given individually, merged into `opts` (e.g. `fb_engine("brms", chains = 4L)`).

Details

`name` and the derived `paradigm` are closed vocabularies. `"auto"` is a routing directive, not an engine. Pass `backend = "auto"` directly for automatic routing.

Value

An `fb_engine` object: a classed list with elements `name`, `paradigm` (one of `mcmc`, `laplace`, `vb`, `map`), `toolchain_status` (one of `ready`, `requires_install`, `unavailable`), and `opts`.

Recognised options

All four are sampler controls, read by the routes that emit through Stan and ignored by the rest. An unrecognised name is an error at construction, not a silently dropped argument.

`n_samples` Post-warmup draws to keep, per chain.

`warmup` Warmup iterations, per chain.

`chains` Number of chains.

`adapt_delta` Target acceptance probability for Stan's step-size adaptation, between 0 and 1. Raising it makes the sampler take smaller steps, which costs time and removes the divergent transitions a smaller step would have avoided. brms's own default is 0.8.

Deprecated

`fb_engine()` is deprecated as of 0.13.0. Pass the backend and engine settings directly to `flexybayes()`.

See Also

`fb_approx()`

Examples

```
plan <- flexybayes(y ~ x, data = data.frame(y = 1:4, x = 4:1),
                   backend = "brms", chains = 4L, plan = TRUE)
plan$backend_chosen
```

---

fb_from_asreml *Ingest an ASReml-format model specification into the flexyBayes IR*

Description

Parses an ASReml-style `fixed` / `random` / `residual` specification into a `fb_terms` object, flexyBayes's backend-agnostic intermediate representation (IR) of a model. The IR is what every engine emits from, so building it explicitly lets a power user inspect the parsed model, cache it, or hand it to a fitting verb. Most users never call this directly: `flexybayes()` and `fb()` build the IR internally from the same arguments. Argument names and defaults mirror `flexybayes()` one-for-one, so this is a drop-in for that function's parsing step.

Usage

```
fb_from_asreml(
  fixed,
  random = NULL,
  residual = NULL,
  data,
  family = "gaussian",
  link = NULL,
  weights = NULL,
  known_matrices = list(),
  prior = NULL,
  prior_fixed_sd = 100,
  prior_vc_sd = 1,
  rcov
)
```

Arguments

`fixed`	Two-sided formula: `response ~ fixed_effects`.
`random`	One-sided formula `~ random_terms` (ASReml syntax), or `NULL`.

residual	One-sided formula ~ residual_structure, or NULL. NULL defaults to iid residuals (list(list(type = "units"))), matching flexybayes().
data	A data.frame containing every variable referenced.
family	Character family name (gaussian, binomial, binary, poisson, negative_binomial, negbinom, gamma, beta, and, for brms only, hurdle_gamma and lognormal), or a family object, which goes through the same family gate and supplies its own link.
link	Character link override, or NULL for the family default.
weights	Optional numeric vector of length nrow(data). When non-NULL, mapped to a single addition term with type = "weights".
known_matrices	Named list of known matrices (e.g. list(Gmat = G)). Names are recorded in data_summary$known_matrices.
prior	An fb_prior() object, or NULL. NULL falls back to the scalar priors prior_fixed_sd / prior_vc_sd.
prior_fixed_sd	Numeric SD for fixed-effect normal priors when prior is NULL.
prior_vc_sd	Numeric hyperparameter for the variance-component priors when prior is NULL.
rcov	Defunct in flexyBayes 0.9.0. The ASReml 3 name for the residual-structure argument, renamed to residual in ASReml 4. supplying it now raises an error. Use residual instead.

Value

An fb_terms object with source = "asreml".

Deprecated

fb_from_asreml() is deprecated as of 0.13.0. Use fb_plan() to inspect the backend-neutral model plan without fitting it.

See Also

flexybayes() and fb() for the universal fitting entry. fb_from_brms() for the other ingest dialect.

Other flexyBayes ingest adapters: fb_from_brms()

Examples

```
set.seed(1L)
df <- data.frame(
  yield = rnorm(20),
  geno  = factor(rep(letters[1:4], 5)),
  env   = factor(rep(c("a", "b"), 10))
)
plan <- fb_plan(yield ~ env, random = ~ geno, data = df)
class(plan)
```

`fb_from_brms` *Ingest a brms-format formula into the flexyBayes IR*

Description

Parses a brms / lme4-style two-sided formula (`response ~ fixed + (1 | g)`) into a `fb_terms` object – flexyBayes's backend-agnostic intermediate representation (IR). The IR is what every engine emits from, so building it explicitly lets a power user inspect the parsed model, cache it, or hand it to a fitting verb. Most users never call this directly: `fb()` and `flexybayes()` detect a brms-style `formula` argument and build the IR internally.

Usage

```
fb_from_brms(
  formula,
  data,
  family = "gaussian",
  link = NULL,
  prior = NULL,
  weights = NULL,
  prior_fixed_sd = 100,
  prior_vc_sd = 1,
  carry_n_rows = NULL,
  ...
)
```

Arguments

`formula`	A base-R formula (recommended) or a `brmsformula` object. `brmsformula` support requires **brms** to be installed.
`data`	A data.frame containing every referenced variable. May be `NULL` only on the advanced metadata-only path (see `carry_n_rows`).
`family`	Character family name or a base-R `family()` object.
`link`	Character link override, or `NULL` for the family default.
`prior`	An `fb_prior()` object, or `NULL`.
`weights`	Optional numeric weights vector of length `nrow(data)`, mapped to a single addition term with `type = "weights"`.
`prior_fixed_sd`	Numeric SD for fixed-effect normal priors when `prior` is `NULL`.
`prior_vc_sd`	Numeric hyperparameter for the variance-component priors when `prior` is `NULL`.
`carry_n_rows`	Advanced: a positive integer enabling the metadata-only IR path (`data = NULL`). The formula's variables are realised as a one-row stand-in data frame and the row count is recorded as `carry_n_rows`, for stress-testing the preflight layer without materialising the full data. Leave `NULL` for ordinary use.
`...`	Reserved for future brms-ingest options (specials, autocor handling); currently unused.

Details

Linear fixed effects, random intercepts (`(1 | g)`, `(1 || g)`) and uncorrelated random intercept+slope (`(x || g)`) are supported; correlated random slopes (`(x | g)`), smoothers, Gaussian processes and autocorrelation terms refuse at ingest with a structured message.

Value

An `fb_terms` object with `source = "brms"`.

Deprecated

`fb_from_brms()` is deprecated as of 0.13.0. Use `fb_plan()` to inspect the backend-neutral model plan without fitting it.

See Also

`fb()` and `flexybayes()` for the universal fitting entry; `fb_from_asreml()` for the other active dialect.

Other flexyBayes ingest adapters: `fb_from_asreml()`

Examples

```
set.seed(1L)
df <- data.frame(
  y = rnorm(30),
  x = rnorm(30),
  g = factor(rep(letters[1:5], 6))
)
plan <- fb_plan(y ~ x + (1 | g), data = df)
class(plan)
```

---

fb_gblup_cv *Genomic-prediction accuracy by cross-validation*

Description

Estimate how accurately a genomic BLUP predicts the performance of genotypes whose phenotypes are held out. For each fold the variance components are estimated by REML on the training genotypes, the held-out breeding values are predicted from the genomic relationship matrix ($\hat{u}_V = K_{VT}(K_{TT} + \delta I)^{-1}(y_T - X_T\hat{\beta})$), and the predictions are compared with the observed phenotypes. The main metric is *prediction accuracy*, the correlation between predicted and observed across the held-out genotypes. *bias* is the slope of observed on predicted (1 is unbiased).

Usage

```
fb_gblup_cv(y, K, X = NULL, folds = 5L, repeats = 1L, seed = NULL, tol = 1e-08)
```

Arguments

y	Numeric vector of phenotypes, one per genotype, aligned to the rows of K.
K	An n x n genomic (or pedigree) relationship matrix.
X	Optional n x p fixed-effect design matrix (an intercept is used when NULL).
folds	Number of cross-validation folds (default 5).
repeats	Number of independent fold partitions to average over (default 1). Repeated CV stabilises the estimate.
seed	Optional integer seed for reproducible fold assignment.
tol	PSD / symmetry tolerance for the spectral decomposition.

Value

An fb_gblup_cv object: accuracy (pooled predicted-observed correlation), accuracy_per_fold, bias (pooled observed-on- predicted slope), predicted (out-of-fold predictions in input order, averaged over repeats), h2 (REML heritability on the full data), and metadata.

Examples

```
set.seed(1)
n <- 80L; m <- 400L
Z <- matrix(rbinom(n * m, 2L, 0.3), n, m)
Zc <- scale(Z, scale = FALSE)
G <- tcrossprod(Zc) / m + diag(n) * 1e-3
u <- as.vector(t(chol(G)) %*% rnorm(n))
y <- u + rnorm(n)
cv <- fb_gblup_cv(y, G, folds = 5L, seed = 1L)
cv$accuracy
```

---

fb_inla *Fit a flexyBayes model via the INLA engine*

Description

Engine pin: fits the model with INLA (integrated nested Laplace approximation) only. This is sugar for flexybayes(..., backend = "inla") and accepts the same arguments and grammars (an ASReml fixed / random / residual specification or a brms-style bar-grouped formula, see flexybayes() for the full argument list). The model must be latent-Gaussian feasible. If it is not, the shared lgm_gate() raises a structured refusal naming the offending term, exactly as flexybayes(backend = "inla") does.

Usage

```
fb_inla(...)
```

Arguments

`...`	Arguments passed to `flexybayes()` (e.g. `fixed`, `random`, `residual`, `data`, `family`, `prior`, `syntax`). The backend argument is pinned to `"inla"` and must not be supplied.

Details

Sampling-control arguments (`n_samples`, `warmup`, `chains`, `seed`, `control`, `mcmc_verbose`) are accepted for call-compatibility with the other engine pins but are inert under INLA's deterministic Laplace approximation. `seed` and `control` say so when supplied: the fit is repeatable without a seed, because nothing in the approximation is drawn.

Value

An object of class `"flexybayes"` (specifically a `flexybayes_inla` fit). See `flexybayes()` for the structure.

Deprecated

`fb_inla()` is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Call `flexybayes(..., backend = "inla")`, which takes the same arguments. A call warns with a condition of class `flexybayes_deprecated`.

See Also

`flexybayes()` for the universal entry that picks a backend. `fb_from_asreml()` / `fb_from_brms()` for the ingest layer.

Other flexyBayes engine pins: `fb_brms()`

Examples

```
set.seed(1L)
df <- data.frame(
  yield = rnorm(40),
  geno  = factor(rep(letters[1:8], 5)),
  env   = factor(rep(c("a", "b"), 20))
)

if (requireNamespace("INLA", quietly = TRUE)) {
  fit <- flexybayes(yield ~ env, random = ~ geno, data = df,
                    backend = "inla")
}
```

`fb_isolated_lapply` *Run jobs, each in its own R process*

Description

`fb_isolated_lapply()` is `lapply()` with each call of `fn` made in a separate R process started through callr, so that a process ended by the operating system or by a crash in compiled code is reported as a refusal and the session goes on. It is the mechanism behind `options(flexyBayes.isolation = "process")`, offered to packages that extend flexyBayes for their own loops of model fits.

Usage

```
fb_isolated_lapply(jobs, fn, plan = list())
```

Arguments

`jobs`	A list or vector, `fn` is called on each element.
`fn`	A function of one element of `jobs`.
`plan`	A list: • `workers`: how many processes run at once, a whole number of one or more (default 1). • `threads_per_worker`: the OpenMP threads each process may use, or `NA` (default) to leave them as the session has them. A process also runs one chain at a time when this is set. • `warm_first`: `TRUE` runs the first job alone before the others start, so that a Stan program compiled by it is cached for the rest (default `FALSE`). • `engine`: the engine the jobs fit with, named in a refusal (default `"none"`). • `entry_point`: the function the user called, named in a refusal (default `"fb_isolated_lapply()"`).

Details

Each process loads flexyBayes, and the package whose namespace `fn` is defined in, from where this session loaded them, and starts from this session's flexyBayes, brms, cmdstanr and INLA settings, its environment variables and its random-number state. Messages, warnings and printed output come back and are replayed in job order, never in completion order. A job that fails stops new jobs from starting. The jobs before it are waited for, and the lowest failing job's error is raised. With one worker, each process starts from the random-number state the previous one left, as `lapply()` would, with several, each starts from the session's state and the session's state is left where it was.

A process that ends without returning raises the refusal `isolated_process_died`, which names the engine, the entry point, the job and the signal or exit status.

Value

A list of the values `fn` returned, in the order of `jobs`.

See Also

fb_refit(), the refit that such jobs usually run.

Examples

```
fb_isolated_lapply(1:2, function(i) i^2, list(workers = 2L))
```

---

fb_kfold *Exact K-fold cross-validation of an ordinal fit*

Description

Refits the model once per fold with that fold's responses held out, and scores each held-out group by its log predictive density, the group's rows taken jointly. With group naming the plot, a plot is scored by the probability of all of its plants' categories together, which is how ordinal models of sub-sampled field trials are compared when leave-one-out approximations are unreliable.

Usage

```
fb_kfold(fit, folds, group = NULL, n_draws = 1000L, seed = NULL, ...)
```

Arguments

fit	An ordinal fit, from flexybayes() with family = "cumulative" or "cratio".
folds	The fold of each row of the data the fit was given: a vector with one label per row and at least two distinct labels.
group	The name of a column of the fit's data whose levels are scored jointly, for example the plot, or NULL (the default) to score each row on its own. The rows of a group must share a fold.
n_draws	The number of posterior draws per fold. Default 1000.
seed	An integer seed, or NULL (the default) to use the session's random number stream. Fold f uses seed + f.
...	Further arguments of fb_refit() for each fold's refit, for example control on brms.

Details

On INLA a fold's responses are set to missing and its rows kept, so every latent node, an AR1 field's included, stays in the model, and the held-out linear predictors are drawn from the refit's joint posterior. On brms the model is refitted to the other folds' rows, and a held-out group's random effect at a level the refit did not see is drawn from its fitted variance (brms's allow_new_levels = TRUE with sample_new_levels = "gaussian").

The score of group g is log(mean over draws s of prod over its rows i of P(y_i | draw s)), the elpd is the sum over groups, and its standard error is sqrt(G) * sd(score) over the G groups. Rows with a missing response are not scored.

Every category must keep a training observation in every fold, or the refit would have fewer thresholds than the model it is scored against.

Value

An object of class `fb_kfold`, a list with `estimates`, a data frame holding `elpd` and its standard error `se`, `pointwise`, one row per scored group with `group`, `fold`, `n_rows` and `elpd`, and `folds`, `engine` and `group`.

References

Cao Z, Megirian R, Nguyen M, Sparks AH (2026). Bayesian ordinal analysis of cereal growth stage data from oat trials in Western Australia. arXiv:2610.37384.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(
    trt = factor(rep(c("a", "b"), each = 60)),
    plot = factor(rep(1:24, each = 5))
  )
  eta <- 0.8 * (d$trt == "b") + rnorm(24, 0, 0.5)[d$plot]
  d$score <- cut(eta + rlogis(120), c(-Inf, -0.5, 0.5, 1.5, Inf),
    labels = c("low", "mid", "high", "top"), ordered_result = TRUE)
  fit <- flexybayes(score ~ trt, random = ~plot, data = d,
    family = "cumulative", backend = "inla", verbose = FALSE)
  folds <- rep(rep(1:4, 6), each = 5)
  fb_kfold(fit, folds, group = "plot", seed = 1L)
}
```

---

fb_marginal_cov *The marginal covariance of the response of a fitted model*

Description

Returns the covariance of the response over the rows of a fit's data, given its fitted variance components,

$$V = R + \sum_k \sigma_k^2 Z_k G_k Z_k^\top + K,$$

as a structured object with one block per variance component. `as.matrix()` assembles it as a dense matrix.

Usage

```
fb_marginal_cov(fit, rows = NULL, variances = TRUE)

## S3 method for class 'fb_marginal_cov'
as.matrix(x, rows = NULL, cols = rows, ...)
```

Arguments

fit	A fitted flexybayes model, or an fb_marginal_cov object to take a subset of or to evaluate at other variances.
rows	Integer positions of the rows to cover: rows of the fit's data, or of fit when it is an fb_marginal_cov object. NULL (the default) covers every row.
variances	TRUE (the default) to use the fit's own variance components, FALSE for the structure alone, with every variance NA. or a list with residual (the residual variance, or one per section named by level), terms (a variance per block, named by block) and, for an AR1 x AR1 field, rho (a list of c(row = , col =) named by block), all on the variance scale.
x	An fb_marginal_cov object.
cols	Integer positions of the columns, for a cross-covariance between two sets of rows. Defaults to rows.
...	Unused.

Details

Each variance enters at its posterior mean, $E[\sigma^2] = \text{estimate}^2 + \text{std.error}^2$, read from the fit's variance-component table, summary(fit)$varcomp, whose estimate and std.error are the posterior mean and standard deviation of each standard deviation. For a structure without a correlation parameter this is the posterior mean of V itself. The two correlations of an AR1 x AR1 field enter at their posterior means (estimate). On brms, the rows of a sectioned residual report the posterior median of each standard deviation, so there the residual variances are median^2 + std.error^2.

The structures built are random intercepts ((1 | g), ide(g)), two-way interactions (a:b), uncorrelated random slopes ((x || g)), diag(f):g, an ar1(row):ar1(col) field on one trial or id(trial):ar1(row):ar1(col) on each trial, and vm() or ped() known covariances, over an independent residual or one variance per level of a factor (dsum(~ units | f)). Observation weights divide the residual variance (σ^2/w_i), a known variance per row is added to it, and a known covariance matrix is added whole.

A model with any other structure is refused with the reason code marginal_cov_not_buildable, naming the terms.

Value

An object of class fb_marginal_cov, a list with

n, rows The number of rows covered, and their positions in the fit's data.

residual A list: variance (one value, or one per section named by level), by (the sectioning column, or NULL), level (each row's section, or NULL), weights (each row's observation weight, or NULL) and known_variance (each row's known variance, or NULL).

blocks A named list, one entry per variance component: name, label, component (the names the variance may carry in summary(fit)$varcomp), variance, index, n_levels, value, key, columns and structure as described above, and for an AR1 x AR1 field rho and kronecker, the factors n_at (trials), row and col (AR1 correlation matrices) with $G = I_{n_{at}} \otimes R_{row} \otimes R_{col}$.

known_covariance The known covariance matrix over the rows, whose diagonal is residual$known_variance, or NULL.

`groups` Whether V is block-diagonal over a grouping column: `applicable`, `column`, `blocks` (row positions per group, the grouping with the most groups that every block nests within) and `reason`.

`varcomp` The variance-component table the values were read from, or `NULL`.

See Also

`fb_variance_draws()` for posterior draws of the variance components.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(site = factor(rep(1:6, each = 5)))
  d$y <- rnorm(6)[d$site] + rnorm(30)
  fit <- flexybayes(y ~ 1, random = ~site, data = d, backend = "inla",
                    verbose = FALSE)
  v <- fb_marginal_cov(fit, rows = 1:10)
  v$blocks$site$variance
  as.matrix(v)
}
```

---

fb_plan *Plan a flexyBayes fit without firing the backend*

Description

Returns the dispatch + preflight + representation + memory decision the routing layer would make, without running MCMC or Laplace approximation. Useful for verifying the backend chosen, memory estimate, and any structural refusals before paying the fit cost.

Usage

```
fb_plan(
  formula,
  data,
  random = NULL,
  residual = NULL,
  backend = c("auto", "inla", "brms"),
  priors = NULL,
  known_matrices = list(),
  family = "gaussian",
  link = NULL,
  weights = NULL,
  aggregate = "auto",
  memory_ceiling_gb = NULL,
  predict_plan = NULL,
```

```
  syntax = c("auto", "asreml", "brms"),
  ...
)
```

Arguments

formula	Either a brms-style two-sided formula such as y ~ x + s(z) + (1 | g), or (when random and/or residual are supplied) the ASReml-grammar fixed-effects formula, e.g. yield ~ loc + yearf. The same formula the fit would receive.
data	A data.frame holding every variable the formula names. Planning reads its dimensions and column types, never its values.
random	An optional one-sided ASReml-grammar random-effects formula (e.g. ~ gen + gen:loc). NULL (default) for a brms-style formula, or for a fixed-effects-only ASReml model.
residual	An optional one-sided ASReml-grammar residual structure formula (e.g. ~ dsum(~ units | env)). NULL (default) for the ordinary independent-residual model.
backend	A single string, one of "auto" (the default), "inla", or "brms". Chooses the engine the plan reports on, with "auto" planning the route dispatch would take.
priors	An optional fb_prior() list. Defaults to the half-normal-on-SD default the fit would inject.
known_matrices	A named list of structured-covariance matrices referenced by vm() or ped() terms. Empty when the model names none.
family, link	Standard family and link arguments, given as single strings.
weights	An optional numeric vector of observation weights, one per row of data.
aggregate	"auto", TRUE, or FALSE, read exactly as on flexybayes().
memory_ceiling_gb	An optional numeric override for the preflight memory ceiling. Defaults to the flexyBayes.preflight_ceiling_gb option, or to flexyBayes.preflight_ram_fraction of available RAM.
predict_plan	An optional list(newdata = ..., chunk_size = ...) requesting a prediction-shape plan. Plan-only, so it never fires predict().
syntax	Grammar override, one of "auto" (default, detected from formula / random / residual exactly as flexybayes() does), "asreml", or "brms".
...	Not used. Any argument fb_plan() does not take, such as a misspelt name, is refused with plan_argument_unknown.

Details

Accepts either grammar flexybayes() accepts: a single brms-style two-sided formula (y ~ x + s(z) + (1 | g)), or the ASReml triple (formula as the fixed-effects side, plus random / residual). Both spellings route through the same grammar detection flexybayes() uses (syntax = "auto" by default), so fb_plan(y ~ x, random = ..., residual = ..., data = ...) and flexybayes(fixed = y ~ x, random = ..., residual = ..., data = ..., plan = TRUE) plan the identical model. The two spellings of the fixed part differ: fb_plan() takes it positionally as formula, the entry point takes it as fixed.

Value

An <fb_plan> classed list. See print.fb_plan() for the one-screen surface, summary.fb_plan() for the verbose dump, and as.data.frame.fb_plan() for the programmatic-consumer shape.

Examples

```
# Planning never fires the backend, so this runs without INLA or
# brms installed.
set.seed(1)
d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))

# brms-style grammar.
p1 <- fb_plan(y ~ x + (1 | g), data = d)
print(p1)

# ASReml-style grammar plans the identical model.
p2 <- fb_plan(y ~ x, random = ~ g, data = d)
identical(p1$backend_chosen, p2$backend_chosen)

# The representation table and the memory estimate are both on the
# plan object without ever fitting.
as.data.frame(p1)[, c("backend_chosen", "gate_outcome")]
```

---

fb_prior *Specify priors via the PC-canonical hybrid DSL*

Description

The flexyBayes prior DSL (domain-specific language) lives on the standard-deviation scale (never precision / variance) and accepts two canonical idioms: distributional (half_normal(scale = 1)) and tail-quantile / PC (penalised complexity) (pc(upper = 1, prob = 0.01) meaning Pr(sigma > 1) = 0.01). The PC idiom is the cross-engine interlingua, it survives every backend because it is a probability statement, not a distributional name.

Usage

```
fb_prior(...)
```

Arguments

...	One or more two-sided formulas of the form target ~ distribution(args), one per parameter being given a prior. Examples: • sigma ~ pc(upper = 2, prob = 0.05) • sd(group = "subject") ~ half_normal(scale = 1) • b("treatment") ~ student_t(df = 4, scale = 2.5) • cor(group = "subject") ~ lkj(eta = 2)

- `sd(group = "subject") ~ uniform(lower = 0, upper = 5)`

Supported distribution families: `pc`, `half_normal`, `half_cauchy`, `student_t`, `normal`, `exponential`, `lkj`, `cauchy`, `gamma`, `uniform`, and `pooled` (see "Partially pooled SDs" below). `uniform()` on a variance component sits outside the PC-canonical interlingua, but both active backends represent it faithfully on the SD scale: the INLA backend via an expression-prior on the log-precision, and brms as a bounded `uniform()` prior on each simple random-effect (and residual) SD.

Details

v0.1 supports the targets and distributions listed above in the file header. Calls outside the supported set raise a structured error with the supported list.

Argument matching is strict. Each distribution has a fixed parameter list, and a call is matched against it by base R's own rules, so `half_normal(1)` and `half_normal(scale = 1)` are the same specification. An argument name the distribution does not have, a duplicated argument, a missing required argument, a non-scalar or non-finite value, or a value outside its domain (a non-positive scale, a tail probability outside `(0, 1)`) is refused at construction with a condition carrying a `flexybayes_refusal_*` class: see `fb_refusals()`. Both halves of a PC prior are required: `pc(upper = 1)` states no probability, and is refused instead of completed with a default the caller never wrote.

Value

An `fb_prior` object, an S3 class inheriting from list, whose `$specs` element carries the parsed `target` and `spec` pairs.

What each backend can carry

`fb_prior()` is engine-neutral: it records what was asked for. Whether a given (target, distribution) pair can be *carried* depends on the backend, and the two differ.

- **brms** takes any of the ten distributions on `sigma` and `sd(group = )`: the parameter is bounded below at zero and brms renormalises a two-sided density over that support, which is why `normal(0, s)` and `half_normal(s)` are the same prior there, and `normal` / `student_t` on `b()`. It carries no `cor()` or `smooth()` prior, because no model this package emits on brms has either parameter.
- **INLA** takes `half_normal`, `uniform`, `pc` and `exponential` on `sigma`, `sd(group = )` and `smooth()` (`exponential` is the PC prior written in its distributional form), `normal` on `b()`, which arrives through `control.fixed`, and `normal` on `loading(group = )`, the loadings of `fa(f, k):group` relative to the anchor level. With no loading prior stated, each loading takes `normal(mean = 1, sd = 0.316)`, which is INLA's own prior on a copy scaling, `prior_summary()` lists it.

A specification the selected backend cannot carry is refused at the fit, naming the row and both remedies, re-express it in a distribution that engine does carry, or switch backend. Before 0.9.2 the INLA route discarded such a row silently while `prior_summary()` printed it as applied, so a prior-sensitivity analysis could vary a scale and compare the engine's default with itself. `prior_summary()`

lists each prior with the engine slot it reached, as the engine reports it after fitting, and names each stated or default prior that reached no slot, with the reason.

A prior naming a coefficient or a variance component the model does not have is likewise refused at the fit, against the engine's own parameter list and in the vocabulary the caller wrote.

The default prior

A fit that states neither `prior` nor `prior_vc_sd` runs under the package default: `half_normal(scale = s)` on the residual standard deviation and on the standard deviation of every variance component the default reaches, on both engines. `s` is `5 * sd(y)` on an identity link, `3` on a log link and `5` on a logit link. On an uncorrelated random slope on a numeric covariate `x` it is `s / sd(x)`. The default has no upper bound. `prior_summary()` prints the scale a fit ran under. A Gaussian identity-link fit on INLA runs on the response divided by `sd(y)`, where the same prior is `half_normal(scale = 5)`. The printed formula and `prior_summary()` show it on both scales.

Partially pooled SDs

`pooled()` puts one shared prior on a set of per-level standard deviations, instead of a separate prior on each. The target names the term, and `by` the factor whose levels are pooled:

- `sd(group = "TRIAL:VARIETY", by = "TRIAL") ~ pooled(scale = half_normal(scale = 0.5))` for the genotype-by-trial SDs of `diag(TRIAL):VARIETY`.
- `sigma(by = "TRIAL") ~ pooled(scale = half_normal(scale = 0.5))` for the residual SDs of `dsum(~ units | TRIAL)`.

Either set can be pooled without the other. The log SDs of a pooled set are normal with an estimated mean and an estimated SD, the pooling SD. `scale` is the prior on the pooling SD and has no default. It can be any prior brms accepts for a standard deviation. The mean has a `normal(log(s / 5), 2)` prior, where `s` is the scale of the package's default SD prior. On an identity link that is `normal(log(sd(y)), 2)`.

Pooling shrinks the per-level SDs toward each other, so the estimates differ from ASReml's, which fits each SD separately. Only brms fits it, because INLA cannot put a shared prior on separate variance parameters: `backend = "inla"` refuses with `pooled_prior_requires_brms`, and `backend = "auto"` uses brms. Printed output labels the fit "per-trial SDs partially pooled", with the `by` factor's name in place of "trial".

Examples

```
p <- fb_prior(
  sigma                ~ pc(upper = 2, prob = 0.05),
  sd(group = "subject") ~ half_normal(scale = 1),
  b("treatment")       ~ student_t(df = 4, scale = 2.5)
)
p

# Model C with both per-trial SD blocks partially pooled
fb_prior(
  sd(group = "TRIAL:VARIETY", by = "TRIAL") ~
    pooled(scale = half_normal(scale = 0.5)),
  sigma(by = "TRIAL") ~ pooled(scale = half_normal(scale = 0.5))
```

```
)
```

`fb_refit` *Refit a model on new data, with added fixed columns*

Description

Refits the model of `fit` on `data`, with the same random structure, family, priors, engine and engine settings, known variance or covariance, observation weights and offsets. A package built on flexyBayes uses it to refit a user's model on a fold of the data, with candidate columns added to the fixed part, or with brms prior rows and Stan code of its own.

Usage

```
fb_refit(
  fit,
  data = NULL,
  add_fixed = NULL,
  extra_prior = NULL,
  stanvars = NULL,
  formula_hook = NULL,
  readout = c("full", "varcomp", "fitted", "draws"),
  fixed = NULL,
  ...
)
```

Arguments

`fit`	A flexyBayes fit.
`data`	A data frame with the columns the model uses and any added ones. `NULL`, the default, refits on the data the fit carries.
`add_fixed`	Fixed-effect terms to append to the fixed formula: a one-sided formula such as `~ z1 + z2`, or a character vector of term labels. `NULL` adds none.
`extra_prior`	A `brmsprior` from `brms::set_prior()` with rows to add to the prior, or `NULL`.
`stanvars`	A `stanvars` object from `brms::stanvar()`, or `NULL`.
`formula_hook`	A function of `formula` and `prior` returning `list(formula = , prior = )`, or `NULL`. See "Extending the brms model".
`readout`	`"full"`, the default, or `"varcomp"`, `"fitted"` or `"draws"`. See "Readout".
`fixed`	A two-sided formula that replaces the fit's fixed formula, for a refit whose fixed part differs by more than added terms. `NULL`, the default, keeps the fit's.
`...`	Named arguments of `flexybayes()` that change how this refit runs, for example `seed`, `backend`, `n_samples`, `control`, `na_action`, `verbose`, `return_code`, `weights` or `known_variance`. On the brms route of "Extending the brms model", arguments that `flexybayes()` does not have go to `brms::brm()`. Arguments that state the model (`random`, `residual`, `family`, `link`, `known_matrices`, `prior`, `prior_fixed_sd`, `prior_vc_sd`) are refused, `update()` refits a changed model.

Value

A fitted flexyBayes object of the engine the refit ran on, or, with `return_code = TRUE`, the code that engine would run.

What the refit keeps

- The fixed formula is the one `fit` was called with, or `fixed`, with the terms of `add_fixed` appended.
- A prior written with `fb_prior()`, `prior_fixed_sd` or `prior_vc_sd` is passed on as written. The package default prior is built again over the refitted model, as `update()` does, so on the same data every term keeps the prior it had and an added term gets the default.
- The engine is the one `fit` was fitted with, `"inla"` or `"brms"`, even when the original call asked for `"auto"`. Its recorded sampler settings, `seed`, `control`, `na_action`, `fixed_alias` and `aggregate` are passed on.
- Weights and a known variance belong to rows. A refit on the fit's own data keeps them. A known variance is read from the column of that name in the fit's data, so rows of `fit$data` carry it. For any other `data`, pass `weights =` or `known_variance =` with one value per row. without them a weighted fit is refused, as is a fit with a known variance whose `data` lacks the column.

Extending the brms model

`extra_prior`, `stanvars` and `formula_hook` are brms objects, and the refit then runs on brms: `fit` must be a brms fit, or `backend = "brms"` must be given. The refit still starts from the brms formula and prior rows the package builds for the model.

`formula_hook` is called as `formula_hook(formula, prior)` with that formula and the `brmsprior` of those rows, and returns `list(formula = , prior = )`, the formula and prior rows to fit. A non-linear formula usually moves the package's rows onto one of its parts, which is why the hook receives them. `extra_prior` rows are then merged in, replacing any row with the same `class`, `coef`, `group`, `resp`, `dpar` and `nlpar`, and `stanvars` is passed to `brms::brm()`. The fitted object reads under the linear model's names.

Readout

`readout` names what the caller reads from an INLA refit. `"varcomp"` (variance components), `"fitted"` (fitted values) and `"draws"` (posterior draws, for example on held-out rows) ask INLA for that readout only: no DIC, WAIC or marginal densities, posterior configurations only for `"draws"`, and no second fit to test an AR1 field. The posterior mode, and the readout taken from it, are those of the full refit. A brms refit is always returned in full.

See Also

`update.flexybayes()` to refit with any argument changed.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
```

```
  d <- data.frame(site = factor(rep(1:12, each = 10)), x = rnorm(120),
                  z = rnorm(120))
  d$y <- rnorm(12)[d$site] + 0.5 * d$x + rnorm(120)
  fit <- flexybayes(y ~ x, random = ~site, data = d, backend = "inla",
                    verbose = FALSE)
  # The same model with z added, on the first nine sites.
  fold <- fb_refit(fit, data = d[d$site %in% 1:9, ], add_fixed = ~z,
                   verbose = FALSE)
  coef(fold)
}
```

`fb_refusals` *List flexyBayes refusal reasons*

Description

`fb_refusals()` exposes the locked refusal-reason registry as a browsable table: the canonical vocabulary of conditions under which flexyBayes declines to fit, route, or validate a model, each with a one-line description and the release it was introduced in. It is the discovery surface for the structured refusals the package raises. Every such refusal carries a condition class `flexybayes_refusal_<reason_code>`, so a reason listed here can be caught precisely, for example with `tryCatch(fit, flexybayes_refusal_precision_not_sy = handler)`.

Usage

```
fb_refusals(reason_code = NULL, since_version = NULL)
```

Arguments

`reason_code`	Optional character vector of exact reason codes to filter to. `NULL` (default) returns all registered reasons.
`since_version`	Optional single version-string prefix to filter to. `NULL` (default) returns all.

Details

Two optional filters narrow the listing. `reason_code` selects rows by exact reason-code match (a single code or a vector). The `since_version` filter selects reasons introduced in a matching release by version-string prefix: `since_version = "0.4"` returns every reason added in the 0.4 series.

Routing-decision reasons (surfaced by `fb_plan()` and `backend_decision()`) and internal control-flow signals are deliberately excluded: this table lists only refusals a user can actually encounter.

Packages that extend flexyBayes add their own reasons with `fb_register_refusals()`. The `package` column names the package each reason belongs to: `"flexyBayes"` for the core, and the plug-in's name for the rest.

Value

A data frame of subclass `fb_refusals_table`, one row per matching refusal reason, with columns `reason_code`, `package`, `description`, `since_version`, and `plan_field`. The print method renders it as a compact checklist.

Examples

```
fb_refusals()
fb_refusals(reason_code = "precision_not_symmetric")
fb_refusals(since_version = "0.4")
```

---

fb_refuse *Raise a registered refusal*

Description

`fb_refuse()` stops with a refusal condition for a reason registered by flexyBayes or by a package through `fb_register_refusals()`. The condition has the classes `flexybayes_refusal_<reason_code>`, any classes given in `class`, and `flexybayes_refusal`, so users catch a plug-in's refusals exactly as they catch the core's.

Usage

```
fb_refuse(reason_code, message = NULL, ..., class = character(0), call = NULL)
```

Arguments

`reason_code`	A registered reason code (see `fb_refusals()`).
`message`	The message shown to the user. Defaults to the reason's registered description. A message that names the user's own variables or values is usually more helpful.
`...`	Further named fields to carry on the condition object, for handlers that need the offending value. The names `message`, `reason_code`, `package` and `call` are set by `fb_refuse()` and cannot be used.
`class`	Additional condition classes, placed between the per-code class and `flexybayes_refusal`.
`call`	The call to record on the condition. Defaults to `NULL`, as for the core's refusals.

Value

Does not return. It signals an error. The condition carries `message`, `reason_code`, `package` (the package that registered the code), `call` and the fields given in `...`.

See Also

`fb_register_refusals()`, `fb_refusals()`.

Examples

```
res <- tryCatch(
  fb_refuse("unsupported_family"),
  flexybayes_refusal = function(e) e
)
res$reason_code
res$package
```

`fb_register_refusals` *Register a package's refusal reasons*

Description

A package that extends flexyBayes calls `fb_register_refusals()` from its own `.onLoad()` to add the reasons it refuses with. Its users then catch those refusals with the same `flexybayes_refusal` condition class as the core's, find them in `fb_refusals()`, and the package raises them with `fb_refuse()`.

Usage

```
fb_register_refusals(refusals, package = NULL)
```

Arguments

`refusals`	A data frame with one row per reason and these character columns: • `reason_code` (required): lower-case letters, digits and underscores, starting with a letter. Prefix codes with something particular to the package so they do not collide with others. • `description` (required): one sentence on what the refusal means, shown by `fb_refusals()` and used by `fb_refuse()` when no message is given. • `since_version` (required): the package version that introduced the reason. • `message_template` (optional, default `"%s"`): the message shape, for documentation. • `registered_in_adr` (optional, default `NA`): the design record the reason traces to. • `plan_field` (optional, default `NA`): the `fb_plan()` slot that reports the refusal before a fit, if any.
`package`	The package that owns the reasons. `NULL` (default) takes the package whose code makes the call, which is the right value inside `.onLoad()`. Outside a package it must be given.

Details

Each package's reasons go into a partition named after the package. A reason code is unique across all partitions: a code that flexyBayes or another package already registered is refused, and so is any attempt to register under the name `"flexyBayes"`. Registering the same code again from the same package with identical fields does nothing, so a package can be loaded twice in one session. Registering it with different fields is refused. A table that fails any check registers none of its rows.

Value

The registered reasons as an `fb_refusals_table`, invisibly.

See Also

`fb_refuse()` to raise a registered reason, `fb_refusals()` to list them.

Examples

```
## Not run:
# In the plug-in's R/zzz.R:
.onLoad <- function(libname, pkgname) {
  flexyBayes::fb_register_refusals(
    data.frame(
      reason_code = "mypkg_threshold_negative",
      description = "The threshold must be zero or positive.",
      since_version = "0.1.0"
    )
  )
}

## End(Not run)
```

---

fb_variance_draws *Joint posterior draws of the variance components of an INLA fit*

Description

Draws the hyperparameters of an INLA fit jointly and returns them on the scale and under the names of the fit's variance-component table (`summary(fit)$varcomp`): a standard deviation for the residual (`sigma`) and for each random term (`sd_<term>`), and a correlation as it is.

Usage

```
fb_variance_draws(object, n, seed = NULL)
```

Arguments

object	A flexybayes_inla fit.
n	The number of draws.
seed	An integer seed, or NULL (the default) to use the session's random number stream.

Details

The draws come from INLA::inla.hyperpar.sample() and are mapped to the response's original scale. A hyperparameter held fixed, such as the precision of a known variance, is not drawn. With seed, the draws from a given fit are repeatable on one INLA thread (options(flexyBayes.inla_threads = "1:1")). A refit of the same model in another R session can reach a different mode (see flexybayes(), section *INLA threads*).

Value

A data frame with n rows and one column per variance component.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(site = factor(rep(1:6, each = 5)))
  d$y <- rnorm(6)[d$site] + rnorm(30)
  fit <- flexybayes(y ~ 1, random = ~site, data = d, backend = "inla",
                    verbose = FALSE)
  head(fb_variance_draws(fit, 100L, seed = 1L))
}
```

---

fitted.flexybayes *Extract fitted values*

Description

Extract fitted values

Usage

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

Arguments

object	A fitted flexybayes object of any backend.
...	Ignored, present for compatibility with the generic.

Value

A numeric vector of in-sample fitted values on the response scale, one per row of the fitted data, each the posterior mean of that observation's conditional expectation. Rows carried as latent under `na_action = "augment"` receive a fitted value like any other.

`fitted.flexybayes_aggregated`
Fitted values of an aggregated count fit

Description

An aggregated binomial or Poisson fit is one INLA model over cell totals. Each row's fitted value is the posterior mean of its cell's inverse-linked linear predictor, random effects included, times the row's own exposure: `exp(offset)` on a Poisson model, and the `exposure` or `trials` column of a streamed fit. The values equal those of a per-row fit of the same model.

Usage

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

Arguments

`object`	A `flexybayes_aggregated` fit.
`...`	Ignored, present for compatibility with the generic.

Details

A streamed fit keeps only the first rows it read. When those are fewer than the rows it was fitted to, the method refuses by name (`streamed_rows_not_retained`), not return them as the data.

Value

A numeric vector of fitted values on the response scale, one per row of the data, in row order.

fitted.flexybayes_inla *In-sample fitted values from a per-row INLA fit*

Description

Returns INLA's posterior-mean fitted values (response scale) for the observed rows, taken from `summary.fitted.values`. Without this method `fitted()` dispatched to `stats::fitted.default`, which silently returned `NULL` for an INLA fit because the object does not populate a `$glm` slot.

Usage

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

Arguments

`object`	A `flexybayes_inla` fit.
`...`	Ignored. Present for compatibility with the generic.

Value

A numeric vector of posterior-mean fitted values, one per observation.

flexybayes *Bayesian Mixed Models with ASReml Syntax*

Description

Specify mixed models using ASReml formula syntax and estimate them on one of two active engines: INLA (integrated nested Laplace approximation) or brms (the Stan passthrough). Returns a three-part result: a GLM-compatible object for use with standard R packages (emmeans, marginaleffects, etc.), the native backend object for engine-specific diagnostics, and extras for secondary analyses.

Usage

```
flexybayes(
  fixed,
  random = NULL,
  residual = NULL,
  data,
  family = "gaussian",
  link = NULL,
  known_matrices = list(),
```

```
  weights = NULL,
  na_action = c("augment", "omit", "fail"),
  n_samples = 1000,
  warmup = 500,
  chains = 4,
  seed = NULL,
  control = NULL,
  prior = NULL,
  prior_fixed_sd = 100,
  prior_vc_sd = 1,
  verbose = TRUE,
  mcmc_verbose = TRUE,
  return_code = FALSE,
  review_code = FALSE,
  backend = c("auto", "inla", "brms"),
  aggregate = "auto",
  plan = FALSE,
  syntax = c("auto", "asreml", "brms"),
  known_variance = NULL,
  fixed_alias = c("refuse", "drop"),
  rcov
)

fb(
  fixed,
  random = NULL,
  residual = NULL,
  data,
  family = "gaussian",
  link = NULL,
  known_matrices = list(),
  weights = NULL,
  na_action = c("augment", "omit", "fail"),
  n_samples = 1000,
  warmup = 500,
  chains = 4,
  seed = NULL,
  control = NULL,
  prior = NULL,
  prior_fixed_sd = 100,
  prior_vc_sd = 1,
  verbose = TRUE,
  mcmc_verbose = TRUE,
  return_code = FALSE,
  review_code = FALSE,
  backend = c("auto", "inla", "brms"),
  aggregate = "auto",
  plan = FALSE,
```

```
  syntax = c("auto", "asreml", "brms"),
  known_variance = NULL,
  fixed_alias = c("refuse", "drop"),
  rcov
)
```

Arguments

fixed	Two-sided formula `response ~ fixed_effects`. This is the universal entry's model slot: it accepts the ASReml `fixed` form (paired with `random` / `residual`) **or** a brms / lme4-style bar-grouped formula such as `response ~ x + (1 | g)` (in which case the grouping lives in the formula and `random` / `residual` must be left `NULL`). The grammar is detected from the call shape. Use `syntax` to force it. **Offsets.** `offset()` terms in the fixed formula enter the linear predictor with a fixed coefficient of one on both engines, and several add. Each is evaluated against `data` and has to be a finite number on every row (`fixed_offset_invalid` otherwise). brms receives the terms in its formula; INLA receives the evaluated offset through `inla()`'s `offset` argument, and a Gaussian identity-link fit reads its centre, scale and default priors from the response minus the offset. The aggregated route carries an offset exactly on the Gaussian family (subtracted before the cell statistics are taken) and the Poisson family (as exposure), and refuses one on the binomial family (`fixed_offset_unsupported`), where `aggregate = "auto"` takes the per-row route. `predict()` on new data evaluates the offset there and refuses when a column it reads is absent (`newdata_offset_missing`). An `offset()` in `random` or `residual` is refused.
random	One-sided formula: `~ random_terms` using ASReml syntax. "Parsed" and "fitted" are not the same claim, so the term catalogue is two lists. **Parsed and fitted** (emitted by at least one active engine): `vm()`, `at()`, `diag()`, `idh()`, `us()`, `ar1()`, `spl()`, `ped()`, `dsum()`, `id()`, and nested colon terms, including the per-trial separable AR1 field described under "Spatial and temporal autoregressive fields" below. **Parsed and refused by name.** `fa(f, k):g`, the factor-analytic genotype-by-environment covariance, is refused on both engines (`factor_analytic_not_repeatable`). On simulated trials with a known covariance, the genetic-correlation intervals of INLA fits covered the true value less often than required (134 and 125 of 150 for k = 1 and 2, against 136), and flexyBayes has no factor-analytic lowering for brms. Write `diag(f):g`, which fits on both engines, or `us(f):g` on brms. The bare spelling `fa(v, k)`, with no grouping factor for the covariance to apply to, keeps its own refusal (`fa_not_representable`). `corh(f):g` is likewise ASReml syntax that flexyBayes parses and refuses by name. It is described just below. **Heterogeneous variances.** `diag(f):g`, `idh(f):g` and `at(f):g` all fit one variance per level of `f` with no covariance between levels: the usual multi-environment structure, where the genotype variance differs by site. ASReml treats `diag()` and `idh()` as the same structure, and so does flexyBayes. The three spellings emit identical code. `us(f):g` estimates every pairwise correlation instead, at `k(k+1)/2` parameters against `diag()`'s `k`.

`corh(f):g`, heterogeneous variances with a single shared correlation, is refused on both engines: flexyBayes has no lowering for it, and approximating it with either neighbour would change the parameter count. With one variance for every level and a correlation of zero or more, `random = ~ g + f:g` is the same covariance and fits on both engines.

Selected levels. `at(f, level):g` gives the random effect `g` only to the rows where `f` takes one of the named levels, with one variance per named level, as in ASReml. `at(design, "split"):mainplot` is a main-plot effect in the split-plot trials alone. Levels are written as values, `at(f, c("a", "b"))`, or as positions in `levels(f)`, `at(f, 2)`. A level the data do not have is refused (`at_level_not_in_data`). It fits on both engines and reports `sd_<g>__<f><level>`, the name `at(f):g` gives the same level.

`diag(f):g` fits on both engines. `us(f):g` fits on brms, and INLA refuses it, so `auto` routes such a model to brms.

Spatial and temporal autoregressive fields. `ar1(t)` and the separable `ar1(row):ar1(col)` are written here, on the random side, and emitted on INLA as a latent autoregressive field plus the Gaussian observation nugget. Four hyperparameters on a field grid: the row correlation, the column correlation, the field SD and the nugget SD, all four printed by `summary()`. Each correlation is named after the factor it runs along, `rho_<factor>`: `rho_row` and `rho_col` for `ar1(row):ar1(col)`, and the same two names for `ar1(col):ar1(row)`. The field is faithful only with one observation per grid node, so an incomplete or replicated grid refuses (keep the unobserved nodes as design cells with the default `na_action = "augment"`). flexyBayes has no brms lowering for the field, so brms refuses.

That four-parameter model is **not** ASReml's separable residual, which is one correlated residual with no independent plot error: three parameters, nested inside this one as the nugget goes to zero. On data with real plot-to-plot noise the two return different correlations, because a nugget-free model must absorb independent noise into the correlated structure and is pulled towards zero by it. The gap grows with the share of the total variance the nugget holds, and it falls on the correlations a reader acts on. Neither model is wrong. The residual form is written on the residual side, `residual = ~ ar1(row):ar1(col)`, described under `residual` below.

Per-trial nested field. `id(trial):ar1(row):ar1(col)` fits one separable AR1 field per level of `trial` on INLA, via the `replicate =` mechanism: one complete grid per trial, each with its own realised field. The row correlation, column correlation and field SD are **shared across every level of** `trial`, not estimated per trial. This is the model ASReml fits under the same spelling.

A field for each trial. `at(trial):ar1(row):ar1(col)` fits a separate field on each trial, with its own SD and correlations, as ASReml does: three parameters per trial, reported as `sd_spatial__trial<level>`, `rho_row__trial<level>` and `rho_col__trial<level>` for `ar1(row):ar1(col)`, plus the shared Gaussian nugget. `at(trial, level):ar1(row):ar1(col)` fits the field on the named trials only. Each trial's plots must fill its own grid. brms refuses both.

Units followed through time. `ar1(t):g`, or the same term written `g:ar1(t)`, fits one AR1 series along `t` within each level of `g`, independent across levels, with one correlation and one field SD shared by every level and the Gaussian nugget on top, three hyperparameters in all. This is the units-by-occasions

repeated-measures covariance of Verbyla and Cullis (1992) written on the random side: `g` indexes the plots, animals or series and `t` the occasions, whose levels must be in time order. An occasion a unit missed stays as a design row with the response `NA` under the default `na_action = "augment"`, and the `t` by `g` grid must be complete in that sense, one row per node. brms refuses, as for the other fields.

`residual` One-sided formula: `~ residual_structure`. Default `~ units` (iid residuals). `~ dsum(~ units | env)`, and the equivalent `~ at(env):units`, give a separate residual variance per level of `env`. Only families that carry a residual scale parameter can have one made heterogeneous. A Poisson's dispersion is a function of its mean, so there is nothing there to vary, and the request is refused.

`~ ar1(row):ar1(col)` is ASReml's separable residual: one correlated residual with no nugget, three parameters (`sd_spatial`, `rho_row`, `rho_col`), and no `sigma` row. `~ dsum(~ ar1(row):ar1(col) | trial)` fits it separately in each trial, with names suffixed `__trial<level>`. Both fit on INLA for a Gaussian response with the identity link and one plot per grid cell, where the observation error is fixed at a variance of `exp(-12)` on the standardised response; brms refuses them. They cannot be combined with a random-side field. `~ ar1(t)` and `~ ar1(row):col` are refused on both engines, with a pointer to the random-side field described under `random` above.

`data` A data.frame containing all variables referenced in the formulas.

`family` Character: `"gaussian"`, `"binomial"`, `"poisson"`, `"negative_binomial"`, `"gamma"`, `"beta"`, `"cumulative"`, `"cratio"`, `"hurdle_gamma"` or `"lognormal"`. The last two are brms only: INLA's likelihood roster has no hurdle gamma, and no INLA lognormal fit is validated, so the family gate refuses both there and `backend = "auto"` routes them to brms. A family object (`binomial()`, `gaussian()`, `brms::lognormal()`) is also accepted, goes through the same gate as its name, and supplies its own link, so `Gamma()` means the inverse link it names, not the log link `family = "gamma"` defaults to. Passing a family object together with a contradicting `link` is refused, not resolved. A family outside this set is refused at the door. Where the boundary is flexyBayes's own and not the engines', the refusal says so and `inst/KNOWN_ISSUES.md` records it. A response outside the family's domain (0 or negative for `"gamma"` or `"lognormal"`, negative for `"hurdle_gamma"`, at or outside [0, 1] for `"beta"`) is refused by name (`family_response_domain_violation`) before any engine runs. A missing (`NA`) response is `na_action`'s concern, not a domain violation. The ordinal families are `"cumulative"` (proportional odds) and `"cratio"` (continuation ratio, the sequential model of Cao et al. 2026). The response is an ordered factor or a whole-number vector, its categories are the values observed, in order, and thresholds follow brms's convention: `P(Y <= k) = F(tau_k - eta)` for `"cumulative"` and `P(Y > k | Y >= k) = F(eta - tau_k)` for `"cratio"`. The model keeps its intercept, whose place the thresholds take. INLA fits both families with the random structures it fits for a binomial response, the `ar1(row):ar1(col)` field included, and the cumulative family with at most 11 categories. brms fits both without the field. A factor that is not ordered, or a character or logical response, is refused (`ordinal_response_unordered`), and so is an offset (`ordinal_offset_not_supported`). Category probabilities come from `fitted()` and `predict()`, and `fb_kfold()` cross-validates the fit.

`link` Character or NULL: override the default link function (e.g., `"probit"` for binomial). Leave it unset when `family` is a family object, which carries its own link. The links fitted, default first: gaussian `identity`, `log`, `inverse`, `logit`, `probit`, `cloglog`, `cauchit`; binomial `logit`, `probit`, `probit_approx`, `cloglog`, `cauchit`, `softit`, `identity`, `log`; poisson `log`, `identity`, `sqrt`; negative_binomial `log`, `identity`, `sqrt`, `softplus`, `squareplus`; gamma `log`, `identity`, `inverse`; hurdle_gamma `log`, `identity`, `inverse`, `softplus`, `squareplus`; beta `logit`; lognormal `identity`; cumulative `logit`, `probit`; cratio `logit`, `probit`, `cloglog`. `backend = "inla"` takes the default link only, refusing any other by name (`inla_gate_refused`), and `backend = "auto"` fits another link on brms. A link outside the list is refused by name (`unsupported_family`) on every backend.

`known_matrices` Named list of matrices referred to in the random formula (e.g., `list(Gmat = G_mat, Amat = A_mat)`). The carrier is declared with the `fb_cov()` constructor inside the random term: a dense covariance (`vm(group, cov = fb_cov(G, type = "dense"))`; brms), a user-supplied lower-triangular Cholesky factor (`vm(group, cov = fb_cov(L, type = "chol"))`; brms), or a sparse precision matrix `Matrix::dgCMatrix` (`vm(group, cov = fb_cov(Q, type = "precision"))`; INLA and brms). INLA takes the precision and pedigree-precision carriers only, and refuses by name a dense or Cholesky carrier, pointing at `solve(V)` or at brms, and a block carrier, which INLA would fit with one standard deviation per block, pointing at the precision carrier of the block-diagonal inverse. The bare dense forms `vm(group, V = ...)` / `ped(group, A = ...)` remain the default. The legacy v0.3.7 keyword carriers (`chol = `, `precision = `, `blocks = `, `low_rank_factor = `) are deprecated and emit a migration warning. See the formula-surface vignette for per-type worked examples.

`weights` Optional numeric weight vector (length N). Lowered for `family = "gaussian"` with the identity link only, in the ASReml / lme4 / `glm(weights=)` precision sense, `Var(y_i) = sigma^2 / w_i`: INLA's `scale = w` per-observation precision multiplier, and on brms a known offset on the log-link sigma distributional parameter (not brms's own `weights()` addition term, which implements a different, likelihood-power quantity). A constant vector (including `rep(1, N)`) is the unweighted model under a different spelling and passes through unchanged. A non-constant vector on any other family, or a non-identity link on Gaussian, is refused by name (`weights_requires_gaussian`) instead of silently producing the unweighted posterior. Likelihood-power, frequency, trials and exposure are different models and cannot share this argument silently. `aggregate = TRUE` together with weights is also refused by name (`weights_not_aggregatable`): the aggregated route's closed-form sufficient statistics do not carry a per-observation weight through the compression.

`na_action` How to treat observations whose response is missing, and, through the list form below, observations whose predictors are missing.

`"augment"` (default) retains missing-response rows, carrying the missing response as a latent quantity the engine marginalises, and completes the design grid where the absent cells are determinable. This is ASReml's `na.method(y = "include")` behaviour, and it matters whenever the model carries a covariance indexed by the design: deleting the row of a lost plot changes the index set a separable AR1 field is built over, so the fitted model is no longer the model that was

written down. "omit" drops the rows (complete-case), and a structured covariance over the resulting broken grid then refuses downstream. "fail" refuses if any response is missing, and leaves the design grid alone.

A response value that is NaN or infinite, as log(0) gives, is not a missing observation. It refuses by name under every setting (response_not_finite). Set it to NA to treat it as missing.

The argument also accepts the object an ASReml user already writes, asreml::na.method(y = , x =), and the bare list of the same shape for readers without an asreml licence:

```
flexybayes(..., na_action = list(y = "include", x = "fail"))
```

The response words map include to "augment", omit to "omit", fail to "fail". The covariate words are ASReml's own. x = "fail" (the default, and ASReml's) refuses a missing predictor. x = "omit" drops the affected rows with a warning naming the count and the columns. x = "include", ASReml's zero-fill (Reference Manual 4.2, section 3.11), is refused by name, because a zero is a value the plot did not have. A value arriving as an unreduced na.method() default vector is read as its first element, which is the policy ASReml itself would use.

Under ignorability, and only when both representations put the same prior on the same model parameters, the posterior for those parameters is the same whether a missing response is augmented or omitted – augmentation preserves the representation, not information. A backend default prior can depend on the realised design matrix, so it does not establish that common-prior condition automatically. The identity is a statement about the posterior and not a promise about what an optimiser returns: the two settings hand the engine an intact design and a ragged one, and on a weakly identified mode they can land in different places. Where missingness depends on the unobserved response itself, both are biased and neither this argument nor any diagnostic here will tell you so.

n_samples — Integer: number of posterior samples per chain.

warmup — Integer: number of warmup (burn-in) iterations per chain.

chains — Integer: number of MCMC chains.

seed — Integer or NULL: the seed for the sampler, on the engine that samples.

On brms it is forwarded to brms::brm(seed =) and fixes Stan's draws: two fits of the same model to the same data under the same seed, chains and backend return identical draws. With NULL (the default) brms passes no seed and Stan runs with one drawn for it. On the cmdstanr backend that draw comes from R's random-number stream, so it follows set.seed() only when nothing else draws from the stream first. Either way the seed Stan ran with is kept at fit$extras$reproducibility$seed$as_run, and a refit under that value returns the same draws. Draws taken later from the fitted object, such as brms::posterior_epred() on new levels, follow R's random-number stream and not this seed.

On INLA the seed has no effect: INLA::inla() takes no seed, and the argument says once that it was not used. A refit in a fresh R session can reach a different mode whatever the seed (see the section *INLA threads*). The thread setting an INLA fit ran with is recorded beside the engine versions, in the same record.

`control` A named list of sampler control settings or `NULL`, forwarded to `brms::brm(control = )`. This is the route to `adapt_delta` and `max_treedepth`: `control = list(adapt_delta = 0.95)` is the standard first response to divergent transitions. `NULL` (the default) uses brms's own settings. A no-op on the INLA path, as `seed` is.

`prior` An optional `fb_prior()` object specifying priors via the PC-canonical hybrid DSL (preferred). When supplied it overrides `prior_vc_sd` for the variance components it covers. See `fb_prior()`. On INLA, a random-effect standard deviation the prior does not name takes the package's default, `half_normal(scale = s)` with the family-aware `s` described under `prior_vc_sd` (divided by the covariate's standard deviation for an uncorrelated slope); `prior_summary()` lists what was filled. A standard deviation the prior does name is left exactly as written.

`prior_fixed_sd` Numeric: SD for fixed-effect normal priors, applied uniformly to the intercept, factor contrasts, continuous slopes, factor x continuous interactions, and `I()`-expression terms. Default `100`: weakly informative on the natural response scale for the vast majority of agricultural / clinical responses (covers responses with central tendency up to several hundred without crushing the posterior toward zero), while still regularising at sample sizes below ~ 30 per coefficient. Set wider (e.g. `1000`) for responses on a larger natural scale, or narrower for explicit shrinkage. A weakly-informative normal prior on the data scale, in the spirit of the weakly-informative-prior literature for regression coefficients (e.g. Gelman et al. 2008, *Annals of Applied Statistics* 2(4):1360-1383). **Applied when you supply it**, on every backend: brms receives one `normal(0, sd)` row per fixed-effect class, and INLA receives `control.fixed` with `prec = 1 / sd^2` on the slopes and the intercept alike. Left unsupplied, brms keeps its own fixed-effect default (flat coefficients and a response-centred `student_t` intercept), because replacing a response-centred intercept prior with `normal(0, 100)` is a different model for any response far from zero. INLA gives each column built from a numeric variable the covariate-scaled default `normal(0, 1e5 * sd(y) / sd(x))` on an identity link and `normal(0, sqrt(1000) / sd(x))` on a log or logit link, with `sd(x)` read over the rows whose response is observed, so a covariate's units do not change the fit, and keeps its own `prec = 0.001` on factor contrasts and its flat intercept. `prior_summary()` names what a fit ran under. A supplied value is never rescaled by a covariate: it is a standard deviation in response units per covariate unit, so on a covariate recorded in small units (a large coefficient) it can bind. Measure the covariate in natural units, or leave the argument unsupplied on INLA.

`prior_vc_sd` Numeric: hyperparameter for the legacy `lognormal(0, prior_vc_sd)` variance-component prior. **Note:** when both `prior` and `prior_vc_sd` are left at their defaults, the package applies a half-normal default on the SD scale (`half_normal(scale = s)` with family-aware `s`: `5 * sd(y)` for Gaussian, `5` on the logit scale for binomial / beta, `3` on the log scale for Poisson / negative-binomial / gamma) to `sigma` and every named random-effect group, on both engines, with the scale on the standard deviation of an uncorrelated random slope divided by the standard deviation of its numeric covariate. It is weakly informative and has no upper bound. A half-t or half-Cauchy (Gelman 2006, *Bayesian Analysis* 1(3):515-534) is stated through `fb_prior()`. Before 0.13.0 the default was `uniform(0, U)` with `U` equal to the scale above. The legacy `lognormal(0, 1)` default fires only

when prior_vc_sd is passed explicitly. Silence the one-time announcement message via options(flexyBayes.silence_default_prior_note = TRUE). When it is passed explicitly it reaches both active engines as the same density on the standard-deviation scale: brms as a lognormal(0, prior_vc_sd) prior row, INLA as the expression prior that writes that density in INLA's log-precision parameterisation. The two engines therefore carry the same variance-component prior.

verbose — Logical: print the generated backend code to the console.

mcmc_verbose — Logical: show the sampler's progress bar. Has no effect on INLA, which does not sample.

return_code — Logical: if TRUE, return the generated backend code without fitting the model: the Stan program on backend = "brms", the INLA formula, family and hyperparameter list on backend = "inla".

review_code — Logical: if TRUE, do not fit the model immediately; instead return a <flexybayes_review> deferred-execution object carrying the generated Stan code, the resolved prior, the parsed intermediate representation (IR), the captured call, and a snapshot of .Random.seed. Inspect the code with cat_code() and run the deferred fit with proceed(). A second proceed() call returns the cached fit. Useful as a teaching / auditing surface before a long MCMC run. Default FALSE preserves the existing run-immediately semantics. A session-level override is available via options(flexyBayes.review_code_default = TRUE). The argument value at call time always wins. Closest published precedent: brms::stancode() plus the chains = 0 idiom for "do everything except sampling". review_code = TRUE and return_code = TRUE are mutually exclusive. brms is the only engine with a code slot, so review_code = TRUE is available under backend = "brms" and under "auto" (which resolves the code modes to brms). Under backend = "inla" it refuses with review_code_backend_unsupported: the deferred-execution token would need an INLA-side code slot, which is queued for a later release.

backend — Character: one of "auto" (**default**), "inla", or "brms". INLA fits the latent-Gaussian model class by nested Laplace approximation, without sampling. brms is the Stan passthrough. Any other value, including a formerly-registered engine name this package withdrew entirely (see NEWS.md), refuses with unknown_backend naming the two active engines.

What "auto" does. The call runs lgm_gate() and routes to INLA when the model is latent-Gaussian feasible and INLA is installed. Otherwise it routes to brms when brms is installed *and* its capability predicate accepts the model. When neither can represent the model, the call refuses with auto_no_active_route instead of fitting something else. Nothing is translated silently. auto does change engine in one case after starting: an INLA program failure is caught and re-routed to brms, and a message reports it. The fit's class and backend field always record which engine produced it. backend_decision() surfaces the full dispatch trace (including rejected_routes) post-fit, and fb_backend_status() reports which engines are usable. The one-time notes on the auto path are silenceable via options(flexyBayes.silence_auto_fallback_note = TRUE) (gate refusal or an INLA numerical failure) and options(flexyBayes.silence_auto_inla_missing_n = TRUE) (INLA not installed).

What each engine represents. INLA fits: Gaussian LMM, simple random intercept; GLMM (binomial, Poisson, negative binomial, gamma, beta), simple random effect; Ordinal regression (cumulative, continuation ratio); Uncorrelated random slope; Factor-by-numeric fixed interaction; Nested / interaction random effects, multi-stratum; Heterogeneous variance by factor level; Heterogeneous residual by factor level; Baseline heteroscedastic MET: iid genotype, iid genotype-by-environment and a sectioned residual; Known-covariance genomic / pedigree random effect; Separable AR1 spatial field; Per-trial separable AR1 field; Grouped temporal AR1 field; Univariate P-spline; Observation weights (Gaussian, identity link); Exact sufficient-statistic aggregation (binomial, Poisson). INLA refuses: Hurdle gamma (zero mass plus a positive gamma part); Correlated random slope; Unstructured genotype-by-environment covariance; Heterogeneous variances with one shared correlation; Factor-analytic genotype-by-environment covariance; Multi-trait covariance; Exact sufficient-statistic aggregation (Gaussian). brms fits: Gaussian LMM, simple random intercept; GLMM (binomial, Poisson, negative binomial, gamma, beta), simple random effect; Hurdle gamma (zero mass plus a positive gamma part); Ordinal regression (cumulative, continuation ratio); Uncorrelated random slope; Factor-by-numeric fixed interaction; Correlated random slope; Nested / interaction random effects, multi-stratum; Heterogeneous variance by factor level; Unstructured genotype-by-environment covariance; Heterogeneous residual by factor level; Baseline heteroscedastic MET: iid genotype, iid genotype-by-environment and a sectioned residual; Known-covariance genomic / pedigree random effect; Observation weights (Gaussian, identity link). brms refuses: Heterogeneous variances with one shared correlation; Factor-analytic genotype-by-environment covariance; Multi-trait covariance; Separable AR1 spatial field; Per-trial separable AR1 field; Grouped temporal AR1 field; Univariate P-spline. The per-class table is in `README.md`, and with a note on each row in `system.file("KNOWN_ISSUES.md", package = "flexyBayes")`.

Caution (small-group random effects). Both engines apply the same default prior, the half-normal on each standard deviation, which the INLA path represents exactly as an expression prior on the log-precision, so they agree on the variance component far more closely than in earlier versions. A model with very few groups nonetheless carries a weakly-identified variance component (the data say little about the between-group spread), and INLA's Laplace approximation is less accurate there than full MCMC. For a flagship random-intercept model with few groups, pin `backend = "brms"` or supply an explicit informative prior (a half-Cauchy, per Gelman 2006, as in the *priors and regularisation* vignette).

Convergence. The brms fit emits a warning when the sampler may not have converged (a parameter with Rhat at or above 1.1, or a low effective sample size). Treat such a posterior with caution: increase `warmup` / `n_samples`, simplify the model, or supply a more informative prior, and inspect the full diagnostics with `summary()`. Silence the warning (for intentionally short fits) via `options(flexyBayes.silence_convergence_warning = TRUE)`. An INLA fit does not sample and carries no such warning.

Code inspection. brms is the only engine with a code slot, so `return_code = TRUE` / `review_code = TRUE` under `"auto"` resolve to brms, but only when brms can represent the model. Where it cannot, the call refuses with `auto_no_active_route`

instead of returning the code for a different model. `backend = "inla"` with `return_code = TRUE` returns the INLA formula, family and hyperparameter list. With `review_code = TRUE` it refuses until the INLA-side code slot lands.

`aggregate` One of `"auto"` (default), `TRUE`, or `FALSE`. Exact sufficient-statistics aggregation gate. INLA is the only engine with an aggregated emit. `"auto"` consults the aggregation plan and routes a binomial-logit or poisson-log model through the per-cell path when the IR is in scope (fixed plus random intercept; productive compression) and the model reaches INLA. A Gaussian model is fitted per row: the aggregated Gaussian route cannot yet carry the residual prior, so `TRUE` on a Gaussian model refuses by name (`aggregation_residual_prior_not_composed`). `TRUE` also raises a structured refusal when the plan declares ineligibility, or when no active engine offers an aggregated route. `FALSE` skips the gate entirely. Aggregated fits carry `$exactness == "aggregated_exact"`, the dispatch trace's `path` slot reads `"aggregated_count"`, and their summary reports no marginal likelihood (`aggregated_evidence_unavailable`). For out-of-core datasets that do not fit in memory, see `flexybayes_stream()`.

`plan` Logical: if `TRUE`, short-circuit after intermediate representation (IR) build and return a `<fb_plan>` object carrying the IR, the routing decision, the representation plan, the aggregation plan, and the prediction plan, *without* emitting backend code or fitting. Equivalent to calling `fb_plan()` with the same formula triple, exposed inline as `flexybayes(plan = TRUE)` so users can reach the planning object without re-typing the call. Default `FALSE` preserves the run- immediately semantics. Mutually exclusive with `return_code` and `review_code`.

`syntax` One of `"auto"` (default), `"asreml"`, or `"brms"`. Selects how `fixed` is interpreted. `"auto"` detects the grammar from the call shape (a bar-grouped formula is read as brms, otherwise ASReml). The other values force a grammar.

`known_variance` Optional numeric vector of length N, or an N x N covariance matrix (below). A vector is a known variance for each response, added to the estimated residual variance, `Var(y_i) = sigma^2 + v_i`. This is the second stage of a two-stage analysis, where each response is a first-stage estimate (a variety mean in one trial, say) and `v_i` is the variance of that estimate. `weights` is a different model, `Var(y_i) = sigma^2 / w_i`. Given both, the residual variance is `sigma^2 / w_i + v_i`. Lowered for `family = "gaussian"` with the identity link: brms fits `y | se(sqrt(v), sigma = TRUE)`, and INLA a row effect with known variance `v_i`. The summaries report `sigma` as the estimated part, and fitted values on both engines leave the known part out. A vector of zeros fits the model without it. Any other family refuses with `known_variance_not_supported`, and a vector that is not one finite, non-negative value per row with `known_variance_invalid`. A symmetric positive-definite N x N matrix `K` is a known covariance, `Var(y) = sigma^2 I + K`, as when the first stage returns the full covariance of its estimates. It is fitted on INLA as a latent effect with precision `K^-1`. brms has no term for it and refuses with `known_variance_not_supported`. A row and column of zeros marks a row without a known part. A matrix that is not square, symmetric and positive definite over the rows of `data` refuses with `known_variance_invalid`.

`fixed_alias` `"refuse"` (the default) or `"drop"`: what to do with a fixed-effect column that is a linear combination of the columns before it on the rows with a response, as in a factorial with empty cells. INLA and brms would report such a column at its

prior, so by default the fit refuses with `fixed_effects_aliased` and names the columns. `"drop"` drops them as asreml 4.2 does, on INLA: terms are taken in model order, and within a term by descending factor levels, the term's first factor varying slowest. A column that is a linear combination of the columns before it is fixed at zero. `summary()` reports it with estimate 0 and no standard error, the columns are listed in `fit$extras$fixed_alias$dropped`, and a prediction whose row is not a combination of the fitting rows is `NA` with status `"Aliased"` (see `predict.flexybayes_inla()`).

`rcov` Defunct in flexyBayes 0.9.0. This was the ASReml 3 name for the residual-structure argument. ASReml 4 renamed it to `residual`, and flexyBayes follows the ASReml 4 name. Supplying `rcov` now raises an error. Use `residual` instead.

Value

An object of class `"flexybayes"`, a list with three components.

`$glm` A GLM-compatible object (class `c("flexybayes_glm", "glm", "lm")`) with posterior mean coefficients, vcov, residuals, fitted values, etc. Works with `summary()`, `emmeans()` and `marginaleffects()`.

`$brms` **or** `$inla` The native backend object: a live `brmsfit` on the brms path, or INLA's own fitted object on the INLA path. Use with `bayesplot`, `posterior::as_draws()`, and each engine's own accessors.

`$extras` Additional outputs: posterior `summary`, `convergence` diagnostics, `variance_comps`, `blups`, `predictions`, generated `code`, `param_names`, `parse_info`, `call_info`, `run_time`, `model_info`, and `reproducibility`: the R, flexyBayes and engine versions, the thread settings and the seed the fit ran with, which `print()` shows.

If `return_code = TRUE`, returns the generated backend code instead: a character string holding the Stan program on the brms path, or a list of the formula, family and hyperparameter specification on the INLA path.

INLA threads

An INLA fit runs on one thread, `num.threads = "1:1"`, whatever `INLA::inla.setOption()` holds. With more threads, identical calls can return different variance components where the data inform a variance only weakly. On a 48-trial barley model with 22 localities, eight identical fits at `"4:1"` gave the locality variance from 0.58 to 42.3, and at `"1:1"` from 0.75 to 0.79, with the fixed effects unchanged. Set `options(flexyBayes.inla_threads = "4:1")` for another plan, or `"inla"` for INLA's own default. The plan a fit ran with is in `$extras$reproducibility`.

One thread does not guarantee the same result in a new R session. Identical calls in fresh R processes can reach different modes. On a 48-trial barley series, four processes reached two modes, 0.10 apart in log marginal likelihood, whose variance component standard deviations differed by at most 3.4%. On six trials of the same series, the modes differed by up to 26%.

Each INLA fit is therefore run three times: once in this session and twice in fresh R processes started through callr. The fit with the highest log marginal likelihood is returned (the first of equal ones). `options(flexyBayes.inla_best_of = k)` sets the number of fits, a whole number of one or more; 1 fits once in this session, as 0.16.0 did, and any other value refuses with `inla_best_of_invalid`. Above 1 without callr installed, the fit refuses with `isolation_requires_callr`. A fit that fails in a

separate process stops the call with that process's message. `$extras$reproducibility$best_of` holds each fit's log marginal likelihood and mode status, the fit kept, the range of the marginal likelihoods, and each standard deviation per fit with its largest difference across the fits, relative to the smallest. `print()` and `summary()` add a line when a standard deviation differs by more than 5%. The restart probe of the numerical-confirm gate examines the fit kept. The fresh-process fits run beside the session fit, at most `options(flexyBayes.inla_best_of_workers)` fits at once (default: all of them) and fewer, with a message, when the free memory holds fewer. The record gives each fit's time in seconds (`wall_s`) and whether they ran at once (`parallel`).

Fitting in a separate process

`options(flexyBayes.isolation = "process")` runs the engine fit in a separate R process started through callr, which is in Suggests. Without callr the request refuses with `isolation_requires_callr` before anything is fitted. The option applies to `flexybayes()`, `fb_brms()`, `fb_inla()` and `proceed()`, and `fb_isolated_lapply()` runs any loop of calls the same way. The default, `"none"`, fits in this session.

If the operating system kills that process, or compiled code crashes inside it, the call refuses with `isolated_process_died`, naming the engine, the function called, and the exit status or signal, and this session carries on. A failure inside the fit, a refusal for instance, arrives exactly as it would without the option. The call is not retried.

The separate process loads the flexyBayes code this session loaded and starts from this session's options, INLA settings, environment variables and random-number state, so it fits the same model with the same seed, and the random-number stream moves on as it would have. Messages, warnings and printed output come back in the order they were raised, and the fit records the process in `$extras$isolation`. A brms fit with a seed returns the same draws either way. INLA does not return identical numbers for two identical calls in one session, and an isolated INLA fit agrees with an in-process one as closely as two in-process fits agree. The separate process starts with an empty global environment, so the formula finds its variables in `data`. Starting the process and moving the data and the fit costs time on every call.

Deprecated

`fb()` is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Call `flexybayes()`, which takes the same arguments. A call to `fb()` warns with a condition of class `flexybayes_deprecated`.

References

Cao Z, Megirian R, Nguyen M, Sparks AH (2026). Bayesian ordinal analysis of cereal growth stage data from oat trials in Western Australia. arXiv:2610.37384.

Examples

```
## Not run:
# live fit, needs an active backend (INLA or brms/Stan)
data(met_example, package = "flexyBayes")
# Simple random intercept model (small budget for example purposes)
fit <- flexybayes(
  fixed  = yield ~ env,
  random = ~ geno,
```

```
  data   = met_example$dat,
  n_samples = 100, warmup = 100, chains = 1, verbose = FALSE
)
summary(fit)
coef(fit)

## End(Not run)
```

`flexybayes_stream` *Fit a mixed model to an out-of-core dataset by streaming aggregation*

Description

`flexybayes_stream()` is the large-data entry point: it reads the data a chunk at a time from `source`, accumulates additive sufficient statistics per design cell, and fits the resulting aggregated model through the same INLA emit path as `flexybayes()`. The aggregated likelihood is algebraically identical to the per-row likelihood, so the posterior is the full-data posterior, not an approximation.

Usage

```
flexybayes_stream(
  fixed,
  random = NULL,
  source,
  family = "gaussian",
  trials = NULL,
  exposure = NULL,
  backend = "inla",
  chunk_rows = 5e+06,
  prior = NULL,
  fit = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`fixed`	A two-sided formula `response ~ fixed_terms`. The fixed terms must be factors or factor interactions. A continuous term forces one cell per row and is refused. `offset()` terms are allowed on the Gaussian and Poisson families (see Details).
`random`	A one-sided formula of random-intercept grouping factors, ASReml style (for example `~ geno`), or `NULL` for no random terms.
`source`	The data source. One of: a length-1 character path to an `.fst` file, a length-1 character path to a delimited file readable by `data.table::fread()`, an in-memory `data.frame` / `data.table` (chunked internally, mainly for testing), or a function of one argument `i` returning the `i`-th chunk as a `data.frame` and `NULL` once the chunks are exhausted.

family	A length-1 character family: "gaussian", "binomial", or "poisson".
trials	For family = "binomial", the name of the trials-count column, or NULL for Bernoulli (one trial per row).
exposure	For family = "poisson", the name of the exposure column, or NULL for unit exposure. An offset(log(E)) in fixed is the same model as exposure = "E".
backend	The estimation backend. "inla" is the default and the only active choice: INLA is the one engine with an aggregated emit. Any other value, including a formerly-registered engine name this package withdrew entirely, refuses with unknown_backend.
chunk_rows	The number of rows to read per chunk. Larger chunks read faster but use more peak memory. The default 5e6 keeps peak memory modest while amortising read overhead.
prior	An optional fb_prior() object. When NULL, each random-effect standard deviation takes the default flexybayes() applies, half_normal(scale = 3) for Poisson and half_normal(scale = 5) for binomial, on the link scale. The fixed effects keep INLA's default priors. prior_summary() lists the prior on every parameter of the fit.
fit	When TRUE (default) the aggregated model is fitted and a <flexybayes> object is returned. When FALSE the function returns the <fb_aggregated> sufficient-statistics object without fitting, which shows the compression a design will achieve.
verbose	When TRUE, report streaming progress and the compression achieved.
...	Further arguments passed to the aggregated emit.

Details

Fit a cell-aggregatable mixed model to a dataset held on disk, reading it in row-range chunks and accumulating exact per-cell sufficient statistics instead of materialising the full table in memory.

The method scales by cell count K (the number of distinct factor design combinations), not by row count N. A replicated factorial design collapses billions of rows to a few thousand cells and fits in seconds. A design that keys on a continuous covariate does not compress and is refused before any chunk is read. Continuous fixed effects, random slopes, structured covariance, and smooth terms break the cell-constant linear-predictor property and are likewise refused. Pass the data to flexybayes() per row for those models. The model needs at least one factor cell-key term (a fixed factor, a factor interaction, or a random grouping factor named in random) to aggregate by. An intercept-only model (fixed = y ~ 1, no random) has nothing to compress and is refused by name (aggregation_route_unavailable).

Supported families for a fit are "binomial" and "poisson". A "gaussian" fit refuses by name (aggregation_residual_prior_not_composed): the aggregated Gaussian route cannot yet carry the residual prior, so fit such data per row with flexybayes(). fit = FALSE still returns the Gaussian sufficient statistics. For binomial data supply trials to name the column of trial counts (a 0/1 response is treated as Bernoulli with one trial per row). For poisson data supply exposure to name an exposure column. A fit on the aggregated representation reports no marginal likelihood (aggregated_evidence_unavailable).

An `offset()` in `fixed` is carried exactly on two of the three families. On `"gaussian"` it is subtracted from the response before the cell statistics are taken, which reproduces the per-row likelihood however the offset varies inside a cell. On `"poisson"`, `offset(o)` multiplies each row's exposure by `exp(o)`, so `offset(log(E))` and `exposure = "E"` fit the same model. Given both, they multiply. On `"binomial"` a cell total of rows whose offsets differ is not binomial, and the call refuses with `fixed_offset_unsupported`.

Value

When `fit = TRUE`, a `<flexybayes>` fit object carrying `extras$aggregation_meta$streamed == TRUE`. When `fit = FALSE`, an `<fb_aggregated>` object.

Author(s)

Max Moldovan, `<max.moldovan@adelaide.edu.au>`

See Also

`flexybayes()` for the in-memory entry point, and `fb_prior()` for prior specification.

Examples

```
# INLA is distributed from its own repository, not from CRAN.
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1L)
  n <- 1e6
  df <- data.frame(
    env = factor(sample(letters[1:6], n, replace = TRUE)),
    geno = factor(sample(1:50, n, replace = TRUE)),
    count = rpois(n, 3)
  )
  path <- tempfile(fileext = ".fst")
  fst::write_fst(df, path)
  fit <- flexybayes_stream(count ~ env, random = ~ geno, source = path,
                           family = "poisson", backend = "inla")
  summary(fit)
}
```

---

formula.flexybayes *Extract model formula*

Description

Extract model formula

Usage

```
## S3 method for class 'flexybayes'
formula(x, ...)
```

Arguments

x	A fitted flexybayes object of any backend.
...	Ignored, present for compatibility with the generic.

Value

The fixed-effect (population-level) formula as a formula object. Random-effect and residual-structure terms are not part of it: read them from the fit's fb_terms intermediate representation, or from fb_plan() before fitting.

---

formula.flexybayes_inla *Fixed-effect model formula of a per-row INLA fit*

Description

The fixed-effect (population-level) formula, recovered from the captured call. Random-effect and residual-structure terms are not part of this formula.

Usage

```
## S3 method for class 'flexybayes_inla'
formula(x, ...)
```

Arguments

x	A flexybayes_inla fit.
...	Ignored. Present for compatibility with the generic.

Value

A formula.

---

genomic_summary *Genomic summary of a fitted relationship model*

Description

Extract the breeder-facing genomic quantities: narrow-sense heritability h^2, genomic estimated breeding values (GEBVs) with posterior reliability, and the genetic / residual variances, from a fitted vm() (genomic / GBLUP) or ped() (pedigree) model. The quantities are read from the posterior draws engine-agnostically: an INLA or brms GBLUP fit returns the same summary object, so the two engines' genomic summaries can be read side by side.

Usage

```
genomic_summary(fit, term = NULL)
```

Arguments

`fit`	A fitted `flexybayes` object carrying at least one `vm()` or `ped()` relationship term.
`term`	Optional grouping-factor name selecting which relationship term to summarise when the model has more than one. Defaults to the first.

Details

The heritability is computed per draw as $h^2 = \sigma^2_g/(\sigma^2_g + \sigma^2_e)$ on the genotype-mean basis. The kinship scaling convention is the analyst's (state it when reporting). Reliability is $1 - \mathrm{PEV}_i/\sigma^2_g$ from the posterior variance of each breeding value. GEBVs are available on both active backends (brms and INLA) natively.

Value

An `fb_genomic_summary` object: `heritability`, `genetic_variance`, `residual_variance` (each a posterior summary), `gebv` (data frame of breeding values with reliability), and metadata.

See Also

`vpredict()` for posterior functionals of the variance components, and `ranef()` for the group-level effects themselves.

Examples

```
## Not run:
fit <- flexybayes(
  yield ~ 1, random = ~ vm(geno, Gmat), data = met,
  known_matrices = list(Gmat = G), backend = "brms"
)
gs <- genomic_summary(fit)
gs$heritability
head(gs$gebv)

## End(Not run)
```

`get_coef.flexybayes` *marginaleffects support: fixed-effect coefficients (default method)*

Description

marginaleffects support: fixed-effect coefficients (default method)

Usage

```
## S3 method for class 'flexybayes'
get_coef(model, ...)

## S3 method for class 'flexybayes_inla'
get_coef(model, ...)
```

Arguments

model	A flexybayes_inla fit.
...	Ignored. Present for compatibility with the generic.

Value

Named numeric vector of coefficients.

---

get_data.flexybayes *Model data accessor (default method)*

Description

Registered for insight::get_data() so marginaleffects (which discovers a model's data through insight) can build reference grids and average predictions without an explicit newdata.

Usage

```
## S3 method for class 'flexybayes'
get_data(x, ...)

## S3 method for class 'flexybayes_inla'
get_data(x, ...)
```

Arguments

x	A flexybayes_inla fit.
...	Ignored. Present for compatibility with the generic.

Value

The model data frame.

get_predict.flexybayes *marginaleffects support: population-level predictions (default method)*

Description

marginaleffects support: population-level predictions (default method)

Usage

```
## S3 method for class 'flexybayes'
get_predict(model, newdata = NULL, type = "response", ...)

## S3 method for class 'flexybayes_inla'
get_predict(model, newdata = NULL, type = "response", ...)
```

Arguments

model	A flexybayes_inla fit.
newdata	Data frame to predict on (default: fit data).
type	Prediction scale (identity link only).
...	Ignored. Present for compatibility with the generic.

Value

A data frame with rowid and estimate.

get_vcov.flexybayes *marginaleffects support: covariance (default method)*

Description

marginaleffects support: covariance (default method)

Usage

```
## S3 method for class 'flexybayes'
get_vcov(model, ...)

## S3 method for class 'flexybayes_inla'
get_vcov(model, ...)
```

Arguments

model	A flexybayes_inla fit.
...	Ignored. Present for compatibility with the generic.

Value

Fixed-effect covariance matrix.

---

glance.flexybayes *Glance at a flexyBayes fit*

Description

Returns a one-row data.frame of model-level statistics: the response family and link, the observation and parameter counts, the log-likelihood, the worst convergence diagnostics across monitored parameters, and the wall-clock run time.

Usage

```
## S3 method for class 'flexybayes'
glance(x, ...)

## S3 method for class 'flexybayes_inla'
glance(x, ...)
```

Arguments

x	A flexyBayes fit (flexybayes or flexybayes_brms).
...	Currently unused. Present for generic compatibility.

Value

A one-row data.frame with nobs, npar, logLik, family, link, chains, samples, max_rhat, min_ess, and run_time.

See Also

tidy.flexybayes()

Examples

```
# Not run: glance() reports sampler diagnostics, so the example needs
# a sampled fit and therefore a Stan compile. The fragment is
# complete and runs as written where a C++ toolchain is present.
## Not run:
set.seed(1)
dat <- data.frame(yield = rnorm(40), env = factor(rep(1:4, each = 10)))
```

```
fit <- flexybayes(yield ~ env, data = dat, backend = "brms",
                  chains = 1L, n_samples = 200L, warmup = 100L)
glance(fit)

## End(Not run)
```

`is_fb_approx` *Test whether an object is an* `fb_approx` *specification*

Description

Test whether an object is an `fb_approx` specification

Usage

```
is_fb_approx(x)
```

Arguments

`x`	Any R object. The test is a class check, not a structural one.

Value

`TRUE` if `x` is an `fb_approx` object, `FALSE` otherwise.

Deprecated

`is_fb_approx()` is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Use `inherits(x, "fb_approx")`. A call warns with a condition of class `flexybayes_deprecated`.

Examples

```
a <- fb_approx("low_rank_smooth")
inherits(a, "fb_approx")
inherits("low_rank_smooth", "fb_approx")
```

is_fb_cov *Test whether an object is an* fb_cov *carrier*

Description

Test whether an object is an fb_cov carrier

Usage

```
is_fb_cov(x)
```

Arguments

x	Any R object. The test is a class check, not a structural one.

Value

TRUE if x is an fb_cov object, FALSE otherwise.

Deprecated

is_fb_cov() is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Use inherits(x, "fb_cov"). A call warns with a condition of class flexybayes_deprecated.

Examples

```
K <- diag(3)
dimnames(K) <- list(letters[1:3], letters[1:3])
inherits(fb_cov(K, type = "dense"), "fb_cov")
inherits(K, "fb_cov")
```

is_fb_engine *Test whether an object is an* fb_engine *specification*

Description

Test whether an object is an fb_engine specification

Usage

```
is_fb_engine(x)
```

Arguments

x	Any R object. The test is a class check, not a structural one.

Value

TRUE if x is an fb_engine object, FALSE otherwise.

Deprecated

is_fb_engine() is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Use inherits(x, "fb_engine"). A call warns with a condition of class flexybayes_deprecated.

Examples

```
inherits(fb_engine("inla"), "fb_engine")
inherits("inla", "fb_engine")
```

---

logLik.flexybayes *Plug-in conditional log-likelihood of a flexyBayes fit*

Description

Evaluates the conditional log-likelihood at the posterior-mean fitted values: a plug-in quantity, not a posterior summary, and not the marginal likelihood. It exists so AIC()-style comparisons of two fits from the same engine have something to read.

Usage

```
## S3 method for class 'flexybayes'
logLik(object, ...)
```

Arguments

object	A flexybayes fit carrying a response vector, fitted values, and a recorded response family.
...	Ignored, present for compatibility with the generic.

Details

The method computes what the object carries and refuses by name when it carries too little. Three requirements are checked before any arithmetic runs: the response vector in glmy, the fitted values from fitted(), and a recorded family. A Gaussian fit uses the residual root-mean-square as its plug-in scale. Families outside the Gaussian, binomial and Poisson set have no plug-in form here and are refused and not reported as NA, because a silent NA propagates into anova() and AIC() as though the comparison had been made.

brms fits reach logLik.flexybayes_brms() instead (brms's own log_lik()), and INLA fits reach logLik.flexybayes_inla(), which states that INLA reports a marginal log-likelihood and returns NA with that message.

Value

A `logLik` object: the scalar log-likelihood with `df` and `nobs` attributes taken from the fit's recorded model information.

`logLik.flexybayes_brms` *Log-likelihood on the brms path*

Description

Delegates to `brms::log_lik()` then sums pointwise log-likelihood across observations and averages across draws. The `df` attribute carries the parameter count from `$extras$model_info`; `nobs` carries the observation count.

Usage

```
## S3 method for class 'flexybayes_brms'
logLik(object, ...)
```

Arguments

`object`	A `flexybayes_brms` object.
`...`	Ignored. Present for compatibility with the generic.

Value

An object of class `logLik`: the pointwise log-likelihood summed over observations and averaged across draws, carrying `df` (parameter count) and `nobs` (observation count) attributes.

`logLik.flexybayes_inla` *Log-likelihood of a per-row INLA fit (not computed)*

Description

INLA reports a *marginal* log-likelihood (the model evidence, available through `summary()`), not the *conditional* model log-likelihood that the `logLik()` generic denotes and that the brms backend exposes. Returning the marginal quantity under the `logLik` name would conflate two different things, so this method returns `NA` (with the degrees of freedom and observation count filled in) and a one-line note. This also lets downstream summaries (for example `glance()`) degrade gracefully instead of erroring with "no applicable method".

Usage

```
## S3 method for class 'flexybayes_inla'
logLik(object, ...)
```

Arguments

object	A flexybayes_inla fit.
...	Ignored. Present for compatibility with the generic.

Value

A logLik object whose value is NA_real_, carrying df and nobs attributes.

---

loo.flexybayes *Approximate leave-one-out cross-validation for a flexyBayes fit*

Description

A method for the **loo** package's loo() generic. On a fit from the **brms** engine the call passes through to brms::loo(), which computes PSIS-LOO from the pointwise log-likelihood Stan stored, and returns **loo**'s own object, elpd_loo, p_loo, looic and the Pareto-k diagnostics, unchanged.

Usage

```
## S3 method for class 'flexybayes'
loo(x, ...)
```

Arguments

x	A fitted flexybayes object.
...	Passed to brms::loo() on the brms path. Ignored on the refusal paths.

Details

On a fit from the **INLA** engine the method refuses by name. INLA returns a nested Laplace approximation to the posterior, not draws of the log-likelihood, so there is no quantity for importance sampling to reweight and no leave-one-out estimate to report. The refusal names the information criteria the fit did compute, WAIC at fit$inla$waic$waic and DIC at fit$inladicdic on a per-row fit, so the alternative is in the message, not left to be found. Neither is a leave-one-out estimate and neither carries Pareto-k diagnostics.

A fit that carries no posterior at all is refused as fit_lacks_posterior_draws instead, so the two states are told apart by the condition class, not by reading the message.

Value

On a brms-engine fit, the loo object brms::loo() returns. Otherwise the method does not return: it raises a refusal of class flexybayes_refusal_loo_requires_sampler_draws, or flexybayes_refusal_fit_lacks_post when the fit carries no posterior.

See Also

summary.flexybayes(), whose $converge slot carries the engine's own diagnostics. as_draws_df.flexybayes() for the posterior itself. fb_refusals() for the refusal vocabulary.

`met_example` *Example Multi-Environment Trial (MET) Dataset*

Description

A synthetic multi-environment trial dataset suitable for demonstrating mixed model formulations. Contains yield data for 10 genotypes evaluated across 4 environments with 3 replicates per environment, plus spatial layout, genomic markers, and multiple response types.

Usage

```
met_example
```

Format

A list with the following components:

`dat` A data frame with 120 rows and the following columns:

- `geno` Factor: genotype identifier (G1–G10)
- `env` Factor: environment identifier (E1–E4)
- `rep` Factor: replicate (R1–R3)
- `block` Factor: incomplete block within rep (B1–B6)
- `row` Factor: row position in field layout
- `col` Factor: column position in field layout
- `yield` Numeric: grain yield (continuous, primary response)
- `x_cov` Numeric: a continuous covariate
- `bin_y` Integer: binary response (0/1)
- `count_y` Integer: count response (Poisson-like)

`G_mat` 10x10 genomic relationship matrix (G = MM'/2pq)

`A_mat` 10x10 pedigree-based relationship matrix

`M_geno` 10x50 marker matrix (50 SNP markers for 10 genotypes)

`n_geno` Integer: number of genotypes (10)

`n_env` Integer: number of environments (4)

Details

The dataset is generated with known variance components:

- Genotype variance: sigma_g^2 = 4
- Environment variance: sigma_e^2 = 9 (varies by environment)
- GxE variance: sigma_ge^2 = 2
- Residual variance: sigma^2 = 1

The genomic relationship matrix is computed from 50 simulated SNP markers using the VanRaden method 1. The pedigree matrix uses a simple half-sib structure.

Source

Generated synthetically for package demonstration purposes.

Examples

```
data(met_example)
str(met_example$dat)
head(met_example$dat)

# Simple model (small budget for example purposes)
## Not run:
# A live fit needs a backend (INLA, or brms/Stan).
fit <- flexybayes(
  fixed  = yield ~ env,
  random = ~ geno,
  data   = met_example$dat,
  n_samples = 100, warmup = 100, chains = 1, verbose = FALSE
)
summary(fit)

## End(Not run)
```

`model.matrix.flexybayes` *Fixed-effect model matrix of a flexyBayes fit*

Description

Rebuilds the population-level design matrix from the formula and data the fit carries. Random-effect and residual-structure terms are not part of it: this is the basis the fixed-effect coefficients are expressed in.

Usage

```
## S3 method for class 'flexybayes'
model.matrix(object, newdata = NULL, ...)
```

Arguments

`object`	A `flexybayes` fit carrying a fixed-effect formula and the data it was fitted to.
`newdata`	Optional data frame of rows to build the design for. The response may be absent.
`...`	Ignored, present for compatibility with the generic.

Details

The formula and data are resolved from whichever slots the object holds. The brms-shaped emit keeps both under `$glm`. An INLA fit keeps its data at `$data` and recovers its fixed-effect formula through `formula.flexybayes_inla()`. An object supplying neither is refused by name, since `model.matrix()` on a `NULL` formula silently returns the intercept-only matrix of the calling frame's data.

An aggregated fit keeps its call, not a formula under `$glm` and recovers the formula the same way. A streamed fit that keeps fewer rows than it was fitted to refuses by name (`streamed_rows_not_retained`).

With `newdata`, the design is built for those rows instead, with each factor coded on the levels of the fitted data, so its columns are the fit's coefficients in their order: `model.matrix(fit, newdata) %*% coef(fit)` is the population-level prediction.

Value

A numeric matrix with one row per observation (of `newdata` where given) and one column per fixed-effect basis column. Without `newdata` it carries the usual `assign` attribute.

`nobs.flexybayes` *Number of observations a flexyBayes fit was fitted to*

Description

A fit whose missing responses were augmented, not dropped was handed more rows than it has observations: the design cell of a lost plot is still a row, carried as a latent quantity so the index set a structured covariance is built over survives. The two counts are therefore different numbers and the argument says which one is wanted.

Usage

```
## S3 method for class 'flexybayes'
nobs(object, type = c("design", "observed"), ...)
```

Arguments

`object`	A `flexybayes` fit of any engine.
`type`	Which count to return: `"design"` (default) or `"observed"`.
`...`	Ignored, present for compatibility with the generic.

Details

`type = "design"` (the default, and the historical behaviour) is the number of rows the engine saw. `type = "observed"` is how many of those carried an observed response, read from the record the missing-response layer left on the fit.

Value

A single integer.

See Also

summary.flexybayes(), whose n_design and n_observed slots are the same two numbers.

---

ordinal_probabilities *Category probabilities of an ordinal fit*

Description

fitted() returns, for each row the model was fitted to, the posterior mean probability of each category, at the fitted values of the random effects. predict() returns the same probabilities for the rows of newdata, with the random effects set to zero, or with marginal = TRUE integrated over their fitted variances, which is the probability for a new unit in a new block.

Usage

```
## S3 method for class 'flexybayes_ordinal'
fitted(object, n_draws = 1000L, seed = NULL, summary = TRUE, ...)

## S3 method for class 'flexybayes_ordinal'
predict(
  object,
  newdata = NULL,
  marginal = FALSE,
  n_draws = 1000L,
  seed = NULL,
  summary = TRUE,
  ...
)
```

Arguments

object	An ordinal fit, from flexybayes() with family = "cumulative" or "cratio".
n_draws	The number of posterior draws. Default 1000. On brms it is at most the number the sampler stored.
seed	An integer seed for the draws, or NULL (the default) to use the session's random number stream.
summary	TRUE (the default) returns the posterior mean. FALSE returns every draw.
...	Ignored by fitted(). In predict(), an argument of the other predict() methods that has no meaning here is refused by name.
newdata	A data frame holding the fixed-effect variables, or NULL (the default) for the rows the model was fitted to.
marginal	FALSE (the default) sets every random effect to zero. TRUE integrates over them.

Details

Each probability is averaged over `n_draws` joint posterior draws of the thresholds, the coefficients and the variances, so it carries the uncertainty of all three. With `marginal = TRUE` the random terms of a new unit enter as one normal deviation whose variance is the sum of the fitted variances (random intercepts, their interactions and an AR1 field, whose marginal variance is its fitted variance), and the probability is integrated over it by 40-node Gauss-Hermite quadrature in each draw. On brms, `fitted()` reads `brms::posterior_epred()`.

The probability of a set of categories is the sum of their columns, for example `rowSums(p[, c("Z5", "Z6")])` for a stage of Z5 or later.

Value

A matrix with one row per row of the data and one column per category, named by the category labels. With `summary = FALSE`, an array, draws by rows by categories.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(
    trt = factor(rep(c("a", "b"), each = 60)),
    block = factor(rep(1:6, times = 20))
  )
  eta <- 0.8 * (d$trt == "b") + rnorm(6, 0, 0.3)[d$block]
  d$score <- cut(eta + rlogis(120), c(-Inf, -0.5, 0.5, 1.5, Inf),
    labels = c("low", "mid", "high", "top"), ordered_result = TRUE)
  fit <- flexybayes(score ~ trt, random = ~block, data = d,
    family = "cumulative", backend = "inla", verbose = FALSE)
  head(fitted(fit, seed = 1L))
  predict(fit, newdata = data.frame(trt = c("a", "b")), marginal = TRUE,
    seed = 1L)
}
```

`plot.flexybayes` *Plot diagnostics for a flexyBayes model*

Description

Draws one of eight standard displays for a fitted model, chosen by `type`. Every backend reaches the same method, and a display that needs a slot a given engine does not populate (posterior draws on an INLA fit, for instance) prints a message naming what is missing and returns invisibly instead of erroring or drawing an empty panel. The one exception is `"pp_check"`, which raises the refusal `pp_check.flexybayes()` raises, so a caller can catch it by class.

Usage

```
## S3 method for class 'flexybayes'
plot(x, type = NULL, ...)

## S3 method for class 'flexybayes_inla'
plot(x, ...)

## S3 method for class 'flexybayes_brms'
plot(x, ...)

## S3 method for class 'flexybayes_aggregated'
plot(x, ...)

## S3 method for class 'flexybayes_glm'
plot(x, ...)
```

Arguments

x	A fitted flexybayes object of any backend. Aggregated and generalised-linear fits reach the same method through their own registrations.
type	A single string naming the display to draw. One of: "diagnostics", trace plots and marginal densities per parameter (needs a sampled posterior), "residuals", residuals against fitted values beside a normal quantile-quantile plot, "effects", a forest plot of the fixed effects with their credible intervals, available on every backend that supplies coef() and confint(). "variance", a bar chart of the variance components with credible intervals, "blups", a caterpillar plot of the random-effect predictions ordered by magnitude, "pp_check", a posterior predictive check overlaying replicated datasets on the observed response, which needs predictive draws and so refuses by name on a fit that carries none (see pp_check.flexybayes()), and "variogram", the empirical semivariance of the residuals over the design index, and "spatial_field", the posterior mean of a separable autoregressive field, random = ~ ar1(row):ar1(col), drawn on the row-by-column array of the trial on the response scale. NULL (the default) resolves to "diagnostics" on a sampled fit and "residuals" otherwise.
...	Further arguments passed to the underlying plotting call, for example variable to restrict which parameters a diagnostic display covers, or type and ndraws for "pp_check". The spatial field takes trial, the identifier printed in its title and the only label it adds to the array's own row and column labels. On a fit with one field per trial, random = ~ id(trial):ar1(row):ar1(col), it names the level whose field is drawn.

Details

The default display depends on what the fit carries. A sampled posterior gets "diagnostics", because the first question about a sampler is whether it converged. A deterministic approximation has no chains to trace, so an INLA fit gets "residuals" instead of a message declining to draw the display it was asked for.

Value

Invisibly, the object the underlying plotting call returns: a **ggplot2** object for the displays built with it, the semivariance table for `"variogram"`, for `"spatial_field"` a list holding `values` (the field, one row per row level and one column per column level), `row_var`, `col_var`, `trial`, `title` and `spelling` (the field's term as written), and `NULL` for those drawn on the base graphics device. A fit with no separable field refuses `"spatial_field"` with `spatial_field_requires_ar1_field`. Called for the plot it draws.

See Also

`pp_check.flexybayes()` for the posterior predictive check `type = "pp_check"` draws, and the refusal it raises where a fit has no predictive draws.

---

pp_check.flexybayes *Posterior predictive check for a flexyBayes fit*

Description

A method for the **bayesplot** package's `pp_check()` generic. On a fit from the **brms** engine the call delegates to `brms::pp_check()`: datasets are simulated from the posterior predictive distribution and shown against the observed response. The default display overlays the densities of the replicated datasets on the density of the response (`type = "dens_overlay"`). Every other **bayesplot** check type, and every argument of `brms::pp_check()`, is reachable through `...`.

Usage

```
## S3 method for class 'flexybayes'
pp_check(object, ...)
```

Arguments

`object`	A fitted `flexybayes` object.
`...`	Passed to `brms::pp_check()` on the brms path, `type`, `ndraws`, `group` and the rest. Ignored on the refusal path.

Details

On a fit from the **INLA** engine the method refuses by name. A nested Laplace approximation returns marginal densities, not simulated datasets, so there is nothing to overlay, and a display of observed against fitted values shown under this name would be a different diagnostic wearing the check's title. The refusal names the residual displays the fit does answer: `plot(fit, type = "residuals")` and, on a fit carrying a design index, `plot(fit, type = "variogram")`.

`plot(fit, type = "pp_check")` runs this same code, so the two entry points cannot disagree about what a check is.

Value

On a brms-engine fit, the **ggplot2** object `brms::pp_check()` returns. Otherwise the method does not return: it raises a refusal of class `flexybayes_refusal_pp_check_requires_predictive_draws`.

See Also

`plot.flexybayes()` for the other displays, including the residual diagnostics this refusal points at. `as_draws_df.flexybayes()` for the posterior itself. `summary.flexybayes()`, whose `$converge` slot carries the engine's own diagnostics.

`predict.flexybayes_brms` *Predict from a brms-passthrough flexybayes fit*

Description

Delegates to brms's `posterior_epred()` (response-scale posterior mean) or `posterior_linpred()` (linear-predictor scale) on the live `brmsfit` carried at `$brms`. The parent `predict.flexybayes()` path uses a `$glm$linear.predictors` point estimate that handles only the original-data case. This subclass override accepts `newdata` and returns the posterior-mean prediction (per-row mean over draws), or the full posterior matrix when `summary = FALSE`.

Usage

```
## S3 method for class 'flexybayes_brms'
predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  re_formula = NULL,
  se.fit = FALSE,
  summary = TRUE,
  classify = NULL,
  level = 0.95,
  allow_new_levels = FALSE,
  mean_by = NULL,
  n_draws = 1000L,
  ...
)
```

Arguments

`object`	A `flexybayes_brms` object.
`newdata`	Optional data.frame at which to predict. When omitted, returns the in-sample posterior summary. On a model with an `offset()` brms evaluates the offset on `newdata`, so `newdata` needs every column the offset reads. An absent one refuses with `newdata_offset_missing` instead of being taken as zero.

type	"response" (default. posterior_epred) or "link" (posterior_linpred).
re_formula	Forwarded to brms; NULL (default) includes all random effects, NA excludes them (population-level).
se.fit	Logical: if TRUE, returns a list with fit (posterior mean) and se.fit (posterior SD).
summary	Logical: if TRUE (default), summarise across draws to a numeric vector. If FALSE, return the draws x rows posterior matrix.
classify	The factors to break a predicted-value table down by: a character value ("Variety", "Variety:env") or a one-sided formula (~ Variety). The factors may be fixed or random, and the table follows ASReml's default rules, described under predict.flexybayes_inla(). On this engine the values come from the posterior draws. A level combination the data never held is drawn from its fitted normal distribution. NULL (the default) is the historical behaviour.
level	Credible level for the classify table's interval, or for the predictive interval with allow_new_levels = TRUE, as a proportion. Default 0.95.
allow_new_levels	Logical. TRUE returns the posterior predictive of the rows of newdata, observation noise included, in which a level the fit did not see is drawn from its fitted standard deviation: brms::posterior_predict(allow_new_levels = TRUE, sample_new_levels = "gaussian"). The table matches the INLA method's. See predict.flexybayes_inla() for the structures it is built for and the refusals. Its estimate, sd, lower and upper are the mean, standard deviation and quantiles of the draws. FALSE (the default) leaves the paths above unchanged.
mean_by	With allow_new_levels = TRUE, the columns of newdata whose combinations define the sets of rows to average, as a character vector or a one-sided formula (~ TRIAL). NULL predicts each row.
n_draws	With allow_new_levels = TRUE, the number of posterior draws used, at most the number the fit holds. Default 1000.
...	Forwarded to brms::posterior_epred() / brms::posterior_linpred().

Details

Population-level vs. group-level prediction follows brms's re_formula convention: the default re_formula = NULL includes all random effects. Pass re_formula = NA for population-level predictions only.

Value

One of five shapes, by argument. With classify set, the marginal-means table for those factors. With allow_new_levels = TRUE, the fb_predict_new table described in predict.flexybayes_inla(). Otherwise with summary = FALSE, the draws x rows posterior matrix. With se.fit = TRUE, a list of fit (posterior mean per row) and se.fit (posterior SD per row). And by default a numeric vector of posterior means, one per row of newdata or of the fitted data.

predict.flexybayes_inla *Population-level predictions from a per-row INLA fit*

Description

Fixed-effect (population-level) predictions: the linear predictor is `X beta` with random effects held at their population mean (zero). On the identity link the response- and link-scale predictions coincide. This is the prediction surface **marginaleffects** uses for average predictions and slopes.

Usage

```
## S3 method for class 'flexybayes_inla'
predict(
  object,
  newdata = NULL,
  type = c("response", "link"),
  se.fit = FALSE,
  classify = NULL,
  level = 0.95,
  allow_new_levels = FALSE,
  mean_by = NULL,
  n_draws = 1000L,
  re_formula = NA,
  summary = TRUE,
  ...
)

## S3 method for class 'fb_predict_new'
print(x, digits = 4L, ...)
```

Arguments

`object`	A `flexybayes_inla` fit.
`newdata`	Optional data frame. Defaults to the fit data. Ignored, with a warning, when `classify` is supplied. On a model with an `offset()` the prediction includes the offset evaluated on `newdata`, so `newdata` needs every column the offset reads. An absent one refuses with `newdata_offset_missing` instead of being taken as zero.
`type`	`"response"` or `"link"`.
`se.fit`	Logical: also return delta-method standard errors from the fixed-effect covariance.
`classify`	The factors to break a predicted-value table down by: a character value (`"Variety"`, `"Variety:env"`) or a one-sided formula (`~ Variety`). The factors may be fixed or random. `NULL` (the default) is the historical behaviour.

level	Credible level for the classify table's interval, or for the predictive interval with allow_new_levels = TRUE, as a proportion. Default 0.95.
allow_new_levels	Logical. FALSE (the default) is the population-level prediction above. TRUE returns the posterior predictive of the rows of newdata (see Details), in which every random effect enters: a level the fit saw at its posterior draw, and a level it did not see drawn from its fitted variance component.
mean_by	With allow_new_levels = TRUE, the columns of newdata whose combinations define the sets of rows to average: a character vector or a one-sided formula. mean_by = ~ TRIAL predicts each trial mean, ~ TRIAL + VARIETY + NRATE each cell mean. NULL (the default) predicts each row.
n_draws	With allow_new_levels = TRUE, the number of posterior draws. Default 1000.
re_formula	NA (the default): the prediction is population-level, the fixed part of the model, as re_formula = NA gives on a brms fit. Any other value refuses with method_argument_not_supported. fitted() returns the predictions at the fitted levels of the random effects.
summary	TRUE (the default): the prediction is the posterior mean. FALSE refuses with method_argument_not_supported. fb_as_draws_simple() returns the posterior draws.
...	Ignored, except that an argument of ASReml's predict() given with classify is refused by name.
x	An fb_predict_new table, printed with a banner naming what its numbers are.
digits	Number of significant digits for the printed table.

Details

With gen random, classify = "gen" returns each genotype's fixed part plus its random effect, and on a multi-environment fit with gen:env random it also adds that genotype's gen:env effects averaged over environments. This matches ASReml's default predict(): equal-weight averaging over the fixed-term variables not in classify, a covariate held at its mean, variables that appear only in random terms ignored, and every random term whose variables are all in classify or averaged over included. The intervals integrate over INLA's hyperparameter configurations, so they include the uncertainty in the variance components. A table with no random term is computed by emmeans, with intervals from the Gaussian approximation of the fixed-effect posterior. The arguments of ASReml's predict() that change these rules (present, average, ignore, use, except, only, associate, levels, vcov, sed, parallel, aliased, design.points) are refused with classify_argument_not_supported.

With allow_new_levels = TRUE, the prediction for a set of rows is the mean response over them, observation noise included, as an observed trial or cell mean at a new locality would be. Given one posterior draw it is normal, with mean x'beta averaged over the rows plus the mean of the effects at levels the fit saw, and variance the sum over random terms of sigma^2 * sum_l (n_l / n)^2 over the unseen levels l (n_l of the n rows at level l), plus sigma_field^2 times the mean AR1 x AR1 correlation over pairs of rows in a trial the fit did not see, plus the residual variance over n. The prediction is the mixture of these normals over the draws: estimate and sd are its exact mean and standard deviation, and lower and upper its quantiles. The fixed effects and the seen effects are drawn jointly from INLA's approximation, and the variances from their continuous joint posterior (as fb_variance_draws() draws them). Every random number comes from R's generator, so set.seed() before the call makes the result repeatable.

The new-level draw is built for random intercepts, interactions (TRIAL:REP), diag(f):g and id(trial):ar1(row):ar1(col over an independent residual or dsum(~ units | f), for a Gaussian response on the identity link. Any other structure is refused with predict_new_levels_structure_not_supported, a new level with no fitted variance (a new level of f in diag(f):g, a new residual section, a row or column off the fitted lattice) with predict_new_levels_no_variance, and another family with predict_new_levels_family_not_supported.

On a fit with fixed_alias = "drop", a row that is not a linear combination of the rows the model was fitted to has no estimable prediction: it is returned as NA, and the status attribute (a status element with se.fit = TRUE, a status column with classify) reads "Aliased" there and "Estimable" elsewhere, as in asreml's predict().

Value

With classify, a data frame of class fb_predict_classify. With allow_new_levels = TRUE, a data frame of class fb_predict_new: one row per set, holding the mean_by columns (or row), n_rows, estimate, sd, lower and upper (and status on a fit with fixed_alias = "drop", where a set holding a row the fit does not estimate is NA), with the joint predictive draws, draws by sets, in attr(, "draws"). Otherwise a numeric vector of predictions, or a list fit / se.fit when se.fit = TRUE.

---

print.fb_approx *Print an* fb_approx *specification*

Description

Print an fb_approx specification

Usage

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

Arguments

x	An fb_approx object as returned by fb_approx().
...	Ignored. Present for compatibility with the generic.

Value

Invisibly, x unchanged. Called for the one-line specification it prints.

print.fb_cov *Print an* fb_cov *carrier*

Description

Print an fb_cov carrier

Usage

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

Arguments

x	An fb_cov object as returned by fb_cov().
...	Ignored. Present for compatibility with the generic.

Value

Invisibly, x unchanged. Called for the one-line carrier description it prints.

print.fb_engine *Print an* fb_engine *specification*

Description

Print an fb_engine specification

Usage

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

Arguments

x	An fb_engine object as returned by fb_engine().
...	Ignored. Present for compatibility with the generic.

Value

Invisibly, x unchanged. Called for the one-line engine description it prints.

`print.fb_plan` *Print an* <fb_plan> *— flight-checklist form*

Description

Print an <fb_plan> — flight-checklist form

Usage

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

Arguments

x	An <fb_plan> object as returned by fb_plan().
...	Ignored. Present for compatibility with the generic.

Value

Invisibly, x unchanged. Called for the checklist it prints.

`print.fb_predict_classify` *Print a classify means table*

Description

Prints the table with a banner naming what the numbers are. The estimate is a posterior mean. The banner also states that the table has no standard errors of difference, which ASReml's predict() prints beside its means.

Usage

```
## S3 method for class 'fb_predict_classify'
print(x, digits = 4L, ...)
```

Arguments

x	An fb_predict_classify table, as returned by predict(fit, classify =).
digits	Number of significant digits for the printed table.
...	Ignored, present for compatibility with the generic.

Details

When the table includes random effects, the banner names the random terms included, the variables averaged over and the variables left out, as ASReml's own notes do. Otherwise, on an INLA fit, the banner says the interval comes from the Gaussian approximation of the joint fixed-effect posterior, not from INLA's own marginal densities.

Value

Invisibly, `x` unchanged. Called for the table it prints.

`print.flexybayes` *Print a compact description of a flexyBayes fit*

Description

Reports the call the fit came from, the model the representation describes, the engine that ran it, the design and observation counts, and the main posterior quantities. Use `summary()` for the full coefficient and variance-component tables.

Usage

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

Arguments

`x`	A fitted `flexybayes` object of any backend.
`...`	Ignored, present for compatibility with the generic.

Details

The header is shared with `print.flexybayes_inla()` and the brms print, so the three cannot disagree about what the fit is. Sampler lines print only on an engine that sampled: a nested Laplace approximation has no chains and no warmup, and reporting them over one tells a reader the fit is stochastic when it is not.

The header closes with three lines read from `fit$extras$reproducibility`: the R, flexyBayes and engine versions. The thread settings (INLA's `num.threads`, or the chains, cores and threads per chain on brms, with `OMP_NUM_THREADS`). And the seed supplied, with the seed Stan ran with on brms. `summary()` opens with the same header.

Value

Invisibly, `x` unchanged. Called for the description it prints.

print.flexybayes_aggregated *Print a flexybayes_aggregated object*

Description

Brief one-screen summary of a fit produced by `flexybayes(..., aggregate = "auto"/TRUE)` or `fb_brms(..., aggregate = ...)`. The header names the fit's own family (`aggregated-gaussian`, `aggregated-binomial`, `aggregated-poisson`). Includes the `exactness` field and the cell compression ratio (when N/K >= 2), and the versions, threads and seed the fit ran with, as `print.flexybayes()` prints them.

Usage

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

Arguments

`x`	A <flexybayes_aggregated> object, as returned by a fit run on the aggregated representation.
`...`	Ignored. Present for compatibility with the generic.

Value

Invisibly, `x` unchanged. Called for the one-screen summary it prints.

print.flexybayes_brms *Print method for the brms-passthrough flexybayes subclass*

Description

Opens with the header every engine's print shares, then adds the sampler diagnostics and a brms-specific footer (the live `brmsfit` lives at `$brms`. The GLM shim at `$glm`; `$extras` carries the same diagnostics as the INLA path).

Usage

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

Arguments

`x`	A `flexybayes_brms` object.
`...`	Ignored. Present for compatibility with the generic.

Value

Invisibly, x unchanged. Called for the description it prints.

`print.summary.flexybayes` *Print a flexyBayes summary*

Description

Prints the fixed-effect table, the variance components with the prior each one carried, the engine's own convergence diagnostics, and whatever engine-native panels the fit earns, INLA's hyperparameter and spatial-field tables, brms's per-level residual table.

Usage

```
## S3 method for class 'summary.flexybayes'
print(x, digits = 4L, ...)
```

Arguments

x	A `summary.flexybayes` object, as returned by `summary()` on a fitted model.
digits	Number of significant digits for the printed tables, including the scales inside the `prior` cell.
...	Ignored, present for compatibility with the generic.

Details

Two sentences on the output are deliberate. The banner above the variance components names the estimator, because the table looks like an ASReml variance-component table and is not one: every number in it is a posterior summary. On a fit carrying a latent field, the sentence below the table names the fourth parameter, because a separable AR1 field plus an independent nugget is a different model from ASReml's three-parameter nugget-free residual and the tables look alike.

The numbers inside the `prior` cell are rounded for display only. The stored cell carries the resolved prior's own string at full precision, which is what `prior_summary()` reports and what the two are compared on.

Value

Invisibly, x unchanged. Called for the tables it prints.

prior_summary *Resolved-prior summary for a flexyBayes fit*

Description

Returns the resolved priors used to fit the model, either the user-supplied `fb_prior()` object, the auto-default half-normal on the standard-deviation scale, or the legacy scalar bridge (`prior_fixed_sd + prior_vc_sd`). The return value is an S3 object with a `print()` method. The underlying `fb_prior` (when applicable) is exposed under `$fb_prior` for programmatic access.

Usage

```
prior_summary(object, ...)

## S3 method for class 'flexybayes_inla'
prior_summary(object, ...)

## S3 method for class 'flexybayes_brms'
prior_summary(object, ...)

## Default S3 method:
prior_summary(object, ...)
```

Arguments

`object`	A `flexybayes_inla` or `flexybayes_brms` object.
`...`	Ignored by current methods (reserved for future per-component selection).

Details

The methods are also registered on **rstantools**' `prior_summary()` generic, which **brms** re-exports, so `prior_summary(fit)` returns this summary when brms is attached after flexyBayes.

Value

A `prior_summary_flexybayes` object, a list carrying the components below.

`kind` One of `"fb_prior"`, `"legacy_scalar"`, `"no_prior_recorded"`.

`backend` The backend the fit ran on: `"inla"` or `"brms"`.

`fb_prior` The `fb_prior` object (when `kind == "fb_prior"`).

`default_origin` `"auto"` when the prior was constructed by the half-normal auto-default. `"user"` when supplied via the `prior` argument; `NA` for legacy / no-prior cases.

`default_scale`, `default_basis` Attributes carried by the auto-default prior naming the half-normal scale and its basis (response-scale `sd(y)`, logit-scale constant, log-scale constant). `NULL` when the prior was user-supplied.

`fixed_sd`, `vc_sd` Legacy scalar values (when `kind == "legacy_scalar"`).

`scalars_supplied` Named logical: whether the caller supplied `prior_fixed_sd` and `prior_vc_sd`. An argument left unsupplied leaves the engine's own default in force, which `fixed_sd_engine_default` names.

`legacy_vc_applied` Named character: the density the legacy scalar bridge put on each variance component it reached, empty when `prior_vc_sd` was not supplied.

`not_applied` Named character: a prior this package declared that the engine has no parameter for, a residual prior on a family whose dispersion is a function of the mean.

`declaration_only` Reserved. Always `FALSE` for every current backend.

`inputs` Data frame: every prior specification the fit was given, stated or chosen by the package, with its source and the number of engine slots that used it.

`slots` Data frame: every engine slot, with the prior the package sent, the input it came from, the prior the engine reports after fitting and whether the two agree. A slot carrying the engine's own prior is listed with that source.

`not_consumed` Named character: each input no engine slot used, with the reason code.

Examples

```
## Not run:
# live brms (Stan) fit, needs a working Stan toolchain
data(sleepstudy, package = "lme4")
fit <- fb_brms(Reaction ~ Days + (1 | Subject),
               data = sleepstudy,
               n_samples = 100, warmup = 100, chains = 1,
               verbose = FALSE, mcmc_verbose = FALSE)
prior_summary(fit)

## End(Not run)
```

---

proceed *Advance a deferred-execution object into its fit*

Description

Generic for the inspect-then-fit pattern. The `<flexybayes_review>` method restores the RNG snapshot captured at review-object construction, runs the deferred fit via the backend driver, caches the result in-place, and returns the fit. A second call returns the cached fit.

Usage

```
proceed(x, ...)

## S3 method for class 'flexybayes_review'
proceed(x, ...)
```

Arguments

x	A <flexybayes_review> object holding the deferred fit.
...	Reserved for future deferred-execution classes.

Value

The fit object the originating call would have returned, of class flexybayes. A second call returns the cached fit.

Deprecated

proceed() is deprecated in flexyBayes 0.13.0 and will be removed in a later release. Call flexybayes() with the same arguments and review_code = FALSE, and set seed for a reproducible brms fit. A call warns with a condition of class flexybayes_deprecated.

Examples

```
# The direct call fits the model proceed() would have fitted. The Stan
# compile is the expensive part of running this, so the fit is given a
# small budget.
if (requireNamespace("brms", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))

    fit <- flexybayes(y ~ x + (1 | g), data = d, backend = "brms",
                      chains = 1L, n_samples = 200L, warmup = 100L,
                      seed = 1L)
    class(fit)

}
```

---

ranef *Random-effect predictions from a flexyBayes fit*

Description

The random-effect table, one data frame per grouping factor, in the same columns on every engine: group, level, estimate, std.error, conf.low, conf.high. Identical to coef(object, what = "random"), and present under this name because it is the name the mixed-model ecosystem uses.

Usage

```
ranef(object, ...)

## S3 method for class 'flexybayes'
ranef(object, ...)
```

Arguments

`object`	A fitted `flexybayes` object of any backend.
`...`	Ignored, present for compatibility with the generic.

Details

The method is registered on both this package's own `ranef()` generic and on **nlme**'s, which is the generic **lme4** and **brms** re-export. Attaching any of those masks the local generic, and the registration on nlme's is what keeps a bare `ranef(fit)` dispatching there. In a session where the masking is ambiguous, call `flexyBayes::ranef(fit)`.

Value

A named list of data frames, one per grouping factor. Empty when the model carries no random terms.

See Also

`coef.flexybayes()`, which this delegates to.

Examples

```
if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))
  fit <- flexybayes(y ~ x + (1 | g), data = d, backend = "inla",
                    verbose = FALSE)
  re <- ranef(fit)
  str(re, max.level = 1)
}
```

`recover_data.flexybayes` *emmeans support: recover model data (default method)*

Description

The bare `flexybayes` method: reached by any fit whose engine has no more specific override, which today is the brms backend.

Usage

```
## S3 method for class 'flexybayes'
recover_data(object, ...)

## S3 method for class 'flexybayes_inla'
recover_data(object, ...)
```

Arguments

object	A flexybayes_inla fit.
...	Passed to emmeans::recover_data().

Value

A data frame of predictors for the reference grid.

---

residuals.flexybayes *Extract residuals*

Description

Extract residuals

Usage

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

Arguments

object	A fitted flexybayes object of any backend.
...	Ignored, present for compatibility with the generic.

Value

A numeric vector of response residuals, the observed value minus the posterior-mean fitted value, one per row of the fitted data. A row whose response was missing and carried as latent has no observed value and returns NA.

---

residuals.flexybayes_aggregated
Residuals of an aggregated count fit

Description

The observed response minus fitted.flexybayes_aggregated(), one value per row. A streamed fit that keeps fewer rows than it was fitted to refuses by name (streamed_rows_not_retained).

Usage

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

Arguments

`object`	A `flexybayes_aggregated` fit.
`...`	Ignored, present for compatibility with the generic.

Value

A numeric vector of response residuals, one per row of the data, in row order.

`residuals.flexybayes_inla` *Response residuals from a per-row INLA fit*

Description

Observed response minus the posterior-mean fitted value (on the response scale). The response is recovered from the fit's fixed-effect formula evaluated against the stored data, so a transformed response (`log(y) ~ ...`) residualises on the modelled scale.

Usage

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

Arguments

`object`	A `flexybayes_inla` fit.
`...`	Ignored. Present for compatibility with the generic.

Value

A numeric vector of response residuals, one per observation.

`set_coef.flexybayes` *marginaleffects support: set coefficients (default method)*

Description

marginaleffects support: set coefficients (default method)

Usage

```
## S3 method for class 'flexybayes'
set_coef(model, coefs, ...)

## S3 method for class 'flexybayes_inla'
set_coef(model, coefs, ...)
```

Arguments

model	A flexybayes_inla fit.
coefs	Replacement coefficient vector.
...	Ignored. Present for compatibility with the generic.

Value

The fit with a coefficient override attached.

---

summary.fb_plan *Summarise an <fb_plan> — verbose form*

Description

Summarise an <fb_plan> — verbose form

Usage

```
## S3 method for class 'fb_plan'
summary(object, ...)
```

Arguments

object	An <fb_plan> object as returned by fb_plan().
...	Ignored. Present for compatibility with the generic.

Value

Invisibly, object unchanged. Called for the preflight and representation tables it prints.

---

summary.flexybayes *Summarise a flexyBayes fit*

Description

Prints the fixed-effect posterior summaries, the variance components with the prior each one carried, the engine's own convergence diagnostics, and whatever engine-native panels the fit earns. On a brms fit with a sectioned residual it additionally prints the residual variance and standard deviation for each level of the sectioning factor, computed from the draws, not by transforming a posterior mean.

Usage

```
## S3 method for class 'flexybayes'
summary(object, ...)
```

Arguments

object	A fitted flexybayes object of any backend.
...	Ignored, present for compatibility with the generic.

Details

The returned object is the same eleven-slot summary.flexybayes object on every active engine, so summary(fit)$varcomp answers whichever engine ran the fit. Before 0.9.1 the two engines returned two incomparable objects, INLA's four-slot list and brms's brmssummary, and neither carried a variance-component table at all.

A fit whose model carries an autoregressive latent field gains one further slot, spatial_field, which is the field's own parameters on the correlation and standard-deviation scales. It is the one engine-native slot the object carries, it is present only where the model has a field, and the eleven above it are on every fit.

Value

Invisibly, an object of class c("summary.flexybayes", "list") carrying the slots below. Printed as a side effect.

fixed Data frame: term, estimate, std.error, conf.low, conf.high. Posterior mean, posterior standard deviation and a credible interval, not a sampling-theory estimate and its confidence interval.

varcomp Data frame: component, estimate, std.error, conf.low, conf.high, prior, note. One row per variance component (and per correlation on a fit carrying a latent field), on the standard-deviation scale, under the canonical component name. prior is a projection of prior_summary(). note is "collapsed" when the component's 97.5% quantile sits below 1% of the posterior-median residual standard deviation, a display heuristic for a posterior piled against zero, not a test.

random Named list, one data frame per grouping factor, with the columns group, level, estimate, std.error, conf.low, conf.high. Empty when the model carries no random terms. The same object ranef() returns.

missing Data frame of the unobserved design cells, ASReml's mv factor, with the columns row, estimate, std.error, conf.low, conf.high, followed by any design index variables the fit recorded. row indexes the data the engine was handed. The posterior is the engine's own: INLA's fitted-value marginal at that row, brms's sampled mi() response. Zero rows (never NULL) when every response was observed. The same table coef() returns for what = "missing".

converge List, in the engine's own terms: R-hat, effective sample sizes and divergent transitions where a sampler ran. Mode status, marginal likelihood and the largest Kullback-Leibler divergence where INLA's Laplace approximation did. No R-hat is invented for an approximation that has none.

n_design Rows the engine was handed.

n_observed How many of those carried an observed response.

na_action The missing-response record the fit carries, or NULL on a fit assembled without that layer.

`model` One-line description of the random and residual structure, derived from the model representation, not from any engine's emitted formula.

`engine` `"inla"` or `"brms"`.

`call` The recorded call.

`spatial_field` **Present only on a fit carrying an autoregressive latent field**, and absent (not `NULL`-valued) otherwise. A data frame of `parameter`, `median`, `lower`, `upper`: the field's correlation and standard-deviation parameters and the nugget standard deviation, on the scale a reader thinks in, not INLA's precision scale. The point estimate is the posterior median, because the standard deviations are read off precision marginals and only a monotone summary survives the reciprocal square root exactly. The same table the printed field panel renders, unrounded.

`offset` **Present only on a fit whose fixed formula carries** `offset()` **terms**, and absent otherwise: the offset calls as written, in formula order. Each has a fixed coefficient of one.

See Also

`prior_summary()` for the full resolved prior, and `nobs()` for the design and observed counts on their own.

---

summary.flexybayes_aggregated

Summarise a fit run on the aggregated representation

Description

Returns the same eleven-slot `summary.flexybayes` object every other engine returns, so `summary(fit)$varcomp` answers on an aggregated fit as it does on a per-row one. Aggregation was the default route for an ordinary Gaussian call with a random term, so until 0.9.1 that slot was `NULL` on the commonest fit the package produces: this method returned its own list of the aggregated posterior's raw pieces and carried no variance-component table at all.

Usage

```
## S3 method for class 'flexybayes_aggregated'
summary(object, ...)
```

Arguments

`object`	A `<flexybayes_aggregated>` object, as returned by a fit run on the aggregated representation.
`...`	Ignored. Present for compatibility with the generic.

Details

Two things are aggregated-specific and both appear in the printed header, not as extra slots. The banner names the representation (`aggregated-gaussian`, `aggregated-binomial`, `aggregated-poisson`) where a per-row fit names its engine, and the `aggregation` line gives the row-to-cell compression the fit ran under. The `model` slot carries the same compression in its text, because a reader comparing the fixed-effect tables of an aggregated and a per-row fit otherwise has nothing on the object saying the two were computed over different numbers of rows.

The variance components come from the engine's own posterior. On the INLA route that is the hyperparameter marginal transformed to the standard-deviation scale, the same construction the per-row route uses, and not the reciprocal square root of a tabulated precision mean. A route that recorded posterior means alone reports the means with the three interval columns `NA` and the row's `note` cell reading `"no interval recorded"`.

The raw aggregated pieces this method used to return, `beta_means`, `beta_vcov`, `sigma_means`, `tau_means`, are unchanged at `fit$extras$summary`.

Value

Invisibly, an object of class `c("summary.flexybayes", "list")` with the slots documented at `summary.flexybayes()`. Printed as a side effect.

See Also

`summary.flexybayes()` for the slot-by-slot contract, and `print.flexybayes_aggregated()` for the one-screen fit description.

---

summary.flexybayes_glm

Summary for flexybayes GLM-compatible object

Description

Similar to `summary.glm()` but with Bayesian posterior statistics instead of p-values.

Usage

```
## S3 method for class 'flexybayes_glm'
summary(object, ...)
```

Arguments

`object`	A `flexybayes_glm` object, reached as `fit$glm` on any fitted `flexybayes` object.
`...`	Ignored. Present for compatibility with the generic.

Value

Invisibly, the printed coefficient table as a data.frame, or `NULL` when the object carries no fixed effects.

`tidy.flexybayes` *Tidy a flexyBayes fit into a one-row-per-term data frame*

Description

Turns a flexyBayes fit into a flat, broom-style `data.frame`: one row per model term, with stable, documented columns. The method is registered against the `tidy()` generic (re-exported by `broom`), so `broom::tidy(fit)` and `generics::tidy(fit)` both dispatch here.

Usage

```
## S3 method for class 'flexybayes'
tidy(
  x,
  conf.int = TRUE,
  conf.level = 0.95,
  effects = c("fixed", "random"),
  ...
)
```

Arguments

`x`	A flexyBayes fit: `flexybayes_brms` (brms backend, which inherits this method via the bare `flexybayes` class).
`conf.int`	Logical. Whether to attach credible intervals. Defaults to `TRUE`.
`conf.level`	Numeric in `(0, 1)`. The credible level for the intervals. Defaults to `0.95`.
`effects`	Character. Which effects to return: `"fixed"` for the population-level (fixed) coefficients or `"random"` for the variance-component summary. Defaults to `"fixed"`.
`...`	Currently unused. Present for generic compatibility.

Details

This is the supported accessor for cross-engine summaries. The hub returns backend-specific objects, `flexybayes_brms` (brms) and `flexybayes_inla` (INLA), whose internal layouts differ. Tidying through this generic yields the same columns across both, so a brms-versus-INLA comparison table can be assembled by `rbind`-ing two `tidy()` outputs instead of reaching into each backend's slots by hand.

The credible intervals are posterior quantile-based intervals, not frequentist confidence intervals. They are reported in the `broom`-canonical `conf.low` / `conf.high` columns. The `std.error` column carries the posterior standard deviation of each term, again under the `broom`-canonical (dotless to the user, dotted in the column name) label.

Value

A `data.frame` with one row per term and columns:

term Character. The coefficient or variance-component name.

estimate Numeric. The posterior mean.

std.error Numeric. The posterior standard deviation.

conf.low Numeric. The lower credible bound (present when `conf.int = TRUE`).

conf.high Numeric. The upper credible bound (present when `conf.int = TRUE`).

An empty `data.frame` is returned when the requested effects are absent (for example `effects = "random"` on a fixed-effects-only fit).

See Also

`glance.flexybayes()`, `augment.flexybayes()`, `tidy.flexybayes_inla()`

Examples

```
# Not run: the brms path compiles a Stan program first, and the
# compile is far longer than an example should take. The fragment is
# complete, so it runs as written on a machine with a C++ toolchain.
## Not run:
set.seed(1)
dat <- data.frame(yield = rnorm(40), env = factor(rep(1:4, each = 10)))
fit <- flexybayes(yield ~ env, data = dat, backend = "brms",
                  chains = 1L, n_samples = 200L, warmup = 100L)
tidy(fit)
tidy(fit, effects = "random")

## End(Not run)
```

`tidy.flexybayes_inla` *Tidy a per-row INLA fit into a one-row-per-term data frame*

Description

The INLA backend returns a `flexybayes_inla` object whose internal layout differs from the other engines', it has no `$glm` shim, so it needs its own `tidy()` method even though it shares the `flexybayes` parent class. The fixed-effect summary is read directly off INLA's `summary.fixed` table, whose `mean`, `sd`, and `0.025quant` / `0.975quant` columns map cleanly onto the `broom`-canonical `estimate`, `std.error`, `conf.low`, and `conf.high`.

Usage

```
## S3 method for class 'flexybayes_inla'
tidy(
  x,
  conf.int = TRUE,
  conf.level = 0.95,
  effects = c("fixed", "random"),
  ...
)
```

Arguments

x	A flexybayes_inla fit.
conf.int	Logical. Whether to attach the credible-interval columns. Defaults to TRUE.
conf.level	Numeric in (0, 1). Accepted for generic compatibility. INLA reports the 95% marginal bounds, so a non-0.95 request is noted.
effects	Character. Which effects to return: "fixed" for the population-level coefficients or "random" for the variance-component summary. Defaults to "fixed".
...	Currently unused. Present for generic compatibility.

Details

Because the INLA fixed-effect intervals come from the marginal posteriors INLA has already integrated, the conf.level argument is accepted for generic compatibility but only the 95% bounds INLA reports are returned. A one-off message notes this when a different level is requested rather than silently ignoring it.

The effects argument is the same one tidy.flexybayes() carries, and means the same thing on both: "fixed" returns the population-level coefficients, "random" the variance-component summary. Before 0.9.1 this method had no such argument, so effects = "random" on an INLA fit was absorbed by ... and the FIXED-effect table came back under a call that asked for variance components, a wrong answer with no error. An unrecognised value now stops.

Value

A data.frame with one row per term and the columns term, estimate, std.error, and (when conf.int = TRUE) conf.low / conf.high. The rows are the fixed-effect terms under effects = "fixed" and the variance components under effects = "random".

See Also

tidy.flexybayes()

Examples

```
set.seed(1)
dat <- data.frame(yield = rnorm(40),
                  env = factor(rep(1:4, each = 10)),
```

```
                  block = factor(rep(1:5, times = 8)))
fit <- flexybayes(yield ~ env, random = ~block, data = dat,
                  backend = "inla", verbose = FALSE)
tidy(fit)
tidy(fit, effects = "random")
```

`update.flexybayes` *Re-fit a flexyBayes model with modified arguments*

Description

Rebuilds the original `flexybayes()` call from the argument record the fit carries, applies the overrides supplied in `...`, and re-fits.

Usage

```
## S3 method for class 'flexybayes'
update(object, ...)
```

Arguments

`object`	A `flexybayes` fit carrying a complete argument record in `$extras$call_info`.
`...`	Named arguments overriding the recorded call, for example `n_samples = 2000L` or a replacement `data` frame.

Details

The record has to be complete for this to be safe. An argument the fit did not record would be re-supplied as its default, so a re-fit could quietly drop a relationship matrix, a prior scale, or a missing-response policy, and return a different model under the same name. When any recorded argument is absent, the method refuses and names what is missing instead of re-fitting a model the user did not ask for. A fit produced before its engine recorded the full set therefore refuses, which is the intended behaviour: there is no pre-0.9.1 object to special-case into a silent default.

Value

A new fitted `flexybayes` object of the same engine as the original, unless an override routes it elsewhere.

The prior a re-fit runs under

A re-fit uses the prior the original fit resolved to, taken from the record on the fit itself, not re-derived from the arguments. This matters because the auto-default half-normal on the standard-deviation scale fires only when the call supplies neither a `prior` nor a `prior_vc_sd`, and a re-fit that re-issues the recorded `prior_vc_sd` supplies one by construction. Before 0.9.1 the auto-default therefore

never fired on a re-fit: an identity `update()` fell through to the engine's own hyperprior and could report a variance component half the size of the one it re-fitted, with nothing on the object to say so.

Precedence, in order:

- A `prior` in `...` is used as given. The recorded prior is not re-issued alongside it.
- A `prior_vc_sd` in `...` is used as given, and the recorded prior is *not* re-issued, the two forms describe the same variance-component scale, and passing both would let the recorded prior silently override the scalar the caller just wrote.
- Otherwise the fit's own resolved prior decides, and *how* it decides depends on where that prior came from, see below.
- A fit carrying the legacy scalar bridge, or no prior record at all, re-fits on the recorded scalars exactly as it did before.

A policy re-fires. A bespoke prior carries. The two things `$extras$fb_terms$priors` can hold are not the same kind of thing, and a re-fit treats them differently.

- **The auto-default** half-normal is the output of a policy: one half-normal per variance component the model has, with the scale read off the data. A re-fit passes neither `prior` nor either scalar, so the policy runs again over the *updated* model and data. On an identity `update()` the same data rebuild the same scale, so nothing moves. On `update(fit, random = ~ g + h)` the added term gets the same half-normal as its siblings.
- **A user-supplied** `fb_prior()` is a statement about the terms it names, and is carried verbatim. A term the user did not name (including one an override has just added) keeps following the engine's own hyperprior, exactly as it did on the first fit. `prior_fixed_sd` and `prior_vc_sd` are re-issued alongside it, so a route that reads them for the terms the prior object does not name keeps the values the original ran under.

The asymmetry is deliberate. Re-applying a policy's old output to a new model leaves a mixed state, the original terms priored, the added one falling to the engine while its siblings keep the default, which is not what either the policy or the user asked for. Re-deciding a *user's* explicit prior, on the other hand, would put words in their mouth. Read `prior_summary()` on the re-fit to see what it ran under, and `summary.flexybayes()`'s `varcomp` table to see it per component.

The engine and representation a re-fit runs on

`backend` and `aggregate` are re-issued from the record alongside everything else. Before 0.9.1 neither was recorded, so a re-fit took the formal defaults: an identity `update()` of a Stan fit came back as an aggregated INLA fit, a different inference engine and a different representation, under the same name and with nothing said.

What is recorded is the value the *call* asked for, not the engine the call resolved to. A fit made under `backend = "auto"` records `"auto"`, and a re-fit routes again from there. A fit that named `"brms"` records `"brms"` and comes back on Stan. The distinction is deliberate: the recorded value is the user's policy, and a re-fit whose model has changed (`update(fit, random = ~ g + h)`) has to be free to route the changed model, not be pinned to the engine that suited the old one. `aggregate` behaves the same way. A recorded `"auto"` lets the aggregation gate re-decide, a recorded `FALSE` keeps the re-fit per-row.

An override in `...` wins, as for every other recorded argument, so `update(fit, backend = "brms")` moves the re-fit to Stan.

What `update()` reproduces is the model and the policy behind it, not the display settings of the session that first ran it. `verbose` is recorded on the fit for completeness and is deliberately *not* re-issued. A re-fit follows the current call's display default, and `update(fit, verbose = FALSE)` quietens one on the spot. The distinction is in what the argument can get wrong. A silently switched engine is a different answer under the same name. A banner printed where the first fit printed none is a banner.

See Also

`prior_summary()` for the resolved prior a fit carries, and `summary.flexybayes()`, whose `varcomp` table names the prior each component ran under.

`validate_approximation` *Validate an approximate model fit against its bias bound*

Description

`validate_approximation()` reports how much of a fitted model's structure was lost to its approximation scheme, measured against the scheme's declared pass threshold. It is the user-facing entry to the per-scheme validation procedure registered for every approximate route, the contract surfaces the realised error number while the user keeps the accept / re-fit judgement.

Usage

```
validate_approximation(fit, ...)
```

Arguments

`fit`	A fitted `flexybayes` object.
`...`	Passed to the per-scheme validation procedure (e.g. `threshold` for `low_rank_smooth`).

Details

Dispatch is on the fit's registered approximation scheme. At present the only registered scheme is `low_rank_smooth` (the rank-K principal-component truncation of an `s()` smooth basis): for such a fit, the procedure reports the realised Frobenius capture $\sum_{i \le K} d_i^2 / \sum_i d_i^2$ of each truncated smooth against the default pass threshold of `0.99`, where d_i are the singular values of the full smooth basis.

A fit carrying no recognised approximation (an exact fit) is refused instead of returned as a vacuous pass.

Value

An `<fb_approximation_validation>` object: the scheme, the overall pass / fail verdict, the pass threshold, one result row per approximated smooth (realised capture, bias bound, per-smooth pass flag), and the registry's fallback hint.

See Also

flexybayes() for fitting. The approximation registry records each scheme's bias bound and fallback.

Examples

```
# An exact fit (no registered approximation) refuses instead of
# returning a vacuous pass. \donttest{} + an INLA guard: this needs
# a live fit and does not fire on a machine without INLA installed.

if (requireNamespace("INLA", quietly = TRUE)) {
  set.seed(1)
  d <- data.frame(y = rnorm(60), x = rnorm(60), g = factor(rep(1:6, 10)))
  fit <- flexybayes(y ~ x + (1 | g), data = d, backend = "inla",
                    verbose = FALSE)
  tryCatch(validate_approximation(fit), error = function(e) {
    message(conditionMessage(e)) # "carries no recognised approximation"
  })
}


# The low_rank_smooth verdict shape, on a plan-only object carrying
# the scheme (no active engine reaches low_rank_smooth on the
# current backends: it was built for a since-withdrawn engine and
# now refuses at dispatch; see R/emit_smooth_low_rank.R). This part
# needs no engine and always runs.
demo_fit <- structure(
  list(
    exactness = "approximate_low_rank_smooth",
    extras = list(parse_info = list(approx = list(
      x = list(
        scheme = "low_rank_smooth", rank = 4L, k = 9L,
        frobenius_capture = 0.995,
        V_K = matrix(0, 9L, 4L), singular_values = rep(1, 9L)
      )
    )))
  ),
  class = c("flexybayes", "list")
)
v <- validate_approximation(demo_fit)
print(v)
v$pass
```

---

vcov.flexybayes *Extract variance-covariance matrix of fixed effects*

Description

Extract variance-covariance matrix of fixed effects

Usage

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

Arguments

`object`	A fitted `flexybayes` object of any backend.
`...`	Ignored, present for compatibility with the generic, except that `n_samples`, which the INLA method takes, is refused with `method_argument_not_supported`: the matrix comes from every stored posterior draw.

Value

The posterior covariance matrix of the fixed effects, square with one row and column per coefficient and dimnames taken from `coef()`. This is a posterior covariance, not a sampling-theory variance estimate, though the downstream packages that consume it treat it as one.

`vcov.flexybayes_inla` *Posterior covariance of a per-row INLA fit's fixed effects*

Description

The joint posterior covariance of the fixed effects, the matrix that contrast and marginal-mean standard errors need. It is computed exactly from the Gaussian approximations INLA stores for the fit, averaged over INLA's hyperparameter configurations, so it involves no random draws: every call on the same fit returns the same matrix and the session's random number stream is left untouched. It is the covariance of the draws `INLA::inla.posterior.sample()` would give, without their Monte Carlo error. The standard deviations on its diagonal agree closely with `summary.fixed$sd`, which comes from INLA's marginal densities.

Usage

```
## S3 method for class 'flexybayes_inla'
vcov(object, n_samples = 2000L, ...)
```

Arguments

`object`	A `flexybayes_inla` fit.
`n_samples`	Not used, as the covariance is exact. Accepted so that calls written for earlier versions, which estimated the matrix from `n_samples` posterior draws, still run.
`...`	Ignored. Present for compatibility with the generic.

Value

Posterior covariance matrix of the fixed effects, with the coefficient names of `coef()` as dimnames.

`vpredict` *Posterior functionals of the variance components*

Description

The exact posterior of any function of the fitted variance components: a heritability, a genetic correlation, a variance ratio, a repeatability. Each functional is written as a two-sided formula whose left side names the result and whose right side is an expression over the component names that `summary()` reports.

Usage

```
vpredict(object, ...)

## S3 method for class 'flexybayes'
vpredict(object, ..., n_draws = 1000L, prob = 0.95, draws = FALSE)
```

Arguments

`object`	A fitted `flexybayes` model.
`...`	One or more two-sided formulas, `name ~ expression`.
`n_draws`	Number of posterior draws to evaluate over.
`prob`	Width of the credible interval.
`draws`	When `TRUE`, return the matrix of functional draws (one column per functional) instead of the summary.

Details

Components are on the **standard-deviation** scale, as everywhere else in this package, so a variance ratio squares them. Writing a name the fit does not carry is refused, and the refusal lists the names it does.

The interval is a quantile interval of the functional's own posterior. Where the delta method would put a heritability's interval below zero, this one cannot: every draw of `sd_g^2 / (sd_g^2 + sigma^2)` lies in [0, 1], so the interval does too.

The method is also registered on **sommer**'s `vpredict()` generic, so `vpredict(fit, ...)` reaches it when sommer is attached after flexyBayes.

Value

A `data.frame` with one row per functional and the columns `functional`, `estimate` (posterior mean), `median`, `std.error` (posterior standard deviation), `conf.low` and `conf.high`. With `draws = TRUE`, a numeric matrix of draws instead.

Examples

```
## Not run:
fit <- flexybayes(yield ~ 1, random = ~ gen, data = trial)
vpredict(fit, h2 ~ sd_gen^2 / (sd_gen^2 + sigma^2))

## End(Not run)
```

Index

∗ **datasets**
met_example, 74
∗ **flexyBayes engine pins**
fb_brms, 20
fb_inla, 29
∗ **flexyBayes ingest adapters**
fb_from_asreml, 25
fb_from_brms, 27

anova.flexybayes, 4
as.data.frame.fb_plan, 5
as.matrix.fb_marginal_cov
*(*fb_marginal_cov*)*, 33
as_draws.flexybayes
*(*as_draws_df.flexybayes*)*, 5
as_draws_df.flexybayes, 5
as_draws_df.flexybayes(), *73*, *81*
as_draws_matrix.flexybayes
*(*as_draws_df.flexybayes*)*, 5
augment.flexybayes, 7
augment.flexybayes(), *102*
augment.flexybayes_inla
*(*augment.flexybayes*)*, 7

backend_decision, 8
backend_decision(), *42*, *56*
brms::brm(), *40, 41*
brms::loo(), *73*
brms::posterior_epred(), *78*
brms::pp_check(), *80, 81*
brms::set_prior(), *40*
brms::stancode(), *10*, *56*
brms::stanvar(), *40*

canonical_names, 9
canonical_names(), *7*
cat_code, 10
cat_code(), *56*
coef(), *79*, *98*, *108*
coef.flexybayes, 11
coef.flexybayes(), *13*, *94*
coef.flexybayes_inla, 12
coef.flexybayes_ordinal
*(*coef.flexybayes*)*, 11
confint(), *79*
confint.flexybayes, 13
confint.flexybayes(), *16*
confint.flexybayes_brms, 14
confint.flexybayes_brms(), *14*
confint.flexybayes_glm, 15
confint.flexybayes_inla, 15
confint.flexybayes_inla(), *14*

data.table::fread(), *61*

emm_basis.flexybayes, 16
emm_basis.flexybayes_inla
*(*emm_basis.flexybayes*)*, 16
emmeans::emmeans(), *12*
emmeans::recover_data(), *95*

family.flexybayes, 17
family.flexybayes_inla, 17
fb *(*flexybayes*)*, 48
fb(), *20*, *25–28*
fb_approx, 18
fb_approx(), *23*, *25*, *85*
fb_as_draws_simple(), *6, 7*, *84*
fb_backend_status, 19
fb_backend_status(), *56*
fb_brms, 20
fb_brms(), *30*
fb_complete_grid, 21
fb_cov, 22
fb_cov(), *53*, *86*
fb_engine, 24
fb_engine(), *18*, *23*, *86*
fb_from_asreml, 25
fb_from_asreml(), *28*, *30*
fb_from_brms, 27

fb_from_brms(), *20*, *26*, *30*
fb_gblup_cv, 28
fb_inla, 29
fb_inla(), *20*
fb_isolated_lapply, 31
fb_isolated_lapply(), *60*
fb_kfold, 32
fb_kfold(), *52*
fb_marginal_cov, 33
fb_plan, 35
fb_plan(), *26*, *28*, *42*, *44*, *58*, *64*
fb_prior, 37
fb_prior(), *23*, *26, 27*, *41*, *55*, *62, 63*
fb_refit, 40
fb_refit(), *32*
fb_refusals, 42
fb_refusals(), *38*, *43–45*, *73*
fb_refuse, 43
fb_refuse(), *44, 45*
fb_register_refusals, 44
fb_register_refusals(), *42, 43*
fb_variance_draws, 45
fb_variance_draws(), *35*, *84*
fitted(), *71*, *84*
fitted.flexybayes, 46
fitted.flexybayes_aggregated, 47
fitted.flexybayes_aggregated(), *95*
fitted.flexybayes_inla, 48
fitted.flexybayes_ordinal
*(*ordinal_probabilities*)*, 77
flexybayes, 48
flexybayes(), *6*, *10*, *19, 20*, *22*, *25–30*, *40*, *46*, *61–63*, *104*, *107*
flexybayes_stream, 61
flexybayes_stream(), *58*
formula.flexybayes, 63
formula.flexybayes_inla, 64
formula.flexybayes_inla(), *76*

genomic_summary, 64
get_coef.flexybayes, 65
get_coef.flexybayes_inla
*(*get_coef.flexybayes*)*, 65
get_data.flexybayes, 66
get_data.flexybayes_inla
*(*get_data.flexybayes*)*, 66
get_predict.flexybayes, 67
get_predict.flexybayes_inla
*(*get_predict.flexybayes*)*, 67
get_vcov.flexybayes, 67
get_vcov.flexybayes_inla
*(*get_vcov.flexybayes*)*, 67
glance(), *72*
glance.flexybayes, 68
glance.flexybayes(), *102*
glance.flexybayes_inla
*(*glance.flexybayes*)*, 68

insight::get_data(), *66*
is_fb_approx, 69
is_fb_cov, 70
is_fb_engine, 70

lapply(), *31*
logLik.flexybayes, 71
logLik.flexybayes(), *4*
logLik.flexybayes_brms, 72
logLik.flexybayes_brms(), *71*
logLik.flexybayes_inla, 72
logLik.flexybayes_inla(), *71*
loo.flexybayes, 73

marginaleffects::predictions(), *12*
met_example, 74
model.matrix.flexybayes, 75

nobs(), *12*, *99*
nobs.flexybayes, 76

ordinal_probabilities, 77

plot.flexybayes, 78
plot.flexybayes(), *81*
plot.flexybayes_aggregated
*(*plot.flexybayes*)*, 78
plot.flexybayes_brms *(*plot.flexybayes*)*, 78
plot.flexybayes_glm *(*plot.flexybayes*)*, 78
plot.flexybayes_inla *(*plot.flexybayes*)*, 78
pp_check.flexybayes, 80
pp_check.flexybayes(), *78–80*
predict(), *12*
predict.flexybayes_brms, 81
predict.flexybayes_inla, 83
predict.flexybayes_inla(), *59*, *82*
predict.flexybayes_ordinal
*(*ordinal_probabilities*)*, 77

print.fb_approx, 85
print.fb_cov, 86
print.fb_engine, 86
print.fb_plan, 87
print.fb_predict_classify, 87
print.fb_predict_new
(predict.flexybayes_inla), 83
print.flexybayes, 88
print.flexybayes(), *89*
print.flexybayes_aggregated, 89
print.flexybayes_aggregated(), *100*
print.flexybayes_brms, 89
print.flexybayes_inla(), *88*
print.summary.flexybayes, 90
prior_summary, 91
prior_summary(), *7*, *38, 39*, *55*, *62*, *90*, *98, 99*, *105, 106*
proceed, 92
proceed(), *56*

ranef, 93
ranef(), *11–13*, *65*, *98*
recover_data.flexybayes, 94
recover_data.flexybayes_inla
(recover_data.flexybayes), 94
residuals.flexybayes, 95
residuals.flexybayes_aggregated, 95
residuals.flexybayes_inla, 96

set_coef.flexybayes, 96
set_coef.flexybayes_inla
(set_coef.flexybayes), 96
summary(), *57*, *72*, *88*, *90*, *109*
summary.fb_plan, 97
summary.flexybayes, 97
summary.flexybayes(), *7*, *12*, *73*, *77*, *81*, *100*, *105, 106*
summary.flexybayes_aggregated, 99
summary.flexybayes_glm, 100

tidy.flexybayes, 101
tidy.flexybayes(), *7*, *68*, *103*
tidy.flexybayes_inla, 102
tidy.flexybayes_inla(), *102*

update(), *40, 41*
update.flexybayes, 104
update.flexybayes(), *41*

validate_approximation, 106
validate_approximation(), *18*, *23*
vcov(), *12*
vcov.flexybayes, 107
vcov.flexybayes_inla, 108
vpredict, 109
vpredict(), *65*