Package 'hbsaems'

May 25, 2026

Type Package

Title Hierarchical Bayesian Area-Level Small Area Estimation Models

Version 1.0.0

Date 2026-05-23

Maintainer Achmad Syahrul Choir <madsyair@stis.ac.id>

Description Fits area-level Hierarchical Bayesian Small Area Estimation models. The methodological foundation follows the standard area-level Small Area Estimation literature, primarily Rao and Molina (2015, ISBN: 9781118735787) <doi:10.1002/9781118735855>, while computational implementation is adapted to the parameterisation and prior-specification conventions of the 'brms' package <doi:10.18637/jss.v080.i01>, which targets the Stan back-end. Supports a principled Bayesian workflow <doi:10.48550/arXiv.2011.01808>, with prior predictive checks, convergence diagnostics, model comparison, spatial random effects, custom distributions, missing-data handling, and a bilingual 'shiny' application for non-programmer analysts.

License GPL (>= 3)

URL https://madsyair.github.io/hbsaems/, https://github.com/madsyair/hbsaems

BugReports https://github.com/madsyair/hbsaems/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.0.0)

Imports brms (>= 2.18.0), coda (>= 0.19-4), ggplot2 (>= 3.3.5), mice (>= 3.14.0), rstantools (>= 2.4.0), stats, utils

Suggests bayesplot, bridgesampling, DT, energy, knitr (>= 1.40), loo, mgcv, minerva, mockery, posterior, priorsense, readxl, rmarkdown (>= 2.16), rstan, sf, shiny, shinydashboard (>= 0.7.2), shinyWidgets, shinytest2, spdep, spelling (>= 2.2), testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Config/roxygen2/version 8.0.0

Author Achmad Syahrul Choir [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7088-0646>),
Saniyyah Sri Nurhayati [aut],
Sofi Zamzanah [aut],
Arsyka Laila Oktalia Siregar [aut]

Repository CRAN

Date/Publication 2026-05-25 02:20:02 UTC

Contents

`adjacency_matrix_car` *Province-level Adjacency Matrix*

Description

A small example adjacency matrix used for fitting Conditional Autoregressive (CAR) random effects on the **province** level. Pairs with `data_fhnorm` and `data_betalogitnorm`, whose `province` column has matching labels.

Usage

```
adjacency_matrix_car
```

Format

A binary symmetric 5×5 matrix with row- and column-names `province_01` .. `province_05`; entries are `1` for adjacent province pairs and `0` otherwise.

Source

Simulated.

`adjacency_matrix_car_regency`
Regency-level Adjacency Matrix (Coarse Spatial Cluster)

Description

A small example adjacency matrix used for fitting Conditional Autoregressive (CAR) random effects on the **regency** (coarse spatial-cluster, kabupaten) level. Pairs with `data_binlogitnorm` and `data_lnln`, whose `regency` column has matching labels.

Usage

```
adjacency_matrix_car_regency
```

Format

A binary symmetric 5×5 matrix with row- and column-names `regency_01` .. `regency_05`; entries are `1` for adjacent regency pairs and `0` otherwise.

Note

The naming `regency_01..05` (two-digit suffix) is reserved in this package for the COARSE 5-level cluster used by `data_binlogitnorm` and `data_lnln`. The 100-level FINE regency column in `data_fhnorm` and `data_betalogitnorm` uses three-digit suffixes (`regency_001..100`) and pairs with the larger `spatial_weight_sar` matrix. See `vignette("hbsaems-spatial")` for the naming convention.

Source

Simulated.

brms_custom_loglogistic *Loglogistic as a Custom Distribution Family for brms*

Description

Returns the `custom_family` + `stanvars` pair required to fit a **brms** model with a Loglogistic response distribution. The Stan code is loaded from `inst/stan/loglogistic.stan` via `build_brms_custom_family`; post-processing functions (`log_lik`, `posterior_predict`, `posterior_epred`) are wired in automatically.

Usage

```
brms_custom_loglogistic()
```

Details

Two parameters:

`mu` Scale (`mu > 0`, log link).

`beta` Shape (`beta > 0`, log link).

Value

A list with elements `custom_family` and `stanvars_family` ready for use with `brms::brm()`. The returned `custom_family` object has `log_lik`, `posterior_predict`, and `posterior_epred` registered so that `brms::loo()`, `brms::posterior_predict()`, and `brms::posterior_epred()` work without further setup.

See Also

`dloglogistic`, `build_brms_custom_family`, `register_hbsae_brms_custom`.

Examples

```
library(hbsaems)
library(brms)
ll <- brms_custom_loglogistic()
# fit <- brm(y ~ x, data = d,
#            family   = ll$custom_family,
#            stanvars = ll$stanvars_family)
# loo(fit)                    # uses log_lik_loglogistic
# posterior_predict(fit)      # uses posterior_predict_loglogistic
# posterior_epred(fit)        # uses posterior_epred_loglogistic
```

`brms_custom_shifted_loglogistic` *Shifted Loglogistic as a Custom Distribution Family for brms*

Description

Returns the `custom_family` + `stanvars` pair required to fit a brms model with a Shifted Loglogistic response distribution. The Stan code is loaded from `inst/stan/shifted_loglogistic.stan` via `build_brms_custom_family`; post-processing functions (`log_lik`, `posterior_predict`, `posterior_epred`) are wired in automatically.

Usage

```
brms_custom_shifted_loglogistic()
```

Details

Three parameters:

`mu` Location parameter (identity link).

`sigma` Scale parameter (`sigma > 0`, log link).

`xi` Shape parameter (identity link).

Value

A list with elements `custom_family` and `stanvars_family` ready for use with `brms::brm()`.

See Also

`dshifted_loglogistic`, `build_brms_custom_family`, `register_hbsae_brms_custom`.

Examples

```
library(hbsaems)
library(brms)
sll <- brms_custom_shifted_loglogistic()
# fit <- brm(y ~ x, data = d,
#            family   = sll$custom_family,
#            stanvars = sll$stanvars_family)
```

build_brms_custom_family *Build a brms Custom Family + Stanvars Pair from a Single Spec*

Description

Convenience function that combines `custom_family` and `stanvar` into the standard `list(custom_family,stanvars_famil` pair returned by every `brms_custom_*` helper in **hbsaems**. The Stan code is read directly from `inst/stan/<name>.stan` – the user does not need to maintain it as an R string.

Usage

```
build_brms_custom_family(
  name,
  dpars,
  links,
  lb = NA,
  ub = NA,
  type = c("real", "int"),
  loop = FALSE,
  log_lik = NULL,
  posterior_predict = NULL,
  posterior_epred = NULL,
  use_stan_native = FALSE
)
```

Arguments

`name`	Character. Family name; must match a `<name>.stan` file under `inst/stan/`.
`dpars`	Character vector of distributional parameter names. The first MUST be `"mu"` (brms convention).
`links`	Character vector of link functions, same length as `dpars`. Common values: `"identity"`, `"log"`, `"logit"`.
`lb, ub`	Numeric vectors of lower / upper bounds; `NA` for none.
`type`	`"real"` (continuous) or `"int"` (discrete).
`loop`	Logical. `FALSE` (default) selects the vectorised brms convention (Stan signatures take vectors of `y` and `mu`); `TRUE` selects scalar Stan signatures. The default matches the neodistr convention. Whichever you choose, the corresponding `.stan` file under `inst/stan/` must use the same convention.
`log_lik`	Optional function for computing observation-level log-likelihoods (used by `loo()`, `waic()`). Signature must be `function(i, prep)`. See the brms vignette "Define Custom Response Distributions" for details.
`posterior_predict`	Optional function for drawing from the posterior predictive distribution (used by `pp_check()`, `posterior_predict()`). Signature `function(i, prep, ...)`.

`posterior_epred`
: Optional function returning the conditional expectation $E[Y \mid X]$ (used by `fitted()`, `conditional_effects()`). Signature `function(prep)`.

`use_stan_native`
: Logical. If `TRUE`, no `stanvars` block is attached – the `<name>_lpdf` function is assumed to exist as a Stan **built-in** (e.g.\ `loglogistic_lpdf` in Stan $\geq$ 2.29). When `FALSE` (the default), the function definition is loaded from `inst/stan/<name>.stan`.

Details

Stan code is built once per call; if your code is reused many times, wrap the returned object in a function and call it lazily.

Value

A list with two elements:

`custom_family` A `brms::customfamily` object.

`stanvars_family` A `brms::stanvars` object with the Stan code in the `functions` block, or `NULL` when `use_stan_native = TRUE`.

See Also

`read_stan_function`, `register_hbsae_brms_custom`.

Examples

```
library(hbsaems)
library(brms)

# Build the loglogistic family.  Stan code is loaded from
# inst/stan/hbsae_loglogistic.stan — the `hbsae_` prefix avoids
# collision with Stan's BUILT-IN `loglogistic_lpdf` (Stan >= 2.29).
ll <- build_brms_custom_family(
  name             = "hbsae_loglogistic",
  dpars            = c("mu", "beta"),
  links            = c("log", "log"),
  lb               = c(0,    0),
  ub               = c(NA,   NA),
  type             = "real"
)
class(ll$custom_family)
```

build_spatial_weight *build_spatial_weight: Construct M for CAR / SAR models*

Description

Constructs a spatial weight / adjacency matrix M suitable for use as the `M` argument of `hbm`. The function accepts either a path to a shapefile (`.shp`) or an `sf` / `Spatial*` object already in memory, and returns a square numeric matrix that is automatically validated against the theoretical requirements of the target model class via `check_spatial_weight`.

Usage

```
build_spatial_weight(
  shp,
  for_model = NULL,
  type = NULL,
  style = NULL,
  k = 4L,
  threshold = NULL,
  id_col = NULL,
  longlat = FALSE,
  validate = TRUE
)
```

Arguments

`shp`	Either a character path to a `.shp` file, an `sf` object, or a `Spatial*` (**sp**) object.
`for_model`	Optional convenience argument: `"car"` or `"sar"`. When supplied, sets sensible defaults for `type` and `style` as documented above. When the user also supplies `type` or `style` explicitly, those take precedence.
`type`	Character. Neighbour definition: `"queen"` Polygons sharing at least one vertex. Symmetric. `"rook"` Polygons sharing an edge. Symmetric. `"knn"` `k` nearest centroids. Asymmetric in general; auto-symmetrised when `style = "B"`. `"distance"` Centroids within `threshold`. Symmetric.
`style`	Character. `"B"` (binary, suitable for CAR) or `"W"` (row-standardised, suitable for SAR).
`k`	Integer. Number of nearest neighbours when `type = "knn"` (default 4).
`threshold`	Numeric. Distance threshold (in CRS units) when `type = "distance"`. When `NULL`, the maximum first-nearest-neighbour distance is used.
`id_col`	Optional character. Column name in the spatial object whose values become the matrix `dimnames`.
`longlat`	Logical. Whether coordinates are in longitude/latitude.
`validate`	Logical. Whether to run `check_spatial_weight` after construction (default `TRUE`). Set to `FALSE` to skip diagnostics.

Details

Build a Spatial Weight or Adjacency Matrix from Shapefile

Choosing type and style for your SAE model:

The combination of `type` (neighbour definition) and `style` (matrix coding) determines whether the resulting matrix is theoretically appropriate for a CAR or SAR model. The recommended combinations are:

Model	type	style	Reference
CAR / ICAR / BYM2	queen or rook	B	Besag (1974, 1991), Riebler et al. (2016)
SAR (lag/error)	knn or distance	W	Anselin (1988), Whittle (1954)
CAR (irregular)	knn or distance	B	kNN auto-symmetrised when `style="B"`

Use the convenience wrapper `for_model = "car"` or `for_model = "sar"` to set both `type` and `style` automatically from the recommendation above. When `type` or `style` is also supplied, the explicit argument wins (the helper only fills in the missing one).

Mathematical contracts:

CAR M must be **binary**, **symmetric**, and have **zero diagonal**. The full distribution is $u \sim \mathcal{N}\big(0, \sigma^2(D - \rho W)^{-1}\big)$ with $D = \mathrm{diag}(W\mathbf{1})$.

SAR M should be **row-standardised** so that ρ lies in $(-1, 1)$. Symmetry is not required.

All matrices are post-checked with `check_spatial_weight` and offending matrices are flagged.

Value

A square numeric matrix with row/column names taken from `id_col` (if supplied) or sequential integers, plus the following attributes: `"hbsaems_type"`, `"hbsaems_style"`, `"hbsaems_for_model"`, `"hbsaems_check"` (the result of `check_spatial_weight`).

Reproducibility note

KNN graphs depend on the order in which ties are broken, so when several centroids are exactly equidistant the resulting matrix may depend on feature ordering.

References

Anselin, L. (1988). *Spatial Econometrics: Methods and Models*. Kluwer Academic Publishers.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems (with discussion). *JRSS B* 36(2), 192–236.

Bivand, R. S., Pebesma, E., & Gomez-Rubio, V. (2013). *Applied Spatial Data Analysis with R* (2nd ed.). Springer.

See Also

`check_spatial_weight`, `hbm`, `adjacency_matrix_car`, `spatial_weight_sar`

Examples

```
# Inspect a pre-built CAR adjacency matrix shipped with the package
data("adjacency_matrix_car")
dim(adjacency_matrix_car)
check_spatial_weight(adjacency_matrix_car, spatial_model = "car",
                     verbose = FALSE)$compatible


# Build a CAR matrix from an sf object (requires sf + spdep)
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("spdep", quietly = TRUE)) {
  library(sf)
  # A small 2x2 grid of polygons
  g <- st_sf(
    id = LETTERS[1:4],
    geometry = st_sfc(
      st_polygon(list(rbind(c(0,0), c(1,0), c(1,1), c(0,1), c(0,0)))),
      st_polygon(list(rbind(c(1,0), c(2,0), c(2,1), c(1,1), c(1,0)))),
      st_polygon(list(rbind(c(0,1), c(1,1), c(1,2), c(0,2), c(0,1)))),
      st_polygon(list(rbind(c(1,1), c(2,1), c(2,2), c(1,2), c(1,1))))
    )
  )
  M_car <- build_spatial_weight(g, for_model = "car")
  M_sar <- build_spatial_weight(g, for_model = "sar", k = 2L)
}
```

---

check_data *Inspect Data Before Fitting an HBSAE Model*

Description

Performs three independent checks on the supplied dataset and returns a structured `hbsaems_data_check` object that summarises the results. This function is intended to be called **before** `hbm` or any of the distribution-specific wrappers.

Usage

```
check_data(
  data,
  response,
  auxiliary = NULL,
  area_var = NULL,
  spatial_var = NULL,
  M = NULL,
  trials = NULL,
  n_var = NULL,
  predictors = NULL,
```

```
  group = NULL,
  sre = NULL
)
```

Arguments

data	A data.frame.
response	Character. Name of the response variable in data.
auxiliary	Character vector. Names of the auxiliary variables (the SAE 'X' covariates).
area_var	Optional character. Name of the area (random-effect grouping) variable.
spatial_var	Optional character. Name of the spatial-grouping variable (column in data that indexes rows of the spatial weight matrix M).
M	Optional spatial weight matrix to dimension-check against data.
trials	Optional character. Name of the trials variable (binomial models).
n_var	Optional character. Name of the sample-size variable (beta / lognormal direct-estimator models).
predictors	**Deprecated.** Use auxiliary instead.
group	**Deprecated.** Use area_var instead.
sre	**Deprecated.** Use spatial_var instead.

Details

1. Variable presence: Verifies that response, every name in auxiliary, and the optional area_var / spatial_var / trials / n_var columns exist in data. Missing variables are reported in $issues.

2. Missing-value pattern: The pattern is one of:

"none" All listed columns are complete.

"y_only" Only the response has NAs.

"x_only" Only the auxiliary variables have NAs.

"both" Both Y and X have NAs.

Based on the pattern, a strategy is recommended:

"y_only" **First, check whether the NA-Y rows are non-sample (out-of-sample) areas** – domains for which a prediction is desired but no direct estimate exists. If yes, do *not* delete them; fit on the complete-Y subset and pass the NA-Y rows to sae_predict via the newdata argument. If they are merely missing observations within sampled areas, use handle_missing = "deleted".

"x_only" handle_missing = "multiple" – multiple imputation via **mice**.

"both" **(continuous outcome)** handle_missing = "model" – joint Bayesian imputation via brms::mi().

"both" **(discrete outcome, binomial)** handle_missing = "multiple".

3. Dimension check: When M is supplied, verifies that it is square and that nrow(M) matches the number of *distinct* levels in data[[spatial_var]] (or nrow(data) when spatial_var is NULL).

Value

An object of class `hbsaems_data_check` with components:

`n_obs` Number of rows in the data.

`missing_summary` Named integer vector: per-variable count of `NA`.

`missing_pattern` Character: `"none"`, `"y_only"`, `"x_only"`, or `"both"`.

`dimension_check` Named list of dimension diagnostics.

`non_sample_warning` Character or `NULL` – a hint to investigate whether NA-Y rows are non-sample (out-of-sample) areas.

`recommended_method` Character: suggested `handle_missing` value (`"deleted"`, `"multiple"`, `"model"`, or `NA` when no missing values are present).

`recommendation_text` Human-readable rationale.

`issues` Character vector of fatal errors (length 0 if OK).

See Also

`hbm`, `sae_predict`

Examples

```
data("data_fhnorm")

# 1. Complete data -> no warnings, no recommendation
chk <- check_data(data_fhnorm,
                  response   = "y",
                  auxiliary  = c("x1", "x2", "x3"))
print(chk)

# 2. Missing-Y pattern -> recommends checking for non-sample areas
d <- data_fhnorm
d$y[1:5] <- NA
chk2 <- check_data(d, response = "y",
                      auxiliary  = c("x1", "x2", "x3"))
summary(chk2)

# 3. Missing-X-only -> recommends multiple imputation
d2 <- data_fhnorm
d2$x1[10:15] <- NA
chk3 <- check_data(d2, response = "y",
                      auxiliary  = c("x1", "x2", "x3"))
chk3$recommended_method

# 4. Spatial dimension check
data("adjacency_matrix_car")
chk4 <- check_data(data_fhnorm[1:5, ],
                   response   = "y",
                   auxiliary  = c("x1", "x2", "x3"),
                   M          = adjacency_matrix_car)
chk4$dimension_check
```

`check_shiny_deps` *Check Shiny App Dependencies*

Description

Classifies and inspects the optional packages used by the Shiny dashboard. The dashboard is launched by `run_sae_app` and the dependencies it touches live in `Suggests`, not `Imports`, so users who only use the modelling functions never have to install them.

Usage

```
check_shiny_deps(verbose = TRUE)
```

Arguments

`verbose`	Logical. When `TRUE` (default), prints a formatted summary of which dependencies are installed and which are missing.

Value

Invisibly, a named list with components:

`critical_missing` Character vector of critical packages not installed. When non-empty, the app cannot start.

`optional_missing` Character vector of optional packages not installed. When non-empty, the app starts but some panels degrade.

`install_cmd` A ready-to-paste `install.packages()` call covering all missing packages, or `NULL` when none are missing.

See Also

`run_sae_app`

Examples

```
check_shiny_deps()
```

`check_spatial_weight` *Validate a Spatial Weight Matrix Against CAR/SAR Theory*

Description

Runs five theoretical compatibility checks on a spatial weight matrix M and reports any deviations from the standard requirements of the chosen model class. Returns a structured object summarising the results.

Usage

```
check_spatial_weight(
  M,
  spatial_model = c("car", "sar"),
  verbose = TRUE,
  sre_type = NULL
)
```

Arguments

`M`	A square numeric matrix.
`spatial_model`	Character. `"car"` or `"sar"` – the model class the matrix is intended for.
`verbose`	Logical. When `TRUE` (default), prints a formatted diagnostic report.
`sre_type`	**Deprecated.** Use `spatial_model` instead. Kept for backward compatibility; will be removed in v2.0.0.

Details

Theoretical requirements: For a **CAR** model (Besag 1974), the joint distribution

$$u \sim \mathcal{N}\big(0,\, \sigma^2 (D - \rho W)^{-1}\big)$$

is well-defined only when W is **symmetric** with **zero diagonal**. By convention, W is taken as the **binary adjacency matrix** (`style = "B"`) so that $D = \mathrm{diag}(\text{row sums})$ has integer entries.

For a **SAR** model (Whittle 1954, Anselin 1988),

$$u = \rho W u + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma^2 I)$$

and W is conventionally **row-standardised** (`style = "W"`) so that the spatial autoregressive parameter ρ can be interpreted as a normalised correlation in $(-1, 1)$. Symmetry is *not* required for SAR.

Style detection heuristic:

- `"B"` (binary): all values in $\{0, 1\}$.
- `"W"` (row-standardised): every non-zero row sums to 1.
- `"other"`: neither of the above.

Value

Invisibly, an object of class `hbsaems_spatial_check` with components:

`is_square` Logical.

`has_zero_diag` Logical.

`is_symmetric` Logical.

`detected_style` Character: `"B"`, `"W"`, or `"other"`.

`n_isolated` Integer: number of areas with no neighbours.

`n_components` Integer: number of connected components.

`issues` Character vector of fatal errors.

`warnings` Character vector of soft warnings.

`compatible` Logical: TRUE if matrix is theoretically compatible with `spatial_model`.

See Also

`build_spatial_weight`, `hbm`

Examples

```
# Build a small valid CAR matrix
M <- matrix(c(0, 1, 1, 0,
              1, 0, 0, 1,
              1, 0, 0, 1,
              0, 1, 1, 0), 4, 4)
check_spatial_weight(M, spatial_model = "car")

# An asymmetric matrix flagged for CAR
M2 <- M; M2[1, 2] <- 2
check_spatial_weight(M2, spatial_model = "car", verbose = FALSE)$issues
```

---

convergence_check *MCMC Convergence Diagnostics for Fitted HBMs*

Description

Computes a battery of convergence tests and diagnostic plots for an `hbmfit` object. This is the primary convergence diagnostic function in **hbsaems** (supersedes the deprecated `hbcc`).

Usage

```
convergence_check(
  model,
  diag_tests = c("rhat", "geweke", "heidel", "raftery"),
  plot_types = c("trace", "dens", "acf", "nuts_energy", "rhat", "neff"),
  ...
)
```

Arguments

`model`	An `hbmfit` or `brmsfit` object.
`diag_tests`	Character vector of tests to run. Any subset of `c("rhat", "geweke", "heidel", "raftery")`.
`plot_types`	Character vector of plot types to generate. Any subset of `c("trace", "dens", "acf", "nuts_energy", "rhat", "neff")`.
`...`	Additional arguments passed to the underlying **brms** or **coda** routines.

Details

For each parameter θ_j, the Gelman-Rubin statistic is computed as

$$\widehat{R}_j = \sqrt{\frac{\widehat{\mathrm{var}}^{+}(\theta_j)}{W_j}}$$

where W_j is the within-chain variance and $\widehat{\mathrm{var}}^{+}$ is the marginal posterior variance estimate. Values close to 1 (typically below 1.1) indicate convergence.

Value

An `hbcc_results` object containing:

`rhat_ess` Matrix with columns `Rhat`, `Bulk_ESS`, `Tail_ESS`.

`geweke,raftery,heidel` Outputs from the corresponding **coda** routines, or `NULL` when the test fails.

`plots` Named list of `ggplot`/`bayesplot` objects.

See Also

`is_converged`, `diagnostic_summary`, `hbm_warnings`

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
# Uses brms's default MCMC settings (chains = 4, iter = 2000,
# warmup = 1000).  For challenging posteriors (e.g. funnel
# geometries in Fay-Herriot with small D_i), consider
# chains = 4, iter = 4000, warmup = 2000 and
# control = list(adapt_delta = 0.99).
model <- hbm(brms::bf(y ~ x1 + x2 + x3),
             data   = data_fhnorm,
             re     = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores  = 1, seed = 123, refresh = 0)

diag <- convergence_check(model)
summary(diag)
```

```
is_converged(model)
is_converged(model, threshold = 1.05)
```

`data_betalogitnorm` *Simulated Beta Logit-Normal Data*

Description

A simulated dataset for 100 regencies under a Beta logit-normal model. The response `y` is a proportion in $(0, 1)$.

Usage

```
data_betalogitnorm
```

Format

A data frame with 100 rows and 9 variables:

`y` Direct estimator of the area-level proportion.

`theta` True area-level proportion.

`x1`, `x2`, `x3` Auxiliary covariates.

`n` Area sample size.

`deff` Design effect.

`regency` Regency identifier (`"regency_001"` .. `"regency_100"`).

`province` Province identifier (`"province_01"` .. `"province_05"`) – spatial cluster level for CAR / SAR.

Source

Simulated.

`data_binlogitnorm` *Simulated Binomial Logit-Normal Data*

Description

A simulated dataset for 100 districts under a Binomial logit-normal model. Each district provides a number of successes (`y`) out of `n` trials.

Usage

```
data_binlogitnorm
```

Format

A data frame with 100 rows and 14 variables:

`n` Number of trials in the district.

`y` Number of successes.

`p` Direct proportion (`y / n`).

`x1`, `x2`, `x3` Auxiliary covariates.

`u_true` True area-level random effect (logit scale).

`eta_true` True linear predictor (logit scale).

`p_true` True success probability.

`psi_i` Sampling variance.

`y_obs`, `p_obs` Observed (direct) values.

`district` District identifier (`"district_001"` .. `"district_100"`) – the 100 small areas to estimate.

`regency` Regency identifier (`"regency_01"` .. `"regency_05"`) – coarse spatial-cluster level (5 regencies each containing 20 districts). Pair with `adjacency_matrix_car_regency`.

Source

Simulated.

`data_fhnorm` *Simulated Fay-Herriot Normal Data*

Description

A simulated dataset for 100 regencies under the Fay-Herriot Normal small-area model. Used as the running example throughout the package documentation and vignettes. The simulation is engineered so that the canonical Fay-Herriot fit (`hbm(..., sampling_variance = "D")`) converges with default brms / Stan settings – no divergent transitions, no manual tuning required.

Usage

```
data_fhnorm
```

Format

A data frame with 100 rows and 9 variables:

`y` Direct (survey) estimator of the area mean.

`D` Sampling variance of the direct estimator (KNOWN from the survey design; treat as input, not as a parameter).

`x1`, `x2`, `x3` Auxiliary covariates at the area level, simulated from $\mathcal{N}(0, 1)$.

`theta_true` True area-level latent value θ_i.

`u` True area-level random effect u_i.

`regency` Regency identifier (`"regency_001"` through `"regency_100"`) used as the IID random-effect grouping variable. Use with `re = ~ (1 | regency)` or `area_var = "regency"`.

`province` Province identifier (`"province_01"` through `"province_05"`) – 20 regencies per province. Used as the spatial random-effect grouping variable for CAR / SAR / BYM2 examples; also serves as the higher level in the hierarchical-area example `area_var = c("province", "regency")`.

Details

Generative model. For each regency $i = 1, \ldots, 100$,

$$y_i = \theta_i + \varepsilon_i, \quad \varepsilon_i \sim \mathcal{N}(0, D_i)$$

$$\theta_i = 10 + 0.8\, x_{1i} - 0.5\, x_{2i} + 0.3\, x_{3i} + u_i, \quad u_i \sim \mathcal{N}(0, \sigma_u^2)$$

with auxiliary covariates $x_j \sim \mathcal{N}(0, 1)$ (already standardised), area RE SD $\sigma_u = 1$, and **known** sampling variances $D_i \sim \text{Gamma}(\text{shape} = 4, \text{rate} = 4)$ – a realistic spread ($\approx [0.2, 3.0]$) that mirrors varying sample sizes across regencies.

Important: pass `D` as the sampling variance. In any fit on this dataset, supply `sampling_variance = "D"`; otherwise the residual σ and the area-RE σ_u compete to explain the same variance, producing weak identifiability and divergent transitions.

Source

Simulated. Reproducible script in `data-raw/data_fhnorm.R`.

`data_lnln` *Simulated Lognormal-Lognormal Data*

Description

A simulated dataset for 100 districts under a Lognormal-Lognormal model. Suitable for strictly positive, right-skewed outcomes.

Usage

```
data_lnln
```

Format

A data frame with 100 rows and 13 variables:

`district` District identifier (`"district_001"` .. `"district_100"`) – the 100 small areas to estimate.

`x1`, `x2`, `x3` Auxiliary covariates.

`u_true` True area-level random effect (log scale).

`teta_true` True linear predictor (log scale).

`mu_orig_true` True mean on the original scale.

`n` Area sample size.

`y_obs` Observed direct estimator.

`lambda_dir` Direct estimator scale parameter.

`y_log_obs` Observed value on the log scale.

`psi_i` Sampling variance on the log scale.

`regency` Regency identifier (`"regency_01"` .. `"regency_05"`) – coarse spatial-cluster level (5 regencies each containing 20 districts). Pair with `adjacency_matrix_car_regency`.

Source

Simulated.

deprecated *Deprecated Functions*

Description

These functions were the main entry point in **hbsaems** $\leq$ 0.2.x. They are retained for backwards compatibility but now call the new primary functions and emit a deprecation warning via `.Deprecated`. They will be removed in **v2.0.0**.

Usage

```
hbcc(
  model,
  diag_tests = c("rhat", "geweke", "heidel", "raftery"),
  plot_types = c("trace", "dens", "acf", "nuts_energy", "rhat", "neff"),
  ...
)

hbmc(
  model,
  model2 = NULL,
  ndraws_ppc = 100,
  moment_match = FALSE,
  moment_match_args = list(),
  reloo_args = list(),
  plot_types = c("pp_check", "params"),
  comparison_metrics = c("loo", "waic", "bf"),
  run_prior_sensitivity = FALSE,
  sensitivity_vars = NULL,
  ...
)

hbpc(model, data, response_var, ndraws_ppc = 50, ...)

hbsae(model, newdata = NULL, ...)
```

Arguments

`model`	An `hbmfit` object.
`diag_tests, plot_types, ndraws_ppc, moment_match, moment_match_args, reloo_args, comparison_metrics, run_prior_sensitivity, sensitivity_vars`	Forwarded unchanged to the replacement function.
`...`	Additional arguments forwarded to the replacement.
`model2`	For `hbmc`: optional second model.
`data`	For `hbpc`: the data `data.frame`.

response_var	For hbpc: name of the response variable.
newdata	For hbsae: optional new data.

Value

Identical to the corresponding replacement function.

Migration guide

Deprecated	Replacement
hbcc(model, ...)	convergence_check(model, ...)
hbmc(model, ...)	model_compare(model, ...)
hbpc(model, data, response_var)	prior_check(model, data, response_var)
hbsae(model, ...)	sae_predict(model, ...)

---

diagnostic_summary *Extract a Diagnostic Summary*

Description

Extract a Diagnostic Summary

Usage

```
diagnostic_summary(x)
```

Arguments

x	An hbmfit or hbcc_results object.

Value

A named list of diagnostic statistics.

`get_hbsae_model` *Inspect a Registered HBSAE Model Specification*

Description

Inspect a Registered HBSAE Model Specification

Usage

```
get_hbsae_model(key)
```

Arguments

`key`	Character. A model key returned by `list_hbsae_models`.

Value

The named list spec, or `NULL` if not found. Useful fields include:

- `family` – brms family name passed to `brmsfamily`.
- `link` – default link function used by the family (`"identity"`, `"logit"`, `"log"`, ...). See `brmsfamily` for the complete set of supported links per family.
- `discrete` – whether the response is discrete.
- `supports_mi` – whether brms-canonical `mi()` imputation is allowed for this family (FALSE for all discrete responses).

See Also

`register_hbsae_model`, `brmsfamily` for the canonical brms family / link reference.

Examples

```
get_hbsae_model("lognormal")
get_hbsae_model("beta")$link  # "logit"
```

hbm *hbm: Hierarchical Bayesian Small Area Models*

Description

Fits a hierarchical Bayesian model for Small Area Estimation (SAE) using the brms package (Stan back-end). The function supports fixed effects, random effects, spatial random effects (CAR/SAR), user-defined priors, and three strategies for handling missing data in auxiliary (predictor) variables.

Usage

```
hbm(
  formula,
  hb_sampling = "gaussian",
  hb_link = "identity",
  link_phi = "log",
  re = NULL,
  spatial_var = NULL,
  spatial_model = NULL,
  car_type = NULL,
  sar_type = NULL,
  M = NULL,
  data,
  prior = NULL,
  fixed_params = NULL,
  sampling_variance = NULL,
  prior_type = "default",
  hs_df = 1,
  hs_df_global = 1,
  hs_df_slab = 4,
  hs_scale_global = NULL,
  hs_scale_slab = 2,
  hs_par_ratio = NULL,
  hs_autoscale = TRUE,
  r2d2_mean_R2 = 0.5,
  r2d2_prec_R2 = 2,
  r2d2_cons_D2 = NULL,
  r2d2_autoscale = TRUE,
  nonlinear = NULL,
  nonlinear_type = "spline",
  spline_k = -1L,
  spline_bs = "tp",
  gp_k = NA_integer_,
  gp_cov = "exp_quad",
  gp_c = NULL,
  gp_scale = NULL,
  handle_missing = NULL,
```

```
  m = 5L,
  mice_args = list(),
  measurement_error = NULL,
  control = list(),
  chains = 4L,
  iter = 4000L,
  warmup = floor(iter/2),
  cores = 1L,
  sample_prior = "no",
  sre = NULL,
  sre_type = NULL,
  stanvars = NULL,
  ...
)
```

Arguments

formula	A brmsformula or standard formula object specifying the model structure. For multi-response or imputation sub-models use bf. Examples: formula(y ~ x1 + x2), bf(y ~ x1 + x2), or bf(y | mi() ~ mi(x1) + x2) + bf(x1 | mi() ~ x2).
hb_sampling	Character string naming the distribution family of the response variable (default: "gaussian"). Any family supported by brmsfamily is accepted.
hb_link	Character string specifying the link function (default: "identity").
link_phi	Character string specifying the link function for the precision/phi parameter (default: "log"). Only used for Beta and related families.
re	An optional one-sided formula specifying group-level (random) effects, e.g. ~(1|area). Must follow standard lme4-style syntax: ~ (1|group1) + (1|group2). If NULL (default) *and* spatial_model is also NULL, the function emits a warning recommending an area-level random effect, since a Fay-Herriot SAE model with neither IID nor spatial area effects degenerates to fixed-effects regression and does not borrow strength across areas. The warning can be silenced with suppressWarnings() if a fixed-effects-only baseline is intentional.
spatial_var	Character. Name of the column in data that identifies the spatial areas (e.g. "regency" or "province"). Must be supplied *together with* spatial_model and M; providing only one of them is an error. Distinct from re: re is a formula for IID random effects, whereas spatial_var is a column name (string) for the spatially-structured random effect.
spatial_model	Character. Type of spatial model: "car" (Conditional Autoregressive) or "sar" (Simultaneous Autoregressive). Must be supplied *together with* spatial_var and M; providing only one of them is an error.
car_type	Character. CAR subtype passed to brms: one of "escar" (exact sparse CAR), "esicar" (exact sparse intrinsic CAR), "icar" (intrinsic CAR), or "bym2". Defaults to "icar" when spatial_model = "car".
sar_type	Character. SAR subtype: "lag" (SAR of the response values) or "error" (SAR of the residuals). Defaults to "lag" when spatial_model = "sar".

`M` Spatial matrix supplied as `data2` to `brms`. For CAR this must be a binary adjacency matrix; for SAR a spatial weight matrix. Row names must match the levels of `spatial_var`.

`data` A data frame containing all variables referenced in `formula`.

`prior` Priors specified via `prior` or a list thereof. If `NULL` (default), `brms` default priors are used.

`fixed_params` Named list pinning distributional parameters to known values instead of sampling them. Each entry maps a parameter name to one of:

A column name (character) The named column in `data` is used as the fixed values.

A scalar (numeric, length 1) Broadcast to all rows.

A vector (numeric, length `nrow(data)`) Used directly.

A one-sided formula (e.g. `~ I(n / deff - 1)`) Evaluated against `data` to produce the vector of fixed values.

Internally, each pinned parameter `<par>` is attached to the data as a column `.hbsaems_<par>_fixed` and added to the brms formula as `<par> ~ 0 + offset(.hbsaems_<par>_fixed`. Using `fixed_params` on a parameter for which the user also supplies an explicit prior is an error. Typical use cases include pinning `phi` for Beta regression from survey design effect (`phi = ~ I(n / deff - 1)`) and pinning `sigma` for Fay–Herriot-style models with known sampling variance.

`sampling_variance`

Optional character. Name of a column in `data` containing the **known** sampling variance D_i for each area (the Fay-Herriot sugar). When supplied, $\sigma_i = \sqrt{D_i}$ is pinned via offset. This is the canonical way to fit a Gaussian Fay-Herriot model: without it, the residual σ and the area-RE σ_u compete to explain the same variance and the model is unidentified, typically producing divergent transitions. Equivalent to `fixed_params = list(sigma = sqrt(data[[<col>]]))`.

Family compatibility. `sampling_variance` requires a continuous family whose response distribution has a residual scale parameter named `sigma`. Supported: `gaussian` (the canonical Fay-Herriot case), `lognormal` (D must be on the log scale; see also `hbm_lnln`), `student`, `skew_normal`, `exgaussian`, `asym_laplace`. A helpful error is thrown when an incompatible family is supplied.

For non-Gaussian SAE families the analogous pinning mechanism is family-specific:

- **Beta**: pin the precision `phi` via the survey design effect, e.g.\ `fixed_params = list(phi = ~ I(n/deff - 1))` (Liu 2009). See `hbm_betalogitnorm`.
- **Binomial**: sampling variability enters through the `trials` addition term, not through a separate `sigma`. See `hbm_binlogitnorm`.
- **Poisson, Gamma, Weibull**: variance is tied algebraically to the mean – no separate scale parameter to pin.

`prior_type` Character. Global-local shrinkage prior applied to all regression coefficients (`class = "b"`). One of:

`"default"` No shrinkage prior is added; the `prior` argument governs everything (default).

"horseshoe" Regularised horseshoe prior (Piironen & Vehtari 2017). Encourages exact sparsity while allowing large signals through. Controlled by hs_df, hs_df_global, hs_df_slab, hs_scale_global, hs_scale_slab, hs_par_ratio, and hs_autoscale.

"r2d2" R2D2 prior (Zhang et al. 2022). Places a prior directly on the model R^2 and distributes explained variance across predictors via a Dirichlet decomposition. Controlled by r2d2_mean_R2, r2d2_prec_R2, r2d2_cons_D2, and r2d2_autoscale.

If prior already contains a global class = "b" entry, prior_type is ignored and a warning is issued.

Cascading to smooth and GP terms. When the formula contains s() or gp() terms, the shrinkage prior is automatically extended to the corresponding parameter classes "sds" (spline SDs) and/or "sdgp" (GP SDs) using the brms-canonical main = TRUE pattern. The resulting prior regularises ALL components jointly – linear coefficients, nonlinear smooth wiggliness, and GP marginal variance – which is the principled approach to global-local shrinkage in models that combine parametric and nonparametric components.

hs_df Numeric > 0. Local half-t degrees of freedom for the horseshoe prior (default 1 = half-Cauchy).

hs_df_global Numeric > 0. Global half-t degrees of freedom (default 1).

hs_df_slab Numeric > 0. Slab half-t degrees of freedom (default 4).

hs_scale_global

Numeric > 0 or NULL. Scale for the global half-t prior. NULL (default) lets brms compute it automatically from the number of predictors via hs_autoscale.

hs_scale_slab Numeric > 0. Scale for the slab component (default 2).

hs_par_ratio Numeric > 0 or NULL. Expected ratio of non-zero to total coefficients. NULL (default) treats all coefficients as potentially non-zero.

hs_autoscale Logical. Whether brms should auto-scale the horseshoe prior using the residual SD σ. Default TRUE; set to FALSE for non-continuous responses (binomial, Poisson, ...) where σ is not defined.

r2d2_mean_R2 Numeric in $(0, 1)$. Prior mean of the model R^2 (default 0.5).

r2d2_prec_R2 Numeric > 0. Prior precision of R^2 (default 2). Higher values concentrate mass around r2d2_mean_R2.

r2d2_cons_D2 Numeric > 0 or NULL. Dirichlet concentration for the D2 component. NULL (default) uses the brms default 0.5.

r2d2_autoscale Logical. Whether brms should auto-scale the R2D2 prior using σ. Default TRUE; set to FALSE for non-continuous responses.

nonlinear Character vector or NULL. Names of predictor variables to model with a smooth nonlinear term. Each listed variable is replaced in the formula RHS with s(var) (spline) or gp(var) (Gaussian process). Variables not listed remain linear. Do *not* also write s(x) in the formula when using this argument – the modification is applied automatically. Default NULL (all predictors remain linear).

nonlinear_type Character. Smooth term family to use. One of "spline" (default, penalised regression spline via mgcv::s()) or "gp" (Gaussian process via brms::gp()).

`spline_k` Integer. Spline basis dimension (number of knots) passed to `mgcv::s(..., k = ...)`. `-1L` (default) lets **mgcv** choose automatically. For SAE typically `k = 8` to `15`. Ignored when `nonlinear_type = "gp"`.

`spline_bs` Character. Spline basis type passed to `mgcv::s(..., bs = ...)`. Default `"tp"` (thin-plate regression spline, the **mgcv** default). Common alternatives: `"cr"` (cubic regression spline; often more stable for SAE with correlated auxiliary variables), `"cs"` (cubic with shrinkage; allows variable selection), `"ps"` (P-splines). Ignored when `nonlinear_type = "gp"`.

`gp_k` Integer or `NA`. Number of basis functions for the Hilbert-space approximate GP (Riutort-Mayol et al.\ 2020), passed to `brms::gp(..., k = ...)`. `NA` (default) = exact GP which scales $O(n^3)$ and is **not recommended for** $n > 100$ areas; an immediate warning is emitted in that case pointing to this argument. Integer values `10`–`25` are typical for SAE and dramatically improve convergence and runtime. Ignored when `nonlinear_type = "spline"`.

`gp_cov` Character. GP covariance function passed to `brms::gp(..., cov = ...)`: `"exp_quad"` (squared exponential / RBF, default), `"matern15"` (Matern 3/2), `"matern25"` (Matern 5/2; often more numerically stable for SAE than RBF), or `"exponential"`.

`gp_c` Numeric > 0 or `NULL`. Hilbert-space GP boundary-scale factor passed to `brms::gp(..., c = ...)`. Default brms value is $5/4$ (= 1.25); increase if the GP appears truncated at the domain boundaries. Only relevant when `gp_k` is supplied.

`gp_scale` **Deprecated.** Use `gp_c` instead. The old name suggested a length-scale interpretation but actually mapped to the HSGP boundary-scale factor. Will be removed in v2.0.0.

`handle_missing` Character or `NULL`. Strategy for missing data. One of `"deleted"`, `"multiple"`, or `"model"` (see **Details**). If `NULL` (default) and missing values exist in the data, an informative error is raised.

`m` Integer. Number of imputations when `handle_missing = "multiple"` (default: `5`). Ignored for other strategies.

`mice_args` A named list of additional arguments forwarded to `mice`, for example `list(method = "pmm", seed = 42)`. Only used when `handle_missing = "multiple"`.

`measurement_error`
Optional named list specifying which auxiliary variables are measured with error and where to find their standard errors. The list maps variable names to columns in `data` containing the SE, e.g. `measurement_error = list(x1 = "se_x1", x2 = "se_x2")`. Listed variables are wrapped on-the-fly with `mi(var, se_col)` in the brmsformula so that brms treats them as latent variables with a Gaussian measurement-error structure, following Ybarra and Lohr (2008). Standard errors must be non-negative and have no missing values; `measurement_error` variables must be a subset of the model's auxiliary (linear) predictors. When the user has already written `mi(...)` explicitly in the formula, the corresponding entries in `measurement_error` are detected and not duplicated. Note that ME inflates the parameter space (each *x_i* becomes latent), which slows sampling and increases the risk of divergent transitions, especially when combined with smooth terms (`nonlinear`). See the "Measurement error" section of the SAE vignette.

control	A named list of sampler control parameters (default: list()). Passed directly to brm.
chains	Integer. Number of MCMC chains (default: 4).
iter	Integer. Total iterations per chain (default: 4000).
warmup	Integer. Warm-up iterations per chain (default: floor(iter / 2)).
cores	Integer. Number of CPU cores for parallel sampling (default: 1).
sample_prior	Character. Whether to draw from the prior distribution. One of "no" (default), "yes", or "only".
sre	**Deprecated.** Use spatial_var instead. Kept for backward compatibility; will be removed in v2.0.0.
sre_type	**Deprecated.** Use spatial_model instead. Kept for backward compatibility; will be removed in v2.0.0.
stanvars	An optional stanvar object (or a list of such objects) supplying additional Stan code, data, or parameters. Passed directly to brm and brm_multiple. Intended for use by wrapper functions such as hbm_betalogitnorm that require custom Stan blocks; end users typically do not need to set this argument. Default: NULL.
...	Additional arguments forwarded to brm or brm_multiple.

Details

Hierarchical Bayesian Model for Small Area Estimation

Spatial Small Area Estimation Models

For spatially correlated areas, hbsaems extends the standard area-level SAE model (Fay-Herriot 1979) by adding a spatially structured random effect:

$$y_i = x_i^\top \boldsymbol{\beta} + u_i + e_i,$$

where u_i is the spatial random effect for area i and e_i the sampling error. Two families of spatial structures are supported.

CAR (Conditional Autoregressive; Besag 1974) Specified by spatial_model = "car". The joint distribution of the spatial effects is

$$u \sim \mathcal{N}\big(0,\, \sigma_u^2 (D - \rho W)^{-1}\big),$$

where W is a binary adjacency matrix (1 if neighbour, 0 otherwise) and $D = \mathrm{diag}(W\mathbf{1})$. Sub-types via car_type: "icar" (intrinsic, $\rho = 1$; Besag 1991); "escar", "esicar" (exact sparse formulations of Morris et al.\ 2019); "bym2" (BYM2 reparameterisation of Riebler et al.\ 2016, recommended for disconnected graphs).

SAR (Simultaneous Autoregressive; Whittle 1954, Anselin 1988) Specified by spatial_model = "sar". The model is

$$u = \rho W u + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 I),$$

where W is row-standardised so that $\rho \in (-1, 1)$ carries an interpretable correlation meaning. Sub-types via sar_type: "lag" (spatial lag of the response, $y = \rho W y + X\boldsymbol{\beta} + \varepsilon$); "error" (spatial error model).

Use check_spatial_weight to verify that M satisfies the theoretical requirements (square, zero diagonal, symmetry for CAR, style-appropriate for the model class). Use build_spatial_weight to construct M from a shapefile.

Missing Data Strategies

The three strategies differ in scope and statistical assumptions:

`"deleted"` Complete-case analysis. Only rows where **all** response variable(s) are observed are used for model fitting. Auxiliary variables must be complete; otherwise an informative error is raised. Appropriate under MCAR (Missing Completely At Random).

`"multiple"` Multiple imputation via `mice` **for auxiliary (predictor) variables only**. The response variable Y is **never** imputed. In a Bayesian model, missing outcomes are naturally marginalised through the posterior predictive distribution:

$$p(\theta \mid Y_{\text{obs}}, X) = \int p(\theta \mid Y_{\text{obs}}, Y_{\text{mis}}, X)\, p(Y_{\text{mis}} \mid Y_{\text{obs}}, X, \theta)\, \mathrm{d}Y_{\text{mis}}.$$

Imputing Y before fitting would replace this integral with a single point substitute, deflate posterior uncertainty, and potentially bias the estimates if the imputation model is misspecified. If Y has missing values, those rows are excluded from model fitting (a `warning` is issued) but are retained in the returned object for subsequent prediction via hbsae. Appropriate under MAR (Missing At Random).

`"model"` Model-based imputation using `brms::mi()`. Missing values in auxiliary variables are jointly estimated with the model parameters. The user must specify imputation sub-models explicitly in the `formula` argument, e.g.: `bf(y | mi() ~ mi(x1) + x2) + bf(x1 | mi() ~ x2)`. Only applicable to **continuous** distributions. Appropriate under MAR.

If data are Missing Not At Random (MNAR), none of the above strategies applies directly; sensitivity analyses and explicit missingness models are recommended.

Value

An object of class `hbmfit`, which is a named list containing:

`model`	The fitted `brmsfit` object (or `brmsfit_multiple` when `handle_missing = "multiple"` with missing predictors).
`handle_missing`	The missing-data strategy used (`NULL` if the data were complete).
`data`	The **original** data frame passed to `hbm()` before any row deletion or imputation. This is intentional: hbsae needs all rows – including those with missing Y – to generate predictions for every small area.

How to pin distributional parameters per family

hbsaems exposes three layers for pinning distributional parameters to known values, in increasing order of generality:

1. **Family-specific sugar** (least typing, most readable). Each wrapper exposes a convenience argument that maps to a well-defined Fay-Herriot-style transformation of survey design quantities:

Wrapper	Sugar argument	Pinned dpar
`hbm(..., hb_sampling = "gaussian")`	`sampling_variance = "D"`	$\sigma_i = \sqrt{D_i}$
`hbm_lnln()`	`sampling_variance = "psi"`	$\sigma_i = \sqrt{\psi_i}$ (on log scale)
`hbm_betalogitnorm()`	`n = "n", deff = "deff"`	$\phi_i = n_i/\mathrm{deff}_i - 1$ (Liu 2009)
`hbm_binlogitnorm()`	`trials = "n"`	(sampling variance built into the family)

2. **Universal** `fixed_params` (works everywhere). A named list pinning any distributional parameter – accepts a column name (character), a scalar (numeric of length 1), a vector of length `nrow(data)`, or a one-sided formula (evaluated in `data`'s environment). Examples:
   - `fixed_params = list(sigma = "D")` – pin sigma to a column.
   - `fixed_params = list(phi = ~ I(n / deff - 1))` – pin phi via formula.
   - `fixed_params = list(nu = 4)` – pin Student-t df to a scalar.
3. **Raw** `stanvars` (for power users authoring custom Stan code). Direct injection of Stan code blocks – see `stanvar`.

Sugar arguments are simply thin wrappers around layer 2: they validate the survey-design inputs (no NAs, strictly positive) and translate to `fixed_params` before delegating to the universal machinery. Hence the conflict policy below applies uniformly to both sugar and explicit `fixed_params`.

Conflict resolution between prior, prior_type, fixed_params, and stanvars

`hbm()` provides four orthogonal mechanisms to influence the prior / parameter specification of the underlying **brms** model:

1. `prior` – explicit `brmsprior` object(s).
2. `prior_type` – global-local shrinkage prior on the regression coefficients (cascades to `"sds"` / `"sdgp"` when splines or GPs are present).
3. `fixed_params` – pin distributional parameters to known values via the offset trick.
4. `stanvars` – inject custom Stan code blocks.

Plus two family-specific *sugar* arguments that translate to `fixed_params` internally:

- `sampling_variance` (continuous families): pins $\sigma_i = \sqrt{D_i}$, equivalent to `fixed_params$sigma = sqrt(data$D)`.
- `n` + `deff` (`hbm_betalogitnorm`): pins $\phi_i = n_i/\mathrm{deff}_i - 1$, equivalent to `fixed_params$phi = n / deff - 1`.

Combining these without rules in mind can produce unidentified models or compile-time errors. `hbm()` therefore enforces the following **conflict matrix**, where each cell describes what happens when the row and column inputs both target the same distributional parameter (e.g.\ both pin `sigma`):

	fixed_params	**prior**	**prior_type**	**stanvars**
sampling_variance	error	error (transitive)	no overlap	error (transitive)
n + deff	error	error (transitive)	no overlap	error (transitive)
fixed_params	–	error (10b.i)	no overlap	error (10b.ii)
prior	error (10b.i)	–	warning, user wins	no check needed
prior_type	no overlap	warning, user wins	–	no check needed

stanvars	error (10b.ii)	no check needed	no check needed	–

Resolution semantics in detail:

- `prior` **vs** `prior_type`**.** If the user supplies a *global* (no `coef =`) prior on `class = "b"`, `"sds"`, or `"sdgp"`, `prior_type` is silently dropped for that class and a warning is emitted. Coefficient-specific user priors (`coef = "x1"`) are kept alongside the shrinkage prior without warning.
- `fixed_params` **vs** `prior`**.** A pinned parameter is removed from the sampler; supplying a prior on that same parameter therefore has no effect and is treated as a user error – an informative `stop()` is issued.
- `fixed_params` **vs** `stanvars`**.** Same logic as above: a sampling statement in `stanvars` that targets a pinned parameter would fail at Stan compile time; `hbm()` catches this and stops with a clear message.
- **Sugar vs** `fixed_params` **on the same parameter.** The sugar translators (`.translate_sampling_variance()`, `.translate_n_deff_to_phi()`) error out if the user has also pre-populated `fixed_params` for the target parameter – there should never be two pin sources for the same dpar.
- **Sugar vs** `prior` / `stanvars` **transitively.** After the sugar -> `fixed_params` translation, the downstream `fixed_params`-vs-prior / -stanvars checks fire automatically. E.g.\ `sampling_variance = "D"` plus `prior = set_prior("normal(0, 1)", class = "sigma")` errors via the `fixed_params` vs `prior` rule.

The intent is to fail fast and explicitly rather than silently producing an unidentified or mis-specified model.

Convergence advice for SAE practitioners

Common convergence pathologies in hierarchical Bayesian SAE models and how to address them. Run `convergence_check()` after fitting to inspect $\hat{R}$, effective sample size (ESS), and divergent transitions.

1. Default sampler settings (recommended starting point).

- `chains = 4`, `iter = 4000`, `warmup = 2000`
- `control = list(adapt_delta = 0.95, max_treedepth = 12)`
- `cores = parallel::detectCores() - 1`

2. Divergent transitions. Most common cause is the *funnel* geometry of hierarchical variance parameters.

- First-line: increase `adapt_delta` to `0.99`.
- If still diverging, increase `warmup` and consider a tighter prior on the area-level standard deviation (`sd` class), e.g.\ `set_prior("normal(0, 0.5)", class = "sd")`.
- For Beta/Binomial logit-normal models, prior on the random intercept SD should also be on the logit scale.
- **Gaussian (Fay-Herriot) only.** Always supply the known sampling variance via `sampling_variance = "<col>"`. Without this, the residual σ and the area-RE σ_u compete to explain the same variance component, producing weak identifiability and divergent transitions almost regardless of `adapt_delta`. This is the single most common cause of divergences in Fay-Herriot fits and should be checked *first* before any sampler-tuning.

3. Low effective sample size (ESS < 1000).

- Increase `iter` (e.g.\ to `6000`); this is the single most reliable fix.
- Centre and scale the auxiliary variables before fitting.
- Check for prior–data conflict via **priorsense**; see `?prior_check`.

4. Gaussian processes.

- **Exact GP scales $O(n^3)$.** For $n > 100$ areas, set `gp_k` to use the Hilbert-space approximate GP (Riutort-Mayol et al.\ 2020). A heuristic is `gp_k = ceiling(min(n / 5, 25))`.
- Try `gp_cov = "matern25"` (Matern 5/2) if the default squared-exponential covariance is numerically unstable.
- The boundary-scale factor `gp_c` (brms default 1.25) may need increasing if the posterior GP is truncated at the domain edges.

5. Splines.

- Start with `spline_k = -1` (auto). Increase only if the residual diagnostics suggest under-smoothing.
- For strongly correlated auxiliary variables, try `spline_bs = "cr"` (cubic regression spline) for better numerical stability than the default thin-plate.
- For variable selection, use `spline_bs = "cs"` (cubic with shrinkage); coefficients on irrelevant smooths shrink toward zero.

6. Spatial models.

- For CAR, `car_type = "bym2"` (Riebler et al.\ 2016) is the modern recommendation; it stabilises the spatial/IID decomposition via a single mixing parameter.
- Verify the weight matrix with `check_spatial_weight()`; isolated areas or multiple disconnected components cause non-identifiability.

7. Prior predictive check first. Always call `prior_check()` (`sample_prior = "only"`) before the full posterior run. Implausible prior predictives are the single most common cause of slow / divergent sampling.

Author(s)

Achmad Syahrul Choir, Saniyyah Sri Nurhayati, and Sofi Zamzanah

References

Rao, J. N. K., & Molina, I. (2015). *Small Area Estimation*. John Wiley & Sons.

Burkner, P. C. (2017). brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1), 1–28.

Riutort-Mayol, G., Burkner, P.-C., Andersen, M. R., Solin, A., & Vehtari, A. (2023). Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming. *Statistics and Computing*, 33, 17. doi:10.1007/s11222022101672

Riebler, A., Sorbye, S. H., Simpson, D., & Rue, H. (2016). An intuitive Bayesian spatial model for disease mapping that accounts for scaling. *Statistical Methods in Medical Research*, 25(4), 1145–1165.

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate imputation by chained equations in R. *Journal of Statistical Software*, 45(3), 1–67.

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
data <- data_fhnorm

# Standard brms-default MCMC settings used throughout these
# examples (chains = 4, iter = 2000, warmup = 1000).  For tougher
# posteriors (funnel geometry, weakly identified priors), bump to
# iter = 4000, warmup = 2000, control = list(adapt_delta = 0.99).
FAST <- list(chains = 4, iter = 2000, warmup = 1000, cores = 1,
             seed = 123, refresh = 0)

# -- Basic model ------------------------------------------------------------
model <- do.call(hbm, c(
  list(formula     = bf(y ~ x1 + x2 + x3),
       hb_sampling = "gaussian",
       hb_link     = "identity",
       re          = ~(1 | regency),
       data        = data),
  FAST
))
summary(model)

# -- Horseshoe prior (sparse coefficients) ----------------------------------
model_hs <- do.call(hbm, c(
  list(formula     = bf(y ~ x1 + x2 + x3),
       re          = ~(1 | regency),
       data        = data,
       prior_type  = "horseshoe",
       hs_df       = 1),
  FAST
))
summary(model_hs)

# -- R2D2 prior (prior on model R-squared) ----------------------------------
model_r2 <- do.call(hbm, c(
  list(formula      = bf(y ~ x1 + x2 + x3),
       re           = ~(1 | regency),
       data         = data,
       prior_type   = "r2d2",
       r2d2_mean_R2 = 0.5,
       r2d2_prec_R2 = 2),
  FAST
))
```

```
summary(model_r2)

# -- Spline smooth for x1 (nonlinear) ----------------------------------------
# x1 is modelled with s(x1); x2 and x3 remain linear.
model_spline <- do.call(hbm, c(
  list(formula        = bf(y ~ x1 + x2 + x3),
       re             = ~(1 | regency),
       data           = data,
       nonlinear      = "x1",
       nonlinear_type = "spline"),
  FAST
))
summary(model_spline)

# -- Gaussian process for x2 (nonlinear) -------------------------------------
model_gp <- do.call(hbm, c(
  list(formula        = bf(y ~ x1 + x2 + x3),
       re             = ~(1 | regency),
       data           = data,
       nonlinear      = "x2",
       nonlinear_type = "gp"),
  FAST
))
summary(model_gp)

# -- Missing data: deletion (Y missing, X complete) --------------------------
data_miss_y        <- data
data_miss_y$y[3:5] <- NA

model_deleted <- do.call(hbm, c(
  list(formula        = bf(y ~ x1 + x2 + x3),
       re             = ~(1 | regency),
       data           = data_miss_y,
       handle_missing = "deleted"),
  FAST
))
summary(model_deleted)

# -- Missing data: multiple imputation (X only -- Y is never imputed) ----------
data_miss_x          <- data
data_miss_x$x1[6:8]  <- NA

model_multiple <- do.call(hbm, c(
  list(formula        = bf(y ~ x1 + x2 + x3),
       re             = ~(1 | regency),
       data           = data_miss_x,
       handle_missing = "multiple",
       m              = 5),
  FAST
))
summary(model_multiple)

# -- Missing data: model-based imputation with mi() --------------------------
```

```
data_miss_x2         <- data
data_miss_x2$x1[6:7] <- NA

model_model <- do.call(hbm, c(
  list(formula        = bf(y | mi() ~ mi(x1) + x2 + x3) +
                        bf(x1 | mi() ~ x2 + x3),
       re             = ~(1 | regency),
       data           = data_miss_x2,
       handle_missing = "model"),
  FAST
))
summary(model_model)

# -- Spatial: CAR (Conditional Autoregressive) --------------------------------
data("adjacency_matrix_car")
model_car <- do.call(hbm, c(
  list(formula     = bf(y ~ x1 + x2 + x3),
       data        = data,
       spatial_var = "province",
       spatial_model    = "car",
       M           = adjacency_matrix_car),
  FAST
))
summary(model_car)

# -- Spatial: SAR (Simultaneous Autoregressive) -------------------------------
# spatial_weight_sar is a 100x100 row-standardised matrix with row-
# names regency_001..regency_100, so it pairs with the fine-grained
# "regency" column (100 levels) -- NOT with "province" (5 levels).
data("spatial_weight_sar")
model_sar <- do.call(hbm, c(
  list(formula     = bf(y ~ x1 + x2 + x3),
       data        = data,
       spatial_var = "regency",
       spatial_model    = "sar",
       M           = spatial_weight_sar),
  FAST
))
summary(model_sar)
```

`hbm-info` *Model Inspection Helpers*

Description

Quick getters for an `hbmfit` object: metadata, training data, and diagnostic warnings.

hbmfit *User-Facing Helper to Build an hbmfit Object*

Description

Convenience constructor that calls new_hbmfit then validate_hbmfit – the safe public entry-point if you ever need to build an hbmfit object manually (e.g.\ when adapting a non-**brms** model). In normal usage you do *not* need to call this: hbm and the distribution-specific wrappers do it for you.

Usage

```
hbmfit(model, data, missing_method = NULL)
```

Arguments

model	A brmsfit or brmsfit_multiple object.
data	The data.frame used to fit model.
missing_method	Character scalar or NULL. One of "deleted", "multiple", "model".

Value

A validated hbmfit object.

See Also

new_hbmfit, validate_hbmfit, hbm

Examples

```
# Uses brms-default MCMC settings (chains = 4, iter = 2000,
# warmup = 1000) -- this toy data is only for verifying the
# hbmfit class structure, not for inference.
raw <- brms::brm(y ~ x1, data = data.frame(y = rnorm(10), x1 = 1:10),
                 chains = 4, iter = 2000, warmup = 1000, refresh = 0)
fit <- hbmfit(model = raw,
              data  = data.frame(y = rnorm(10), x1 = 1:10),
              missing_method = NULL)
validate_hbmfit(fit)
```

hbmfit-class *The hbmfit S3 Class*

Description

`hbmfit` is the result class returned by all model-fitting functions in **hbsaems**: `hbm`, `hbm_betalogitnorm`, `hbm_binlogitnorm`, and `hbm_lnln`. It wraps a `brmsfit` object together with the original data and fitting metadata.

Slots

`model` A `brmsfit` (or `brmsfit_multiple`) object.

`missing_method` Character scalar – the missing-data strategy used (`"deleted"`, `"model"`, `"multiple"`) or `NULL` when the data were complete.

`data` The **original** `data.frame` passed to the fitting function before any row deletion or imputation. Downstream functions (e.g.\ `sae_predict`) need all rows – including those with missing Y – to produce area-level predictions.

`handle_missing` Backwards-compatibility alias for `missing_method`; identical value.

hbmfit-methods *Standard S3 Methods for hbmfit*

Description

These methods allow `hbmfit` objects to be used with familiar base-R generics – `summary()`, `plot()`, `coef()`, etc. – in the same way as `brmsfit` objects from **brms**.

Arguments

`object`, `x`	An `hbmfit` object.
`type`	Plot type. See Details for available options.
`newdata`	Optional new data frame for predictions.
`...`	Additional arguments passed to the underlying **brms** method.

Value

Varies by method; see **brms** documentation for the underlying return types.

`hbm_betalogitnorm` *Small Area Estimation Under a Beta Likelihood (Logit-Normal Link)*

Description

Convenience wrapper around `hbm` for SAE problems where the response $y_i \in (0, 1)$ is modelled as

$$y_i \mid \theta_i, \phi_i \sim \mathrm{Beta}(\theta_i\phi_i, (1-\theta_i)\phi_i),$$
$$\mathrm{logit}(\theta_i) = x_i^\top \boldsymbol{\beta} + u_i, \quad u_i \sim \mathcal{N}(0, \sigma_u^2).$$

Usage

```
hbm_betalogitnorm(
  response,
  auxiliary = NULL,
  data,
  n = NULL,
  deff = NULL,
  area_var = NULL,
  area_re_structure = c("nested", "crossed"),
  spatial_var = NULL,
  spatial_model = NULL,
  car_type = NULL,
  sar_type = NULL,
  M = NULL,
  link_phi = NULL,
  prior = NULL,
  stanvars = NULL,
  handle_missing = NULL,
  m = 5L,
  control = list(),
  chains = 4L,
  iter = 4000L,
  warmup = floor(iter/2),
  cores = 1L,
  sample_prior = "no",
  fixed_params = NULL,
  predictors = NULL,
  group = NULL,
  sre = NULL,
  sre_type = NULL,
  ...
)
```

Arguments

`response`	Character. Name of the response column (must lie strictly between 0 and 1).

`auxiliary` Character vector of auxiliary (fixed-effect) variable names; corresponds to area-level covariates in SAE literature (see Rao & Molina 2015 Ch. 4).

`data` A data frame.

`n` Character or `NULL`. Name of the column giving the per-area sample size used to compute the fixed ϕ. Must be supplied together with `deff`. When supplied, $\phi_i = n_i/\text{deff}_i - 1$ is pinned via offset. Default: `NULL` (treats `phi` as random with brms's default prior).

`deff` Character or `NULL`. Name of the design-effect column. Required when `n` is supplied (and vice versa).

`area_var` Character vector or `NULL`. Name(s) of the area-grouping column(s). Length 1 adds an IID random intercept (`1 | area_var`); length ≥ 2 supports hierarchical areas (e.g.\ `c("province", "regency")`) – see `?hbm_flex` for the nested vs.\ crossed structures.

`area_re_structure`
Either `"nested"` (default) or `"crossed"`; controls how multiple area columns combine.

`spatial_var`, `spatial_model`, `car_type`, `sar_type`, `M`
Spatial random-effect arguments, forwarded to `hbm`.

`link_phi` Character or `NULL`. Link function for `phi`. **Default `NULL` resolves automatically**: `"identity"` when `phi` is pinned via the survey design (the `n` + `deff` sugar or `fixed_params$phi`, both of which produce raw positive offsets) and `"log"` otherwise (the brms default for `phi` in the Beta family; keeps $\phi > 0$ while letting NUTS sample on $\mathbb{R}$). Manually setting `"identity"` together with an estimated `phi` can let NUTS propose negative values and trigger divergent transitions; an informative warning is emitted in that case.

`prior` Optional `brmsprior` object. If `NULL`, sensible defaults are filled in:

- `Intercept ~ student_t(4, 0, 10)`
- `b ~ student_t(4, 0, 2.5)`
- `phi` – brms default `gamma(0.01, 0.01)` (in random mode; ignored in fixed mode).

The user may pass a partial prior: missing default classes are filled in automatically. To put a custom prior on `phi` (random mode), set `prior = set_prior("gamma(2, 0.5)",class = "phi")` or similar.

`stanvars` Optional `brmsstanvars` object for power users who need to inject custom Stan code blocks (e.g.\ transformed data, generated quantities, model-block statements not expressible via the `prior` argument). Passed through to brms verbatim. **Note:** As of v1.0.0, this wrapper no longer declares `alpha` or `beta` as Stan parameters, so legacy `stanvars` blocks containing sampling statements on `alpha` / `beta` (left over from the pre-v1.0.0 hierarchical-phi construction) will now raise an informative error.

`handle_missing`, `m`, `control`, `chains`, `iter`, `warmup`, `cores`, `sample_prior`, `...`
Passed through to `hbm`.

`fixed_params` Optional named list pinning distributional parameters to known values. See `hbm` for the spec format. Allows power-user access to the same machinery used by the `n`/`deff` arguments.

predictors	**Deprecated.** Use auxiliary instead. Kept for backward compatibility; will be removed in v2.0.0.
group	**Deprecated.** Use area_var instead.
sre	**Deprecated.** Use spatial_var instead.
sre_type	**Deprecated.** Use spatial_model instead.

Details

The precision parameter ϕ_i can either be pinned to a survey design effect (n + deff) or sampled with brms's default weakly-informative prior $\phi \sim \mathrm{Gamma}(0.01, 0.01)$.

Value

An object of class hbmfit.

Migration note (v1.0.0)

Earlier versions of this wrapper introduced a hierarchical construction $\phi \sim \mathrm{Gamma}(\alpha, \beta)$, $\alpha \sim \mathrm{Gamma}(1, 1)$, $\beta \sim \mathrm{Gamma}(1, 1)$ for the random-phi mode, declaring alpha and beta as Stan parameters with hyperpriors injected via stanvars. Starting v1.0.0, that construction has been removed in favour of brms's own default prior, $\phi \sim \mathrm{Gamma}(0.01, 0.01)$ with lower bound 0 (mean 1, variance 100; weakly informative on the precision scale). Three reasons:

- **Brittle priors on alpha/beta.** The hyperprior $\mathrm{Gamma}(1, 1)$ on α has prior mean 1, which is on the boundary of the parameter space declared as real<lower=1>. This routinely produced divergent transitions when the data were not informative about phi.
- **Parameter blow-up.** Estimating two extra Stan parameters per area-level model with limited data inflated the effective posterior dimension and slowed convergence.
- **Cleaner brms semantics.** Letting brms apply its own default $\mathrm{Gamma}(0.01, 0.01)$ means that passing prior = NULL now does exactly what the user expects: "brms defaults". No surprises.

If you need to reproduce the old behaviour, supply stanvars yourself to declare alpha, beta and their hyperpriors, and pass prior = set_prior("gamma(alpha,beta)", class = "phi"). Pre-v1.0.0 code that supplied stanvars with hyperpriors on alpha/beta will now raise an informative error.

Conflict policy

When the precision parameter ϕ is pinned via n + deff (or via fixed_params$phi), the function refuses any additional specification that would also set ϕ. Specifically, all of the following are rejected with an informative error at construction time:

- n supplied without deff, or vice versa.
- n + deff *and* fixed_params$phi.
- n + deff *and* a user prior on class = "phi".
- n + deff *and* a stanvars hyperprior on alpha or beta (the $\mathrm{Gamma}(\alpha, \beta)$ hyperparameters used in random-ϕ mode).
- auxiliary *and* the deprecated predictors in the same call.

References

Liu, B. (2009). *Hierarchical Bayes Estimation and Empirical Best Prediction of Small-Area Proportions*. University of Maryland.

Rao, J. N. K., & Molina, I. (2015). *Small Area Estimation*, 2nd ed. Wiley, p. 390.

See Also

hbm_flex, hbm

Examples

```
library(hbsaems)
library(brms)
data("data_betalogitnorm")
data <- data_betalogitnorm

# -- 1. Basic model (phi random, with default hyperprior) --------------------
model1 <- hbm_betalogitnorm(
  response   = "y",
  auxiliary  = c("x1", "x2", "x3"),
  area_var   = "regency",
  data       = data,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)
summary(model1)

# -- 2. Fixed phi via survey design (n + deff) -------------------------------
model2 <- hbm_betalogitnorm(
  response   = "y",
  auxiliary  = c("x1", "x2", "x3"),
  n          = "n",
  deff       = "deff",
  area_var   = "regency",
  data       = data,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)
summary(model2)

# -- 3. Custom prior on phi via the standard `prior` argument ----------------
#
# Starting v1.0.0, hbm_betalogitnorm() no longer declares its own
# alpha / beta hyperparameters; phi uses brms's native default
# Gamma(0.01, 0.01).  To override that prior, pass a `brms::set_prior()`
# entry to the `prior` argument -- the legacy
# stanvar("alpha ~ gamma(...); beta ~ gamma(...)") pattern would now
# error because alpha/beta are no longer Stan parameters.
model3 <- hbm_betalogitnorm(
  response   = "y",
  auxiliary  = c("x1", "x2", "x3"),
  area_var   = "regency",
  data       = data,
```

```
  prior      = brms::set_prior("gamma(2, 0.5)", class = "phi"),
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)

# -- 4. Spatial CAR model ------------------------------------------------
data("adjacency_matrix_car")
model4 <- hbm_betalogitnorm(
  response   = "y",
  auxiliary  = c("x1", "x2", "x3"),
  n          = "n",
  deff       = "deff",
  spatial_var = "province",
  spatial_model   = "car",
  M          = adjacency_matrix_car,
  data       = data,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)
```

---

hbm_binlogitnorm *Small Area Estimation under a Binomial Logit-Normal Model*

Description

Convenience wrapper that fits a Hierarchical Bayesian Small Area Estimation model with a **binomial** likelihood and logit link. Internally delegates to hbm_flex with family_key = "binomial".

Usage

```
hbm_binlogitnorm(
  response,
  trials,
  auxiliary = NULL,
  data,
  area_var = NULL,
  area_re_structure = c("nested", "crossed"),
  spatial_var = NULL,
  spatial_model = NULL,
  car_type = NULL,
  sar_type = NULL,
  M = NULL,
  fixed_params = NULL,
  predictors = NULL,
  group = NULL,
  sre = NULL,
  sre_type = NULL,
  ...
)
```

Arguments

response	Character. Name of the successes variable (non-negative integer, $y \leq n$).
trials	Character. Name of the trials variable (positive integer).
auxiliary	Character vector of auxiliary (fixed-effect) variable names; corresponds to area-level covariates in SAE literature (see Rao & Molina 2015 Ch. 4).
data	A data.frame.
area_var	Optional character vector. Name(s) of column(s) in data identifying the small area / domain. Length 1 fits an IID area-level random intercept (1 | area_var); length ≥ 2 supports hierarchical areas – see ?hbm_flex for the nested vs.\ crossed structures. Default: NULL.
area_re_structure	Either "nested" (default) or "crossed"; controls how multiple area columns combine.
spatial_var	Optional character. Name of a column identifying the spatial cluster. Must be supplied together with spatial_model and M. Default: NULL.
spatial_model	Optional character. Spatial dependence: "car" (default car_type = "icar") or "sar" (default sar_type = "lag"). Default: NULL.
car_type	Optional character. CAR sub-type passed to **brms**: "escar", "esicar", "icar", or "bym2".
sar_type	Optional character. SAR sub-type: "lag" or "error".
M	Optional numeric matrix. Spatial weight matrix. Required when spatial_model is supplied.
fixed_params	Optional named list pinning distributional parameters to known values. See hbm for the spec format.
predictors	**Deprecated.** Use auxiliary instead. Kept for backward compatibility; will be removed in v2.0.0.
group	**Deprecated.** Use area_var instead.
sre	**Deprecated.** Use spatial_var instead.
sre_type	**Deprecated.** Use spatial_model instead.
...	Additional arguments forwarded to hbm_flex (e.g.\ prior_type, handle_missing, sampler controls).

Details

Let y_i denote the number of successes in area i out of n_i trials. The model is

$$y_i \mid p_i, n_i \sim \text{Binomial}(n_i, p_i),$$

$$\text{logit}(p_i) = x_i^\top \boldsymbol{\beta} + u_i, \quad u_i \sim \mathcal{N}(0, \sigma_v^2).$$

Value

An object of class hbmfit.

Conflict policy

- `auxiliary` *and* the deprecated `predictors` in the same call are rejected with an informative error.
- `handle_missing = "model"` is not supported (binomial is a discrete family; see *Notes on missing data*).

Notes on missing data

The binomial family does not support `handle_missing = "model"` (joint Bayesian imputation via `brms::mi()`). When NA values are detected and `handle_missing` is left `NULL`, the wrapper auto-selects `"multiple"` (multiple imputation via **mice** on the predictors only).

See Also

`hbm_flex`, `hbm`

Examples

```
library(hbsaems)
library(brms)
data("data_binlogitnorm")

# -- 1. Standard binomial logit-normal SAE ---------------------------------
fit <- hbm_binlogitnorm(
  response   = "y",
  trials     = "n",
  auxiliary  = c("x1", "x2", "x3"),
  area_var   = "district",        # area-level random effect (1 | district)
  data       = data_binlogitnorm,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)

# -- 2. With spatial CAR random effect -------------------------------------
data("adjacency_matrix_car_regency")
fit_car <- hbm_binlogitnorm(
  response      = "y",
  trials        = "n",
  auxiliary     = c("x1", "x2", "x3"),
  spatial_var   = "regency",
  spatial_model = "car",
  M             = adjacency_matrix_car_regency,
  data          = data_binlogitnorm,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)
```

hbm_control *Sampler Configuration for HBSAE Models*

Description

Bundles the MCMC sampler arguments of hbm (and the hbm_* family wrappers) into a single named list. Use this when you want a reusable sampler profile or to reduce the size of long hbm() calls.

Usage

```
hbm_control(
  chains = 4L,
  iter = 4000L,
  warmup = NULL,
  thin = 1L,
  cores = 1L,
  seed = NULL,
  refresh = NULL,
  adapt_delta = NULL,
  max_treedepth = NULL,
  control = NULL
)
```

Arguments

chains	Integer. Number of Markov chains (default 4L).
iter	Integer. Total iterations per chain (default 4000L).
warmup	Integer. Warm-up iterations per chain. Default floor(iter / 2).
thin	Integer. Thinning interval (default 1L).
cores	Integer. Number of cores for parallel chains (default 1L).
seed	Optional integer seed for reproducibility.
refresh	Integer. Stan progress refresh frequency (default NULL: **brms** default).
adapt_delta	Numeric in $(0, 1)$. When supplied, included in the control list as control = list(adapt_delta = ...).
max_treedepth	Integer. Max tree depth, included in control when supplied.
control	Optional list of additional NUTS control options. Merged with any adapt_delta / max_treedepth above.

Details

This is entirely **opt-in**: the flat signatures of hbm(), hbm_lnln(), etc.\ continue to work exactly as before. Pass the result directly to hbm() – it is auto-spliced via ....

Value

A named list whose elements are valid arguments of hbm.

See Also

hbm_priors, hbm_nonlinear, hbm

Examples

```
# Build a reusable "high-quality" profile
hq <- hbm_control(chains = 4, iter = 8000, cores = 4,
                  adapt_delta = 0.99, seed = 1)
str(hq)

# Quick draft profile
draft <- hbm_control(chains = 2, iter = 1000)
```

---

hbm_data *Return the Data Used to Fit an hbmfit*

Description

Return the Data Used to Fit an hbmfit

Usage

```
hbm_data(model)
```

Arguments

model	An hbmfit object.

Value

The original data.frame passed to the fitting function.

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
head(hbm_data(model))
```

hbm_flex *Fit a Flexible HBSAE Model with Any Registered Family*

Description

Flexible factory that fits an HBSAE model using any distribution currently registered in the family registry, together with the full set of cross-cutting features: spatial random effects (CAR/SAR), shrinkage priors (horseshoe, R2D2), smooth terms (penalised splines, Gaussian processes), auxiliary-parameter hyperpriors, and missing-data strategies. Distribution-specific wrappers (hbm_lnln, hbm_binlogitnorm, hbm_betalogitnorm) are thin signature shims around this function with preset family_key values. Advanced users can also call it directly once a custom family has been registered via register_hbsae_model.

Usage

```
hbm_flex(
  family_key,
  response,
  auxiliary = NULL,
  data,
  addition_var = NULL,
  area_var = NULL,
  area_re_structure = c("nested", "crossed"),
  spatial_var = NULL,
  spatial_model = NULL,
  car_type = NULL,
  sar_type = NULL,
  M = NULL,
  prior = NULL,
  fixed_params = NULL,
  sampling_variance = NULL,
  prior_type = "default",
  hs_df = 1,
  hs_df_global = 1,
  hs_df_slab = 4,
  hs_scale_global = NULL,
  hs_scale_slab = 2,
  hs_par_ratio = NULL,
  hs_autoscale = TRUE,
  r2d2_mean_R2 = 0.5,
  r2d2_prec_R2 = 2,
  r2d2_cons_D2 = NULL,
  r2d2_autoscale = TRUE,
  nonlinear = NULL,
  nonlinear_type = "spline",
  spline_k = -1L,
  spline_bs = "tp",
  gp_k = NA_integer_,
```

```
  gp_cov = "exp_quad",
  gp_c = NULL,
  gp_scale = NULL,
  handle_missing = NULL,
  m = 5L,
  mice_args = list(),
  control = list(),
  chains = 4L,
  iter = 4000L,
  warmup = floor(iter/2),
  cores = 1L,
  sample_prior = "no",
  link = NULL,
  aux_args = NULL,
  stanvars = NULL,
  predictors = NULL,
  group = NULL,
  sre = NULL,
  sre_type = NULL,
  ...
)
```

Arguments

family_key	Character. The registry key of the desired family (e.g.\ "lognormal", "binomial", "gamma_log").
response	Character. Name of the response variable column.
auxiliary	Character vector. Names of auxiliary (fixed-effect) variables; corresponds to area-level covariates in SAE literature.
data	A data.frame.
addition_var	Character or NULL. Name of the addition term variable (e.g.\ trials for binomial). Required when the family spec has has_addition_term = TRUE.
area_var	Character vector or NULL. Name(s) of the column(s) in data identifying the areas. Three usage modes: • Length 1 (default behaviour): a single area-level random intercept (1 | area_var). • Length ≥ 2 with area_re_structure = "nested" (default): a hierarchy of areas given from the *highest* to the *lowest* level, e.g. c("province", "regency") yields (1 | province / regency) which brms expands to (1 | province) + (1 | province:regency). This is the canonical multi-stage SAE setup. • Length ≥ 2 with area_re_structure = "crossed": non-nested levels, e.g.\ adding separate effects for (1 | province) + (1 | urbanrural). Use only when the levels are truly crossed rather than hierarchically nested.
area_re_structure	Either "nested" (default) or "crossed". Only consulted when area_var has length ≥ 2. See above.

`spatial_var`, `spatial_model`, `car_type`, `sar_type`, `M`
: Spatial random-effect arguments forwarded to `hbm`. See `?hbm` for a full description.

`prior`, `prior_type`, `hs_df`, `hs_df_global`, `hs_df_slab`, `hs_scale_global`, `hs_scale_slab`, `hs_par_ratio`, `hs_autoscale`, `r2d2_mean_R2`, `r2d2_prec_R2`, `r2d2_cons_D2`, `r2d2_autoscale`
: Shrinkage prior arguments forwarded to `hbm`. When the formula contains `s()` or `gp()` terms, the prior is automatically cascaded to the corresponding parameter classes (`"sds"`, `"sdgp"`) via the brms `main = TRUE` pattern.

`fixed_params`
: Optional named list pinning distributional parameters to known values. See `hbm` for the spec format. Allows power-user access to the generic fixed-parameter machinery (works with custom families too).

`sampling_variance`
: Optional character. Name of a column in `data` containing the **known** sampling variance D_i for each area (the Fay-Herriot sugar). When supplied, $\sigma_i = \sqrt{D_i}$ is pinned via offset. Forwarded to `hbm`, where a family-compatibility check ensures the family exposes a residual SD parameter named `sigma` (gaussian, lognormal, student, skew_normal, exgaussian, asym_laplace). Incompatible families (beta, binomial, poisson, etc.) raise an explicit error pointing at the family-specific alternative. See `?hbm` for details.

`nonlinear`
: Character vector of variable names to include as smooth/nonlinear terms (rather than linear). Variables listed here that also appear in `auxiliary` are modelled nonlinearly only.

`nonlinear_type`
: Character. `"spline"` (penalised regression spline via **mgcv**, default) or `"gp"` (Gaussian process via **brms**).

`spline_k`
: Integer. Spline basis dimension passed to `mgcv::s(..., k = ...)`. `-1` (default) lets **mgcv** choose automatically. For SAE typically `k = 8` to `15`.

`spline_bs`
: Character. Spline basis type passed to `mgcv::s(..., bs = ...)`. Defaults to `"tp"` (thin-plate regression spline). Set `"cr"` (cubic regression) for better numerical stability with correlated auxiliary variables. Other choices: `"cs"` (cubic with shrinkage), `"ps"` (P-splines).

`gp_k`
: Integer or `NA`. Number of basis functions for the Hilbert-space approximate GP (Riutort-Mayol et al.\ 2020). `NA` (default) = exact GP, scales $O(n^3)$ and is **not recommended for** $n > 100$ areas. Integer `gp_k = 10–25` is typical for SAE applications and dramatically improves convergence and runtime.

`gp_cov`
: Character. Covariance function: `"exp_quad"` (squared exponential / RBF, default), `"matern15"` (Matern 3/2), `"matern25"` (Matern 5/2, often more numerically stable than RBF), `"exponential"`.

`gp_c`
: Numeric. Hilbert-space GP boundary-scale factor passed to `brms::gp(c = ...)`. Default brms value is 5/4 (= 1.25); increase if the GP appears truncated at domain boundaries. Only relevant when `gp_k` is set.

`gp_scale`
: **Deprecated.** Use `gp_c` instead. The old name suggested a length-scale interpretation but actually mapped to the boundary-scale factor.

`handle_missing`, `m`, `mice_args`
: Missing-data arguments. When `handle_missing = NULL` the wrapper auto-selects a strategy based on the family registry's `supports_mi` flag.

`control`, `chains`, `iter`, `warmup`, `cores`, `sample_prior`, `link`
: Sampler and model-spec arguments forwarded to `hbm`.

`aux_args`
: Optional named list of family-specific auxiliary arguments (e.g.\ `list(n = "n", deff = "deff")` for the Beta family's phi hyperprior). Forwarded to the family's `aux_param_hyperprior` callback if it has one.

`stanvars`
: Optional `stanvar` object passed through to **brms**. When the family has an `aux_param_hyperprior` callback that returns its own `stanvars`, the two are concatenated.

`predictors`
: **Deprecated.** Use `auxiliary` instead. Kept for backward compatibility; will be removed in v2.0.0.

`group`
: **Deprecated.** Use `area_var` instead.

`sre`
: **Deprecated.** Use `spatial_var` instead.

`sre_type`
: **Deprecated.** Use `spatial_model` instead.

`...`
: Additional arguments forwarded to `brm`.

Details

The factory performs five duties that were previously duplicated across wrappers:

1. Validate that `response`, `auxiliary`, and optional variables exist in `data`.
2. Run the family's `response_check` on the response and report a human-readable error on failure.
3. Auto-select a missing-data strategy that respects the family's `supports_mi` flag (e.g.\ binomial cannot use `"model"`).
4. Build the brms formula with optional addition terms and apply spline/GP transformations.
5. Invoke the family's `aux_param_hyperprior` callback (if defined) so distributions with a hyperprior on auxiliary parameters – e.g.\ phi for the Beta family, shape for Gamma, nu for Student-t – can inject Stan code without writing a thick wrapper file.

Value

An object of class `hbmfit`.

See Also

`register_hbsae_model`, `list_hbsae_models`, `hbm`

Examples

```
library(hbsaems)
data("data_lnln")
# Equivalent to hbm_lnln(...)
fit <- hbm_flex(
  family_key = "lognormal",
  response   = "y_obs",
  auxiliary  = c("x1", "x2", "x3"),
  area_var   = "district",         # area-level random effect: (1 | district)
```

```
  data       = data_lnln,
  chains = 4, iter = 2000, refresh = 0
)
```

---

hbm_info *Get Comprehensive Model Information*

Description

Returns a one-page summary of the fitted model's metadata: number of observations, family, link function, formula, MCMC settings, missing-data strategy, and so on.

Usage

```
hbm_info(model)
```

Arguments

model	An hbmfit object.

Value

A named list of metadata.

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
hbm_info(model)
```

hbm_lnln *Small Area Estimation under a Lognormal-Lognormal Model*

Description

Convenience wrapper that fits a Hierarchical Bayesian Small Area Estimation model with a **lognormal** likelihood. Internally delegates to hbm_flex with family_key = "lognormal".

Usage

```
hbm_lnln(
  response,
  auxiliary = NULL,
  data,
  area_var = NULL,
  area_re_structure = c("nested", "crossed"),
  spatial_var = NULL,
  spatial_model = NULL,
  car_type = NULL,
  sar_type = NULL,
  M = NULL,
  sampling_variance = NULL,
  fixed_params = NULL,
  predictors = NULL,
  sampling_var = NULL,
  group = NULL,
  sre = NULL,
  sre_type = NULL,
  ...
)
```

Arguments

response	Character. Name of the response column (must be strictly positive).
auxiliary	Character vector of auxiliary (fixed-effect) variable names; corresponds to area-level covariates in SAE literature (see Rao & Molina 2015 Ch. 4).
data	A data.frame.
area_var	Optional character vector. Name(s) of a column (or columns) in data identifying the small area / domain. Length 1 adds an IID area-level random intercept (1 | area_var); length ≥ 2 supports hierarchical areas – see ?hbm_flex for the nested vs.\ crossed structures. Default: NULL.
area_re_structure	Either "nested" (default) or "crossed". Controls how multiple area columns combine. See ?hbm_flex.
spatial_var	Optional character. Name of a column in data identifying the spatial cluster (e.g. province). Must be supplied together with spatial_model and M. Default: NULL.

`spatial_model` Optional character. Type of spatial dependence: `"car"` (conditional autoregressive) or `"sar"` (simultaneous autoregressive). Default: `NULL`.

`car_type` Optional character. CAR sub-type passed to **brms**: `"escar"`, `"esicar"`, `"icar"` (intrinsic CAR; default when `spatial_model = "car"`), or `"bym2"`.

`sar_type` Optional character. SAR sub-type: `"lag"` (spatial-lag, default when `spatial_model = "sar"`) or `"error"` (spatial-error).

`M` Optional numeric matrix. Spatial weight matrix. Required when `spatial_model` is supplied.

`sampling_variance` Optional character. Name of a column in `data` containing the known per-area sampling variance ψ_i on the **log scale**, i.e.\ the variance of $\log(\hat{y}_i)$. When supplied, $\sigma_i = \sqrt{\psi_i}$ is pinned via offset, recovering the Fay–Herriot lognormal model. If your survey software produces $\mathrm{Var}(\hat{y}_i)$ on the original scale, convert with the delta-method approximation $\psi_i \approx \mathrm{Var}(\hat{y}_i)/\hat{y}_i^2$ before passing. Default: `NULL` (sigma is random).

`fixed_params` Optional named list pinning distributional parameters to known values. See `hbm` for the spec format. Allows power-user access to the same machinery used by `sampling_variance`.

`predictors` **Deprecated.** Use `auxiliary` instead. Kept for backward compatibility; will be removed in v2.0.0.

`sampling_var` **Deprecated.** Use `sampling_variance` instead. Kept for backward compatibility; will be removed in v2.0.0.

`group` **Deprecated.** Use `area_var` instead.

`sre` **Deprecated.** Use `spatial_var` instead.

`sre_type` **Deprecated.** Use `spatial_model` instead.

`...` Additional arguments forwarded to `hbm_flex` (e.g.\ `prior_type`, `nonlinear`, `handle_missing`, sampler controls such as `chains`, `iter`, `cores`, `seed`).

Details

The response y_i in area i is assumed to follow

$$y_i \mid \theta_i \sim \mathrm{Lognormal}(\theta_i, \sigma^2),$$

with the log-mean linked to auxiliary variables via

$$\log(\theta_i) = x_i^\top \boldsymbol{\beta} + u_i, \quad u_i \sim \mathcal{N}(0, \sigma_v^2).$$

When the user supplies `sampling_variance = "psi_i"` (the column name of the known per-area sampling variance ψ_i), $\sigma_i = \sqrt{\psi_i}$ is pinned for each area via an offset. This recovers the Fay-Herriot-style lognormal model in which residual variability is fully determined by the survey design.

Value

An object of class `hbmfit`.

Conflict policy

When the residual standard deviation $\sigma_i = \sqrt{\psi_i}$ is pinned via `sampling_variance` (or via `fixed_params$sigma`), the function refuses any additional specification that would also set σ. Specifically, all of the following are rejected with an informative error at construction time:

- `sampling_variance` *and* `fixed_params$sigma`.
- `sampling_variance` *and* a user `prior` on `class = "sigma"`.
- `sampling_variance` *and* a `stanvars` sampling statement involving `sigma`.
- `auxiliary` *and* the deprecated `predictors` in the same call.

See Also

`hbm_flex`, `hbm`

Examples

```
library(hbsaems)
data("data_lnln")

# -- 1. Standard lognormal SAE with area random effect ----------------------
fit1 <- hbm_lnln(
  response   = "y_obs",
  auxiliary  = c("x1", "x2", "x3"),
  area_var   = "district",
  data       = data_lnln,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)

# -- 2. Fay-Herriot style with known sampling variance ----------------------
#     (assumes psi_i column is available)
fit2 <- hbm_lnln(
  response     = "y_obs",
  auxiliary    = c("x1", "x2", "x3"),
  area_var     = "district",
  sampling_variance = "psi_i",
  data         = data_lnln,
  chains = 4, iter = 2000, warmup = 1000, refresh = 0
)
```

---

hbm_nonlinear *Nonlinear-Term Configuration for HBSAE Models*

Description

Bundles the nonlinear-term arguments of `hbm` into a single named list. Same opt-in pattern as `hbm_control`.

Usage

```
hbm_nonlinear(
  terms,
  type = c("spline", "gp"),
  k = -1L,
  spline_bs = "tp",
  gp_cov = "exp_quad",
  gp_c = NULL,
  gp_scale = NULL
)
```

Arguments

terms	Character vector of predictor names to be wrapped in `s()` or `gp()`.
type	Character. `"spline"` (default) or `"gp"`.
k	Integer. Basis dimension. For splines: passed to `mgcv::s(..., k = ...)`; `-1L` (default) lets **mgcv** choose. For GP: passed to `brms::gp(..., k = ...)` for the Hilbert-space approximate GP (Riutort-Mayol et al.\ 2020); `NA` = exact GP (slow, not recommended for $n > 100$).
spline_bs	Character. Spline basis type (`"tp"`, `"cr"`, `"cs"`, `"ps"`, ...). Default `"tp"`.
gp_cov	Character. GP covariance function: `"exp_quad"`, `"matern15"`, `"matern25"`, `"exponential"`. Default `"exp_quad"`.
gp_c	Numeric or NULL. HSGP boundary-scale factor for `brms::gp(c = ...)`.
gp_scale	**Deprecated.** Use gp_c instead.

Value

A named list of arguments for hbm, with class `c("hbm_config_nonlinear", "hbm_config", "list")`.

See Also

hbm_control, hbm_priors, hbm

Examples

```
# Penalised regression spline (auto-chosen basis dimension):
nl_spline <- hbm_nonlinear(c("x1", "x3"), type = "spline")

# Spline with cubic-regression basis and fixed k:
nl_cr <- hbm_nonlinear("x1", type = "spline", k = 8, spline_bs = "cr")

# Hilbert-space approximate GP with Matern 5/2 covariance (recommended):
nl_gp <- hbm_nonlinear("x2", type = "gp", k = 20, gp_cov = "matern25")
```

hbm_priors *Prior Configuration for HBSAE Models*

Description

Bundles the shrinkage-prior arguments of hbm into a single named list. Same opt-in pattern as hbm_control.

Usage

```
hbm_priors(
  prior_type = c("default", "horseshoe", "r2d2"),
  prior = NULL,
  hs_df = 1,
  hs_df_global = 1,
  hs_df_slab = 4,
  hs_scale_global = NULL,
  hs_scale_slab = 2,
  hs_par_ratio = NULL,
  hs_autoscale = TRUE,
  r2d2_mean_R2 = 0.5,
  r2d2_prec_R2 = 2,
  r2d2_cons_D2 = NULL,
  r2d2_autoscale = TRUE
)
```

Arguments

prior_type	Character. One of "default", "horseshoe", "r2d2".
prior	Optional brmsprior for explicit priors that override the registry default.
hs_df, hs_df_global, hs_df_slab, hs_scale_global, hs_scale_slab, hs_par_ratio, hs_autoscale	Horseshoe-prior hyperparameters; see hbm.
r2d2_mean_R2, r2d2_prec_R2, r2d2_cons_D2, r2d2_autoscale	R2D2-prior hyperparameters; see hbm.

Value

A named list of arguments for hbm.

See Also

hbm_control, hbm_nonlinear, hbm

Examples

```
p_hs   <- hbm_priors(prior_type = "horseshoe", hs_df = 1, hs_df_slab = 4)
p_r2d2 <- hbm_priors(prior_type = "r2d2",      r2d2_mean_R2 = 0.5)
```

`hbm_warnings` *Get Model Warnings*

Description

Inspects the fitted model for common convergence problems and returns a character vector of human-readable warnings. When no warnings apply, the string `"No warnings detected."` is returned.

Usage

```
hbm_warnings(model)
```

Arguments

`model`	An `hbmfit` object.

Value

A character vector of warning messages.

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
hbm_warnings(model)
```

is-hbsaems *Test Whether an Object Belongs to an hbsaems Result Class*

Description

Lightweight type-checking predicates for all five result classes produced by **hbsaems**. Each predicate returns a single logical.

Usage

```
is.hbmfit(x)

is.hbcc_results(x)

is.hbmc_results(x)

is.hbpc_results(x)

is.hbsae_results(x)
```

Arguments

x	Any R object.

Value

A single logical value.

Examples

```
is.hbmfit("not a model")   # FALSE
```

is.hbsaems_check *Test Whether an Object Is an hbsaems Check Result*

Description

Predicate for the "hbsaems_check" base class. All pre-fit inspection functions in **hbsaems** (check_data, check_spatial_weight, check_shiny_deps) return an object that inherits from this class, enabling generic handling.

Usage

```
is.hbsaems_check(x)
```

Arguments

x	Any R object.

Value

A single logical.

See Also

check_data, check_spatial_weight, check_shiny_deps

Examples

```
chk <- check_data(data.frame(y = 1:5, x = 1:5),
                  response = "y", auxiliary = "x")
is.hbsaems_check(chk)              # TRUE
is.hbsaems_check("not a check")    # FALSE
```

---

is_converged *Test Whether a Fitted HBM Has Converged*

Description

Returns a single TRUE / FALSE based on the Gelman-Rubin statistic.

Usage

```
is_converged(model, threshold = 1.1, ...)
```

Arguments

model	An hbmfit or hbcc_results object.
threshold	$\widehat{R}$ threshold (default 1.1; use 1.05 for a stricter check, as recommended by Vehtari et al. 2021).
...	Currently unused.

Value

A single logical.

list_hbsae_models *List Registered HBSAE Models*

Description

Returns the keys of all model specs currently registered in the **hbsaems** model registry. Built-in models (`"gaussian"`, `"beta"`, `"binomial"`, `"lognormal"`, etc.) are always included; user-registered models appear in addition.

Usage

```
list_hbsae_models(verbose = FALSE)
```

Arguments

`verbose`	Logical. If `TRUE`, return a data.frame summarising each registered model: family name, default brms link function, whether the family is discrete, and whether brms-canonical `mi()` imputation is supported. Default `FALSE` returns a plain character vector of keys (backward compatible).

Value

Character vector of model keys, or a `data.frame` with columns `key`, `family`, `link`, `discrete`, `supports_mi` when `verbose = TRUE`.

See Also

`register_hbsae_model`, `hbm_flex`

Examples

```
list_hbsae_models()
list_hbsae_models(verbose = TRUE)
```

loglogistic *Loglogistic Distribution Functions*

Description

Density, distribution function, quantile function, and random generation for the loglogistic (Fisk) distribution with scale parameter `mu` > `0` and shape parameter `beta` > `0`.

Usage

```
dloglogistic(x, mu = 1, beta = 1, log = FALSE)

ploglogistic(q, mu = 1, beta = 1, lower.tail = TRUE, log.p = FALSE)

qloglogistic(p, mu = 1, beta = 1, lower.tail = TRUE, log.p = FALSE)

rloglogistic(n, mu = 1, beta = 1)
```

Arguments

x, q	Vector of quantiles (x > 0 for non-zero density).
mu	Scale parameter (mu > 0; equals the median).
beta	Shape parameter (beta > 0).
log, log.p	Logical; if TRUE, return the log density / log probability.
lower.tail	Logical; if TRUE (default) probabilities are $P[Y \leq q]$, otherwise $P[Y > q]$.
p	Vector of probabilities (0 <= p <= 1).
n	Number of random draws.

Value

Numeric vector of the same length as the input.

Parameterisation

This implementation follows the canonical Wikipedia / **flexsurv** / **eha** parameterisation (Jackson 2016; Bennett 1983):

$$Y \sim \text{LogLogistic}(\mu, \beta), \quad \mu > 0, \quad \beta > 0,$$

with probability density function

$$f(y \mid \mu, \beta) = \frac{(\beta/\mu)(y/\mu)^{\beta-1}}{\{1 + (y/\mu)^{\beta}\}^2}, \quad y > 0,$$

cumulative distribution function

$$F(y \mid \mu, \beta) = \{1 + (y/\mu)^{-\beta}\}^{-1},$$

median μ, and mean $E[Y] = \mu\pi/[\beta \sin(\pi/\beta)]$ when $\beta > 1$. Equivalently, $\log(Y) \sim \text{Logistic}(\log \mu,\ 1/\beta)$.

Why not match the brms `lognormal` convention? The `brms::lognormal()` family parameterises μ on the log scale (so μ is unconstrained and uses an identity link). Doing the same for the log-logistic would require redefining $\mu = \log(\text{median}(Y))$ – which deviates from every standard R reference (**flexsurv**, **eha**, Wolfram, scipy, Stata). We deliberately follow the survival-analysis convention instead: μ is the median (positive, log link), keeping interpretation simple and posterior summaries comparable with the rest of the R survival ecosystem.

References

Bennett, S. (1983). Log-logistic regression models for survival data. *Journal of the Royal Statistical Society, Series C*, 32(2), 165-171. doi:10.2307/2347295

Jackson, C. H. (2016). flexsurv: A platform for parametric survival modelling in R. *Journal of Statistical Software*, 70(8), 1-33. doi:10.18637/jss.v070.i08

Kleiber, C., & Kotz, S. (2003). *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley.

Examples

```
dloglogistic(c(0.5, 1, 2),  mu = 1, beta = 2)
ploglogistic(c(0.5, 1, 2),  mu = 1, beta = 2)
qloglogistic(c(0.25, 0.75), mu = 1, beta = 2)
set.seed(1); rloglogistic(5, mu = 1, beta = 2)
```

`model-compare` *Compare Fitted HBMs*

Description

Primary model-comparison functions in **hbsaems**. `model_compare` handles one or two models (supersedes deprecated `hbmc`); `model_compare_all` handles N models (analogous to `loo_compare`).

`model_average` *Bayesian Model Averaging on Small-Area Estimates*

Description

Averages the area-level predictions across multiple fitted HBMs. Weights may be supplied manually *or* computed automatically from leave-one-out cross-validation via `loo_model_weights` – the canonical Bayesian stacking / pseudo-BMA approach of Yao et al.\ (2018).

Usage

```
model_average(
  ...,
  weights = NULL,
  method = c("manual", "stacking", "pseudobma"),
  newdata = NULL
)
```

Arguments

...	Two or more hbmfit objects.
weights	Numeric weights of the same length as the number of models, or NULL. When NULL and method = "manual", equal weights are used.
method	Character. Weighting method: "manual" (default), "stacking" (Yao et al. 2018), or "pseudobma". When "stacking" or "pseudobma", weights must be NULL; an error is raised otherwise.
newdata	Optional new data.frame forwarded to sae_predict.

Details

Three weighting modes are supported:

- method = "manual" (default when weights is supplied): use the user-supplied weights vector directly. Internally normalised to sum to 1.
- method = "stacking": weights are obtained from loo::loo_model_weights(loo_list, method = "stacking"), which optimises a log-score over a simplex. Recommended when models are well-specified but capture different features of the data (Yao et al.\ 2018).
- method = "pseudobma": weights are obtained from loo::loo_model_weights(loo_list, method = "pseudobma"), a smoothed pseudo-BMA+ that resembles classical BMA but uses PSIS-LOO log scores. Use as a robust default when one model is much better than the others.

Internally calls sae_predict on each model and then sae_aggregate with method = "weighted".

Value

An hbsae_results object of averaged predictions. The computed weights are attached as an attribute "weights".

References

Yao, Y., Vehtari, A., Simpson, D., & Gelman, A. (2018). Using stacking to average Bayesian predictive distributions (with discussion). *Bayesian Analysis*, 13(3), 917–1007. doi:10.1214/17-BA1091

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5), 1413–1432.

Examples

```
# See ?model_compare_all for the full example fitting m1 / m2.
# Manual:    avg <- model_average(m1, m2, weights = c(0.6, 0.4))
# Stacking:  avg <- model_average(m1, m2, method = "stacking")
# Pseudo-BMA: avg <- model_average(m1, m2, method = "pseudobma")
```

`model_compare` *Compare One or Two Fitted HBMs*

Description

Computes LOO, WAIC, and posterior predictive check diagnostics. With a single model this gives goodness-of-fit metrics; with two models it adds a pairwise comparison.

Usage

```
model_compare(
  model,
  model2 = NULL,
  ndraws_ppc = 100,
  moment_match = FALSE,
  moment_match_args = list(),
  reloo_args = list(),
  plot_types = c("pp_check", "params"),
  comparison_metrics = c("loo", "waic", "bf"),
  run_prior_sensitivity = FALSE,
  sensitivity_vars = NULL,
  ...
)
```

Arguments

`model`	An `hbmfit` or `brmsfit` object.
`model2`	Optional second `hbmfit` for pairwise comparison.
`ndraws_ppc`	Number of draws for the posterior-predictive plot (default `100`).
`moment_match`	Logical; use moment matching for LOO (default `FALSE`).
`moment_match_args`	Named list of arguments for moment matching.
`reloo_args`	Named list of arguments for `reloo`.
`plot_types`	Character vector. Any subset of `c("pp_check", "params")`.
`comparison_metrics`	Character vector. Any subset of `c("loo", "waic", "bf")`.
`run_prior_sensitivity`	Logical; run prior sensitivity analysis using **priorsense** (default `FALSE`).
`sensitivity_vars`	Variables for the sensitivity analysis.
`...`	Additional arguments.

Value

An `hbmc_results` object with components `loo1`, `waic1`, `pp_check`, `params`, and – when `model2` is given – also `loo2`, `waic2`, `bf`, and `model2`.

See Also

model_compare_all, model_average

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
FAST <- list(chains = 4, iter = 2000, warmup = 1000, cores = 1,
             seed = 123, refresh = 0)

m1 <- do.call(hbm, c(list(formula = brms::bf(y ~ x1 + x2 + x3),
                          data = data_fhnorm), FAST))
m2 <- do.call(hbm, c(list(formula = brms::bf(y ~ x1 + x2),
                          data = data_fhnorm), FAST))

model_compare(m1)           # single-model goodness-of-fit
model_compare(m1, m2)       # pairwise comparison
```

---

model_compare_all *Compare Multiple Fitted HBMs*

Description

Ranks N models by LOO and/or WAIC, returning a sorted hbm_table. Analogous to loo_compare.

Usage

```
model_compare_all(..., criterion = c("loo", "waic", "both"))
```

Arguments

...	Named hbmfit objects.
criterion	One of "loo" (default), "waic", or "both".

Value

A hbm_table (a sorted data.frame) with columns Model, ELPD_LOO, LOO_SE, LOO_rank (and analogous *_WAIC* columns when requested).

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
FAST <- list(chains = 4, iter = 2000, warmup = 1000, cores = 1,
             seed = 1, refresh = 0)
m1 <- do.call(hbm, c(list(formula = brms::bf(y ~ x1),
                          data = data_fhnorm), FAST))
m2 <- do.call(hbm, c(list(formula = brms::bf(y ~ x1 + x2),
                          data = data_fhnorm), FAST))
model_compare_all(simple = m1, medium = m2)
```

`plot.hbmfit` *Plot a Fitted hbmfit Object*

Description

Wraps `brms::mcmc_plot()` and `brms::pp_check()` with named plot types.

Usage

```
## S3 method for class 'hbmfit'
plot(
  x,
  type = c("trace", "density", "acf", "nuts_energy", "rhat", "neff", "pp_check"),
  ...
)
```

Arguments

`x`	An `hbmfit` object.
`type`	Plot type, one of: `"trace"` (trace plots), `"density"` (posterior densities), `"acf"` (autocorrelation), `"nuts_energy"` (NUTS energy), `"rhat"` (R-hat distribution), `"neff"` (effective sample size), `"pp_check"` (posterior predictive check).
`...`	Additional arguments passed to the underlying **brms** plotting function.

Value

A `ggplot` or `bayesplot` object.

`posterior-methods` *Posterior and Prior Extraction Methods for hbmfit*

Description

Convenience wrappers around **brms** draw extractors.

posterior_draws *Extract Posterior Draws as a Matrix*

Description

Extract Posterior Draws as a Matrix

Usage

```
posterior_draws(model, params = NULL, ...)
```

Arguments

model	An hbmfit object.
params	Optional character vector of parameter names; NULL (default) returns all parameters.
...	Additional arguments forwarded to as_draws_matrix.

Value

A draws matrix (rows = MCMC iterations, columns = parameters).

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
draws <- posterior_draws(model)
dim(draws)
```

posterior_interval *Compute Credible Intervals for an hbmfit Object*

Description

The posterior_interval generic is re-exported from **rstantools** and an S3 method is provided that dispatches on hbmfit objects. This lets users call posterior_interval(fit) on the return value of hbm just as they would on a brmsfit.

Usage

```
## S3 method for class 'hbmfit'
posterior_interval(object, prob = 0.95, params = NULL, ...)
```

Arguments

`object`	An `hbmfit` object.
`prob`	Coverage probability in $(0, 1)$ (default `0.95`; note that `rstantools::posterior_interval`'s own default is `0.9`).
`params`	Optional character vector of parameter names to keep.
`...`	Additional arguments forwarded to `posterior_draws`.

Value

A matrix with two rows giving lower and upper bounds.

See Also

`posterior_interval`

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
posterior_interval(model, prob = 0.90)
```

`posterior_summary_hbm` *Comprehensive Posterior Summary*

Description

Returns fixed effects, random effects, and model-fit statistics in a single named list.

Usage

```
posterior_summary_hbm(model, probs = c(0.025, 0.975), ...)
```

Arguments

model	An hbmfit object.
probs	Probability bounds for credible intervals (default c(0.025, 0.975)).
...	Additional arguments forwarded to the underlying **brms** functions.

Value

A named list with components fixed_effects, random_effects, and model_fit (containing loo, waic, and R2).

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
s <- posterior_summary_hbm(model)
s$fixed_effects
```

---

prior_check *Prior Predictive Check for Fitted HBMs*

Description

Generates prior predictive samples from a model fit with sample_prior = "only" and compares them to the observed data. This is the primary prior-check function (supersedes the deprecated hbpc).

Usage

```
prior_check(model, data, response_var, ndraws_ppc = 50, ...)
```

Arguments

model	An hbmfit or brmsfit object fit with sample_prior = "only" (see hbm).
data	A data.frame containing the response variable.
response_var	Character scalar. Name of the response variable column.
ndraws_ppc	Integer. Number of prior predictive draws to overlay on the plot (default 50).
...	Currently unused; reserved for future extensions.

Details

The prior predictive distribution is

$$p(y_{\text{rep}}) = \int p(y_{\text{rep}} \mid \theta)\, p(\theta)\, \mathrm{d}\theta,$$

that is, the marginal distribution of new data y_{rep} under the prior alone. Comparing this to the observed data is a fast sanity check: if the prior predictive places no mass anywhere near the data, the priors are likely too tight or in the wrong location.

Value

An `hbpc_results` object with components:

`prior_predictive_plot` A ggplot from `pp_check`, or `NULL` if it could not be generated.

`prior_draws` A draws matrix from `posterior_predict` sized `ndraws_ppc \times nrow(data)`.

`observed` The observed response vector.

See Also

`hbm`, `convergence_check`

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
# `sample_prior = "only"` requires all coefficients to have a proper
# prior; supply explicit priors on the regression class.
model_prior <- hbm(
  formula      = brms::bf(y ~ x1 + x2 + x3),
  data         = data_fhnorm,
  sample_prior = "only",
  prior        = c(
    brms::prior(normal(0, 1), class = "b"),
    brms::prior(normal(0, 5), class = "Intercept")
  ),
  chains = 4, iter = 2000, warmup = 1000, cores = 1,
  seed = 42, refresh = 0
)
pc <- prior_check(model_prior,
                  data         = data_fhnorm,
                  response_var = "y")
print(pc)
plot(pc)
```

prior_draws *Extract Prior Draws*

Description

The prior_draws generic is re-exported from **brms** and an S3 method is provided that dispatches on hbmfit objects. Requires the model to have been fit with sample_prior = "yes" or sample_prior = "only".

Usage

```
## S3 method for class 'hbmfit'
prior_draws(x, ...)
```

Arguments

x	An hbmfit object.
...	Additional arguments forwarded to prior_draws.

Value

A data.frame of prior draws or NULL if no prior samples were stored in the model.

See Also

prior_draws

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
# `sample_prior = "yes"` works best when all coefficients have a
# proper prior; supply explicit priors on the regression class.
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             sample_prior = "yes",
             prior        = c(
               brms::prior(normal(0, 1), class = "b"),
               brms::prior(normal(0, 5), class = "Intercept")
             ),
             chains = 4, iter = 2000, warmup = 1000,
             cores = 1, seed = 1, refresh = 0)
pd <- prior_draws(model)
head(pd)
```

`prior_sensitivity` *Power-Scale Prior Sensitivity Diagnostics for Fitted HBMs*

Description

Computes prior and likelihood power-scaling sensitivity diagnostics for a fitted `hbmfit` model using the **priorsense** package. Useful for assessing whether posterior conclusions are driven by the prior or the data – a critical step in any principled Bayesian SAE workflow.

Usage

```
prior_sensitivity(model, ...)
```

Arguments

`model`	An `hbmfit` object returned by `hbm` (or one of its wrappers) or a `brmsfit` object directly.
`...`	Additional arguments forwarded to `priorsense::powerscale_sensitivity()`, e.g.\ `variable = c("b_x1", "sd_regency__Intercept")` to restrict the report to specific parameters.

Details

Prior sensitivity analysis answers the question: "If I had used a slightly different prior, would the substantive conclusions change?". The power-scaling approach of Kallioinen et al.\ (2023) detects:

- **Prior–likelihood conflict**: the posterior moves non-negligibly when the prior is up- or down-weighted. Often indicates an overly informative or misspecified prior.
- **Weak likelihood**: the posterior is dominated by the prior. Common in SAE for areas with few sampled units.

Reported diagnostics include the Kullback–Leibler divergence between the original posterior and the power-scaled posterior (`prior`, `likelihood`) and a categorical flag (`prior-data conflict`, `strong prior`, `-`).

Computational cost. No re-sampling is required: importance sampling reuses the existing posterior draws. Hence a typical run costs only a few seconds even for large hierarchical models.

Value

A `powerscale_sensitivity_summary` object (data frame) with one row per monitored parameter and columns `variable`, `prior`, `likelihood`, `diagnosis`. `NULL` (with a message) when the **priorsense** package is not installed.

When to run prior sensitivity

Always. Specifically:

- After every model fit, before drawing substantive conclusions.
- Whenever convergence diagnostics from `convergence_check()` are clean but the posterior seems implausibly narrow or implausibly wide.
- When comparing models with shrinkage priors – horseshoe and R2D2 are both informative, and small differences in their hyperparameters can move estimates noticeably.

References

Kallioinen, N., Paananen, T., Burkner, P.-C., & Vehtari, A.\ (2024). Detecting and diagnosing prior and likelihood sensitivity with power-scaling. *Statistics and Computing*, 34, 57. doi:10.1007/s11222023103665

See Also

`prior_check`, `convergence_check`, `model_compare`

Examples

```
if (requireNamespace("priorsense", quietly = TRUE)) {
  data("data_fhnorm")
  fit <- hbm(brms::bf(y ~ x1 + x2),
             data = data_fhnorm, re = ~(1 | regency),
             chains = 4, iter = 2000, refresh = 0)
  ps  <- prior_sensitivity(fit)
  print(ps)
}
```

---

read_stan_function *Read the Stan Function Code for a Custom Distribution*

Description

Loads the contents of `inst/stan/<name>.stan` from the installed **hbsaems** (or from the local source tree when running tests). This is a thin wrapper around `readLines()` that resolves the package path once and validates that the file exists.

Usage

```
read_stan_function(name)
```

Arguments

`name`	Character. The distribution name – must match a `<name>.stan` file shipped under `inst/stan/`.

Value

A character scalar containing the Stan code (functions block).

See Also

build_brms_custom_family, register_hbsae_brms_custom.

Examples

```
code <- read_stan_function("hbsae_loglogistic")
cat(code)
```

`register_hbsae_brms_custom` *Register a brms Custom Family with the hbsaems Model Registry*

Description

Wraps a `brms::custom_family` + `stanvars` pair into a model spec usable by `hbm` and the flexible factory `hbm_flex`. The custom likelihood is then available throughout the package – in `run_sae_app`, in `sae_predict`, in the Shiny application, and so on – as if it were a built-in family.

Usage

```
register_hbsae_brms_custom(
  key,
  custom_family,
  stanvars,
  response_check = NULL,
  response_check_msg = NULL,
  supports_mi = FALSE,
  discrete = FALSE,
  overwrite = FALSE
)
```

Arguments

`key`	Character. Unique registry key (e.g.\ `"loglogistic"`).
`custom_family`	A `brms::custom_family` object describing the parameters, links, and constraints of the distribution. Typically produced by `brms::custom_family()` or by a helper such as `brms_custom_loglogistic`.
`stanvars`	A `brms::stanvars` object containing the Stan-side function definitions (log-PDF / log-CDF / RNG).
`response_check`	Optional function `function(y) -> logical` for response-domain validation (e.g.\ positivity check for the loglogistic).

`response_check_msg`
: Character. Error message if `response_check` fails.

`supports_mi`
: Logical. Whether `brms::mi()` can impute the response under this likelihood (default `FALSE`; custom families typically do not support `mi()`).

`discrete`
: Logical. Whether the family is discrete (default `FALSE`).

`overwrite`
: Logical. If `TRUE`, replace an existing entry with the same `key`.

Details

After registration, the family is usable in any of the following ways:

```
# Direct via the registry key
fit <- hbm_flex("loglogistic", response = "y",
                auxiliary = c("x1", "x2"),
                data = d, area_var = "area")

# Direct via hbm() (canonical):
fit <- hbm(brms::bf(y ~ x1 + x2 + (1 | area)),
           data = d,
           hb_sampling = "loglogistic")
```

Internally, `hbm` detects that the registered family is a `brms::custom_family` and (i) passes the family object directly to `brms::brm()` instead of constructing a built-in `brms::brmsfamily()`, and (ii) merges the family's stanvars with any user-supplied `stanvars`.

Value

Invisibly returns the registered model spec (a list).

See Also

`register_hbsae_model`, `brms_custom_loglogistic`, `brms_custom_shifted_loglogistic`, brms vignette on custom families.

Examples

```
library(hbsaems)
library(brms)

# Loglogistic (built in to hbsaems 1.0.0; this just shows the
# registration mechanism).  We use a key with a "_user" suffix to
# avoid shadowing the built-in "loglogistic" entry, and unregister
# at the end so re-running this example block keeps the registry
# clean.
ll <- brms_custom_loglogistic()
if ("loglogistic_user" %in% list_hbsae_models()) {
  # Idempotent rerun: remove any previous registration first.
  rm("loglogistic_user", envir = hbsaems:::.hbsae_model_env)
}
register_hbsae_brms_custom(
```

```
  key             = "loglogistic_user",
  custom_family   = ll$custom_family,
  stanvars        = ll$stanvars_family,
  response_check  = function(y) all(y > 0, na.rm = TRUE),
  response_check_msg = "Loglogistic response must be positive."
)
"loglogistic_user" %in% list_hbsae_models()

# Cleanup so the registry is unchanged after the example runs.
rm("loglogistic_user", envir = hbsaems:::.hbsae_model_env)
```

`register_hbsae_model` *Register a Custom HBSAE Model*

Description

Adds a new model spec to the **hbsaems** model registry so that it can be used by `hbm` and the flexible factory `hbm_flex`. Useful for extending the package with new likelihoods (e.g.\ *Gamma*, *Tweedie*, *Skew-Normal*), new link functions, or new auxiliary-parameter hyperpriors without modifying **hbsaems** source.

Usage

```
register_hbsae_model(
  key,
  family,
  link = "identity",
  discrete = FALSE,
  supports_mi = !discrete,
  has_addition_term = FALSE,
  addition_template = NULL,
  response_check = function(y) TRUE,
  response_check_msg = NULL,
  default_priors = function(...) NULL,
  aux_param_hyperprior = NULL,
  overwrite = FALSE
)
```

Arguments

`key`	Character. Unique identifier (e.g.\ `"gamma_log"`).
`family`	Character. The **brms** family name passed to `hb_sampling`.
`link`	Character. Default link function (default `"identity"`).
`discrete`	Logical. Is the response discrete? Affects whether `handle_missing = "model"` (joint Bayesian imputation via `brms::mi()`) is allowed (default `FALSE`).

`supports_mi` Logical. Whether `brms::mi()` can impute the response variable for this family (default `!discrete`).

`has_addition_term` Logical. Whether the LHS uses an addition term such as `|\ trials(n)` (default `FALSE`).

`addition_template` Character. An `sprintf` template used when `has_addition_term = TRUE`. Must contain three `%s` slots for response, addition variable, and RHS. Example: `"%s | trials(%s) ~ %s"`.

`response_check` Function `function(y)` returning `TRUE` when the response domain is valid, `FALSE` otherwise (default: accept anything).

`response_check_msg` Character. Error message displayed when `response_check(y)` returns `FALSE`.

`default_priors` Function `function(...)` returning a `brmsprior` object, or `NULL` to use **brms** defaults.

`aux_param_hyperprior` Optional function `function(args, data)` returning a list with components `prior` (a `brmsprior`) and `stanvars` (a `stanvar` object). Used by distributions that have an auxiliary parameter (e.g.\ ϕ for Beta, *shape* for Gamma) requiring a hyperprior expressed in raw Stan code. The `args` list contains family-specific user inputs forwarded through `aux_args` in `hbm_flex`. Return `NULL` to skip injection for a given call.

`overwrite` Logical. Permit overwriting an existing key (default `FALSE`).

Details

After registering, you can fit a model directly with `hbm_flex`:

```
register_hbsae_model(
  key            = "gamma_log",
  family         = "Gamma",
  link           = "log",
  discrete       = FALSE,
  supports_mi    = TRUE,
  response_check = function(y) all(y > 0, na.rm = TRUE),
  response_check_msg = "Gamma response must be strictly positive."
)

fit <- hbm_flex(
  family_key = "gamma_log",
  response   = "expenditure",
  auxiliary  = c("x1", "x2"),
  data       = my_data
)
```

Value

Invisibly returns the registered model spec (a named list).

See Also

hbm_flex, list_hbsae_models

---

run_sae_app *run_sae_app: Interactive Small Area Estimation Application*

Description

Opens the interactive HBSAE application in the default web browser. The application provides a graphical interface for data upload, exploratory analysis, model specification, fitting, and result visualisation, covering all modelling functions in **hbsaems**.

Usage

```
run_sae_app(check_deps = TRUE)
```

Arguments

check_deps	Logical. Whether to verify dependencies before launching (default TRUE).

Details

Launch the HBSAE Shiny Application

The application is located in `inst/shiny/sae_app/app.R` within the installed package. It depends on several packages that are listed in `Suggests` (rather than `Imports`) so they are not required for users who only need the modelling functions.

Two classes of dependencies are checked at launch:

Critical Packages without which the app cannot start (**shinydashboard**, **DT**). Missing critical packages raise an error.

Optional Packages that enable individual panels and features (**shinyWidgets**, **readxl**, **energy**, **minerva**, **sf**, **spdep**, **bridgesampling**). Missing optional packages produce a warning and an in-app banner; the corresponding feature degrades gracefully.

Use `check_shiny_deps()` to inspect dependency status without launching the app.

Value

Does not return a value; called for its side effect of launching a Shiny server in the current R session.

See Also

check_shiny_deps

Examples

```
if (interactive()) {
  run_sae_app()
}
```

---

sae_aggregate *Aggregate Predictions from Multiple hbsae_results*

Description

Combines area-level predictions across multiple hbsae_results objects. All objects must report predictions for the same number of areas (in the same order).

Usage

```
sae_aggregate(..., method = c("mean", "median", "weighted"), weights = NULL)
```

Arguments

...	Two or more hbsae_results objects.
method	One of "mean" (default), "median", or "weighted".
weights	Numeric vector of weights, required when method = "weighted". Internally normalised to sum to 1.

Value

An hbsae_results object containing the combined predictions.

Examples

```
p1  <- structure(list(result_table = data.frame(Prediction = 1:3,
                                                RSE_percent = c(5, 5, 5)),
                      rse_model = 5, pred = 1:3),
                 class = "hbsae_results")
p2  <- structure(list(result_table = data.frame(Prediction = 2:4,
                                                RSE_percent = c(4, 4, 4)),
                      rse_model = 4, pred = 2:4),
                 class = "hbsae_results")
sae_aggregate(p1, p2, method = "mean")
sae_aggregate(p1, p2, method = "weighted", weights = c(0.6, 0.4))
```

sae_benchmark *Benchmark Small-Area Estimates to Known Totals*

Description

Adjusts area-level predictions so that their weighted sum matches one or more known aggregate "benchmark" totals (e.g.\ official provincial or national figures). Two modes are supported:

Usage

```
sae_benchmark(
  predictions,
  target,
  weights = NULL,
  target_type = c("total", "mean"),
  method = c("ratio", "difference", "raking"),
  groups = NULL,
  posterior = NULL,
  probs = c(0.025, 0.5, 0.975),
  max_iter = 100L,
  tol = 1e-08
)
```

Arguments

predictions	An hbsae_results object produced by sae_predict.
target	Numeric. For method = "ratio" or "difference": a single benchmark total T. For method = "raking": a numeric vector of group totals (one per group; see groups).
weights	Optional numeric vector of length equal to the number of areas in predictions. In Official-Statistics practice this is normally the population size N_i of each area, so that $\sum_i w_i \hat{\theta}_i$ is the implied population total. When NULL, a safe default is chosen based on target_type; an informational message is emitted so users can see exactly which weighting was applied.
target_type	Character. Either "total" (default) or "mean". Consulted ONLY when weights = NULL to choose a safe default: "total" sets weights = rep(1, n) so that $\sum_i w_i \hat{\theta}_i = \sum_i \hat{\theta}_i$; "mean" sets weights = rep(1/n, n) so that the weighted sum equals the mean. Ignored when weights is supplied. **Always prefer to pass explicit** weights **(typically the population size N_i) in production code:** the default heuristic is provided only to avoid silent scale corruption when the user forgets the argument.
method	Character. One of: "ratio" Multiplicative: $\hat{\theta}_i^B = \hat{\theta}_i \times T / \sum_j w_j \hat{\theta}_j$. "difference" Additive: $\hat{\theta}_i^B = \hat{\theta}_i + (T - \sum_j w_j \hat{\theta}_j) / \sum_j w_j$. "raking" Iterative proportional fitting to multiple group totals. Requires groups.

`groups` Optional integer/character vector of length equal to the number of areas, assigning each area to a benchmarking group. Required for `method = "raking"`; ignored otherwise.

`posterior` Optional logical or matrix. Controls Bayesian mode:

`NULL` **or** `FALSE` **(default)** Point-estimate mode – same behaviour as v1.0.0.

`TRUE` Bayesian mode. Posterior draws are extracted from `predictions` automatically (requires `predictions$model` to be available).

a numeric *matrix* **(**$D \times n$**)** Bayesian mode with user-supplied draws (D draws, n areas).

`probs` Numeric vector of quantile probabilities to summarise the adjusted posterior with (default `c(0.025, 0.5, 0.975)`). Only used in Bayesian mode.

`max_iter` Integer. Maximum iterations for raking (default `100L`).

`tol` Numeric. Convergence tolerance for raking (default `1e-8`).

Details

Point-estimate mode (default) Applies the adjustment only to the posterior mean. The RSE column is left unchanged as a working approximation.

Fully Bayesian mode (`posterior = TRUE` **or supply a** `posterior` **matrix)** Applies the adjustment to every posterior draw and recomputes SD, quantiles, and RSE from the adjusted draws. This is the statistically correct procedure and produces proper uncertainty intervals after benchmarking.

When to benchmark: Benchmarking is widely used in official statistics to ensure consistency between model-based small-area estimates and published aggregate totals from a more reliable source.

Why fully Bayesian benchmarking matters: Applying a deterministic adjustment factor to a posterior point estimate distorts the uncertainty structure: a multiplicative factor scales both the mean and the SD by the same amount (so the RSE percentage is preserved), but an additive shift does *not* change the SD, so the RSE percentage shrinks at small areas and grows at large ones. Neither of these post-hoc fixes is guaranteed to be correct in general. In Bayesian mode every draw is benchmarked independently, so the resulting posterior carries the right uncertainty.

Value

An `hbsae_results` object with benchmarked `Prediction` values, plus an additional element `$benchmark_info` that records `method`, `target`, `weights`, the implied `adjustment` factor, and `converged` (logical, raking only). In Bayesian mode the `result_table` also contains updated `SD` and `RSE_percent` columns reflecting the post-benchmark posterior uncertainty, plus quantile columns named after `probs`.

References

Pfeffermann, D. (2013). New important developments in small area estimation. *Statistical Science* 28(1), 40–68.

Wang, J., Fuller, W. A., & Qu, Y. (2008). Small area estimation under a restriction. *Survey Methodology* 34, 29–36.

See Also

sae_predict, sae_aggregate, model_average

Examples

```
# Synthetic predictions (point-estimate mode)
p <- structure(
  list(
    result_table = data.frame(Prediction  = c(10, 12, 9, 11),
                              SD          = c(1, 1, 1, 1),
                              RSE_percent = c(10, 8, 11, 9)),
    rse_model    = 9.5,
    pred         = c(10, 12, 9, 11)
  ),
  class = "hbsae_results"
)
bm1 <- sae_benchmark(p, target = 50, method = "ratio")

# Fully Bayesian mode with user-supplied draws
set.seed(1)
D <- 1000
draws <- matrix(rnorm(D * 4, mean = c(10, 12, 9, 11), sd = 1),
                nrow = D, byrow = TRUE)
bm2 <- sae_benchmark(p, target = 50, method = "ratio",
                     posterior = draws)
bm2$result_table              # SD, RSE updated from draws
```

---

sae_filter *Filter SAE Predictions by a Logical Condition*

Description

Filter SAE Predictions by a Logical Condition

Usage

```
sae_filter(x, condition)
```

Arguments

x	An hbsae_results object.
condition	Logical vector of length equal to the number of areas.

Value

A new hbsae_results object containing only rows where condition is TRUE.

Examples

```
p <- structure(list(result_table = data.frame(Prediction = 1:5,
                                              RSE_percent = rep(5, 5)),
                    rse_model = 5, pred = 1:5),
               class = "hbsae_results")
sae_filter(p, p$pred > 2)
```

`sae_predict` *Generate Small Area Estimates*

Description

Primary SAE prediction function in **hbsaems** (supersedes deprecated `hbsae`). Computes area-level posterior predictive means, standard deviations, and relative standard errors (RSE).

Usage

```
sae_predict(model, newdata = NULL, ...)
```

Arguments

`model`	An `hbmfit` or `brmsfit` object.
`newdata`	Optional new `data.frame` for prediction at unsampled areas. If `NULL` (default), the original data are used.
`...`	Additional arguments forwarded to `posterior_predict` (e.g.\ `ndraws`, `re_formula`).

Details

For each area $i = 1, \ldots, n$, the function computes

$$\widehat{y}_i = \frac{1}{S}\sum_{s=1}^{S} y_i^{(s)}, \qquad \widehat{\mathrm{sd}}_i^{\,2} = \frac{1}{S-1}\sum_{s=1}^{S}\left(y_i^{(s)} - \widehat{y}_i\right)^2,$$

where $y_i^{(s)}$ are draws from the posterior predictive distribution and S is the number of draws. The relative standard error is $\mathrm{RSE}_i = 100 \cdot |\widehat{\mathrm{sd}}_i / \widehat{y}_i|$.

Value

An `hbsae_results` object with components:

`result_table` A `data.frame` with columns `Prediction`, `SD`, `RSE_percent`.

`rse_model` Mean of `RSE_percent` across all areas.

`pred` Numeric vector of point predictions (= `result_table$Prediction`).

See Also

sae_aggregate, model_average, sae_transform, sae_scale, sae_filter

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(
  formula = brms::bf(y ~ x1 + x2 + x3),
  data    = data_fhnorm,
  chains = 4, iter = 2000, warmup = 1000, cores = 1,
  seed = 123, refresh = 0
)
est <- sae_predict(model)
summary(est)
plot(est, type = "predictions")
plot(est, type = "uncertainty")
```

---

sae_scale *Standardise SAE Predictions*

Description

Standardise SAE Predictions

Usage

```
sae_scale(x, center = TRUE, scale = TRUE)
```

Arguments

x	An hbsae_results object.
center	Logical or numeric centering (passed to base::scale).
scale	Logical or numeric scaling (passed to base::scale).

Value

A new hbsae_results object with standardised predictions.

Examples

```
p <- structure(list(result_table = data.frame(Prediction = 1:5,
                                              RSE_percent = rep(5, 5)),
                    rse_model = 5, pred = 1:5),
               class = "hbsae_results")
sae_scale(p)
```

sae_transform *Apply a Transformation to SAE Predictions*

Description

Apply a Transformation to SAE Predictions

Usage

```
sae_transform(x, fun, ...)
```

Arguments

x	An hbsae_results object.
fun	A function applied element-wise to the predictions.
...	Additional arguments passed to fun.

Value

A new hbsae_results object.

Examples

```
p <- structure(list(result_table = data.frame(Prediction = c(2, 4, 8),
                                              RSE_percent = c(5, 5, 5)),
                    rse_model = 5, pred = c(2, 4, 8)),
               class = "hbsae_results")
sae_transform(p, log)
```

shifted_loglogistic *Shifted (3-Parameter) Loglogistic Distribution*

Description

Density, distribution function, quantile function and random generation for the shifted log-logistic (generalised log-logistic) distribution with location mu (real), scale sigma > 0 and shape xi (real). The two-parameter logistic distribution is recovered as xi -> 0.

Usage

```
dshifted_loglogistic(x, mu = 0, sigma = 1, xi = 0, log = FALSE)

pshifted_loglogistic(
  q,
  mu = 0,
  sigma = 1,
  xi = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

qshifted_loglogistic(
  p,
  mu = 0,
  sigma = 1,
  xi = 0,
  lower.tail = TRUE,
  log.p = FALSE
)

rshifted_loglogistic(n, mu = 0, sigma = 1, xi = 0)
```

Arguments

x, q	Numeric vector of quantiles.
mu	Location parameter (real; equals the median).
sigma	Scale parameter (sigma > 0).
xi	Shape parameter (real; xi = 0 gives the logistic distribution).
log, log.p	Logical. See Distributions.
lower.tail	Logical. See Distributions.
p	Vector of probabilities.
n	Number of random draws.

Value

Numeric vector.

Parameterisation

This implementation uses the **GEV-style parameterisation** of Hosking & Wallis (1997) and the Flood Estimation Handbook (Robson & Reed 1999), in which μ is a pure location parameter (the median), σ a pure scale parameter and ξ a pure shape parameter:

$$F(x \mid \mu, \sigma, \xi) = \{1 + (1 + \xi z)^{-1/\xi}\}^{-1}, \qquad z = (x - \mu)/\sigma,$$

with corresponding density

$$f(x \mid \mu, \sigma, \xi) = \frac{(1 + \xi z)^{-(1/\xi+1)}}{\sigma\{1 + (1 + \xi z)^{-1/\xi}\}^2}.$$

The support depends on ξ:

- $\xi > 0$: $x \geq \mu - \sigma/\xi$ (bounded below).
- $\xi < 0$: $x \leq \mu - \sigma/\xi$ (bounded above).
- $\xi = 0$: $x \in \mathbb{R}$ (logistic limit).

The median is always μ; the mean exists when $|\xi| < 1$ and is $\mu + \sigma(\alpha \csc \alpha - 1)/\xi$, $\alpha = \pi\xi$. Reducing further, the family contains:

- the standard log-logistic when $\xi = 1$ (reparameterised);
- the logistic distribution as $\xi \to 0$;
- the generalised Pareto family at $\xi = -1$.

Why this parameterisation? An alternative "simple-shift" form, $Y - \delta \sim \mathrm{LogLogistic}$, exists in the literature (Geskus 2001) and is closer in spirit to `brms::shifted_lognormal()`'s positive shift `ndt`. We deliberately follow the GEV-style parameterisation because

1. it provides a *smooth* limit to the logistic distribution at $\xi = 0$;
2. the parameters (μ, σ, ξ) are orthogonally interpretable (location / scale / shape);
3. it is the canonical form in hydrology and extreme-value applications (Hosking & Wallis 1997).

References

Geskus, R. B. (2001). Methods for estimating the AIDS incubation time distribution when date of seroconversion is censored. *Statistics in Medicine*, 20(5), 795-812.

Hosking, J. R. M., & Wallis, J. R. (1997). *Regional Frequency Analysis: An Approach Based on L-Moments*. Cambridge University Press. ISBN 0-521-43045-3.

Robson, A., & Reed, D. (1999). Flood Estimation Handbook, Volume 3: Statistical Procedures for Flood Frequency Estimation. Institute of Hydrology, Wallingford, UK.

Examples

```
dshifted_loglogistic(c(1, 2, 5),    mu = 0, sigma = 1, xi = 0.5)
pshifted_loglogistic(c(1, 2, 5),    mu = 0, sigma = 1, xi = 0.5)
qshifted_loglogistic(c(0.25, 0.75), mu = 0, sigma = 1, xi = 0.5)
set.seed(1); rshifted_loglogistic(5, mu = 0, sigma = 1, xi = 0.5)
```

spatial_weight_sar *Spatial Weight Matrix for Simultaneous Autoregressive Models*

Description

A row-standardised spatial weight matrix for 100 regencies, used for fitting Simultaneous Autoregressive (SAR) random effects. Pairs with `data_fhnorm`'s `regency` column.

Usage

```
spatial_weight_sar
```

Format

A 100×100 numeric matrix with row- and column-names `regency_001` .. `regency_100` and row sums equal to one (when at least one neighbour is present).

Source

Simulated.

summary.hbsaems_check *Generic Summary Method for hbsaems Check Results*

Description

Provides a fall-back summary that simply calls the underlying `print()` method. Subclasses that need a more detailed summary (e.g.\ `summary.hbsaems_data_check`) override this.

Usage

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

Arguments

`object`	An object inheriting from `"hbsaems_check"`.
`...`	Unused.

Value

The input `object`, invisibly.

tr — *Translate a UI String for the Shiny SAE App*

Description

Looks up a translation key in the hbsaems translation dictionary. When the requested language does not contain the key, it falls back to English; when English also lacks the key, it returns the key itself wrapped in brackets so missing strings stand out during development.

Usage

```
tr(key, lang = "en")
```

Arguments

key	Character. Translation key (e.g. "menu_home").
lang	Character. Language code; currently "en" or "id" (default "en").

Value

A character scalar with the translated UI string.

See Also

tr_langs, tr_keys

Examples

```
tr("menu_home", "en")    # "Home"
tr("menu_home", "id")    # "Beranda"
tr("nonexistent", "id")  # "[nonexistent]"
```

tr_keys — *List All Translation Keys (for a Reference Language)*

Description

Useful when adding a new language: enumerates every key that needs a translation.

Usage

```
tr_keys(lang = "en")
```

Arguments

lang	Character. Reference language (default "en").

Value

Sorted character vector of translation keys.

Examples

```
head(tr_keys())
```

`tr_langs` *List Available Languages*

Description

List Available Languages

Usage

```
tr_langs()
```

Value

Character vector of supported language codes.

Examples

```
tr_langs()
```

`update_hbm` *Update a Fitted HBM*

Description

Refits an `hbmfit` model with one or more arguments changed. Useful for re-running with longer chains, more iterations, or new data without retyping the full `hbm` call.

Usage

```
update_hbm(
  object,
  newdata = NULL,
  formula. = NULL,
  iter = NULL,
  warmup = NULL,
  chains = NULL,
  cores = NULL,
  control = NULL,
  ...
)
```

Arguments

object	An hbmfit object.
newdata	Optional replacement data.frame.
formula.	Optional new formula (note the trailing dot, following stats::update). Pass . ~ . + new_predictor to add a term.
iter	Optional new total number of iterations.
warmup	Optional new warm-up length.
chains	Optional new number of MCMC chains.
cores	Optional new number of cores.
control	Optional new control list (e.g.\ list(adapt_delta = 0.99)).
...	Additional arguments forwarded to update.brmsfit.

Value

An updated hbmfit object.

Auto-fallback for new formula terms

When you supply a new formula. that references variables not in the original model frame, **brms**'s default update.brmsfit refuses with *"New variables found ...; supply data again via newdata"*. update_hbm catches this specific error and automatically retries with newdata = object$data (the data frame stored on the original hbmfit). Pass an explicit newdata to override this behaviour.

Fixed-parameter columns

Models fitted with sampling_variance, n + deff (in hbm_betalogitnorm), or fixed_params carry hidden offset columns named .hbsaems_<par>_fixed in their data frames. When update_hbm receives a newdata that *does not* have these columns, brms refuses to refit with *"variables can neither be found in 'data' nor in 'data2'"*. update_hbm now detects this case and:

- warns the user when offset columns are present in the original data but missing in newdata;
- either copies the columns over from the original object$data (when nrow(newdata) == nrow(object$data), which is the typical "same areas, updated covariates" case), or
- raises an informative error pointing the user to either supply the columns in newdata explicitly or refit from scratch with a fresh hbm() call.

Examples

```
library(hbsaems)
library(brms)
data("data_fhnorm")
model <- hbm(brms::bf(y ~ x1), data = data_fhnorm,
             re = ~ (1 | regency),    # area-level random effect
             chains = 4, iter = 2000, warmup = 1000, cores = 1,
             seed = 1, refresh = 0)

# Re-run with slightly more iterations (no data change needed).
```

```
# In production you would jump to e.g. iter = 4000, warmup = 2000.
model2 <- update_hbm(model, iter = 1000, warmup = 500)

# Add a predictor: the auto-fallback transparently retries with the
# stored data frame.  Equivalent to passing newdata = data_fhnorm.
model3 <- update_hbm(model, formula. = . ~ . + x2)
```

`validate_hbmfit` *Validate an hbmfit Object*

Description

Public validator for the `hbmfit-class{hbmfit}` S3 class. Runs all invariants that the cheap constructor (`new_hbmfit`) is permitted to skip. Useful when reconstructing an `hbmfit` object manually, reading one back from disk, or testing custom family wrappers.

Usage

```
validate_hbmfit(x)
```

Arguments

`x`	An object to validate.

Details

Invariants verified:

1. Object is a list with class `"hbmfit"`.
2. Has mandatory slots: `model`, `missing_method`, `data`.
3. `model` inherits from `brmsfit` or `brmsfit_multiple`.
4. `missing_method` is `NULL` or a single character string in `c("deleted", "multiple", "model")`.
5. `data` is a `data.frame` with ≥ 1 row.
6. The `handle_missing` alias (if present) equals `missing_method`.

Value

The input `x`, invisibly, when all invariants hold. Otherwise raises an informative error.

See Also

`new_hbmfit`, `hbmfit`

Examples

```
# Minimal example without area-level RE (fixed-effects baseline) --
# suppress the area-RE advisory because this 5-row toy dataset cannot
# meaningfully estimate a random effect.  Uses brms-default MCMC
# settings (chains = 4, iter = 2000, warmup = 1000); on this toy
# data the fit is only used to verify the hbmfit class structure,
# not for inference.
fit <- suppressWarnings(
  hbm(brms::bf(y ~ x1), data = data.frame(y = rnorm(5), x1 = 1:5),
      chains = 4, iter = 2000, warmup = 1000, refresh = 0)
)
validate_hbmfit(fit)
```

Index