Package 'BayesTools'

November 4, 2026

Title Tools for Bayesian Analyses

Version 0.4.0

Description Provides tools for conducting Bayesian analyses and Bayesian model averaging (Kass and Raftery, 1995, <doi:10.1080/01621459.1995.10476572>, Hoeting et al., 1999, <doi:10.1214/ss/1009212519>). The package contains functions for creating a wide range of prior distribution objects, mixing posterior samples from 'JAGS' and 'Stan' models, plotting posterior distributions, and etc... The tools for working with prior distribution span from visualization, generating 'JAGS' and 'bridgesampling' syntax to basic functions such as rng, quantile, and distribution functions.

Maintainer František Bartoš <f.bartos96@gmail.com>

URL https://fbartos.github.io/BayesTools/

BugReports https://github.com/FBartos/BayesTools/issues

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

SystemRequirements JAGS (>= 4.3.0, < 5.0.0) (https://mcmc-jags.sourceforge.io/)

NeedsCompilation yes

Depends R (>= 4.3.0), stats

Imports graphics, grid, extraDistr, mvtnorm, coda, bridgesampling, parallel, ggplot2, Rdpack, rlang (>= 0.4.10), reformulas (>= 0.2.0)

Suggests R6, scales, testthat (>= 3.3.0), waldo, tibble, withr, vdiffr, covr, digest, knitr, pkgload, rstan, brms, rjags, runjags, lme4 (>= 2.0-1), BayesFactor, RoBMA, rmarkdown

RdMacros Rdpack

VignetteBuilder knitr

Config/testthat/edition 3

Author František Bartoš [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0018-5573>)

Repository CRAN

Date/Publication 2026-11-03 23:00:37 UTC

Contents

add_column . . . 5
as_marginal_inference . . . 6
as_mixed_posteriors . . . 8
BayesTools . . . 9
BayesTools_ensemble_tables . . . 10
BayesTools_load_JAGS_module . . . 15
BayesTools_model_tables . . . 15
BF_carrier_drop . . . 24
bridgesampling_object . . . 25
check_input . . . 26
contr.BayesTools . . . 28
density.prior . . . 29
ensemble_inference . . . 32
fit_backend_fingerprint . . . 34
format_BF . . . 35
format_BF_from_log . . . 35
formula_add_intercept . . . 37
geom_prior . . . 37
geom_prior_list . . . 39
harrell_davis_quantile . . . 41
hypothesis_alias_label . . . 43
hypothesis_ast . . . 44
hypothesis_BF . . . 46
hypothesis_BF_model . . . 52
hypothesis_cache_clear . . . 54
hypothesis_density_backend . . . 55
hypothesis_linear_target . . . 59
hypothesis_linear_target_side . . . 60
hypothesis_parse_point_reference . . . 61
hypothesis_provider . . . 62
hypothesis_quantities . . . 65
hypothesis_stop . . . 66
inclusion_BF . . . 67
interpret . . . 68
interpret_records . . . 69
invert_BF_bound_operator . . . 70
is.prior . . . 71
is_prior_bias . . . 72
is_prior_phacking . . . 73
JAGS_add_priors . . . 73

JAGS_bridgesampling . . . 74
JAGS_bridgesampling_posterior . . . 81
JAGS_check_and_list . . . 82
JAGS_check_convergence . . . 84
JAGS_deterministic_nodes . . . 88
JAGS_diagnostics . . . 90
JAGS_draw_geometry . . . 93
JAGS_evaluate_formula . . . 95
JAGS_fit . . . 97
JAGS_fit_contract . . . 103
JAGS_formula . . . 104
JAGS_formula_coefficient_route . . . 109
JAGS_formula_coefficient_transform . . . 110
JAGS_formula_coordinate_dependencies . . . 112
JAGS_formula_design . . . 112
JAGS_formula_draws . . . 113
JAGS_formula_point_finalize . . . 115
JAGS_formula_predictor_basis . . . 117
JAGS_formula_random_marginal_covariance . . . 118
JAGS_get_inits . . . 119
JAGS_lkj_corr_cholesky . . . 120
JAGS_marglik_parameters_formula . . . 121
JAGS_marglik_priors . . . 122
JAGS_ordered_density_kernel . . . 124
JAGS_ordered_parameter_spec . . . 125
JAGS_package_builds . . . 126
JAGS_predict_formula . . . 127
JAGS_runtime_cluster . . . 129
JAGS_to_monitor . . . 130
kitchen_rolls . . . 131
lines.prior . . . 131
lines_prior_list . . . 133
marginal_inference . . . 136
marginal_posterior . . . 137
mean.prior . . . 139
mix_posteriors . . . 140
mpoint . . . 142
parameter_catalog . . . 143
parameter_gate_states . . . 149
parameter_labels . . . 150
parameter_map . . . 151
parameter_map_cache . . . 152
parameter_mixed_posterior . . . 153
parameter_names . . . 155
parameter_source . . . 157
phack_pi_null . . . 158
plot.prior . . . 159
plot.prior_linear_density . . . 162

plot_marginal . . . 164
plot_models . . . 166
plot_posterior . . . 168
plot_prior_list . . . 172
plot_transformed_prior . . . 175
point . . . 176
posterior_atoms_free . . . 177
posterior_atom_attribute . . . 178
posterior_density_attribute . . . 178
posterior_density_method_match . . . 181
posterior_formula_measure_check . . . 181
posterior_log_ordinate_attribute . . . 183
posterior_measure_check . . . 184
posterior_metadata . . . 186
posterior_support_attribute . . . 189
posterior_transform . . . 190
print.BayesTools_table . . . 192
print.prior . . . 193
prior . . . 194
prior_bias . . . 197
prior_density_has_provenance . . . 197
prior_density_ordinate . . . 198
prior_factor . . . 201
prior_factor_levels . . . 204
prior_functions . . . 205
prior_functions_methods . . . 208
prior_informed . . . 209
prior_informed_medicine_names . . . 211
prior_mixture . . . 211
prior_ordered . . . 212
prior_ordered_total_name . . . 214
prior_ordinate_status . . . 214
prior_phacking . . . 216
prior_PP . . . 217
prior_random . . . 219
prior_spike_and_slab . . . 229
prior_weightfunction . . . 230
random_effects_compile . . . 231
random_effects_correlation_draws . . . 232
random_effects_dependency_matrix . . . 234
random_effects_formula . . . 234
random_effects_level_roles . . . 236
random_effects_marginal_diagonal_factor . . . 237
random_effects_marginal_factor_diagonal . . . 238
random_effects_marginal_factor_product . . . 239
random_effects_marginal_factor_states . . . 239
random_effects_marginal_factor_vcov . . . 240
random_effects_marginal_update_grid . . . 241

random_effects_marginal_update_plan . . . 242
random_effects_marginal_variance_factors . . . 244
random_effects_marginal_vcov . . . 245
random_effects_source_roles . . . 247
random_effects_summary_posterior . . . 248
random_group_component_names . . . 249
random_group_covariance . . . 251
range.prior . . . 252
remove_column . . . 252
Savage_Dickey_BF . . . 253
sd . . . 255
sd.prior . . . 256
selection_backend_spec . . . 257
selection_context_validate . . . 258
selection_event_support . . . 259
selection_model . . . 260
selection_qmc_design . . . 263
simplify_as_mixed_posterior_bias . . . 264
transform_factor_samples . . . 265
transform_meandif_samples . . . 265
transform_orthonormal_samples . . . 266
transform_prior_samples . . . 267
transform_scale_samples . . . 269
update.BayesTools_table . . . 271
var . . . 272
var.prior . . . 272
weightfunctions . . . 273
weightfunctions_mapping . . . 275

Index **277**

`add_column` *Adds column to BayesTools table*

Description

Adds column to a BayesTools table while not breaking formatting, attributes, etc...

Usage

```
add_column(
  table,
  column_title,
  column_values,
  column_position = NULL,
  column_type = NULL
)
```

Arguments

table	BayesTools table
column_title	title of the new column
column_values	values of the new column
column_position	position of the new column (defaults to NULL which appends the column to the end)
column_type	type of values of the new column table (important for formatting, defaults to NULL = the function tries to guess numeric / character based on the column_values but many more specific types are available)

Value

returns an object of 'BayesTools_table' class.

---

as_marginal_inference *Model-average marginal posterior distributions and marginal Bayes factors based on BayesTools JAGS model via* marginal_inference

---

Description

Creates marginal model-averaged and conditional posterior distributions based on a BayesTools JAGS model, vector of parameters, formula, and a list of conditional specifications for each parameter. Computes inclusion Bayes factors for each marginal estimate via a Savage-Dickey density approximation.

Usage

```
as_marginal_inference(
  model,
  marginal_parameters,
  parameters,
  conditional_list,
  conditional_rule,
  formula,
  null_hypothesis = 0,
  normal_approximation = FALSE,
  n_samples = 10000,
  silent = FALSE,
  force_plots = FALSE,
  density_method = "KDE",
  compute_BF = TRUE
)
```

Arguments

model	model fit via the JAGS_fit function with this version of BayesTools. Fits without its parameter map and fit contract (such as fits created by BayesTools 0.3.0) must be refitted: they stop with an error of class BayesTools_refit_required (see JAGS_validate_fit_contract()).
marginal_parameters	parameters for which the the marginal summary should be created
parameters	all parameters included in the model_list that are relevant for the formula (all of which need to have specification of is_null_list)
conditional_list	list of conditional parameters for each marginal parameter Unnamed entries bind to marginal_parameters in supplied order; uniquely named complete lists bind by key. Partial or mismatched names and duplicate, missing or empty marginal keys are refused.
conditional_rule	a character string specifying the rule for conditioning. Either "AND" or "OR". Defaults to "AND".
formula	model formula (needs to be specified if parameter was part of a formula)
null_hypothesis	point null hypothesis to test. Defaults to 0
normal_approximation	whether the height of the posterior density at the null should be approximated via a normal distribution (rather than kernel density). The prior height is always the structural prior-density ordinate. Defaults to FALSE.
n_samples	controls the numerical grid used for model-averaged prior densities (a grid has at least 16 points). It is not used when the samples carry a stored prior-density context. For a simple parameter, an explicitly attached prior_density is authoritative and is propagated instead of being reconstructed from prior_list.
silent	whether warnings should be returned silently. Defaults to FALSE
force_plots	temporal argument allowing to generate conditional posterior samples suitable for prior and posterior plots. Only available when conditioning on a single parameter.
density_method	posterior density method used for Savage-Dickey Bayes factors. Currently only "KDE" is supported by marginal_inference() and as_marginal_inference() because generic model-averaged marginal posteriors do not have a well-defined precomputed density source.
compute_BF	whether to compute inclusion Bayes factors. When FALSE, the averaged and conditional marginal posteriors are returned and the inference list is empty.

Details

For as_marginal_inference(), conditional_list is applied separately to each output marginal or level. A requested conditional parameter is active only for levels whose linear combination has a nonzero weight for that parameter. Requested conditionals with zero weight are ignored for that level; if no requested conditionals are active, the level uses the fully averaged posterior

and prior context. Level comparisons require the compared levels to use the same effective conditional subset and rule. As in `marginal_inference()`, a parameter or level whose conditional posterior has a declared point mass at the null hypothesis gets an `NA` inclusion Bayes factor with the reason in its `"warnings"` attribute. Known prior-ordinate refusals and a numerically unavailable posterior ordinate also produce `NA` plus the original reason, including scalar marginal table results; direct scalar `Savage_Dickey_BF()` remains strict. A level whose effective conditional subset is non-empty and leaves no draws is unavailable: it gets the warning `"No samples left after conditioning."` (once per distinct conditioning event) and is absent from the `conditional` and `inference` entries of its parameter, while the other levels and the `averaged` marginals are kept. A parameter without an available level has no `conditional` or `inference` entry.

Value

`as_marginal_inference` returns an object of class 'marginal_inference'.

See Also

marginal_inference as_mixed_posteriors

`as_mixed_posteriors` *Export BayesTools JAGS model posterior distribution as model-average posterior distributions via* `mix_posteriors`

Description

Creates a model-averages posterior distributions on a single model that allows mimicking the mix_posteriors functionality. This function is useful when the model-averaged ensemble is based on prior_spike_and_slab or prior_mixture priors - the model-averaging is done within the model.

Usage

```
as_mixed_posteriors(
  model,
  parameters,
  conditional = NULL,
  conditional_rule = "AND",
  force_plots = FALSE,
  transform_scaled = FALSE,
  n_prior_samples = 10000
)
```

Arguments

`model`	model fit via the JAGS_fit function with this version of BayesTools. Fits without its parameter map and fit contract (such as fits created by BayesTools 0.3.0) must be refitted: they stop with an error of class `BayesTools_refit_required` (see `JAGS_validate_fit_contract()`).

`parameters` vector of parameters names for which inference should be drawn

`conditional` a character vector of parameters to be conditioned on

`conditional_rule`
a character string specifying the rule for conditioning. Either "AND" or "OR". Defaults to "AND".

`force_plots` temporal argument allowing to generate conditional posterior samples suitable for prior and posterior plots. Only available when conditioning on a single parameter.

`transform_scaled`
whether to transform samples from standardized (scaled) to original (unscaled) scale. When `TRUE`, posterior samples are transformed, and the result can be directly passed to plot_posterior which will automatically detect the transformation and use transformed deterministic prior densities. Requires a model fitted with `formula_scale_list`. Defaults to `FALSE`. The transformed prior densities are computed for the requested `parameters` only and are kept for the fitted object, so repeated calls with the same fit, priors, conditioning, and `n_prior_samples` reuse them. The prior density of a random-effect SD is that of the SD on the original scale: the fitted law rescaled when the original-scale SD is a rescaling of one fitted SD, and otherwise (an SD that combines several fitted SDs, such as the intercept SD of a block with a non-centered scaled slope) unavailable, which `hypothesis_BF()`, `marginal_posterior()` with `prior_samples = TRUE` and `plot_posterior()` with `prior = TRUE` report with an error of class `BayesTools_formula_measure_unavailable` (see `posterior_measure_check()`).

`n_prior_samples`
controls the numerical grid used for transformed prior densities when `transform_scaled = TRUE`. Defaults to 10000.

Value

`as_mix_posteriors` returns a named list of mixed posterior distributions (either a vector of matrix).

See Also

mix_posteriors

`BayesTools` *BayesTools*

Description

BayesTools: Provides tools for conducting Bayesian analyses. The package contains functions for creating a wide range of prior distribution objects, mixing posterior samples from JAGS and Stan models, plotting posterior distributions, and etc... The tools for working with prior distribution span from visualization, generating JAGS and bridgesampling syntax to basic functions such as rng, quantile, and distribution functions.

Details

Random-effect operations use the `BayesTools.random_effects_memory_limit_bytes` option as a deterministic pre-allocation guard. The default is `16 * 1024^3` bytes (16 GiB). The estimate includes the primary numeric payload, simultaneous working copies, row maps, group state, and covariance state known before allocation. Set a different positive byte ceiling with `options(BayesTools.random_effects_me = value)`. Experts may explicitly use `Inf` to disable the guard after verifying the operation's memory budget. Passing the guard does not promise that an allocation will succeed because R, JAGS, and the operating system have additional memory requirements. An operation above the ceiling stops with an error of class `BayesTools_random_effects_memory_limit` (also `BayesTools_random_effects_error`) whose fields `operation`, `peak_bytes`, and `limit_bytes` report the estimate.

Author(s)

František Bartoš <f.bartos96@gmail.com>

See Also

Useful links:

- https://fbartos.github.io/BayesTools/
- Report bugs at https://github.com/FBartos/BayesTools/issues

---

`BayesTools_ensemble_tables`

Create BayesTools ensemble summary tables

Description

Creates estimate summaries based on posterior distributions created by mix_posteriors, inference summaries based on inference created by ensemble_inference, or ensemble summary/diagnostics based on a list of models_inference models (or marginal_inference in case of marginal_estimates_table).

Usage

```
ensemble_estimates_table(
  samples,
  parameters,
  probs = c(0.025, 0.975),
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  transform_factors = FALSE,
  formula_prefix = TRUE,
  transform_scaled = FALSE,
  formula_scale = NULL
)
```

```
ensemble_inference_table(
  inference,
  parameters,
  logBF = FALSE,
  BF01 = FALSE,
  title = NULL,
  footnotes = NULL,
  warnings = NULL
)

ensemble_summary_table(
  models,
  parameters,
  logBF = FALSE,
  BF01 = FALSE,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_spike_0 = TRUE,
  short_name = FALSE
)

ensemble_diagnostics_table(
  models,
  parameters,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_spike_0 = TRUE,
  short_name = FALSE
)

ensemble_estimates_empty_table(
  probs = c(0.025, 0.975),
  title = NULL,
  footnotes = NULL,
  warnings = NULL
)

ensemble_inference_empty_table(title = NULL, footnotes = NULL, warnings = NULL)

ensemble_summary_empty_table(title = NULL, footnotes = NULL, warnings = NULL)

ensemble_diagnostics_empty_table(
  title = NULL,
  footnotes = NULL,
  warnings = NULL
)
```

```
marginal_estimates_table(
  samples,
  inference,
  parameters,
  probs = c(0.025, 0.5, 0.975),
  logBF = FALSE,
  BF01 = FALSE,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  formula_prefix = TRUE,
  transform_scaled = FALSE,
  formula_scale = NULL
)
```

Arguments

samples	posterior samples created by mix_posteriors. For ensemble_estimates_table(), a sample vector or matrix may contain missing draws only if it declares them as undefined through its undefined_draws metadata (posterior_metadata(x,"undefined_draws") <- ; a single value for a vector, or values named by column for a matrix), as set by parameter_draws() from the catalog's definedness ("correlation" for original-scale random-effect correlations, "multiple_correlation" for their cor_total multiple correlations, "allocation_active" for variance shares of gated total-variance allocations). Such a row is summarized over its defined draws and a table footnote reports their share, e.g., "summarized over k of n draws where the correlation is defined, i.e. both SDs are positive". Any other missing draw is an error.
parameters	character vector of parameters (or a named list with of character vectors for summary and diagnostics tables) specifying the parameters (and their grouping) for the summary table
probs	quantiles for parameter estimates
title	title to be added to the table
footnotes	footnotes to be added to the table
warnings	warnings to be added to the table
transform_factors	whether factors with orthonormal/meandif prior distribution should be transformed to differences from the grand mean, and factors with prior_ordered() prior distribution to their level effects (without the reference level of a "cumulative" contrast, which is fixed at zero). Untransformed ordered factors show the sampled parameters: runjags_estimates_table() reports the total effect (<term>{total}) and the allocation shares of the random allocations (<term>[share: <level>], the share of the total added when reaching <level>; fixed allocations have no rows). ensemble_estimates_table() reports retained fitted totals and the union of allocation shares sampled by any declared ordered model. Fixed ordered allocations fill that union with their declared constants. Models without

the ordered parameterization have undefined (NA) shares, with their defined fraction reported in a footnote. Shares remain defined on zero-total and excluded spike draws of an ordered model. Legacy mixed objects without retained sources must be recreated from source fits.

`formula_prefix` whether the parameter prefix from formula should be printed. Defaults to TRUE.

`transform_scaled`
: whether coefficients from standardized continuous predictors should be transformed back to the original scale. For `runjags_estimates_table()`, FALSE keeps fixed and random effects on the fitted predictor scale, while TRUE transforms both to the original predictor scale when the fit contains `formula_scale` metadata: random-effect SDs and correlations are computed from the transformed covariance draws. Subtraction-centering can leave a numerically nonzero predictor mean; its induced effects are retained. Composite original-scale SDs have available draws but may have unavailable prior laws. Fitted-scale coefficients and semantic random-effect summaries remain available without this transformation. A block whose missing-term fallback mixes categorical coding or omits a representable contribution has no available original-scale output, with reason `"original_scale_not_representable"`. Use fitted-scale output or include the needed lower-order terms in the block. Mixed posterior samples are transformed through the fitted design of `formula_scale`: every column is mapped to its fitted coefficient by its `quantities` draw metadata (see `posterior_metadata()`), and columns that are not fitted coefficients are refused (a `posterior_transform()` result, nonlinear or affine with an offset, stops with reason `"transformed_column_not_fitted"`: unscale first, then transform), as are random-effect SDs that the random-effect structure of `formula_scale` does not cover (SDs it does not contain, blocks it contains only in part, or blocks it leaves unchanged while the samples' original-scale SDs differ from the fitted ones). The SDs of a block are transformed through the fitted block design that the formula state of the samples carries (`as_mixed_posteriors()`, and `mix_posteriors()` when fixed formula coefficients or random-effect SDs are among its parameters); samples without it can pair coefficient names only, and the SDs of a block that this does not map exactly (e.g., a factor in an interaction with a standardized slope whose margin is not in the block, or the indicator columns of a logical predictor in such an interaction) are refused. The SD columns of a block with a standardized random slope are labelled by their fitted coordinates (`(mu) g: sd(x)`) until they are transformed to the catalog's original-scale SDs. For a standardized slope that interacts with a factor whose level effects are not columns of the block (`diag(1 + f:x | g)`: the level intercepts; `diag(1 + z + f:x:z | g)`: the level slopes of `z`), the shifts that centering induces by level are not block coefficients. The surviving-coordinate transformation retains the shifted coefficients (here the intercept, or the slope of `z`) at the predictor mean; the SDs of the columns that no centering shifts are exact. `runjags_estimates_table()` instead reports the unavailable original-scale quantities of the shifted columns as NA with a row-specific note, and retains the identifiable summaries. Estimated marginal means of `marginal_estimates_table()` are predictions at the stated predictor values, which do not depend on the standardization, and are not changed. Defaults to FALSE.

`formula_scale` named list containing standardization information (mean and sd) for each stan-

dardized predictor, together with the fitted design of each formula: the `formula_scale` attribute of the fitted model. Required when `transform_scaled = TRUE` for ensemble/marginal tables (unless the samples were unscaled by `as_mixed_posteriors()`; an empty list means that no predictor is scaled): without it the tables stop with `BayesTools_formula_transform_unavailable` (reason `"formula_scale_missing"`), as standardization information without the fitted design does. For `runjags_estimates_table`, this is automatically extracted from the fit object's `formula_scale` attribute.

`inference` — model inference created by ensemble_inference. `ensemble_inference_table()` refuses conditional inference, of the whole object or of any of its parameters.

`logBF` — whether the Bayes factor should be on log scale. When model inference objects store log-space inclusion Bayes factors, log-scale values are taken from them, so they remain finite beyond the range of natural-scale Bayes factors.

`BF01` — whether the Bayes factor should be inverted

`models` — list of models_inference model objects, each of which containing a list of `priors` and `inference` object, The `inference` must be a named list with information about the model: model number `m_number`, marginal likelihood `marglik`, prior and posterior probability `prior_prob` and `post_prob`, inclusion Bayes factor `inclusion_BF`, and fit summary generated by runjags_estimates_table for the diagnostics table

`remove_spike_0` — whether prior distributions equal to spike at 0 should be removed from the `prior_list`. In JAGS estimates tables with `transform_scaled = TRUE`, a coefficient with a spike at 0 is kept when it is non-zero on the original scale.

`short_name` — whether the prior distribution names should be shortened. Defaults to `FALSE`.

Value

`ensemble_estimates_table` returns a table with the model-averaged estimates, `ensemble_inference_table` returns a table with the prior and posterior probabilities and inclusion Bayes factors, `ensemble_summary_table` returns a table with overview of the models included in the ensemble, and `ensemble_diagnostics_table` returns an overview of the MCMC diagnostics for the models included in the ensemble. All of the tables are objects of class 'BayesTools_table'. The estimates tables (`ensemble_estimates_table` and `marginal_estimates_table`) carry the per-row quantity table `attr(table, "quantities")`: a data frame with one row per table row and the columns `row` (the row label), `quantity_id` (the `parameter_catalog()` quantity id the summarized draws declare in their `quantities` metadata, see `posterior_metadata()`; `""` when they declare none, e.g. for estimated marginal means), and `label_parts` (the label parts the row label is rendered from; see `parameter_labels()`). Subsetting the table subsets its quantity table.

See Also

ensemble_inference mix_posteriors BayesTools_model_tables

BayesTools_load_JAGS_module

Load the BayesTools JAGS module

Description

Loads the package-shipped JAGS extension module that provides compiled distributions and helper functions used by generated BayesTools JAGS syntax.

Usage

```
BayesTools_load_JAGS_module(quiet = TRUE, warn = TRUE)
```

Arguments

quiet	logical scalar. Whether module-loading messages should be suppressed.
warn	logical scalar. Whether a warning should be emitted when the module cannot be loaded.

Value

Invisibly returns TRUE when the module is loaded and FALSE otherwise.

BayesTools_model_tables

Create BayesTools model tables

Description

Creates model summary based on a model objects or provides estimates table for a runjags fit.

Usage

```
model_summary_table(
  model,
  model_description = NULL,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_spike_0 = TRUE,
  short_name = FALSE,
  formula_prefix = TRUE,
  remove_parameters = NULL
)
```

```
runjags_estimates_table(
  fit,
  transformations = NULL,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  conditional = FALSE,
  probs = c(0.025, 0.5, 0.975),
  remove_spike_0 = TRUE,
  transform_factors = FALSE,
  formula_prefix = TRUE,
  remove_inclusion = FALSE,
  remove_parameters = NULL,
  remove_formulas = NULL,
  keep_parameters = NULL,
  keep_formulas = NULL,
  return_samples = FALSE,
  transform_scaled = FALSE,
  remove_diagnostics = FALSE,
 diagnostic_columns = getOption("BayesTools.JAGS_estimates_diagnostic_columns", if
    (remove_diagnostics) "none" else "all"),
  random_effects_summary = c("standard", "full", "raw", "none"),
  simplify_names = FALSE,
  random_effects_metadata = FALSE,
  remove_random_effects = NULL,
  keep_random_effects = NULL,
  remove_random_structures = NULL,
  keep_random_structures = NULL
)

runjags_inference_table(
  fit,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  formula_prefix = TRUE,
  logBF = FALSE,
  BF01 = FALSE,
  BF_diagnostics = FALSE,
  BF_diagnostic_columns = getOption("BayesTools.JAGS_BF_diagnostic_columns", if
    (BF_diagnostics) "all" else "none")
)

JAGS_estimates_table(
  fit,
  transformations = NULL,
  title = NULL,
  footnotes = NULL,
```

```
  warnings = NULL,
  conditional = FALSE,
  probs = c(0.025, 0.5, 0.975),
  remove_spike_0 = TRUE,
  transform_factors = FALSE,
  formula_prefix = TRUE,
  remove_inclusion = FALSE,
  remove_parameters = NULL,
  remove_formulas = NULL,
  keep_parameters = NULL,
  keep_formulas = NULL,
  return_samples = FALSE,
  transform_scaled = FALSE,
  remove_diagnostics = FALSE,
 diagnostic_columns = getOption("BayesTools.JAGS_estimates_diagnostic_columns", if
    (remove_diagnostics) "none" else "all"),
  random_effects_summary = c("standard", "full", "raw", "none"),
  simplify_names = FALSE,
  random_effects_metadata = FALSE,
  remove_random_effects = NULL,
  keep_random_effects = NULL,
  remove_random_structures = NULL,
  keep_random_structures = NULL
)

JAGS_inference_table(
  fit,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  formula_prefix = TRUE,
  logBF = FALSE,
  BF01 = FALSE,
  BF_diagnostics = FALSE,
  BF_diagnostic_columns = getOption("BayesTools.JAGS_BF_diagnostic_columns", if
    (BF_diagnostics) "all" else "none")
)

JAGS_summary_table(
  model,
  model_description = NULL,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_spike_0 = TRUE,
  short_name = FALSE,
  formula_prefix = TRUE,
  remove_parameters = NULL
```

```
)

model_summary_empty_table(
  model_description = NULL,
  title = NULL,
  footnotes = NULL,
  warnings = NULL
)

runjags_estimates_empty_table(
  probs = c(0.025, 0.5, 0.975),
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_diagnostics = FALSE,
 diagnostic_columns = getOption("BayesTools.JAGS_estimates_diagnostic_columns", if
    (remove_diagnostics) "none" else "all")
)

runjags_inference_empty_table(
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  logBF = FALSE,
  BF01 = FALSE,
  BF_diagnostics = FALSE,
  BF_diagnostic_columns = getOption("BayesTools.JAGS_BF_diagnostic_columns", if
    (BF_diagnostics) "all" else "none")
)

JAGS_estimates_empty_table(
  probs = c(0.025, 0.5, 0.975),
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_diagnostics = FALSE,
 diagnostic_columns = getOption("BayesTools.JAGS_estimates_diagnostic_columns", if
    (remove_diagnostics) "none" else "all")
)

JAGS_inference_empty_table(
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  logBF = FALSE,
  BF01 = FALSE,
  BF_diagnostics = FALSE,
  BF_diagnostic_columns = getOption("BayesTools.JAGS_BF_diagnostic_columns", if
```

```
    (BF_diagnostics) "all" else "none")
)

stan_estimates_table(
  fit,
  transformations = NULL,
  title = NULL,
  footnotes = NULL,
  warnings = NULL
)
```

Arguments

model	model object containing a list of priors and inference object, The inference must be a named list with information about the model: model number m_number, marginal likelihood marglik, prior and posterior probability prior_prob and post_prob, and model inclusion Bayes factor inclusion_BF
model_description	named list with additional description to be added to the table
title	title to be added to the table
footnotes	footnotes to be added to the table
warnings	warnings to be added to the table
remove_spike_0	whether prior distributions equal to spike at 0 should be removed from the prior_list. In JAGS estimates tables with transform_scaled = TRUE, a coefficient with a spike at 0 is kept when it is non-zero on the original scale.
short_name	whether the prior distribution names should be shortened. Defaults to FALSE.
formula_prefix	whether the parameter prefix from formula should be printed. Defaults to TRUE.
remove_parameters	parameters to be removed from the summary. Can be NULL (default, no removal), a character vector of parameter names to remove, or TRUE to remove all parameters that are not part of any formula. For formula random effects, character filters also accept semantic selectors "random", "random_sd", "random_cor", "random_var_prop", "random_var_mult", "random_allocation", and "random_sd_mult".
fit	runjags model fit or a derived-draw view from JAGS_with_draws(). Estimates and inference describe its supplied draws; chain diagnostics use the actual retained chain geometry.
transformations	named list of transformations to be applied to specific parameters
conditional	summarizes estimates conditional on being included in the model for spike and slab priors. Defaults to FALSE. The standard and full random-effect summaries condition the SD, variance and multiplier rows of a random SD with a spike-and-slab or null/alternative mixture prior on that prior's own inclusion event, as the raw summary does. A mixture with other components has no single inclusion event: its rows are summarized over all draws and a footnote says so.
probs	quantiles for parameter estimates

transform_factors
: whether factors with orthonormal/meandif prior distribution should be transformed to differences from the grand mean, and factors with prior_ordered() prior distribution to their level effects (without the reference level of a "cumulative" contrast, which is fixed at zero). Untransformed ordered factors show the sampled parameters: runjags_estimates_table() reports the total effect (<term>{total}) and the allocation shares of the random allocations (<term>[share: <level>], the share of the total added when reaching <level>; fixed allocations have no rows). ensemble_estimates_table() reports retained fitted totals and the union of allocation shares sampled by any declared ordered model. Fixed ordered allocations fill that union with their declared constants. Models without the ordered parameterization have undefined (NA) shares, with their defined fraction reported in a footnote. Shares remain defined on zero-total and excluded spike draws of an ordered model. Legacy mixed objects without retained sources must be recreated from source fits.

remove_inclusion
: whether estimates of the inclusion probabilities should be excluded from the summary table. Defaults to FALSE. The indicator rows of spike-and-slab and mixture priors, of spike-and-slab totals of ordered-factor priors (labelled <term>{total} (inclusion)), and the raw variance-allocation gates are removed. The backend _variable and _inclusion rows of an ordered prior with a spike-and-slab total are not recognized as inclusion rows and remain in the table. Retained inclusion rows (spike-and-slab and mixture indicators, random-effect and variance-allocation inclusion gates) report the posterior inclusion probability as Mean together with the MCMC diagnostics; their SD and quantile cells are empty.

remove_formulas
: character vector of formula names whose parameters should be removed from the summary. Defaults to NULL.

keep_parameters
: character vector of parameter names to keep. All other parameters will be removed unless they belong to formulas specified in keep_formulas. The random-effect aliases listed for remove_parameters can also be used here, for example keep_parameters = c("random_sd", "random_cor").

keep_formulas
: character vector of formula names whose parameters should be kept. All other parameters will be removed unless they are specified in keep_parameters. Defaults to NULL.

return_samples
: whether to return the transoformed and formated samples instead of the table. Defaults to FALSE.

transform_scaled
: whether coefficients from standardized continuous predictors should be transformed back to the original scale. For runjags_estimates_table(), FALSE keeps fixed and random effects on the fitted predictor scale, while TRUE transforms both to the original predictor scale when the fit contains formula_scale metadata: random-effect SDs and correlations are computed from the transformed covariance draws. Subtraction-centering can leave a numerically nonzero predictor mean; its induced effects are retained. Composite original-scale SDs have available draws but may have unavailable prior laws. Fitted-scale coefficients and semantic random-effect summaries remain available without this

transformation. A block whose missing-term fallback mixes categorical coding or omits a representable contribution has no available original-scale output, with reason "original_scale_not_representable". Use fitted-scale output or include the needed lower-order terms in the block. Mixed posterior samples are transformed through the fitted design of formula_scale: every column is mapped to its fitted coefficient by its quantities draw metadata (see posterior_metadata()), and columns that are not fitted coefficients are refused (a posterior_transform() result, nonlinear or affine with an offset, stops with reason "transformed_column_not_fitted": unscale first, then transform), as are random-effect SDs that the random-effect structure of formula_scale does not cover (SDs it does not contain, blocks it contains only in part, or blocks it leaves unchanged while the samples' original-scale SDs differ from the fitted ones). The SDs of a block are transformed through the fitted block design that the formula state of the samples carries (as_mixed_posteriors(), and mix_posteriors() when fixed formula coefficients or random-effect SDs are among its parameters); samples without it can pair coefficient names only, and the SDs of a block that this does not map exactly (e.g., a factor in an interaction with a standardized slope whose margin is not in the block, or the indicator columns of a logical predictor in such an interaction) are refused. The SD columns of a block with a standardized random slope are labelled by their fitted coordinates ((mu) g: sd(x)) until they are transformed to the catalog's original-scale SDs. For a standardized slope that interacts with a factor whose level effects are not columns of the block (diag(1 + f:x | g): the level intercepts; diag(1 + z + f:x:z | g): the level slopes of z), the shifts that centering induces by level are not block coefficients. The surviving-coordinate transformation retains the shifted coefficients (here the intercept, or the slope of z) at the predictor mean; the SDs of the columns that no centering shifts are exact. runjags_estimates_table() instead reports the unavailable original-scale quantities of the shifted columns as NA with a row-specific note, and retains the identifiable summaries. Estimated marginal means of marginal_estimates_table() are predictions at the stated predictor values, which do not depend on the standardization, and are not changed. Defaults to FALSE.

`remove_diagnostics`
: whether to exclude MCMC diagnostics (MCMC error, ESS, R-hat) from the output table. Defaults to FALSE. Setting to TRUE will exclude diagnostics columns regardless of the conditional setting.

`diagnostic_columns`
: MCMC diagnostic columns to display in JAGS estimates tables. Can be "all", "none", TRUE, FALSE, or a character vector containing any subset of "MCMC_error", "MCMC_SD_error", "ESS", and "R_hat". Defaults to the BayesTools.JAGS_estimates_diagnostic_co option, or all diagnostics unless remove_diagnostics = TRUE.

`random_effects_summary`
: random-effect reporting mode for JAGS estimates tables. "standard" replaces raw implementation parameters with the semantic quantities corresponding to the fitted prior parameterization: directly specified block scales, correlations, inclusion probabilities, allocation aggregate SDs, and variance proportions or multipliers. Inclusion indicators do not add allocation-derived component SDs to this standard set. "full" additionally includes deterministic representations

such as aggregate variances, allocation-derived component SDs, and alternate SD or variance multipliers. `"raw"` keeps the historical monitored implementation parameters, and `"none"` removes random-effect parameters from the table. The `"standard"` and `"full"` modes always obtain public semantic quantities from the fitted parameter map, and omit the internal (unnormalized gamma) allocation nodes of ordered-factor priors. In every reporting mode, `transform_scaled = FALSE` keeps both fixed coefficients and random-effect SDs, variances, and correlations on the fitted predictor scale; `transform_scaled = TRUE` transforms them to the original predictor scale. Centering changes the definition of a random intercept and can therefore change its SD and its correlations with random slopes. Allocation totals and weights retain their declared parameterization. A scaled random-effect block whose design is linearly dependent has no identified original-scale quantities: `transform_scaled = TRUE` reports their rows as `NA` with a note explaining the unavailable transformation. A block whose missing-term fallback mixes categorical coding or omits a representable contribution likewise has `NA` original-scale rows. An SD shared by several columns, as by the indicator columns of a logical predictor's interaction with a scaled slope or the columns of a meandif or orthonormal factor term, needs its columns to be certified as having one original-scale SD: in a correlated block every column must be its own fitted column rescaled by one common factor, and in a diagonal block the columns need equal variance weights. Such an SD is `NA` with its variance, while the other rows of the block keep their original-scale values. If centering a factor-by-slope interaction induces level-specific effects absent from the block (the intercepts of `f:x`, the `z` slopes of `f:x:z` without `f:z`), only the SDs, variances and correlations of the columns that would absorb them (the intercept, or the lower-order slope) are unavailable; the summaries of the other columns remain. These table notes also accompany `return_samples = TRUE` as a `footnotes` attribute. Inference accessors retain their own transformation and availability contracts. Use `transform_scaled = FALSE` for fitted-scale summaries in either `"standard"` or `"full"` mode, or include the needed lower-order terms for an uncertified fallback. The `"raw"` rows of random-effect and allocation implementation coordinates (block SDs, Cholesky factors and correlation matrices, LKJ primitives, standardized group effects, and allocation nodes) are backend coordinates shown for inspection: their labels describe the coordinate, but they are not catalog quantities and do not select one in `parameter_catalog_resolve()`, hypotheses, or plots (the shares of the ordered-factor allocations, which the table shows as `<term>[share: <level>]`, are catalog quantities). Internal latent and realized group-coefficient coordinates are omitted when `transform_scaled = TRUE`, including in `"raw"` mode, because those coordinates remain on the fitted standardized scale.

`simplify_names` whether semantic random-effect row names should use simplified display labels. A sole random intercept is shown as `sd`, while canonical names such as `sd(intercept)` remain stored in the parameter map. Defaults to `FALSE`.

`random_effects_metadata`

whether to add random-effect metadata columns to JAGS estimates tables. When `TRUE`, the table includes the user-facing random-effect name, grouping label, and covariance structure type for random-effect rows. The rows owned by a variance allocation (its totals, proportions, and inclusion indicators) belong to no single

block and leave these columns empty. Defaults to FALSE.

remove_random_effects, keep_random_effects
: character vectors of random-effect names/blocks used to remove or keep random-effect rows. These match explicit name = values, generated block names, and grouping labels of blocks; they address blocks only, so they never match the rows owned by a variance allocation (keep_random_effects removes those rows, remove_random_effects keeps them). Non-random parameters are left unaffected; combine with keep_parameters = "random" to return only the selected random-effect rows.

remove_random_structures, keep_random_structures
: character vectors of random-effect covariance structures (for example "diag", "us", "ar1", or "hcs") used to remove or keep random-effect rows. Non-random parameters are left unaffected.

logBF
: whether the Bayes factor should be on log scale. When model inference objects store log-space inclusion Bayes factors, log-scale values are taken from them, so they remain finite beyond the range of natural-scale Bayes factors.

BF01
: whether the Bayes factor should be inverted

BF_diagnostics
: whether to add MCMC diagnostics for Bayes factors computed from model indicator frequencies. The Bayes factor error is reported as a relative Monte Carlo standard error percentage. Defaults to FALSE.

BF_diagnostic_columns
: MCMC diagnostic columns to display in JAGS inclusion Bayes factor tables. Can be "all", "none", TRUE, FALSE, or a character vector containing any subset of "ESS", "MCMC_error", and "BF_error_percent". Defaults to the BayesTools.JAGS_BF_diagnostic option, or all diagnostics when BF_diagnostics = TRUE and none otherwise.

Details

For product-space JAGS inclusion Bayes factors, posterior inclusion probabilities of exactly 0 or 1 cannot produce a finite point estimate of the model odds ratio. In that case, inclusion BFs marked with "<" or ">" are finite-sample bounds: posterior inclusion probabilities of 0 or 1 were replaced by 1/S or (S - 1)/S, where S is the number of posterior samples. This is a reporting convention, not an unbiased finite Bayes-factor estimate. If the prior inclusion probability is exactly 0 or 1, the inclusion Bayes factor is undefined and reported as NA, because the corresponding inclusion/exclusion comparison was not tested.

The inference table has a row for every spike-and-slab and mixture prior, for the spike-and-slab total of an ordered-factor prior (reported under the term label), and for every random-effect inclusion gate.

An original-scale random-effect correlation is defined in a posterior draw whenever both of its SDs are positive, including draws with a singular covariance (for example, a zero scaled intercept SD with a nonzero scaled slope SD gives an intercept-slope correlation of ± 1). It is missing in draws with a zero SD (for example, an excluded spike-and-slab SD of a scaled slope). The affected rows are summarized over the defined draws, and a table footnote reports the share of defined draws for each affected row. Raw Cholesky and LKJ-primitive coordinates (random_effects_summary = "raw") exist only for positive-definite correlation matrices and are also missing in singular draws.

Value

model_summary_table returns a table with overview of the fitted model, runjags_estimates_table returns a table with MCMC estimates, and runjags_estimates_empty_table returns an empty estimates table. All of the tables are objects of class 'BayesTools_table'. The estimates table carries the per-row quantity table attr(table, "quantities"): a data frame with one row per table row and the columns row (the row label), quantity_id (the parameter_catalog() quantity id of the quantity the row's label names, its canonical name or an exact alias such as a transformed factor level <parameter>[dif: level]; "" for rows that are not catalog quantities, such as inclusion rows, mixture components, backend coordinates, fitted random-effect summaries whose catalog quantity is on the original scale, ordered-factor allocation shares, and rows whose values transformations changed, whose label parts record the transformation as "custom"), and label_parts (the label parts the row label is rendered from; parameter_labels() renders them, also with another formula parameter or random-effect quantity names). With transform_scaled = TRUE, the rows of standardized coefficients (including the exponentiated log intercept, an alias of the intercept) show their catalog quantity on the original predictor scale. Subsetting the table subsets its quantity table.

See Also

BayesTools_ensemble_tables

BF_carrier_drop *Remove the carrier attributes of a Bayes factor*

Description

Removes the private attributes with which a Bayes-factor carrier (see format_BF_from_log()) retains its canonical logarithms, so that the values can be stored in a plain data frame or compared as ordinary numbers.

Usage

```
BF_carrier_drop(x)
```

Arguments

x	a numeric vector, typically a Bayes factor created by format_BF_from_log() or format_BF(). A numeric matrix or array is also accepted and keeps its dimensions.

Details

The names and the public attributes name, logBF, BF01, and bound_operator are kept. The class BayesTools_BF is kept only while a non-missing bound operator remains, as for the result of format_BF() on a plain vector. A numeric vector without carrier attributes is returned with the same rule applied to its class.

Value

BF_carrier_drop() returns x without its carrier attributes.

See Also

format_BF_from_log() format_BF()

Examples

```
BF <- format_BF_from_log(c(a = 2, b = 800))
names(attributes(BF))
names(attributes(BF_carrier_drop(BF)))
```

---

bridgesampling_object *Create a BayesTools marginal-likelihood object*

Description

Creates a marginal-likelihood result from a supplied scalar natural-log marginal likelihood. This is useful for analytically known or deliberately zero-evidence models that were not evaluated by bridge sampling.

Usage

```
bridgesampling_object(logml = -Inf)
```

Arguments

logml	scalar natural-log marginal likelihood. Defaults to -Inf, representing intentional zero evidence (posterior model probability zero under model averaging). Use NA only when a marginal-likelihood computation failed: the result is flagged as failed (success = FALSE, logml = NA) and the on_failure policy of ensemble_inference(), models_inference(), and mix_posteriors() applies to that model. Do not store failed bridge-sampling attempts as -Inf unless zero evidence is scientifically intended.

Value

A BayesTools_marglik object (with the class bridge of released versions, so that bridgesampling::bf(), post_prob() and logml() apply) with no repetition rows and the "supplied_scalar" aggregation rule, or the "supplied_failure" rule for a failed computation (logml = NA).

check_input *Check input*

Description

A set of convenience functions for checking objects/arguments to a function passed by a user.

Usage

```
check_bool(
  x,
  name = deparse(substitute(x)),
  check_length = 1,
  allow_NULL = FALSE,
  allow_NA = FALSE,
  call = ""
)

check_char(
  x,
  name = deparse(substitute(x)),
  check_length = 1,
  allow_values = NULL,
  allow_NULL = FALSE,
  allow_NA = TRUE,
  call = ""
)

check_real(
  x,
  name = deparse(substitute(x)),
  lower = -Inf,
  upper = Inf,
  allow_bound = TRUE,
  check_length = 1,
  allow_NULL = FALSE,
  allow_NA = TRUE,
  call = ""
)

check_int(
  x,
  name = deparse(substitute(x)),
  lower = -Inf,
  upper = Inf,
  allow_bound = TRUE,
  check_length = 1,
```

```
  allow_NULL = FALSE,
  allow_NA = TRUE,
  call = ""
)

check_list(
  x,
  name = deparse(substitute(x)),
  check_length = 0,
  check_names = NULL,
  all_objects = FALSE,
  allow_other = FALSE,
  allow_NULL = FALSE,
  call = ""
)
```

Arguments

x	object to be checked
name	name of the object that will be print in the error message.
check_length	length of the object to be checked. Defaults to 1. Set to 0 in order to not check object length.
allow_NULL	whether the object can be NULL. If so, no checks are executed.
allow_NA	whether the object can contain NA or NaN values.
call	string to be placed as a prefix to the error call.
allow_values	names of values allowed in a character vector. Defaults to NULL (do not check).
lower	lower bound of allowed values. Defaults to -Inf (do not check).
upper	upper bound of allowed values. Defaults to Inf (do not check).
allow_bound	whether the values at the boundary are allowed. Defaults to TRUE.
check_names	names of entries allowed in a list. Defaults to NULL (do not check).
all_objects	whether all entries in check_names must be present. Defaults to FALSE.
allow_other	whether additional entries then the specified in check_names might be present

Value

returns NULL, called for the input check.

NA values

allow_NA defaults to TRUE for check_char, check_int, and check_real, but to FALSE for check_bool: a missing value cannot be used as a switch, so an NA flag is rejected unless allow_NA = TRUE is requested.

Examples

```
# check whether the object is logical
check_bool(TRUE, name = "input")

# will throw an error on any other type
## Not run:
  check_bool("TRUE", name = "input")

## End(Not run)
```

`contr.BayesTools` *BayesTools Contrast Matrices*

Description

BayesTools provides several contrast matrix functions for Bayesian factor analysis. These functions create different types of contrast matrices that can be used with factor variables in Bayesian models.

Usage

```
contr.orthonormal(n, contrasts = TRUE)

contr.meandif(n, contrasts = TRUE)

contr.independent(n, contrasts = TRUE)

contr.ordered_cumulative(n, contrasts = TRUE)

contr.ordered_cumulative_levels(n, contrasts = TRUE)
```

Arguments

`n`	a vector of levels for a factor, or the number of levels
`contrasts`	logical indicating whether contrasts should be computed

Details

The package includes the following contrast functions:

`contr.orthonormal` Return a matrix of orthonormal contrasts: a normalised Helmert basis, whose columns have unit length and sum to zero and which is identical on every platform. Any orthonormal basis of the centred space implies the same exchangeable prior on the factor level effects under a spherical prior on the coefficients, which corresponds to the description by Rouder et al. (2012). Returns a matrix with n rows and k columns, with k = n - 1 if `contrasts = TRUE` and k = n if `contrasts = FALSE`.

`contr.meandif` Return a matrix of mean difference contrasts. This is an adjustment to the `contr.orthonormal` that ascertains that the prior distributions on difference between the gran mean and factor level are identical independent of the number of factor levels (which does not hold for the orthonormal contrast). Furthermore, the contrast is re-scaled so the specified prior distribution exactly corresponds to the prior distribution on difference between each factor level and the grand mean – this is approximately twice the scale of `contr.orthonormal`. Returns a matrix with n rows and k columns, with k = n - 1 if `contrasts` = `TRUE` and k = n if `contrasts` = `FALSE`.

`contr.independent` Return a matrix of independent contrasts – a level for each term. Returns a matrix with n rows and k columns, with k = n if `contrasts` = `TRUE` and k = n if `contrasts` = `FALSE`.

`contr.ordered_cumulative` Return a cumulative increment basis for ordered factors, with the first level fixed to zero and the last level equal to the full total effect.

`contr.ordered_cumulative_levels` Return a cumulative level basis for ordered factors, where the first level receives the first allocation share and the last level equals the full total effect.

References

Rouder JN, Morey RD, Speckman PL, Province JM (2012). "Default Bayes factors for ANOVA designs." *Journal of Mathematical Psychology*, **56**(5), 356–374. doi:10.1016/j.jmp.2012.08.001.

Examples

```
# Orthonormal contrasts
contr.orthonormal(c(1, 2))
contr.orthonormal(c(1, 2, 3))

# Mean difference contrasts
contr.meandif(c(1, 2))
contr.meandif(c(1, 2, 3))

# Independent contrasts
contr.independent(c(1, 2))
contr.independent(c(1, 2, 3))

# Ordered cumulative contrasts
contr.ordered_cumulative(c("low", "mid", "high"))
contr.ordered_cumulative_levels(c("low", "mid", "high"))
```

`density.prior` *Prior density*

Description

Computes density of a prior distribution across a range of values.

Usage

```
## S3 method for class 'prior'
density(
  x,
  x_seq = NULL,
  x_range = NULL,
  x_range_quant = NULL,
  n_points = 1000,
  n_samples = 10000,
  force_samples = FALSE,
  individual = FALSE,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  truncate_end = TRUE,
  ...
)
```

Arguments

x	a prior
x_seq	sequence of x coordinates
x_range	vector of length two with lower and upper display limits (used if x_seq is unspecified)
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
individual	should individual densities be returned (e.g., in case of weightfunction)
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation

, or a list containing the transformation function `fun`, inverse transformation function `inv`, and derivative of the transformation `jac`, evaluated on the original support. A custom transformation can also declare `output_support = c(lower, upper)`. Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

`transformation_arguments`
: a list with named arguments for the `transformation`

`transformation_settings`
: boolean indicating whether the explicitly supplied `x_seq` or `x_range` is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

`truncate_end`
: whether the density should be set to zero for endpoints of truncated distributions

`...`
: additional arguments

Details

Sample-based density estimates for continuous priors with finite support use boundary-reflected kernel density estimates. The plotting range controls the evaluation grid, while reflection is based on the prior's true truncation bounds. Automatic transformed prior-coordinate ranges keep the existing source range and equally spaced source grid. Where the existing marginal and transformed density route supports it, a source display endpoint with an infinite transformed image uses the prior's existing marginal tail quantile, controlled by `x_range_quant`. Prior-list plots use the union of participating component marginal quantiles. This is a finite display window; the prior support, density, probability masses and mixture weights are unchanged. Invalid atom images remain unavailable. When an endpoint quantile is unavailable or its transformed image is still not finite (for example the 0.995 quantile of a very wide prior under `"exp"`), the automatic display range of the transformed prior is unavailable: `density()` stops with `BayesTools_prior_range_unavailable`, parent `BayesTools_transformation_domain` and `BayesTools_transformation`, with `call = NULL`, and the message names the remedy of explicit display limits (`x_range` or `x_seq`; `xlim` in the plotting functions). In `plot_posterior()` only the prior curve is omitted, with a warning of class `BayesTools_prior_curve_unavailable` (see there), and the posterior is drawn alone. Explicit source or transformed coordinates keep their supplied interpretation. Individual weightfunction and ordered routes retain their existing transformation behavior and conditions. Non-individual PET/PEESE standard-error axes and weightfunction p-value axes retain their specialized interpretation. Declared point locations and finite builtin support bounds must have representable images. Finite-source exponential underflow, tanh saturation, positive-power underflow and nonfinite point images stop with `BayesTools_transformation_image_unavailable`, parent `BayesTools_transformation`, with `call = NULL` and fields `transformation`, `source_values`, and `images` for the matched failing finite sources. True zero under positive powers and infinite mathematical support limits retain their meaning. Continuous display grids retain their existing handling. Direct ordered Dirichlet-product densities use stable log-share arithmetic for literal, untruncated Normal totals with mean zero and finite positive SD, retaining the existing quadrature budget and tolerances. This guarded route does not certify tiny or nonrepresentable ranges. Every other total takes the same quadrature at each display value and accepts it when it converges within its tolerances and exceeds the absolute tolerance (the integral of a total concentrated away from

zero can miss the band where the integrand is positive and give zero); the display values it does not accept take the structural route of the level (its total times a Beta share), built without a numerical grid. A quadrature that does not converge is not repeated for the rest of the level. Small second Beta shapes remain a numerical risk of the quadrature. A display value whose density neither the quadrature nor the route resolves (an NA height inside the support) is left out of the level and reported by a warning of class BayesTools_prior_curve_unavailable with the fields unresolved_values, structural_values (the limits at support ends without a structural classification, which are not warned about) and levels.

Value

density.prior returns an object of class 'density'. For Dirichlet simplex priors it returns a named list of beta-marginal density objects, one for each simplex coordinate. Ordered priors return one component per factor level. When a level has both discrete and continuous probability, its atoms table stores exact location and mass values, while its continuous table stores a density already weighted to integrate to the remaining continuous mass over its full support. A clipped plotting grid preserves density heights; diagnostics$continuous_integral reports its numerical trapezoid integral, which can differ from the exact continuous mass because of clipping and grid discretization. Nonfinite continuous ordinates are omitted by the structural mixed-measure route; the curve is not renormalized. A supported scalar point total times a random allocation share has a continuous scaled-Beta distribution at intermediate levels, with declared atoms retained for zero-total branches, reference levels, fixed shares, and the full-total level. A spike-and-slab prior returns a list of class density.prior.spike_and_slab with the slab density (variable) and the density of the spike at zero (inclusion), each scaled by its prior mass. Under a transformation, a prior with inclusion probability 1 has no prior mass in the spike, returns inclusion = NULL, and does not require a representable image of the spike location (for example log(0)); any other inclusion probability does.

See Also

prior()

---

ensemble_inference *Compute posterior probabilities and inclusion Bayes factors*

Description

Computes prior probabilities, posterior probabilities, and inclusion Bayes factors based either on (1) a list of models, vector of parameters, and a list of indicators the models represent the null or alternative hypothesis for each parameter, (2) on prior model odds, marginal likelihoods, and indicator whether the models represent the null or alternative hypothesis, or (3) list of models for each model.

Usage

```
compute_inference(
  prior_weights,
```

```
  margliks,
  is_null = NULL,
  conditional = FALSE,
  on_failure = c("error", "drop", "zero")
)

ensemble_inference(
  model_list,
  parameters,
  is_null_list,
  conditional = FALSE,
  on_failure = c("error", "drop", "zero")
)

models_inference(model_list, on_failure = c("error", "drop", "zero"))
```

Arguments

prior_weights	vector of prior model odds
margliks	vector of natural-log marginal likelihoods
is_null	logical vector of indicators specifying whether the model corresponds to the null or alternative hypothesis (or an integer vector indexing models corresponding to the null hypothesis; use `0` or `integer(0)` when no models are null)
conditional	whether prior and posterior model probabilities should be returned only for the non-null models. Defaults to `FALSE`. Inclusion Bayes factors stay the unconditional inclusion odds; after conditioning, posterior inclusion is 1, so recomputing the Bayes factor from the renormalized probabilities would be `Inf`.
on_failure	policy for missing (`NA`) marginal likelihoods in models with positive prior probability. `"error"` aborts (the default), `"drop"` removes failed models from the prior model space and renormalizes, and `"zero"` explicitly assigns the failed models zero evidence. Non-default policies emit a warning and return audit metadata. Only `NA` is treated as a computation failure under this policy. In model lists, store a failed computation as `bridgesampling_object(NA)`, a result flagged as failed whose marginal likelihood is read as `NA`. A finite failure must be reported as `NA`; by contrast, `-Inf` is an intentional zero-evidence shortcut (for example via `bridgesampling_object()`) and quietly receives posterior model probability zero without consulting `on_failure`.
model_list	list of models, each of which contains marginal likelihood estimated with bridge sampling `marglik` and prior model odds `prior_weights`
parameters	vector of parameters names for which inference should be drawn
is_null_list	list with entries for each parameter carrying either logical vector of indicators specifying whether the model corresponds to the null or alternative hypothesis (or an integer vector indexing models corresponding to the null hypothesis; use `0` or `integer(0)` when no models are null) Exact matching unique list names bind by parameter name; other ensemble lists retain positional order. Marginal inference requires unnamed lists or unique complete names matching its requested parameters.

Details

Dropping every positive-prior-probability model stops because no finite marginal likelihood is available; it does not renormalize an empty model space. Original positive model odds retain their logarithmic declaration even when the visible probability rounds to zero. Evidence is centered before model probabilities and inclusion Bayes factors are evaluated. Finite evidence differences outside the logarithmic range stop with `BayesTools_numerical_condition`. The returned numeric fields remain ordinary probabilities; passing a visible zero to `inclusion_BF()` retains that function's exact supplied-zero meaning.

Value

`compute_inference` returns a named list of prior probabilities, posterior probabilities, and Bayes factors. `ensemble_inference` gives a list of named lists of inferences for each parameter, and `models_inference` returns a list of models, each expanded by the inference list.

See Also

mix_posteriors BayesTools_ensemble_tables

`fit_backend_fingerprint` *Fingerprint the loaded BayesTools fitting backend*

Description

Returns a deterministic fingerprint of the BayesTools implementation that owns the currently loaded namespace. Downstream packages can store this value with serialized fits and invalidate caches when the generic fitting backend changes.

Usage

```
fit_backend_fingerprint()
```

Details

The fingerprint covers the package's executable R and native surface and an explicit fit-backend schema version. It is captured when the namespace loads and memoized for the R session so source files edited afterward cannot silently restamp fits produced by the already loaded implementation. Development workflows should reload BayesTools after changing its backend.

The scope is intentionally conservative: any executable R/native or package build artifact change can change the fingerprint. Installed lazy-load databases and native libraries make the value build- and platform-local; it is not a cross-platform cache key. External runtimes and dependencies such as JAGS are outside this BayesTools implementation fingerprint.

Value

A length-one 32-character hexadecimal fingerprint, or `NA_character_` when the loaded implementation cannot be resolved.

format_BF *Format Bayes factor*

Description

Formats Bayes factor

Usage

```
format_BF(BF, logBF = FALSE, BF01 = FALSE, inclusion = FALSE)
```

Arguments

BF	Bayes factor(s)
logBF	log(BF)
BF01	1/BF
inclusion	whether the Bayes factor is an inclusion BF (for naming purposes)

Details

Valid live Bayes-factor carriers retain their canonical BF10 logarithms when reformatted into either direction and scale. This preserves finite logs and representable reciprocals when the natural display is zero or infinite. Replaced or arithmetically modified vectors follow ordinary numeric formatting.

Value

format_BF returns a formatted Bayes factor.

format_BF_from_log *Format Bayes factors from natural-log values*

Description

Builds a Bayes factor from natural-logarithm BF10 values without passing them through exp(). Output on the logarithmic scale (logBF = TRUE) and in the inverse orientation (BF01 = TRUE) is computed in log space, so Bayes factors beyond the double range (an exp(log_BF) equal to 0 or Inf) keep finite logarithms and representable reciprocals. format_BF() cannot recover them from the natural-scale numbers.

Usage

```
format_BF_from_log(
  log_BF,
  logBF = FALSE,
  BF01 = FALSE,
  inclusion = FALSE,
  bound_operator = NULL,
  diagnostics = NULL
)
```

Arguments

log_BF	numeric vector of natural-logarithm BF10 values (the Bayes factor in favor of the alternative hypothesis). Names are kept. Missing values (NA, NaN) are propagated.
logBF	whether the logarithm of the Bayes factor is returned.
BF01	whether the inverse Bayes factor (the BF01 orientation) is returned.
inclusion	whether the Bayes factor is an inclusion Bayes factor (for naming purposes).
bound_operator	NULL (all values exact), or a character vector with one element per value of log_BF: NA for an exact value, "<" or ">" when the value is an upper or lower bound of the Bayes factor. A logical vector of NA with one element per value (all values exact) is accepted as well. The operators refer to the BF10 values in log_BF and are stored in the orientation of the output (inverted when BF01 = TRUE, see invert_BF_bound_operator()).
diagnostics	NULL, or a list with one element per value of log_BF (elements can be NULL) holding numerical diagnostics of the individual Bayes factors, stored with the carrier.

Details

The result is the Bayes-factor carrier that format_BF() returns when it receives such a carrier: reformatting it with format_BF() keeps the canonical logarithms. A plain numeric vector given to format_BF() returns an unclassed vector (it has the class BayesTools_BF only while a non-missing bound operator is present). The carrier stores the logarithms in private attributes; BF_carrier_drop() removes them. Subsetting keeps the logarithms of the selected values, arithmetic on the carrier returns ordinary numbers, and replacing values removes the carrier attributes.

Value

format_BF_from_log() returns a numeric vector of class c("BayesTools_BF", "numeric") with the names of log_BF. Its public attributes are name (for example "BF", "1/BF", "Inclusion BF", "Exclusion BF", wrapped in log() when logBF = TRUE), logBF, BF01, and bound_operator.

See Also

format_BF() BF_carrier_drop() invert_BF_bound_operator()

Examples

```
# a Bayes factor beyond the double range keeps its finite logarithm
BF <- format_BF_from_log(c(a = 2, b = 800))
format_BF(BF, logBF = TRUE)
format_BF(BF, BF01 = TRUE, logBF = TRUE)
```

---

formula_add_intercept *Add an Intercept to a Formula*

Description

Converts a no-intercept formula to the corresponding formula with an intercept while preserving the formula environment. Additive, parenthesized, and unary-plus no-intercept encodings such as `- 1`, `+ 0`, and `0 +` are removed without editing transformed calls such as `I(x - 1)` or `offset(x - 1)`.

Usage

```
formula_add_intercept(formula)
```

Arguments

`formula`	a formula object.

Value

A formula object with an intercept.

---

geom_prior *Add prior object to a ggplot*

Description

Add prior object to a ggplot

Usage

```
geom_prior(
  x,
  xlim = NULL,
  x_seq = NULL,
  x_range_quant = NULL,
  n_points = 1000,
  n_samples = 10000,
  force_samples = FALSE,
  transformation = NULL,
```

```
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  show_parameter = if (individual) 1 else NULL,
  individual = FALSE,
  rescale_x = FALSE,
  scale_y2 = NULL,
  ...
)
```

Arguments

x	a prior
xlim	plotting range of the prior
x_seq	sequence of x coordinates
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation , or a list containing the transformation function fun, inverse transformation function inv, and derivative of the transformation jac, evaluated on the original support. A custom transformation can also declare output_support = c(lower, upper). Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.
transformation_arguments	a list with named arguments for the transformation
transformation_settings	boolean indicating whether the explicitly supplied x_seq or x_range is on the transformed scale. Automatically selected prior-coordinate ranges are always

	obtained on the source scale before the density family's existing transformation handling.
show_parameter	which parameter should be returned in case of multiple parameters per prior. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions, k selects the publication weight omega of the k-th p-value interval in ascending order, as show_figures in plot.prior() and the order of the summary tables; 1 is the reference weight fixed at 1.
individual	should individual densities be returned (e.g., in case of weightfunction)
rescale_x	allows to rescale x-axis in case a weightfunction is plotted.
scale_y2	scaling factor for a secondary axis. The default NULL reuses the probability mapping from the active mixed plot when one is available.
...	additional arguments

Value

geom_prior returns a ggplot2 layer for a continuous density (a list of layers for a weight function), and an overlay of class BayesTools_prior_overlay for a point mass, a mixture, or a spike and slab prior. Both are added to a prior plot with +; the overlay takes the scaling of the probability axis of the plot it is added to.

See Also

plot.prior() lines.prior()

---

geom_prior_list *Add list of prior objects to a plot*

Description

Add list of prior objects to a plot

Usage

```
geom_prior_list(
  prior_list,
  xlim = NULL,
  x_seq = NULL,
  x_range_quant = NULL,
  n_points = 500,
  n_samples = 10000,
  force_samples = FALSE,
  individual = FALSE,
  show_figures = if (individual) 1 else NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
```

```
  rescale_x = FALSE,
  scale_y2 = NULL,
  prior_list_mu = NULL,
  effect_direction = "positive",
  ...
)
```

Arguments

prior_list	list of prior distributions
xlim	x plotting range
x_seq	sequence of x coordinates
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
individual	should individual densities be returned (e.g., in case of weightfunction)
show_figures	which figures should be returned in case of multiple plots are generated. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions plotted individually by plot.prior() and plot_posterior(), figure k shows the publication weight omega of the k-th p-value interval in ascending order, the order in which summary tables (including RoBMA's) print the omegas; figure 1 is the reference weight of the most significant interval. In plot.prior(), a negative k omits figure -k (the default -1 omits the reference weight fixed at 1). For a two-sided weight function prior, the intervals are its two-sided p-value bins; tables on the one-sided p-value scale print these weights first and then repeat them for the mirrored intervals.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation , or a list containing the transformation function fun, inverse transformation function inv, and derivative of the transformation jac, evaluated on the original support. A custom transformation can also declare output_support =

c(lower, upper). Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

transformation_arguments
: a list with named arguments for the transformation

transformation_settings
: boolean indicating whether the explicitly supplied x_seq or x_range is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

rescale_x
: allows to rescale x-axis in case a weightfunction is plotted.

scale_y2
: scaling factor for a secondary axis

prior_list_mu
: list of priors for the mu parameter, required for the PET-PEESE regression bands (individual = FALSE) and refused otherwise: an individual plot shows the density of one coefficient and does not use it

effect_direction
: direction of the effect for PET-PEESE regression. Use "positive" (default) for mu + PET*se + PEESE*se^2 or "negative" for mu - PET*se - PEESE*se^2.

...
: additional arguments

Value

geom_prior_list returns an overlay of class BayesTools_prior_overlay, which is added to a prior plot with + and takes the scaling of the probability axis of that plot.

See Also

plot_prior_list() lines_prior_list()

---

harrell_davis_quantile

Harrell-Davis quantiles of posterior draws

Description

Estimates quantiles of equally weighted draws with the Harrell-Davis estimator (Harrell and Davis 1982), a weighted average of all order statistics. Applied pointwise to the columns of a draws-by-grid matrix, it gives smooth credible bands: the kinks that the empirical quantile (stats::quantile()) shows when the draws that define the quantile change from one grid point to the next are attenuated. The estimand is the same as that of the empirical quantile; in simulations of smooth curves the accuracy was similar (see the details).

Usage

```
harrell_davis_quantile(x, probs, names = FALSE)
```

Arguments

x	a numeric vector of draws, or a numeric matrix with the draws in rows and one quantity (e.g., a grid point) per column. Missing values are not allowed.
probs	probabilities in [0, 1], in any order; duplicates are allowed.
names	whether the result is named by the probabilities (as stats::quantile() names them). Defaults to FALSE.

Details

For m draws with order statistics y(1), ..., y(m) and a probability p in (0, 1), the estimate is the sum of w_i * y(i) with weights w_i = I_(i/m)(a, b) - I_((i-1)/m)(a, b), where I is the regularized incomplete beta function, a = p * (m + 1), and b = (1 - p) * (m + 1). The weights depend only on the number of draws and p and are computed once for all columns that need them. The estimate is exact for constant columns and does not overflow for finite draws. The probabilities 0 and 1 return the minimum and the maximum, and a single draw is returned as it is.

Caveats of the estimator:

- The quantiles of the columns are pointwise: the result is not a simultaneous band of the underlying curve.
- Smoothness gives no general accuracy guarantee relative to the empirical quantile, including at extreme probabilities (e.g., 0.0005). Increasing the number of draws reduces Monte Carlo error; smoothing alone does not remove it. In simulations of smooth curves (lines with normal coefficients, probability 0.025, 1,000 and 10,000 draws), the root mean squared error was about as large as that of the empirical quantile (median ratios of 0.92 and 0.97) and the roughness of the error along the grid was several times smaller; this is no guarantee for other draws or settings.
- The input draws have equal weights (there are no importance weights), but the estimator gives every draw a positive and unequal weight that depends on its rank and is largest for the draws closest to the target rank. A small number of draws (below roughly 200) and heavy tails can therefore move the estimate far from the empirical quantile.
- Draws with a point mass (an atom, e.g., a parameter fixed at zero in part of a model-averaged posterior) are not treated separately: within about two Monte Carlo standard errors (in probability) of the edge of a point mass, the estimate blends the point mass and the continuous draws.
- The weighted sum is computed in double precision. Its rounding error is of the order of the machine precision times the largest absolute draw. Where the sorted draws jump close to the target rank (e.g., at the edge of a point mass or in a sparse tail), the rounding of the cell boundaries i/m can make it larger by a factor that grows roughly in proportion to sqrt(m * p / (1 - p)). Measured binary-step examples with zeros and the top r draws equal to one give about 40 with m = 10000, r = 249, p = 0.975, and about 1,400 with m = 100000, r = 50, p = 0.9995. Differences between draws that are much smaller than this error (e.g., the middle draw of c(-1e16, 1, 1e16)) are not resolved.

An infinite draw has a positive weight at every probability strictly between 0 and 1, which makes the weighted average infinite (or undefined for infinite draws of both signs). A column that contains an infinite draw is therefore summarized by the empirical quantile of that column (stats::quantile(), type 7). If that quantile is NaN (an interpolation between draws of -Inf and Inf), or if the weights

of a probability cannot be computed numerically (at tiny probabilities, with a limit that depends on the draw count and distribution-function implementation), the function stops with an error of class `BayesTools_harrell_davis_undefined` (also of the family class `BayesTools_harrell_davis`). The weights are computed only when a column with finite draws needs them, so columns of infinite draws never raise the second error.

Value

a numeric vector of the same length as `probs` for a vector `x`, and a matrix with `length(probs)` rows and one column per column of `x` (keeping its column names) for a matrix `x`.

References

Harrell FE, Davis CE (1982). "A new distribution-free quantile estimator." *Biometrika*, **69**(3), 635–640. doi:10.1093/biomet/69.3.635.

See Also

`stats::quantile()`

Examples

```
set.seed(1)
draws <- rnorm(500)
harrell_davis_quantile(draws, c(.025, .5, .975))
quantile(draws, c(.025, .5, .975), names = FALSE)

# pointwise credible band of a line from draws of its intercept and slope
intercept <- rnorm(500)
slope     <- rnorm(500, sd = .5)
x_grid    <- seq(-1, 1, length.out = 5)
line      <- outer(intercept, rep(1, 5)) + outer(slope, x_grid)
harrell_davis_quantile(line, c(.025, .5, .975), names = TRUE)
```

---

hypothesis_alias_label

Helpers of hypothesis tables

Description

Small helpers that the entry points of a package share with `hypothesis_BF_model()`. `hypothesis_alias_label()` is the name under which the rows of a hypothesis table show a tested parameter: the first alias of the parameter other than its own name, or the parameter. `hypothesis_unique_named_warnings()` drops repeated warnings of a table (the same text for the same row).

Usage

```
hypothesis_alias_label(aliases, parameter)

hypothesis_unique_named_warnings(warnings)
```

Arguments

aliases	a list named by aliases, each element the parameter the alias names.
parameter	the name of the tested parameter.
warnings	a character vector of warnings, named by the rows of a hypothesis table they belong to.

Value

`hypothesis_alias_label()` returns a string. `hypothesis_unique_named_warnings()` returns the unique, nonempty warnings, or `NULL` without any.

See Also

`hypothesis_BF_model()`

---

hypothesis_ast *Versioned hypothesis syntax trees*

Description

`hypothesis_parse()` parses the restricted language used by `hypothesis_BF()` into a validated, serializable syntax tree. An explicit `vs` comparison requires two nonempty statements. `hypothesis_render()` renders that tree with token-safe quoting. `hypothesis_symbols()` returns exact parameter roots or an occurrence table. Quoted reserved names and ordinary identifiers retain their exact identity; numeric labels preserve the parsed values when rendered and parsed again.

`hypothesis_rewrite()` replaces only exact symbol roots. Function names, literals, level labels, and longer identifiers are never edited. `hypothesis_resolve()` resolves every occurrence through `parameter_catalog_resolve()` without accessing posterior draws. A bracketed symbol such as `mu_f[b]` or `(mu) f[b]` is first resolved as an exact catalog name or alias; otherwise its bracketed level is used as the catalog component of the parameter before the bracket. The bracket always holds a level label: `mu_f[2]` selects the level labelled `"2"`, never the second fitted coordinate. A symbol that does not resolve uniquely stops with the classed errors of `parameter_catalog_resolve()`, a hypothesis without parameter symbols (e.g. `"1 > 0"`) stops with an error of class `BayesTools_hypothesis_no_parameters`, and a level-qualified symbol whose level differs from `component` stops with an error of class `BayesTools_hypothesis_component_mismatch` (fields `symbol` and `component`); both are also `BayesTools_parameter_resolution_error`.

Usage

```
hypothesis_parse(
  hypothesis,
  catalog = NULL,
  namespace = NULL,
  component = NULL,
  simplify_names = FALSE
)

hypothesis_ast_schema()

hypothesis_render(ast)

hypothesis_symbols(ast, occurrences = FALSE)

hypothesis_rewrite(ast, mapping)

hypothesis_resolve(
  ast,
  catalog,
  namespace = NULL,
  component = NULL,
  simplify_names = FALSE
)
```

Arguments

hypothesis	character vector of hypothesis statements.
catalog	optional BayesTools_parameter_catalog. When supplied to hypothesis_parse(), exact non-syntactic public aliases are recognized without requiring the caller to add backticks, and a contrast-coefficient selector term{j} whose coordinate is a level stops with the error of class BayesTools_selector_unavailable (also BayesTools_hypothesis_target) that names the level form, as in parameter_catalog_resolve(), instead of failing to parse.
namespace	optional exact catalog namespace filter.
component	optional exact catalog component filter for unqualified symbols. A level-qualified symbol such as term[level] supplies its own per-occurrence component and must agree with this value when both are used.
simplify_names	whether to recognize centrally generated simplified random-effect aliases. Defaults to FALSE.
ast	a BayesTools_hypothesis_ast object.
occurrences	whether to return one row per symbol occurrence.
mapping	named character vector from exact old roots to new roots.

Value

hypothesis_parse() and hypothesis_rewrite() return a BayesTools_hypothesis_ast. hypothesis_render() returns character text. hypothesis_symbols() returns unique roots or an occurrence data frame. hypothesis_resolve() returns a versioned BayesTools_hypothesis_resolution containing one row per occurrence.

---

hypothesis_BF *Hypothesis Bayes Factors*

Description

Computes Bayes factors for scalar hypotheses written as expressions, for example "theta = 0", "theta > 0", or "theta = 0 vs theta > 0". hypothesis_BF() is a generic: the default method evaluates posterior draws and marginal posteriors, and the method for a fitted model (BayesTools_fit) tests the quantities of its parameter catalog by their public names (see Fitted models). Factor/posterior-list levels can be referenced as "mu_alloc[alternate] > mu_alloc[random]" when the marginal posterior carries joint prior information. Level names that contain brackets, such as the cut() level (0,1], are referenced by quoting the whole reference in backticks, "`mu[(0,1]]` > `mu[(1,2]]`", or with the parameter catalog's escaped level form mu["(0,1%5D"]. Point-null hypotheses use Savage_Dickey_BF when a marginal_posterior object is supplied, including precomputed qCMDE/IWMDE posterior ordinates when density_method = "precomputed". Level-specific marginal_posterior.* subclass vectors with attached density metadata are accepted as marginal posterior inputs; parent-level posterior_density, posterior_densities, posterior_ordinate, or posterior_ordinates metadata on a marginal-posterior list is reused for matching level hypotheses.

Usage

```
hypothesis_BF(posterior, ...)

## Default S3 method:
hypothesis_BF(
  posterior,
  prior = NULL,
  hypothesis,
  parameter = NULL,
  logBF = FALSE,
  BF01 = FALSE,
  seed = NULL,
  density_method = c("KDE", "normal", "precomputed"),
  columns = "default",
  ...
)

## S3 method for class 'BayesTools_fit'
hypothesis_BF(
  posterior,
```

```
  hypothesis,
  namespace = NULL,
  component = NULL,
  transform_scaled = FALSE,
  conditional = FALSE,
  logBF = FALSE,
  BF01 = FALSE,
  seed = NULL,
  density_method = "KDE",
  density_control = NULL,
  n_samples = 10000,
  columns = "default",
  ...
)

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

Arguments

posterior	posterior draws, a marginal_posterior, a marginal_inference object, or a data frame/matrix of posterior draws, or a model fitted with JAGS_fit() (a BayesTools_fit). Posterior draws must be finite and quantity names must be unique.
...	for the cbind() method, the tables to bind; unused by the as.data.frame() method. The method for draws takes no further arguments and stops on unknown ones; the method for a fitted model warns about them.
prior	prior draws for numeric/data-frame inputs, or for numeric draws a scalar BayesTools prior object (including mixture and spike-and-slab priors). Quantity names in draw tables must be unique. Ignored when posterior already contains deterministic prior density information.
hypothesis	character vector with scalar hypothesis statements written in the restricted grammar described in Details, or, for posterior draws and marginal posteriors, a validated object from hypothesis_parse(). The method for a fitted model takes character strings only and stops with the BayesTools_hypothesis_statement condition for a parsed hypothesis.
parameter	optional scalar quantity name for numeric vectors or marginal_posterior objects.
logBF	whether to display the Bayes factor on the log scale.
BF01	whether to display the inverse Bayes factor.
seed	optional finite seed used by downstream helpers that sample. The caller's random-number generator state is restored after the call.
density_method	posterior density source for point-null tests. "KDE" uses kernel density estimates. "normal" uses a normal approximation to the posterior density at the null. "precomputed" requires valid posterior_ordinate or posterior_density metadata (posterior_metadata()) and errors if no valid precomputed source

is available. KDE point-null tests use boundary reflection when exact support metadata is attached to the marginal posterior or to matched `posterior_density` metadata, and, for numeric draws with a prior object, at the bounds that prior object declares (its truncation, or the support of its family) when the draws lie within them; other numeric and data-frame expression tests use standard Gaussian sample KDE. For numeric draws with a prior object that declares point masses, the posterior ordinate is that of the absolutely continuous part: the share of the draws not at a declared point-mass location times the KDE of those draws. A point hypothesis on a linear expression of marginal-posterior parameters or levels (sums, differences, and constant multiples, e.g., `mu[B] - mu[A] = 0`; functions and powers only of constants) is tested as a Savage-Dickey ratio of that linear combination: the prior density and exact support come from the joint prior context and the posterior ordinate is the normal approximation (`"normal"`) or the KDE with boundary reflection at bounded supports (per mixture component, see Savage_Dickey_BF). An affine expression of a numeric quantity with a prior object uses the exact density of that transformed prior. Only user-supplied prior draws use the sample KDE of the prior expression draws, with a classed warning (see Value). Finite sample and KDE evaluation ranges are not treated as exact support, so finite point hypotheses outside those ranges use kernel-tail density estimates.

`columns` output columns. `"default"` returns `Alternative`, `Null`, `BF`, and `BF_error`. `"all"` also returns `prior`, `posterior`, and `method` columns. The `prior` and `posterior` columns are diagnostics, not always probabilities: point tests report density heights, region tests report odds, and transitive point-vs-region tests report `NA`. The default `BF_error` column is printed as `error%(BF)`. Individual columns are selected by these column names; other spellings are rejected.

`namespace, component, transform_scaled, conditional, density_control, n_samples`

arguments of the method for a fitted model, passed to `hypothesis_BF_model()`, which documents them: the component of the model that the statements refer to (`namespace`: the formula parameter such as `"mu"`, `"model"`, or `"random"`), the level of a factor term that references to the term without a level stand for (`component`, which combines with `namespace`), the scale (`transform_scaled = FALSE`, the default, tests the fitted scale as in `JAGS_estimates_table()`), the conditioning of the tested quantities on their inclusion events, the settings of the density backend, and the number of prior samples.

`x` a hypothesis Bayes factor table.

`row.names` optional row names for the complete exported frame.

`optional` logical; exported column names are always syntactic and unique.

Details

Level combinations require the same complete declared joint prior context. Fixed reference levels use the shared typed ordinate refusals; whole-factor requests retain results for eligible sibling levels. Region Bayes factors are calculated from their canonical log ratios even when intermediate diagnostic odds are outside the natural-scale range. A transitive ratio with both a numerically zero posterior point ordinate and zero posterior region mass is undefined and returns `NA` with a warning recorded in the result.

The hypothesis language deliberately accepts only a small, non-programmable subset of R expressions:

```
hypothesis := statement [ "vs" statement ]
statement  := point | region
point      := arithmetic ( "=" | "==" | "!=" ) finite_number
region     := relation
            | "(" region ")"
            | "!" region
            | region ( "&" | "|" ) region
relation   := arithmetic ( "<" | "<=" | ">" | ">=" ) arithmetic
```

Arithmetic expressions may contain parameter identifiers, finite numeric literals, parentheses, `+`, `-`, `*`, `/`, `^`, and the one-argument functions `abs()`, `exp()`, `log()`, `sqrt()`, `plogis()`, and `qlogis()`. Every constant arithmetic subexpression must evaluate to one finite number. A point value may be one finite numeric literal, optionally preceded by one unary sign, or another parameter expression. Symbolic right-hand sides are normalized to a difference from zero. Relations with a constant on the left and a parameter expression on the right are normalized to the equivalent parameter-left form.

Equality defines a point hypothesis and may appear only as the top-level relation of a statement. It cannot be parenthesized inside, negated, or combined as a region. Region relations may be parenthesized, negated, and combined with the element-wise operators `&` and `|`; `&&` and `||` are not supported. Backticks permit non-syntactic parameter names. In particular, escaped identifiers such as `` `Inf` `` refer to parameters, while unescaped `Inf`, `NaN`, `NA`, `TRUE`, and `FALSE` are reserved literals and are rejected. Whitespace-delimited `vs` (in any case) outside backticks is the separator word of a hypothesis, so a parameter named `vs` is written with backticks (`` `vs` ``), as `hypothesis_render()` writes it.

Form	Status	Reason
`theta = -0.5`	accepted	point with a finite literal
`theta = phi`	accepted	normalized to `theta - phi = 0`
`0 > theta`	accepted	normalized to `theta < 0`
`(theta > 0)`	accepted	parenthesized region
`!(theta > 0)`	accepted	negated region
`theta > 0 & abs(phi) < 2`	accepted	combined regions
`` `Inf` > 0 ``	accepted	escaped parameter identifier
`!(theta == 0)`	rejected	point equalities cannot be negated
`theta = 1 + 1`	rejected	constant point value is not one literal
`theta > 0 && phi < 1`	rejected	scalar boolean operator
`sin(theta) > 0`	rejected	function outside the whitelist

Prior region masses of deterministic prior densities use the structure that `prior_density_ordinate()` classifies, for regions whose relations are linear in the quantity (e.g., `theta > 0`, `2 * theta < 1`, and their combinations with `&`, `|`, and `!`): point masses contribute their exact probability, scalar and normal-sum priors their exact distribution functions, and Gaussian convolutions and conditional-normal scale mixtures the conditional-normal quadrature of the ordinate over the other term, split at the same breakpoints, with the Gaussian peak (where the ordinate uses one) at every finite region bound; mixture, model, and design-row components are summed with their probabilities. A quadrature that fails its diagnostics, or evaluates to exactly zero, stops with an error. Other prior

densities and regions use the prior-density grid, which resolves the region boundaries on that grid: a boundary is located by bisection within each grid cell where the condition changes, but region features narrower than the grid spacing may be missed.

Affine expressions retain an unshifted linear numerator and a checked constant divisor. Point inference evaluates that numerator at the translated null; diagnostic density heights retain the requested quantity units. Numerical coefficient, boundary or log-ordinate resolution loss raises `BayesTools_hypothesis_numerical_` (ordinate parent) or `BayesTools_hypothesis_region_numerical_unavailable` (region parent), also `BayesTools_numerical_unavailable` and `BayesTools_numerical_condition`. The log-resolution screen is a conservative floating-point heuristic, not an estimator or backend error bound. Finite log values may correspond to displayed natural-scale zero or infinity. `raw_log_BF` and `numerical_diagnostics` retain aligned diagnostics; BF formatting uses the live column carrier, which replacement, arithmetic and Math operations invalidate. Supported scalar affine and Boolean regions use deterministic prior laws even when prior draws are available, as do regions over explicit factor levels whose comparisons are all of one linear combination of the levels (e.g., `mu_fac[A] > mu_fac[B] + 0.5`), which use the exact structural mass of that combination. Unsupported regions (nonlinear expressions or comparisons of different combinations, e.g., `mu_fac[A] > 0 & mu_fac[B] > 0`) retain their sampled route, or their existing no-draw grid route. Deterministic prior numerical diagnostics (method, absolute error and convergence when available) can be inspected with `attr(result, "prior_numerical_diagnostics")`; they are row-aligned through subsetting. `BF_error` remains Monte Carlo-only and omits prior Monte Carlo variance only for the actual deterministic route. Accepted positive scalar prior-region probabilities retain their backend log probabilities for canonical BF ratios, including positive subnormal masses. Natural zero or non-finite prior masses remain unavailable.

Data-frame coercion returns a plain long frame with leading `component`, `parameter` (the displayed row identity), and `warning` fields. Statistical values retain full precision. Bayes-factor names are `BF10`, `BF01`, `logBF10`, or `logBF01` according to the displayed column attributes; `BF_error` retains its existing percent relative-error units. Declared nonmissing bounds are in `BF_bound_operator`. Unmatched/global warnings have component `"hypothesis/warnings"`. Compact printing retains its existing layout. `cbind()` binds the displayed columns of hypothesis tables (with their row names, at full precision) into a plain data frame.

Value

A BayesTools table of class `BayesTools_hypothesis_BF`. The `BF_error` column reports approximate relative Monte Carlo error percentage when available. Region odds errors are computed on `log(BF)` from prior/posterior region indicators using an iid delta-method approximation on the flattened draws; they do not adjust for MCMC autocorrelation. Prior region masses computed deterministically from a prior object's distribution function contribute no Monte Carlo error. Point-vs-region errors combine the available point-density and region-mass errors on the `log(BF)` scale. Point-null tests require a regular prior ordinate with finite log density at the null value that `prior_density_ordinate()` classifies exactly from the prior's structure, on every route. Otherwise they stop with a classed error condition, which callers should match by class rather than by message: `BayesTools_point_mass_at_null` (a prior point mass at the null), `BayesTools_infinite_ordinate`, `BayesTools_zero_ordinate`, `BayesTools_undefined_ordinate`, or `BayesTools_inexact_ordinate` (no exact structural ordinate, e.g., a nonlinear expression of a deterministic prior, a prior-density combination evaluated only on a numerical grid, a structural quadrature rejected by its diagnostics, or a density grid without recorded provenance). Each of these conditions also has class `BayesTools_hypothesis_ordinate`. For a whole-factor statement expanded into several quanti-

ties, known ordinate errors produce an NA row with method "unavailable" and the original reason; valid siblings are computed. Declared formula measure refusals of class BayesTools_formula_measure_unavailable behave likewise, preserving their original reason and diagnostics. Only the declared formula-measure cause allow-list is localized; metadata, refit, and programming errors propagate. Direct scalar and explicitly indexed tests remain strict. Known numerical prior-region errors behave likewise, with classes BayesTools_prior_region_mass_unavailable, BayesTools_prior_region_route_unavailable, BayesTools_prior_region_grid_unavailable, or BayesTools_prior_region_probability_rejected, all also BayesTools_hypothesis_region. Missing deterministic provenance for a region remains a metadata error, including in whole-factor tables. The specific BayesTools_transformation_image_unavailable leaf also produces an unavailable whole-factor row with its reason; other transformation and input errors remain strict. Scalar tests retain the original refusal. User-supplied prior draws (numeric or data-frame inputs without a prior object) have no structural prior density: their prior ordinate is the kernel (or normal) estimate of the prior expression draws, returned with a warning of classes BayesTools_inexact_ordinate and BayesTools_hypothesis_ordinate. Region tests require positive prior mass for every compared region. An implicit region statement is compared with its complement, which therefore also needs positive prior mass. Independently evaluated complements remain valid even when the other natural probability rounds to one. Explicit comparisons such as "theta > 0.5 vs theta > 0" accept an encompassing region with prior mass one. Rows are labelled by quantity; when several statements refer to the same quantity, the statement number is appended, e.g. theta (2). A call cannot combine statements that reference explicit levels of a factor (mu_fac[A] > 0) with statements that reference the whole factor (mu_fac = 0); it stops with an error naming the restriction, and the two kinds are evaluated in separate calls. A symbolic comparison on both sides uses its centered scalar expression: "theta = phi vs theta > phi" and "theta - phi = 0 vs theta - phi > 0" are supported. Reversed spelling "theta = phi vs phi < theta" remains conservatively unavailable; use the supported orientation or centered spelling. A reference to a quantity or level that posterior does not contain, and a parameter that it does not contain, stop with an error of class BayesTools_parameter_not_found (also BayesTools_parameter_resolution_error), as unresolved selectors of parameter_catalog_resolve() do; its fields alias and available name the unresolved and the available references.

Fitted models

hypothesis_BF(fit, "(mu) cor = 0.9", BF01 = TRUE, seed = 1) tests the quantities of the parameter catalog (parameter_catalog()) of a fitted model by the names that JAGS_estimates_table() shows, without backticks (coefficients such as "x" or "x_fac[B]", random-effect quantities such as "(mu) sd(intercept)" or "(mu) cor", scalar parameters such as "sigma", and the total and the allocation shares of an ordered term, "f{total}" and "f[share: mid]", also by their earlier names "f_ordered_total" and "f_ordered_allocation[mid]"), and statements with vs work as for draws. The total of a spike-and-slab or mixture prior has a point mass at its null: its point tests need conditional = TRUE, and the total of a point prior is fixed by the model. On the original scale (transform_scaled = TRUE) a total is tested when the scaling leaves the coefficients of its term unchanged; the shares are unitless. A point null on a quantity with a prior point mass at that value (a spike-and-slab coefficient at its spike) stops with BayesTools_point_mass_at_null; conditional = TRUE tests the quantity within its inclusion event, and region hypotheses work without it. The fit method evaluates the Savage-Dickey density ratio (Dickey 1971; Wagenmakers et al. 2010) at the posterior and prior ordinates of the tested quantity; hypothesis_BF_model() describes the engine, its refusals, and the density backends that evaluate point hypotheses (hypothesis_density_backend()). The result is the same as extracting the quantity (parameter_mixed_posterior() or as_mixed_posteriors() with marginal_posterior()) and testing its draws.

References

Dickey JM (1971). "The weighted likelihood ratio, linear hypotheses on normal location parameters." *The Annals of Mathematical Statistics*, 204–223. doi:10.1214/aoms/1177693507.

Wagenmakers E, Lodewyckx T, Kuriyal H, Grasman R (2010). "Bayesian hypothesis testing for psychologists: A tutorial on the Savage-Dickey method." *Cognitive Psychology*, **60**(3), 158–189. doi:10.1016/j.cogpsych.2009.12.001.

`hypothesis_BF_model` *Hypothesis Bayes factors of a fitted model*

Description

`hypothesis_BF_model()` is the engine behind `hypothesis_BF()` of a `BayesTools_fit`, exported for packages that test hypotheses on the parameters of their own models. It parses the statements with the parameter catalog of the model, resolves each to one tested parameter, plans how it can be evaluated (the draws and the prior density of the tested quantity, the classification of its point values), checks every plan against the requested density backend, evaluates the statements in groups that share a tested quantity, and binds the results into one table in the order of the statements. What the engine needs to know about the model comes from its provider (`hypothesis_provider()`); the methods to evaluate a point hypothesis are the density backends (`hypothesis_density_backend()`).

Usage

```
hypothesis_BF_model(
  object,
  hypothesis,
  namespace = NULL,
  component = NULL,
  transform_scaled = FALSE,
  conditional = FALSE,
  logBF = FALSE,
  BF01 = FALSE,
  seed = NULL,
  density_method = NULL,
  density_control = NULL,
  n_samples = 10000,
  columns = "default",
  parameter = NULL,
  conditional_omitted = FALSE
)
```

Arguments

`object`	a model object with a `hypothesis_provider()` method.

`hypothesis` character vector with hypothesis statements (see `hypothesis_BF()`). Names of the parameter catalog of the model need no backticks. The statements of a model are character strings: a parsed hypothesis (`hypothesis_parse()`) stops with an error of class `BayesTools_hypothesis_statement`.

`namespace` the component of the model the statements refer to, in the vocabulary of its provider (for a `BayesTools_fit`: `"auto"`, the formula parameters such as `"mu"`, `"model"`, or `"random"`); `NULL` selects the default.

`component` a catalog component (the level token of a factor term) passed to `hypothesis_resolve()`: a reference to a factor term without a level stands for this level, so that `"x_fac = 0.1"` with `component = "B"` is the statement `"x_fac[B] = 0.1"`. It combines with `namespace`, which selects the formula parameter (for example when the factor is shared by several formulas).

`transform_scaled` whether coefficients of scaled predictors, and random-effect SDs, variances, and correlations of such predictors, are tested on the original scale (`TRUE`) or on the fitted scale (`FALSE`, the default), as in `JAGS_estimates_table()`.

`conditional` whether to test each quantity conditionally on its inclusion event: spike-and-slab and mixture coefficients through `as_mixed_posteriors()` (`conditional` parameter, rule `"AND"`), random-effect quantities through `parameter_mixed_posterior()`. Without it a point null on a quantity with a prior point mass at that value stops with `BayesTools_point_mass_at_null`; region hypotheses are tested.

`logBF`, `BF01`, `seed`, `columns` see `hypothesis_BF()`.

`density_method` the name of a registered density backend (`hypothesis_density_backends()`); matching ignores case and accepts an unambiguous partial name. `NULL` selects the first backend of the provider (the stored method of a marginal-means provider).

`density_control` the settings of the density backend (a named list that the backend validates), or `NULL`.

`n_samples` the number of prior samples or grid points for the deterministic marginal prior densities and for the prior draws of random-effect quantities.

`parameter` for a marginal-means provider, the marginal-means parameter that the statements refer to.

`conditional_omitted` whether the caller omitted `conditional`; the engine then warns when the full ensemble of a mixture prior is tested (`full_ensemble()` of the provider).

Details

Every statement is planned before any is evaluated: the plan records the tested target (its weights on the fitted coefficients, its prior density and declared atoms), the classification of each point value by `prior_ordinate_status()`, and what refuses it. A statement that the requested density backend cannot evaluate stops with its refusal before anything is computed. The refusals are classed conditions; callers match the class and never the message:

`BayesTools_hypothesis_statement` a statement that has to be restated, or an argument that has to change: point expressions that are no direct reference (for example `"2 * mu = 0"`), non-linear expressions of factor levels, one quantity named by several names, and references that BayesTools refuses (`BayesTools_parameter_resolution_error`, for example `BayesTools_parameter_not_found` for an unknown name or level, which keeps its fields `alias` and `available`). `BayesTools_hypothesis_ambiguous` (parent `BayesTools_hypothesis_statement`) marks a statement whose references still name several model parameters, and `BayesTools_hypothesis_component_mismatch` a statement whose parameters belong to another component than the requested one; `BayesTools_hypothesis_parameter_mismat` a marginal-means statement that refers to another parameter than `parameter`.

`BayesTools_hypothesis_unavailable` a test that cannot be computed, with the more specific `BayesTools_hypothesis_fixed` (the quantity is fixed by the fitted model), `BayesTools_hypothesis_target` (no supported target or prior density), or `BayesTools_hypothesis_method` (the density backend is unavailable for the target). A point hypothesis whose prior ordinate is not exact, finite, and regular stops with the class of `prior_ordinate_status()` (for example `BayesTools_point_mass_at_null`) and `BayesTools_hypothesis_ordinate`.

A density backend owns its own conditions (a failed ordinate, a model it does not support).

Random numbers: the draws of every statement are extracted while it is planned, without `seed`; `seed` only reaches the evaluation of the Bayes factors of the draws (`hypothesis_BF()`) and the prior draws of random-effect quantities that have no prior density.

Value

A `BayesTools_hypothesis_BF` table, as returned by `hypothesis_BF()`, with the rows labelled as the statements were written. When an ordinate backend computed ordinates, the table can carry the `density_diagnostics` attribute of the backend.

References

There are no references for Rd macro `\insertAllCites` on this help page.

See Also

`hypothesis_BF()`, `hypothesis_provider()`, `hypothesis_density_backend()`, `hypothesis_quantities()`

---

hypothesis_cache_clear

Drop the caches of the hypothesis engine

Description

Drops the cached plans of `hypothesis_BF_model()` and `hypothesis_quantities()` (draws, selections, and plans are kept per model object for the session) and the caches of the registered density backends (`hypothesis_density_backend()`). A package calls it when a function the plans are built with has been replaced (for example after reloading).

Usage

```
hypothesis_cache_clear()
```

Value

NULL, invisibly.

---

hypothesis_density_backend

Density backends of fit-level hypothesis Bayes factors

Description

The density backend of a point hypothesis (the `density_method` of `hypothesis_BF()` and `hypothesis_BF_model()`) decides how the posterior ordinate at the null value is obtained. The built-in backends `"KDE"` and `"normal"` estimate it from the posterior draws, and `"precomputed"` uses ordinates that are stored on the draws (it is for draw inputs only, not for fitted models). A package that computes posterior ordinates itself, for example by a likelihood-aware estimator that cannot be written as a function of the draws, registers an *ordinate* backend. `hypothesis_density_backend()` creates a backend object, `register_hypothesis_density_backend()` adds it to the registry (a package registers its backends in `.onLoad()` and removes them with `unregister_hypothesis_density_backend()` in `.onUnload()`), and `hypothesis_density_backends()` lists the registered backends.

Usage

```
hypothesis_density_backend(
  name,
  type = c("ordinate", "draws", "stored"),
  available = NULL,
  control = NULL,
  attach = NULL,
  reuse = NULL,
  warnings = NULL,
  diagnostics = NULL,
  bind_diagnostics = NULL,
  rescale_failure = NULL,
  refusal_classes = NULL,
  clear_cache = NULL,
  method = NULL,
  policy = list(),
  advertised = TRUE,
  package = NULL
)

register_hypothesis_density_backend(backend, overwrite = FALSE)

unregister_hypothesis_density_backend(name)
```

hypothesis_density_backends()

Arguments

name	the name of the density method (a single string that starts with a letter).
type	"ordinate", "draws", or "stored".
available, control, attach, reuse, warnings, diagnostics, bind_diagnostics, rescale_failure, refusal_classes, clear_cache	the callbacks of the backend, functions or NULL for the default; see Details.
method	for a "draws" backend, the evaluation of hypothesis_BF() that it stands for: "KDE" or "normal".
policy	a list of declared settings of the backend.
advertised	whether hypothesis_quantities() lists the backend among the methods that evaluate a quantity.
package	the name of the package that provides the backend.
backend	a backend created with hypothesis_density_backend().
overwrite	whether registering replaces a registered backend of the same name. Replacing a built-in backend is not possible.

Details

A backend has a type: "draws" (evaluated from the posterior draws by hypothesis_BF() with the density method method), "ordinate" (the backend attaches the posterior ordinates of the tested values to the draws; they are then evaluated as "precomputed" ordinates), or "stored" (the ordinates are already attached to the draws). Matching of density_method to the registered names ignores case and accepts unambiguous partial names.

The engine calls the callbacks of a backend as follows; each is optional except attach of an ordinate backend.

available(object, plan, control) whether the backend can evaluate the statement plan of the model object (the capability_object of the provider) with the normalized control, the settings that control() returns for the call (for the stored marginal means of a model, for the settings of the call or the stored settings that the provider inherits). For the stored marginal means, settings that control() refuses reach available() as they were supplied, and the engine stops with the failure of control() after the plans are checked; for a fitted model, the failure of control() stops the call before any plan is checked. It is evaluated on every call and never cached. It returns NULL when the backend can, and otherwise a refusal: a list with the reason (a string) and optionally class (additional condition classes appended to BayesTools_hypothesis_method and BayesTools_hypothesis_unavailable) and fields (a named list of further fields of the condition). The engine stops with the refusal before any statement is evaluated.

control(control) the normalized density_control of the backend (defaults filled in), stopping on invalid settings. It is called with the settings of the call (or the stored settings of the provider) on every call with an ordinate backend, and for the settings that a call supplies to another backend, after the check_density_control() of the provider (hypothesis_provider()). The default accepts no settings: NULL and an empty list give NULL, anything else is an error.

`attach(posterior, request, control, cache)` the whole ordinate step of an ordinate backend. `posterior` is the marginal posterior of the tested quantity (a vector, or for the request of a factor level the list of the posteriors of the levels of the factor term); it returns it with the ordinates of the requested values stored in its `posterior_ordinate` metadata (`posterior_metadata()`, `posterior_ordinate_attribute()`, `posterior_ordinate_append()`), one entry per value, and the `parameter`, `conditional`, `conditional_rule`, and `condition_key` metadata of the draws, or stops with its own typed condition when an ordinate cannot be constructed. `cache` is an environment that lives for the evaluation of one group of statements, for objects that the requests of the group share. The `request` is a list with the elements `kind` (`"scalar"`: the values of the vector `posterior`; `"level"`: the values of the level `level`), `route` (the kind of the statement plan: `"scalar"`, `"linear"`, `"random"`, `"exp_affine"`, or `"marginal_mean"`), `object` (the model object, or the object that the provider names as its `capability_object`), `model` (the object of the provider), `parameter`, `label` (the tested quantity, as the table names it), `level`, `values` (the point values to compute), `conditional` (the conditioning of the draws: the parameter whose inclusion event the draws are conditioned on, or `NULL`), `condition` (the declared conditioning of the draws that the ordinates belong to, read from their metadata (`posterior_metadata()`, `"condition"`): the elements `conditional` (the effective conditioning, a character vector that is empty for averaged draws), `conditional_rule`, `condition_key`, `condition_event`, and `resolved_condition_event`, those the draws declare; an empty list for draws without a declaration), `spec` (the structure of the tested quantity that the provider declares: for example `type = "primitive"` or `"linear"` with the `weights` on the fitted coordinates and the `prior_density`), `display_transform` (the transformation from the evaluated to the tested scale, or `NULL`), `density_method` (the name of the backend), and `inference_type` (marginal means: the marginal inference the draws belong to).

`reuse(entry, request, control, posterior)` whether the ordinate `entry` that is stored on the draws `posterior` at a requested value is current for the `request` (the method, control, source of the fit, target, and conditioning agree), so that it is not computed again. `posterior` are the draws that carry `entry` (for the request of a factor level, the draws of that level), with their metadata; `request$condition` is their declared conditioning. The default reuses nothing: stored ordinates at the requested values are replaced.

`warnings(posterior, request)` the warnings of the ordinates stored on `posterior` (recursively over the levels of a list), as a data frame with the columns `warning`, `parameter`, `level`, `label`, and `value`. The engine puts each warning on the rows of the statements it belongs to. The default reads the `warning` element of the diagnostics of each stored ordinate. The `request` is the request that `attach` received for the group of statements, before the values whose stored ordinates `reuse` accepted were left out of `values`; for a group that is attached level by level (the levels of a factor term, marginal means) it is the request of the first level, and the requests of the group differ in their `level`, `values`, `label`, `spec`, and `condition` only.

`diagnostics(posterior, request)` the public diagnostics of the stored ordinates (a data frame; `request` as for `warnings`), stored as the `density_diagnostics` attribute of the table, and `bind_diagnostics(list)`, which combines the diagnostics of several groups of statements. Without `diagnostics` the table has no such attribute; the default of `bind_diagnostics` rowbinds the data frames.

`rescale_failure(condition, requested, evaluated, label, density_method)` restates a condition that `attach` raised for the evaluated values `evaluated` of a request for the values `requested` of a quantity that is evaluated through a transformation of itself (a variance through its standard deviation); it returns the condition to signal, or `NULL` to signal the original one.

`refusal_classes(cause)` additional condition classes for the refusals of the ordinate backends for a generic cause (`"original_scale"`, `"conditional_random"`, or `"random_target"`); they follow the classes of the refusal of the engine (for example `BayesTools_hypothesis_target` and `BayesTools_hypothesis_unavailable`). A provider names the cause of the refusal of a random-effect quantity (`hypothesis_provider()`).

`clear_cache()` drops the caches of the backend; the engine calls it when it drops its own caches.

`policy` is a list of the settings that the backend declares for its acceptance of ordinates (thresholds on its error estimates, for example). It is reported by `hypothesis_density_backends()` and not used by the engine: the backend applies its own acceptance rules in `attach`, and the engine only checks that the attached ordinates are valid (`posterior_ordinate_supports_bf()`).

Value

`hypothesis_density_backend()` returns an object of class `BayesTools_hypothesis_density_backend`. `register_hypothesis_density_backend()` returns the backend invisibly, and `unregister_hypothesis_density_backe` returns whether a backend was removed (invisibly). `hypothesis_density_backends()` returns a data frame with one row per registered backend in the order of registration: `name`, `type`, `package`, `advertised`, and the list column `policy`.

See Also

`hypothesis_BF()`, `hypothesis_BF_model()`, `hypothesis_provider()`

Examples

```
hypothesis_density_backends()

# a toy backend that evaluates the ordinate of a normal posterior exactly
normal_ordinate <- hypothesis_density_backend(
  name   = "exactNormal",
  type   = "ordinate",
  attach = function(posterior, request, control, cache) {
    draws    <- as.numeric(posterior)
    ordinate <- stats::dnorm(request$values, mean(draws), stats::sd(draws))
    posterior_metadata(posterior, "posterior_ordinate") <-
      posterior_ordinate_attribute(
        value = request$values, ordinate = ordinate,
        method = "exact normal", density_method = "exactNormal",
        parameter = attr(posterior, "parameter")
      )
    posterior
  }
)
register_hypothesis_density_backend(normal_ordinate)
hypothesis_density_backends()$name
unregister_hypothesis_density_backend("exactNormal")
```

`hypothesis_linear_target` *Compile a hypothesis linear in the levels of one parameter*

Description

When structured quantity labels are available, hypothesis tables display the compiled combination's label rather than its internal scalar name. Compiles point and simple region statements that all refer to one linear combination $t = \sum_l c_l L_l$ of levels L_l of the same parameter of one marginal posterior, e.g. a difference of two levels, a scaled level, or a weighted average of several levels. The returned scalar marginal posterior of t retains the exact joint-prior density and the structural conditioning metadata needed by downstream inference methods (e.g. precomputed posterior-ordinate estimators that need the combined linear weights).

Every referenced level must have one fixed named linear-weight row over the fitted coordinates and a declared posterior-atom status, and the levels must share a joint prior context and conditioning event. The prior density of t is the joint-context density at the combined weights $\sum_l c_l w_l$ shifted by the combined level offsets: on the fitted scale it equals `parameter_prior_density()` at those coordinate weights, and on the original scale of scaled formula coefficients `JAGS_formula_prior_density()` with the corresponding `weights`. It must be atom-free; by absolute continuity the posterior of t is then atom-free as well. Affine transformations already applied to the marginal levels retain their scale and offset; nonlinear transformed marginal levels, nonlinear expressions, row-varying level weights, and statements about different combinations are rejected.

Usage

```
hypothesis_linear_target(posterior, hypothesis, parameter)
```

Arguments

`posterior`	a factor-like marginal posterior containing the referenced levels.
`hypothesis`	hypothesis text or a `BayesTools_hypothesis_ast`. Each side is a point statement (`=`, `!=`) or a simple comparison (`<`, `<=`, `>`, `>=`) whose expressions are linear in the level references: sums and differences of level references and numbers, level references multiplied or divided by numbers, and parentheses.
`parameter`	exact parameter root used by the level references.

Value

A list with `posterior`, `hypothesis`, `parameter`, and `weights`. `posterior` is a scalar `marginal_posterior` of the linear combination whose draw metadata (`posterior_metadata()`) hold the combined `linear_weights` and `linear_offset`, its `prior_density` and `prior_context`, declared (empty) `atoms`, the `condition` of the levels, and, when the levels carry label parts, `quantities` labelling the target by the combination of the levels' labels (e.g. `(mu) 2*f[A] - f[C]`, see `parameter_labels()`); `hypothesis` is an equivalent AST written against that scalar target, `parameter` its name, and `weights` the combined linear weights. A target that cannot be certified stops with an error of class `BayesTools_linear_target_unavailable` (also `BayesTools_hypothesis_target`) whose field `reason` is `"posterior_atoms"` (the target's

prior is not structurally atom-free), "atom_declarations" (a level lacks its posterior-atom declaration), "prior_context" (no valid joint prior context, including level weights on columns the joint prior context does not contain), "weight_space" (the levels' linear weights are in different weight spaces), or "component_alignment" (the repeated mixture component keys of the levels do not match the original posterior rows). A reference to a level that posterior does not contain stops with an error of class BayesTools_parameter_not_found (also BayesTools_parameter_resolution_error), as in hypothesis_BF(). A translated threshold is the correctly rounded quotient of the statement's value and divisor. Numerical loss in coefficients, or a threshold quotient that is not finite, underflows or is subnormal, stops with BayesTools_hypothesis_numerical_unavailable (also BayesTools_hypothesis_ordinate, BayesTools_numerical_unavailable, and BayesTools_numerical_condition). Unrepresentable target units are unavailable; the compiler does not round or clamp them into a valid target.

---

hypothesis_linear_target_side

Linear form of one side of a hypothesis statement

Description

Reads one side of a statement of a hypothesis syntax tree as a linear combination of the levels of one parameter, $\sum_l c_l L_l$, compared with a value by the operator of the side. It is the per-side reading that hypothesis_linear_target() uses, and it identifies the combination of levels that a statement is about without a posterior: statements with identical coefficients refer to the same linear combination, however their level expressions are written. The coefficients are not normalized, so a multiple of a combination has other coefficients.

Usage

```
hypothesis_linear_target_side(side, parameter)
```

Arguments

side	one side (left or right) of a statement of a BayesTools_hypothesis_ast, for example hypothesis_parse("f[A] > f[B]")$statements[[1]]$left.
parameter	the parameter whose levels the side references, as a single non-empty string ("f" for f[A]).

Details

The side is a left or right element of a statement of a BayesTools_hypothesis_ast (see hypothesis_parse() and hypothesis_ast_schema()): a point side (=, !=), or a region side with a simple comparison (<, <=, >, >=). In a comparison the right expression is subtracted from the left expression. The expressions are linear in the level references of parameter (for example f[A]) and numbers: sums and differences, level references multiplied or divided by numbers, and parentheses. Levels of other parameters, other terms, and other kinds of sides stop with an error. The value is the correctly rounded quotient of the statement's constant and divisor. Numerical loss in the coefficients, or a value that is not finite, underflows or is subnormal, stops with an error of class BayesTools_hypothesis_numerical_unavailable, as in hypothesis_linear_target().

Value

`hypothesis_linear_target_side()` returns a list with

`operator` the operator of the side: `"="` or `"!="` for point sides, and the comparison operator for region sides.

`coefficients` a named numeric vector of the coefficients of the levels divided by `divisor`, named by level, sorted by level, without zero coefficients.

`numerator_coefficients` the coefficients of the levels before the division by `divisor`.

`value` the value that the combination is compared with, in the unit of `coefficients` (the value after moving the constants of the expressions to it and dividing by `divisor`).

`divisor` the divisor of the expression.

See Also

`hypothesis_linear_target()` `hypothesis_parse()`

Examples

```
side <- hypothesis_parse("f[A] - 2*f[B] > 0.5")$statements[[1]]$left
hypothesis_linear_target_side(side, "f")

side <- hypothesis_parse("(f[A] + f[B])/2 = 0")$statements[[1]]$left
hypothesis_linear_target_side(side, "f")
```

---

hypothesis_parse_point_reference

Hypothesis parser helpers

Description

Utilities for parsing point-null hypothesis references and level references in the same syntax accepted by `hypothesis_BF`.

Usage

```
hypothesis_parse_point_reference(hypothesis, allow_compound = TRUE)

hypothesis_parse_level_reference(text)

hypothesis_normalize_level_references(text)
```

Arguments

hypothesis	character vector with hypothesis statements or a validated BayesTools_hypothesis_ast.
allow_compound	whether compound left-hand side expressions such as "theta + 0 = 0" should be returned with direct = FALSE instead of rejected.
text	character vector with possible parameter[level] references. Formula interaction names such as factor:moderator[level] may be supplied without backticks.

Value

hypothesis_parse_point_reference() returns a data frame with columns hypothesis, side, symbol, parameter, level, value, operator, and direct. hypothesis_parse_level_reference() returns a data frame with columns input, symbol, parameter, level, and direct. hypothesis_normalize_level_refere returns a character vector.

---

hypothesis_provider *Hypothesis provider of a fitted model*

Description

hypothesis_provider() is the single hook through which hypothesis_BF_model() reaches a model. Its method returns a provider, a list of the callbacks and values that the engine needs for the statements it plans: the parameter catalog and its entries, the draws of the tested quantities, the random-effect quantities, and the density backends that can evaluate them. The method for a BayesTools_fit returns the default provider, which implements every callback with the functions of BayesTools (parameter_catalog(), as_mixed_posteriors(), marginal_posterior(), parameter_mixed_posterior(), JAGS_formula_coefficient_transform(), and JAGS_formula_prior_density()). A downstream model class that wraps a fit starts from hypothesis_provider(<its fit>) and replaces the callbacks that differ with utils::modifyList(), so that it overrides only what is specific to its model.

Usage

```
hypothesis_provider(object, ...)

## Default S3 method:
hypothesis_provider(object, ...)

## S3 method for class 'BayesTools_fit'
hypothesis_provider(object, ...)
```

Arguments

object	a model object.
...	further arguments for methods.

Details

A provider is a list of class BayesTools_hypothesis_provider; the engine checks it before use. Its elements are the following. The callbacks close over the model object.

kind "model" (default) or "marginal_means".

object the model object. It is hashed for the cache of plans and passed to the density backends.

fit its BayesTools_fit, from which the transforms of formula coefficients and their prior densities are read.

capability_object the object the density backends receive in available() and in their requests (default object).

build, refit_class the name of the build in the requests to refit a model with outdated metadata ("Refit the model with the current <build> build."), and classes that precede BayesTools_refit_required in those conditions.

caller, random_point_mass_advice, transform_scaled_default the name of the entry point in the reasons of refusals (default "hypothesis_BF()"); the sentence that ends the reason of a refused point null at a prior point mass of a random-effect quantity; and the scale of hypothesis_quantities() when transform_scaled is NULL.

ordinate_label how the messages of refusals name the ordinate backends (default "ordinate density methods").

component_argument the name of the argument through which the caller selects a value of components (default "namespace", the argument of hypothesis_BF_model() and of the method of hypothesis_BF() for a BayesTools_fit). The messages of the engine that name that argument ("The 'hypothesis' argument selects namespace = 'mu' but 'namespace' was set to 'random'.") use it; a model class that passes the selection under another name sets it, for example "component".

density_methods the names that density_method can match, in order (default: the registered backends that are not of type "stored").

components the selection vocabulary of the namespace argument as data: values (the accepted values, in the order of the error message), synonyms (a named character vector, for example c(location = "mods")), default ("auto"), refused (a named character vector of the reasons for which a component cannot be tested), random (the value that selects the random-effect quantities), and advice (the sentence that follows the message of an ambiguous statement).

metadata() a list with the catalog (a parameter_catalog(), possibly extended with parameter_catalog_extend(); the engine parses and resolves with it) and the entries: a data frame with one row per tested parameter and the columns quantity_id, parameter (the name of the tested parameter), component (a value of components$values), term, source, formula_parameter, role ("formula_coefficient_group" for a whole factor term, whose levels are its member_quantity_ids), quantity, status, fixed_value, aliases (a list column), and member_quantity_ids (a list column). The alias rows of the catalog of the entries carry the component as their namespace and component.

uncovered(resolution, quantity_ids, symbol, metadata) the refusal for resolved quantities that no entry covers: a list with reason and either type ("target" or "statement") or the complete condition class.

sample_parameters(parameter), mixed_posteriors(parameters, conditional, conditional_rule, transform_sc the draws of a fixed coefficient: the prior-list names whose draws are mixed (the parameter and the owners of the fitted coordinates that its prior density combines), the mixed posteriors

(as_mixed_posteriors()), the element of the samples that is the tested parameter, and the map of the formula coefficients to the fitted names.

random the callbacks for the random-effect quantities: select(parameter, transform_scaled) (a list with the element spec, a list with the label and status of the quantity, and any elements the other callbacks need), mixed_posterior(parameter, transform_scaled, conditional, selected) (the mixed posterior as the one-element list returned by as_mixed_posteriors()), support(selected) (the exact bounds on the scale requested from select(), which are those of the draws), density_target(parameter, transform_scaled, operation) (a list with the quantity that a density backend evaluates, its parameter_spec, and display_transform, or the refusal of a quantity without one: its reason, and optionally its type ("target", the default, "method", or "fixed": the engine's refusal BayesTools_hypothesis_<type> and its parent BayesTools_hypothesis_unavailable) and its cause ("random_target", the default, or "original_scale"), for which the density backend appends its classes (refusal_classes(cause))), sd_pair(selected) (the standard deviation of which a variance is the square, or NULL), zero_boundary(selected) (NULL, or a list with the statement that tests the omission of a component), defined_share(values, selected, conditional), defined_footnote(label, samples, posterior_defined, prior_defined), defined_draws(values, samples, label) (the treatment of the draws at which a quantity is undefined), and prior_draws(parameter, selected, transform_scaled, n_samples, seed) (prior draws of a quantity without a prior density with provenance).

point_mass_reason() NULL, or the text that replaces the reason of a point hypothesis at a prior point mass.

full_ensemble(parameter) whether omitting conditional tests the parameter within the full ensemble, including the models in which it is fixed by its null component (the engine then warns); combination_hint(plan): NULL, or text appended to the refusal of a linear combination of the levels of such a term.

check_plans(plans) NULL, or a function that validates the plans of all statements of a request and stops with a refusal.

check_density_control(density_method, density_control) NULL, or a function that checks the density_control of a call before the control() of the density backend sees it. It receives the name of the selected backend and the settings as supplied (a list, possibly empty, or NULL) wherever control() is called: on every call with an ordinate backend, and on every call that supplies settings to another backend (for stored marginal means also where the stored settings are normalized). It stops with the condition that refuses the settings; its value is ignored. Without it the built-in backends accept NULL and an empty list and refuse other settings with the message 'density_control' is not used by density_method = 'KDE'., and the control() of a backend validates its own settings.

A provider of kind = "marginal_means" describes the stored marginal means of a model instead: inference (the marginal inference with the elements averaged and conditional), model_averaged (logical; model-averaged point statements are tested on the conditional draws and cannot mix with region statements), capability_object (the fit that the backends compute ordinates from, or NULL), select(hypothesis, parameter) (the marginal-means parameter and its aliases that a statement names), density_method_default() and density_control_default(density_method) (used when the arguments are NULL), and metadata() for hypothesis_quantities().

Value

A provider, a list of class `BayesTools_hypothesis_provider`.

See Also

`hypothesis_BF_model()`, `hypothesis_density_backend()`

`hypothesis_quantities` *Available Hypothesis Quantities*

Description

Lists the parameter names and aliases that `hypothesis_BF()` accepts for a fitted model, with the tests that it runs on each of them.

Usage

```
hypothesis_quantities(object, ...)

## Default S3 method:
hypothesis_quantities(object, transform_scaled = NULL, ...)

## S3 method for class 'BayesTools_fit'
hypothesis_quantities(object, transform_scaled = NULL, ...)
```

Arguments

`object`	a fitted model with a `hypothesis_provider()` method, for example a `BayesTools_fit`.
`...`	unused; unused arguments warn.
`transform_scaled`	whether the rendered statements test coefficients of scaled predictors on the original scale (`TRUE`) or the fitted scale (`FALSE`). The default `NULL` is the default of the provider (`FALSE` for a `BayesTools_fit`).

Details

The eligibility columns render the plans that `hypothesis_BF_model()` executes (see its details) for the statements `<q> = <null>` (point tests), `<q> > <null>` (directional and interval tests), and, for factor terms, `<level> = <other level>` (level contrasts) of every quantity `q`. The null value is 0; when 0 is a prior point mass, a support boundary without a regular prior ordinate, or outside the support (for example, a gated random-effect standard deviation or a variance proportion), the point test is rendered at an interior value of the support instead (the midpoint of a bounded support, one unit inside a half-bounded support, and 1 otherwise): point hypotheses at such values follow the per-value rules of `hypothesis_BF_model()`. A factor term lists the point-test methods available for all of its levels that the contrast does not fix, and the contrast-test methods available for all pairs of its levels. Region tests do not use a density method. The methods are the registered density backends that are advertised (`hypothesis_density_backends()`; the normal approximation is a rough check and is not listed).

Value

A data frame with the columns `alias` (a name that `hypothesis_BF()` resolves to the row's quantity when the row's `component` is passed as the argument that selects the component of the model, which the provider names in its `component_argument`: the `namespace` argument for a `BayesTools_fit`; a name shared by several quantities of a component is not listed), `parameter`, `component`, `term`, `bracket` (a `parameter[level]` template for fitted factor terms, or a concrete level selector for each marginal-mean row), `point_test`, `direction_test`, `contrast_test` (`NA` for quantities without levels), `point_test_methods`, `contrast_test_methods`, and `reason` (the refusals of the rendered statements).

See Also

`hypothesis_BF()`, `hypothesis_BF_model()`

---

hypothesis_stop *Stop with a refusal of a hypothesis*

Description

Raises the error with which the hypothesis engine (`hypothesis_BF_model()`) refuses a statement or a test that it cannot compute, so that other entry points of a package refuse the same cause with the same condition classes.

Usage

```
hypothesis_stop(
  reason,
  type = c("target", "fixed", "method", "statement", "ambiguous"),
  class = NULL,
  fields = NULL
)
```

Arguments

`reason`	the message of the condition.
`type`	the kind of refusal: `"target"` (no supported target or prior density), `"fixed"` (the quantity is fixed by the fitted model), or `"method"` (the density method is unavailable for the target) have the classes `BayesTools_hypothesis_<type>` and `BayesTools_hypothesis_unavailable`; `"statement"` (the statement has to be restated) has the class `BayesTools_hypothesis_statement`, and `"ambiguous"` (the statement names several parameters) the classes `BayesTools_hypothesis_ambiguous` and `BayesTools_hypothesis_statement`.
`class`	additional condition classes appended to those of the type.
`fields`	a named list of further fields of the condition.

Value

Does not return; it signals the error.

See Also

`hypothesis_BF_model()`

`inclusion_BF` *Compute inclusion Bayes factors*

Description

Computes inclusion Bayes factors based on prior model probabilities, posterior model probabilities (or natural-log marginal likelihoods), and an indicator of whether the models represent the null or alternative hypothesis.

Usage

```
inclusion_BF(
  prior_probs,
  post_probs,
  margliks,
  is_null,
  on_failure = c("error", "drop", "zero")
)
```

Arguments

`prior_probs`	vector of prior model probabilities summing to one.
`post_probs`	vector of posterior model probabilities summing to one.
`margliks`	vector of natural-log marginal likelihoods.
`is_null`	logical vector of indicators whether the model corresponds to the null or alternative hypothesis (or an integer vector indexing models corresponding to the null hypothesis; use `0` or `integer(0)` when no models are null)
`on_failure`	policy for missing marginal likelihoods in models with positive prior probability. See `compute_inference()`.

Details

Supplying `margliks` as the input is preferred since it is better at dealing with under/overflow (posterior probabilities are very close to either 0 or 1). In case that both the `post_probs` and `margliks` are supplied, the results are based on `margliks`. If the prior probability of either the null or alternative hypothesis is zero, the Bayes factor is undefined and `NA` is returned.

Value

`inclusion_BF` returns a Bayes factor.

`interpret` *Interpret ensemble inference and estimates*

Description

Provides textual summary for posterior distributions created by mix_posteriors and ensemble inference created by ensemble_inference.

Usage

```
interpret(inference, samples, specification, method)

interpret2(specification, method = NULL)
```

Arguments

`inference`	model inference created by ensemble_inference
`samples`	posterior samples created by mix_posteriors
`specification`	list of lists specifying the generated text. Each inner list carries: (1) `inference` specifying the name of in the `inference` entry and optionally `inference_name` as a name to use in the text and `inference_BF_name` as a symbol to be used instead of the default `"BF"`. Finite-sample BF bounds can be supplied either as a `bound_operator` attribute on the numeric BF or as `inference_BF_bound_operator` / `BF_bound_operator` with value `"<"` or `">"`. A BF or bound that rounds to zero at the printed three decimals is printed as its reciprocal (for example `BF < 1/10000`), or as `exp(<log>)` when the reciprocal is beyond the double range. (2) `samples` specifying the name of in the `samples` entry and optionally `samples_name` as a name to use in the text, `samples_units` as a unit text to be appended after the estimate, and `samples_conditional` specifying whether the estimate is conditional or model-averaged.
`method`	character specifying name of the method to be appended at the beginning of each sentence.

Value

`interpret` returns character.

See Also

ensemble_inference mix_posteriors BayesTools_model_tables BayesTools_ensemble_tables

`interpret_records` *Normalize structured interpretation sources*

Description

`interpret_records` normalizes interpretation inputs from one or more structured sources into traceable evidence, estimate, header, note, prior, and footnote records. It is intended for downstream packages that already own the domain-specific ordering and wording of an interpretation, but want BayesTools to consistently extract table rows, preserve Bayes-factor metadata, and carry traceability information.

Usage

```
interpret_records(
  sources,
  plan,
  output = c("records", "text"),
  missing = c("error", "skip", "warn"),
  method = NULL,
  digits = 3
)

interpret_tables(sources, spec, ...)
```

Arguments

`sources`	named list of interpretation sources. Each source can be a `data.frame`/`BayesTools_table`, or a list with entries `data`, optional `type`, and optional per-source `schema`. Supported source types are `"table"`, `"record"`, and `"records"`. Table schemas can define columns such as `row`, `BF`, `central`, `lower`, `upper`, and metadata such as `BF_orientation`, `BF_scale`, `BF_name`, `BF_bound_operator`, `lower_prob`, `upper_prob`, `units`, and `conditioning`. `interval_level` is not accepted as input; normalized estimate records derive it from `upper_prob - lower_prob`, or retain `NA` when endpoint probabilities are unknown.
`plan`	ordered list specifying which records to create. Plan items can have `kind = "evidence"`, `"estimate"`, `"pair"`, `"test_estimate"`, `"for_each"`, `"header"`, `"note"`, `"prior"`, or `"footnote"`. Pair items contain `evidence` and/or `estimate` references. Dynamic `"for_each"` items generate records from source row order.
`output`	whether to return normalized `"records"` or a simple generic `"text"` rendering. Downstream packages should usually request `"records"` and render domain-specific text themselves.
`missing`	how missing optional sources or rows should be handled.
`method`	optional method name used only by the generic text renderer.
`digits`	number of digits used only by the generic text renderer.
`spec`	alias for `plan` used by `interpret_tables`.
`...`	additional arguments passed from `interpret_tables` to `interpret_records`.

Value

`interpret_records` returns a data frame with class `"BayesTools_interpret_records"` when `output = "records"`, or a character vector when `output = "text"`. Evidence records keep the supplied `BF_value` and `BF_scale`; `BF_canonical_value` is the Bayes factor on the natural scale in the canonical orientation (`0` or `Inf` for a log-scale value beyond the double range). The text rendering classifies and prints a log-scale value from its logarithm, so such values keep their evidence strength and are printed as `exp(<log>)`. `interpret_tables` is a convenience wrapper around `interpret_records`.

`invert_BF_bound_operator` *Invert the orientation of Bayes factor bound operators*

Description

Swaps `"<"` and `">"`: the bound operators of a Bayes factor in the BF10 orientation become those of its reciprocal in the BF01 orientation, and the other way round. Missing operators (exact values) are kept.

Usage

```
invert_BF_bound_operator(operator)
```

Arguments

`operator`	a character vector of bound operators (`"<"`, `">"` or `NA`), or a vector of only `NA` values. Other values are rejected.

Value

`invert_BF_bound_operator()` returns a character vector of the length of `operator` (without names) with the operators swapped.

See Also

`format_BF_from_log()` `format_BF()`

Examples

```
invert_BF_bound_operator(c("<", ">", NA))
```

`is.prior` *Reports whether x is a a prior object*

Description

Reports whether x is a a prior object. Note that point priors inherit the prior.simple property

Usage

```
is.prior(x)

is.prior.point(x)

is.prior.none(x)

is.prior.simple(x)

is.prior.discrete(x)

is.prior.vector(x)

is.prior.simplex(x)

is.prior.PET(x)

is.prior.PEESE(x)

is.prior.weightfunction(x)

is.prior.factor(x)

is.prior.ordered(x)

is.prior.orthonormal(x)

is.prior.treatment(x)

is.prior.independent(x)

is.prior.spike_and_slab(x)

is.prior.meandif(x)

is.prior.mixture(x)
```

Arguments

x	an object of test

Value

returns a boolean indicating whether the test object is a prior (of specific type).

Examples

```
# create some prior distributions
p0 <- prior(distribution = "point",  parameters = list(location = 0))
p1 <- prior_PET(distribution = "normal", parameters = list(mean = 0, sd = 1))

is.prior(p0)
is.prior.simple(p0)
is.prior.point(p0)
is.prior.PET(p0)

is.prior(p1)
is.prior.simple(p1)
is.prior.point(p1)
is.prior.PET(p1)
```

---

is_prior_bias *Reports whether x is a composed publication-bias prior*

Description

Reports whether x is a composed publication-bias prior

Usage

```
is_prior_bias(x)
```

Arguments

x	object to test.

Value

A logical value.

is_prior_phacking *Reports whether x is a p-hacking prior*

Description

Reports whether x is a p-hacking prior

Usage

```
is_prior_phacking(x)
```

Arguments

x	object to test.

Value

A logical value.

JAGS_add_priors *Add 'JAGS' prior*

Description

Adds priors to a 'JAGS' syntax.

Usage

```
JAGS_add_priors(syntax, prior_list)
```

Arguments

syntax	JAGS model syntax
prior_list	named list of prior distribution (names correspond to the parameter names)

Details

Syntax containing moment, inverse-moment, or inverse-gamma priors (dbt_invgamma) uses the BayesTools JAGS module. JAGS_fit() loads this module automatically; users running syntax returned by JAGS_add_priors() directly in JAGS need to load the module first.

Value

JAGS_add_priors returns a JAGS syntax.

JAGS_bridgesampling *Compute marginal likelihood of a 'JAGS' model*

Description

A wrapper around bridge_sampler that automatically computes likelihood part dependent on the prior distribution and prepares parameter samples. `log_posterior` must specify a function that takes a named parameter list, the data, and any additional arguments, and returns the log likelihood of the model part.

Usage

```
JAGS_bridgesampling(
  fit,
  log_posterior,
  data = NULL,
  prior_list = NULL,
  formula_list = NULL,
  formula_data_list = NULL,
  formula_prior_list = NULL,
  formula_scale_list = NULL,
  add_parameters = NULL,
  add_bounds = NULL,
  maxiter = 10000,
  silent = TRUE,
  formula_random_prior_list = NULL,
  formula_random_effects_compile_list = NULL,
  formula_random_effects_marginalize_list = NULL,
  bridge_context = FALSE,
  bridge_context_node_names = NULL,
  repetitions = 1L,
  method = c("normal", "warp3"),
  nonfinite = c("error", "drop"),
  cores = 1,
  seed = NULL,
  ...
)
```

Arguments

`fit`	model fitted with runjags, or posterior samples obtained with rjags-package. Derived-draw views are refused with `BayesTools_draws_view_inference_unavailable` (also `BayesTools_draws_view_unavailable`); use `fit$original_fit`.
`log_posterior`	function that takes a named list of samples, the data, and additional list of parameters passed as `...` as input and returns the log of the unnormalized posterior density of the model part

`data` list containing data to fit the model (not including data for the formulas)

`prior_list` named list of prior distributions (names correspond to the parameter names) of parameters not specified within the `formula_list`. For `BayesTools_fit` objects, stored non-formula priors are used when `prior_list = NULL`; a supplied prior must specify the same prior as the stored prior of the same name (distribution, parameters, truncation, `multiply_by`, and mixture components and weights; an error of class `BayesTools_refit_required` otherwise), while stored priors omitted from `prior_list` stay allowed. Formula priors are taken from the stored formula metadata: a formula prior supplied here is ignored when it specifies the same prior as the fitted formula prior (distribution, parameters, truncation, contrast, `multiply_by`, and mixture components and weights) and is an error otherwise.

`formula_list` named list of formulas to be added to the model (names correspond to the parameter name created by each of the formula). For `BayesTools_fit` objects with stored formula-design metadata, formula inputs can be omitted; if supplied, they are rebuilt only to check exact consistency with the fitted design and never replace fitted replay metadata. Fits without versioned formula-design metadata must be refitted: they stop with an error of class `BayesTools_refit_required` (see `JAGS_validate_fit_contract()`).

`formula_data_list` named list of data frames containing data for each formula (names of the lists correspond to the parameter name created by each formula). When supplied for a fitted formula, these data must exactly match the fitted original-scale formula source snapshot.

`formula_prior_list` named list of named lists of prior distributions (names of the lists correspond to the parameter name created by each of the formula and the names of the prior distribution correspond to the parameter names) of parameters specified within the `formula`

`formula_scale_list` named list of named lists for standardizing continuous predictors (names of the lists correspond to the parameter name created by each of the formula). Each entry should be a named list where continuous predictors with `TRUE` values will be standardized. Defaults to stored fit metadata when available and to `NULL` (no standardization) otherwise.

`add_parameters` character vector of additional monitored posterior parameter names to include in the bridge-sampling state and in the `parameters` object passed to `log_posterior`. These are for free stochastic JAGS nodes that are not owned by `prior_list`, such as special likelihood parameters, and `log_posterior` must add their prior density. Deterministic nodes computed from other parameters are not valid bridge coordinates; reconstruct them inside `log_posterior` or, for row-shaped external SD sources, with a `parameter_source(values = ...)` callback. Linearly dependent bridge coordinate draws are rejected. The `parameters` object is a named superset of prior-owned and additional values; user code should index it by name, for example `parameters[["tau"]]`. Parameters already covered by `prior_list` must not be listed here. For Dirichlet priors generated through BayesTools, add the monitored auxiliary `prior_par_eta_*` coordinates when needed, not the normalized simplex coordinates owned by the prior.

add_bounds — list with two named numeric vectors, "lb" and "ub", containing lower and upper bounds for every add_parameters entry. The vectors are named by the add_parameters entries (every entry, no others), contain no NA, and satisfy lb < ub; -Inf and Inf are allowed. Must be supplied whenever add_parameters is non-empty. Bounds of the coordinates BayesTools adds for formula random effects are inferred; a supplied bound for such a coordinate must agree with the inferred one.

maxiter — maximum number of iterations for the bridge_sampler

silent — whether the bridge-sampling progress output is suppressed, defaults to TRUE. Known iteration counts exceeding maxiter always signal a diagnostic warning, independently of progress output.

formula_random_prior_list — optional named list of prior_random() objects for random effects in formula_list. Bridge sampling for formula random effects requires the prior_random() interface because the stochastic bridge coordinates are the standardized latent effects and correlation primitives; a fit without their posterior samples (random_monitor(latent = TRUE)) stops with an error of class BayesTools_refit_monitoring (also BayesTools_refit_required). For BayesTools_fit objects with stored formula-design metadata, this can be omitted unless formula inputs are being supplied for a strict consistency check. Supplied callbacks never replace callbacks stored in the fitted formula design.

formula_random_effects_compile_list — optional named list of random_effects_compile() objects. When formula inputs are supplied for bridge-sampling rebuild/validation, this must match the fitted random-effect compilation policy; otherwise a fitted marginalized model would not be rebuilt as the same model.

formula_random_effects_marginalize_list — optional named list identifying fitted formula random-effect blocks whose covariance is required by the bridge target. Selected sampled blocks are integrated analytically only in the bridge target; blocks already fitted as marginalized expose the same covariance contract without changing the fitted model. Each formula-parameter entry can be a character vector of block names, which requests a dense covariance, or a named list with blocks and optional row_blocks. row_blocks must partition the formula rows and may not separate any structurally nonzero selected random-effect covariance; it requests the exact factorized block representation described below. With row_blocks, factor_state = TRUE additionally requests a compact exact contract that separates invariant factor plans from draw-varying coefficient factors and row scales. Every selected sampled Gaussian latent block is removed from the bridge coordinates and formula predictor. Already-marginalized blocks have no latent coordinates to remove. All SD, allocation, correlation, and other covariance parameters and their priors remain in the target. This requires bridge_context = "marginal" or the full context. The likelihood callback is responsible for adding the supplied covariance to its observation covariance.

bridge_context — whether the log_posterior callback receives a read-only bridge_context argument. Defaults to FALSE, preserving the historical log_posterior(parameters, data, ...) call. TRUE supplies the complete context. "nodes" supplies a

lightweight context containing only its exact flat named nodes vector. "marginal" supplies those nodes plus exact observation-level covariance contributions for bridge-marginalized Gaussian random effects. A context contains the current independent bridge state and BayesTools-resolved deterministic formula/random-effect nodes; it is not necessarily an original MCMC posterior row. For a formula block fitted with random_effects_compile(marginalized = ...), the SD coordinates and priors stay in the bridge target, but without a context (FALSE, the default) the callback receives neither the SD values nor the covariance: it must reconstruct them from primitive coordinates (add_parameters or its own evaluator) or request "marginal" or "full" (TRUE); the evidence of a callback that ignores the fitted random-effect covariance is the evidence of a model without it.

bridge_context_node_names
: optional character vector selecting an exact subset of the complete context's named nodes vector for bridge_context = "nodes" or "marginal". The default, NULL, retains every node. The complete context cannot be subset. Missing requested names are errors.

repetitions
: number of independent bridge-sampling repetitions.

method
: bridge transformation passed to bridge_sampler; either "normal" or "warp3".

nonfinite
: handling of non-finite repetition-level log marginal likelihoods. The default, "error", aborts with an error of class BayesTools_marglik_repetition_failure that names the cause of every failed repetition. "drop" aggregates only the remaining finite repetitions, emits a warning, and records every excluded repetition in the returned diagnostics; it stops with the same error class when no repetition is finite.

cores
: number of cores used by bridge_sampler. Defaults to one. Parallel workers must be able to load every package used by log_posterior; these can be supplied through the upstream packages argument in ....

seed
: optional integer seed for the random bridge-sampling proposal draws. The caller's random-number state (.Random.seed and RNGkind()) is restored afterwards. The default, NULL, draws from the caller's random-number stream, so the estimate depends on that stream.

...
: additional argument to the bridge_sampler and log_posterior function. The upstream-only use_neff logical control defaults to TRUE; an explicitly supplied value is passed to the sampler. The upstream-only controls packages, use_neff, varlist, envir, rcppFile, param_types, and verbose are not forwarded to log_posterior, including for exact zero-dimensional evaluation. Warnings signaled in the calling R process while evaluating callback arguments or the callback body are recorded in diagnostics$upstream_warnings and muffled on both the exact and sampler routes. Callback warnings from parallel worker processes are not relayed to this diagnostic field. Errors retain the Bridge sampling failed: context.

Details

Row-shaped external random-effect SD sources, such as random_sd_source("tau", shape = "row"), must be reconstructable during bridge sampling. A source computed from other parameters, for example tau[i] <- s * tau_data[i], is deterministic and must be supplied as parameter_source("tau",

shape = "row", values = function(parameters, data, n_rows) ..., inputs = ...). The values function is evaluated from the parameters it declares in inputs and the original-scale formula data stored at fit time; it must return finite, non-negative row values on the support of the model. Any callback data must therefore be included in formula_data_list when fitting. Data supplied only to JAGS_bridgesampling() do not extend or replace the fitted source snapshot. Posterior columns named tau[1], ..., tau[n] with non-negative lower bounds in add_bounds are valid only when every tau[i] is a free stochastic node whose prior density log_posterior adds. Supplying deterministic nodes as bridge coordinates makes the target improper; such rank-deficient bridge coordinate draws are rejected with an error. Rank is checked both on the supplied coordinates before bound transformations and on the transformed coordinates used by the proposal. Each rank check rescales a local diagnostic copy of finite, nonzero columns before standardization to avoid variance overflow or underflow. Supplied coordinates and sampler bounds retain their original values. Constant columns retain their existing sampler handling.

If every bridge coordinate is fixed by a point prior, JAGS_bridgesampling() evaluates log_posterior once at the reconstructed fixed parameter values. The returned marginal likelihood is exact and uses the aggregation rule "exact_zero_dimensional"; no bridge repetitions are performed. A target that declares no coordinate at all (no non-formula prior, formula prior, or add_parameters entry, for example raw draws with prior_list omitted) is an error of class BayesTools_marglik_input, not an exact evaluation.

use_neff is passed to bridge_sampler as the logical flag that upstream defines. **bridgesampling** computes the effective sample size itself, from the merged integrand matrix, and accepts no caller-supplied value: a numeric use_neff is a length->1 condition upstream and aborts the sampler. Per-chain effective sample sizes therefore cannot be injected here; the merged-chain ESS is conservative, because concatenating chains that have not mixed inflates the estimated autocorrelation and so lowers the ESS.

When bridge_context = TRUE, the callback receives an object of class BayesTools_bridge_context with fields state, state_matrix, nodes, prior_parameters, formula_prior_parameters, formula_parameters, add_parameters, random, marginalized_random, node_info, and metadata. The state field contains the independent bridge coordinates and the completed fixed-zero random-effect latents (latent coordinates of fixed-zero blocks, which are not bridge dimensions). The add_parameters field contains only the add_parameters supplied by the caller: the coordinates BayesTools adds for formula random effects (standardized latent effects and correlation primitives) are not additional parameters, so a callback that adds densities for additional parameters does not count them twice. They are part of state and nodes, and node_info labels them source = "random_latent", owner = "formula", role = "random_latent". The nodes field is a flat named numeric vector that also includes deterministic BayesTools-resolved nodes such as normalized Dirichlet allocation weights reconstructed from prior_par_eta_* coordinates. The node_info field records the owner and role of each exposed node. The random field contains structured random-effect block state including resolved SDs, allocation weights, correlations, Cholesky factors, covariance matrices where row-invariant, and row-indexed SD source values where applicable. Each random block is keyed by formula parameter and block name, and contains block_name, compile_mode, dimensions, levels, scale, allocation, correlation, covariance, latent, and nodes fields.

When bridge_context = "nodes", the callback receives an object inheriting from BayesTools_bridge_context with only the nodes field. Its values and names are identical to the nodes field in the complete context for the same bridge state. Random-effect replay metadata that is invariant across states is compiled once before bridge sampling.

When bridge_context = "marginal", the callback receives the same exact nodes vector plus

marginalized_random. If bridge_context_node_names is supplied, nodes is the requested exact subset in the requested order. The latter is keyed by formula parameter and contains representation, blocks, structures, row_names, and dimension. Without row_blocks, representation = "dense" and the covariance field equals the sum of ZGZ' over exactly the requested random-effect blocks. With validated row_blocks, representation = "factor"; row_blocks gives the exact observation partition and factors contains dense, ordinary grouped $Z_b G_b Z_b'$, or known-group covariance factors. This avoids materializing zero cross-block entries. For requests with factor_state = TRUE, the marginal context instead reports representation = "factor_state", with invariant factor_plans and draw-varying factor_states; the complete context and direct covariance evaluator retain the full representation. All representations define the same covariance without approximation.

Value

A BayesTools_marglik object that also has the class bridge of released versions, so that bridgesampling::bf(), post_prob() and logml() apply to it. For a single repetition, niter and method of the upstream result are kept at the top level, and the result prints as that object did (one line added when maxiter was exceeded). Failed, manual, exact and repeated results print a BayesTools summary. summary() and bridgesampling::error_measures() of a result forward to the sampler result stored in diagnostics$upstream (the bridge summary and error measures of one repetition, the bridge_list ones of repetitions); a failed, manual or exact result stores none and stops with an error of class BayesTools_marglik_sampler_result_unavailable (parent BayesTools_marglik_unavailable). logml is one scalar natural-log marginal likelihood, aggregated as the median of finite repetition-level log marginal likelihoods. repetitions contains one diagnostic row per bridge repetition, aggregation records the aggregation and failure policy, and diagnostics$upstream retains the original bridgesampling result. Top-level success records estimate availability. The separate logical iteration_limit_exceeded is TRUE when any known repetition count exceeds maxiter, FALSE when all counts are known and none exceeds it, and NA when counts are unavailable or partly unknown without a known excess. Manual/imported and exact zero-dimensional results have NA; historical schema-1 objects may omit the field, meaning unavailable telemetry. This flag does not assert convergence: counts can include upstream retries, including a successful retry whose total count exceeds maxiter.

The columns of repetitions are repetition, logml, niter (iterations used), mcse (Monte Carlo standard error of logml), finite (whether logml is finite), within_maxiter (whether niter did not exceed maxiter; TRUE when the iteration count is unknown), success (finite and within_maxiter), warning and error (the description of an exceeded iteration limit or of a non-finite estimate, NA otherwise), method, n_chains, and n_draws. Only non-finite repetitions are excluded from the aggregate, so aggregation$n_included counts the finite repetitions: a finite repetition that exceeded maxiter has success = FALSE and is still aggregated. Known excess counts always warn with class BayesTools_marglik_iteration_limit (parent BayesTools_marglik_diagnostic; fields repetitions, the affected diagnostic rows, maxiter, and upstream_warnings). A flagged result also prints a visible iteration-count diagnostic. Finite aggregation and top-level success availability are unchanged; legacy repetition success and within_maxiter retain their unknown-count convention.

Rank refusals have class BayesTools_marglik_rank_deficiency (parent BayesTools_marglik_input), with fields reason, rank, n_coordinates, n_draws, and dependent_coordinates. These describe the finite, varying diagnostic sample checked by the existing QR tolerance. The reason "insufficient_draws" means that n_draws <= n_coordinates prevents full centered rank; "dependent_coordinates"

means observed numerical rank loss despite a sufficient raw row count. These observations are nonexclusive and neither certifies a structural cause or a guaranteed remedy. Duplicate rows remain counted. Coordinate names are the complete, basis-dependent QR-pivot tail when available, otherwise `character(0)`; they do not identify uniquely faulty or deterministic nodes.

Condition classes: a non-finite repetition under `nonfinite = "error"`, and the absence of any finite repetition under `"drop"`, stop with an error of class `BayesTools_marglik_repetition_failure` (fields `repetitions`, the failed rows, and `upstream_warnings`). Errors of the likelihood callback keep their own classes and fields, with the `Bridge sampling failed:` context in the message and the muffled warnings in `upstream_warnings`. A state outside the support of the model (for example an out-of-support row-indexed random-effect SD source value) signals an error of class `BayesTools_marglik_out_of_support` while the parameters are reconstructed; the bridge target treats it as zero density, and the direct reconstruction functions (such as `JAGS_marglik_parameters_formula()`) let it escape. The two classes belong to no shared parent class; callers match the class, not the message.

A `NA` returned by `log_posterior` on the sampler route is no density. At a posterior draw the model cannot be evaluated where it was sampled: the call stops with an error of class `BayesTools_marglik_callback_na` (parent class `BayesTools_marglik_input`; field `n_posterior_na`, the number of affected posterior draws, and `upstream_warnings`). At a proposal draw the `NA` is evaluated as `-Inf` (zero density) and the call signals a warning of the same classes that reports the share of proposal draws (fields `n_proposal_na` and `n_proposal`); the estimate and its `success` are not otherwise affected. The counts are those that **bridgesampling** reports for the replaced evaluations. The exact zero-dimensional evaluation and `NA` priors keep their own errors.

This policy is defined for `method = "normal"`; `method = "warp3"` is outside it. `bridgesampling` evaluates the callback at warped draws and takes their exponentials in extended-range arithmetic, which stops on an `NA` with the unclassed upstream error "Bridge sampling failed: NAs are not allowed in subscripted assignments" (no `BayesTools_marglik_callback_na`), and the warnings that it words for `warp3` evaluations are not classified either. A callback used with `warp3` must not return `NA`.

Examples

```
## Not run:
# simulate data
set.seed(1)
data <- list(
  x = rnorm(10),
  N = 10
)
data$x

# define priors
priors_list <- list(mu = prior("normal", list(0, 1)))

# define likelihood for the data
model_syntax <-
  "model{
    for(i in 1:N){
      x[i] ~ dnorm(mu, 1)
    }
```

```
  }"

# fit the models
fit <- JAGS_fit(model_syntax, data, priors_list, seed = 1)

# define log posterior for bridge sampling
log_posterior <- function(parameters, data){
  sum(dnorm(data$x, parameters$mu, 1, log = TRUE))
}

# get marginal likelihoods
marglik <- JAGS_bridgesampling(
  fit, log_posterior, data, priors_list, seed = 1
)

## End(Not run)
```

`JAGS_bridgesampling_posterior` *Prepare 'JAGS' posterior for 'bridgesampling'*

Description

prepares posterior distribution for 'bridgesampling' by removing unnecessary parameters and attaching lower and upper bounds of parameters based on a list of prior distributions.

Usage

```
JAGS_bridgesampling_posterior(
  posterior,
  prior_list,
  add_parameters = NULL,
  add_bounds = NULL
)
```

Arguments

`posterior`	matrix of mcmc samples from the posterior distribution.
`prior_list`	named list of prior distributions. Names correspond to parameter names owned by BayesTools priors and determine the posterior columns retained for bridge sampling together with their bounds.
`add_parameters`	character vector of additional monitored posterior parameter names to retain in addition to the prior-owned parameters. These parameters are passed through as named entries and must be indexed by name by downstream likelihood code.
`add_bounds`	list with two named numeric vectors, `"lb"` and `"ub"`, containing lower and upper bounds for every `add_parameters` entry. The vectors are named by the `add_parameters` entries (every entry, no others), contain no `NA`, and satisfy `lb < ub`; `-Inf` and `Inf` are allowed. Must be supplied whenever `add_parameters` is non-empty.

Value

JAGS_bridgesampling_posterior returns a matrix of posterior samples with 'lb' and 'ub' attributes carrying the lower and upper boundaries.

JAGS_check_and_list *Check and list 'JAGS' fitting settings*

Description

Checks and lists settings for the JAGS_fit function.

Usage

```
JAGS_check_and_list_fit_settings(
  chains,
  adapt,
  burnin,
  sample,
  thin,
  autofit,
  parallel,
  cores,
  silent,
  seed,
  check_mins = list(chains = 1, adapt = 50, burnin = 50, sample = 100, thin = 1),
  call = ""
)

JAGS_check_and_list_autofit_settings(
  autofit_control,
  skip_sample_extend = FALSE,
  call = ""
)
```

Arguments

chains	number of chains to be run, defaults to 4
adapt	number of samples used for adapting the MCMC chains, defaults to 500
burnin	number of burnin iterations of the MCMC chains, defaults to 1000
sample	number of sampling iterations of the MCMC chains, defaults to 4000
thin	thinning interval for the MCMC samples, defaults to 1
autofit	whether the models should be refitted until convergence criteria specified in autofit_control. Defaults to FALSE.
parallel	whether the chains should be run in parallel FALSE

cores — number of cores used for multithreading if parallel = TRUE, defaults to chains

silent — whether the function should proceed silently, defaults to TRUE

seed — seed for random number generation by JAGS_fit(): an integer between -.Machine$integer.max and .Machine$integer.max. Draws are reproducible for the same seed, chains, and R random-number generator settings (RNGkind()), and, with parallel = TRUE, the same cores. Parallel draws need not equal serial draws with the same seed, because the backend passes each chain's initial state to its worker in a serialized text form. Initial values are drawn from the priors after set.seed(seed), and each chain's JAGS random-number seed is drawn from the same seed with sample.int(.Machine$integer.max, chains) (see JAGS_get_inits). A chain's JAGS seed therefore does not depend on the number of chains, but the JAGS generator does: with more than four chains it changes from "base::Super-Duper" to "lecuyer::RngStream", so the draws of a chain differ between fits with up to four and with more chains. The chains of adjacent seeds (e.g., seed and seed + 1) do not share random-number streams. Automatic restarts after a failed initialization (see restarts in autofit_control) draw their seeds from a separate stream of the same seed (the first L'Ecuyer-CMRG substream), so the i-th restart does not repeat the fit with seed + i. A seeded fit restores the caller's random-number state (.Random.seed and RNGkind()) when it returns. With seed = NULL, the initial values are drawn from one seed taken from the caller's random-number stream (see JAGS_get_inits), which the fit then continues to use.

check_mins — named list of minimal values for which should some input be checked. Defaults to:

chains 1

adapt 50

burnin 50

sample 100

thin 1

call — string to be placed as a prefix to the error call.

autofit_control

a list of arguments controlling the autofit function. Convergence thresholds and max_time$time must be finite and non-missing. Set a convergence threshold or max_time to NULL to disable that check. Possible options are:

max_Rhat maximum R-hat error for the autofit function. Defaults to 1.05. R-hat is the larger of the point estimate and the upper confidence limit of the potential scale reduction factor, whereas the summary tables show the point estimate (see JAGS_check_convergence()).

min_ESS minimum effective sample size. Defaults to 500.

max_error maximum MCMC error. Defaults to 0.01.

max_SD_error maximum MCMC error as the proportion of standard deviation of the parameters. Defaults to 0.05.

max_time list with fields time and unit, specifying when the automatic fitting function is stopped. The unit value corresponds to the units argument of difftime.

max_extend number of times after which the automatic fitting function is stopped.

sample_extend number of samples between each convergence check. Defaults to `1000`.

restarts maximum number of fitting attempts when the model fails to initialize; every failed attempt but the last is retried with new initial values (`restarts = 1` fits once, without a retry). A failed attempt that is retried is reported as a warning of the fit. Defaults to `10`.

check_indicators whether model indicator variables should be included in convergence checks, including when `monitor` selects a narrower parameter set. Auxiliary inclusion-probability coordinates remain excluded unless requested explicitly. Defaults to `FALSE`.

monitor optional character vector selecting parameters for convergence checks. Base names select all indexed elements. Requests are resolved against the monitored nodes that the summary tables show, including the derived and auxiliary ones that the default selection excludes. Base names that are not monitored are rejected before sampling (error class `BayesTools_parameter_not_found`); an indexed element such as `"mu[2]"` is resolved against the fitted columns after the first run and fails then when it does not exist. Defaults to `NULL`, which checks the default selection of `JAGS_check_convergence()`: the sampled parameters of the priors, `add_parameters` that depend on a stochastic node, and the stochastic nodes that formulas monitor (such as latent random effects and LKJ primitives). Deterministic nodes that formulas monitor (such as random-effect correlation matrices and their Cholesky factors) are derived and not checked by default. The role of every fitted coordinate is stored in the `convergence_role` column of `parameter_coordinates()`.

allow_not_assessable whether undefined diagnostics for requested sampled parameters may be ignored. Defaults to `FALSE`.

`skip_sample_extend`

whether `sample_extend` is allowed to be NULL and skipped in the check

Value

`JAGS_check_and_list_fit_settings` invisibly returns a list of checked fit settings. `JAGS_check_and_list_autofit_set` invisibly returns a list of checked autofit settings.

`JAGS_check_convergence`

Assess convergence of a runjags model

Description

Checks whether the supplied runjags-package model satisfied convergence criteria. Derived-draw views are refused with `BayesTools_draws_view_inference_unavailable` (also `BayesTools_draws_view_unavailable`) assess `fit$original_fit`.

Usage

```
JAGS_check_convergence(
  fit,
  prior_list = NULL,
  max_Rhat = 1.05,
  min_ESS = 500,
  max_error = 0.01,
  max_SD_error = 0.05,
  add_parameters = NULL,
  fail_fast = FALSE,
  check_indicators = FALSE,
  monitor = NULL,
  allow_not_assessable = FALSE
)
```

Arguments

fit	a 'BayesTools_fit' object created by JAGS_fit() with this version of BayesTools. Fits without its parameter map and fit contract (such as fits created by BayesTools 0.3.0) must be refitted: they stop with an error of class BayesTools_refit_required (see JAGS_validate_fit_contract()).
prior_list	optional named list of prior distributions (names correspond to the parameter names). When supplied, it must be the prior list stored with fit (attr(fit, "prior_list")); another list is an error. The classification of the fitted parameters always comes from the convergence roles of the fit's coordinates (see Details), not from this argument.
max_Rhat	maximum R-hat error for the autofit function. Defaults to 1.05. R-hat is the larger of the point estimate and the upper confidence limit of the potential scale reduction factor (coda::gelman.diag()), whereas the summary tables show the point estimate; the Rhat column of the diagnostics attribute holds the same value. With one chain, this criterion is skipped with a warning; the remaining enabled criteria are still assessed.
min_ESS	minimum effective sample size. Defaults to 500.
max_error	maximum MCMC error. Defaults to 0.01.
max_SD_error	maximum MCMC error as the proportion of standard deviation of the parameters. Defaults to 0.05.
add_parameters	vector of parameter names that are excluded from the default selection. Parameters named in monitor are checked even when they are listed here. Automatic fitting in JAGS_fit() and JAGS_extend() excludes nothing through this argument; it uses the default selection described in Details.
fail_fast	whether the function should stop after the first failed convergence check.
check_indicators	whether model indicator variables should be included in convergence checks. Binary indicators are checked as Bernoulli occupancies and categorical indicators are checked separately for every observed state. When monitor is supplied, eligible indicators are added to that selection. Defaults to FALSE.

`monitor` optional character vector selecting parameters for convergence checks. A base name selects all of its indexed elements. A name that is not available stops with an error of class `BayesTools_parameter_not_found`. During automatic fitting, base names are checked against the monitored nodes before sampling, whereas an indexed request such as `"mu[2]"` is resolved against the fitted columns after the first run and fails then when the element does not exist. Requests are resolved against the monitored parameters that the summary tables show, including derived and auxiliary ones that the default selection leaves out. Private implementation nodes that the summary tables hide cannot be requested. `NULL` selects the default parameters; `character()` requests no parameters.

`allow_not_assessable` whether requested sampled parameters with undefined diagnostics may be ignored. Defaults to `FALSE`. A sampled column that never changes when its chains are pooled, including a constant model indicator, is not evidence of convergence and remains not assessable, as does a diagnostic whose computation failed. A diagnostic computed on draws that vary is a value, not an undefined result, and is assessed even when it is infinite or zero: chains that are each constant but at different values give an R-hat of `Inf` and an ESS of 0, which fail `max_Rhat` and `min_ESS` whatever this argument is. Their MCMC error (the standard error SD / sqrt(ESS) of an ESS of 0) is `Inf`, so `max_error` and `max_SD_error` fail as well.

Details

`JAGS_fit()` assigns every fitted coordinate one convergence role, stored in the `convergence_role` column of `parameter_coordinates()` and derived from the declared model, never from the draws:

`"sampled"` stochastic parameters, including the user's `add_parameters` that depend on a stochastic node; checked by default.

`"indicator"` declared model indicators of mixture, spike-and-slab, and variance-allocation priors; checked with `check_indicators = TRUE`.

`"structural"` declared constants: point priors, one-state indicators, reference and fixed publication-weight bins, a p-hacking kind shared by every mixture branch, the point total of an ordered prior and the coefficients it fixes, unit correlation diagonals and Cholesky constants, monitored fully observed data, and monitored deterministic nodes (`<-` or `=`) whose ancestors in the model syntax are all data or constants. They are reported as `"structural_constant"`.

`"derived"` deterministic functions of sampled nodes that BayesTools generates for formulas (such as random-effect correlation matrices, their Cholesky factors, and derived latent effects or SDs) and point priors whose location is an expression; checked only when named in `monitor`.

`"auxiliary"` inclusion probabilities and the backend anchor of models without monitored parameters, checked only when named in `monitor`, and private implementation nodes that the summary tables hide (such as Dirichlet gamma draws), which are never checked.

Mirrored two-sided publication-weight bins are hidden in the same way. The default selection (`monitor = NULL`) consists of the sampled and structural parameters, and of the indicators when `check_indicators` is set. Automatic fitting in `JAGS_fit()` and `JAGS_extend()` uses the same selection.

Value

JAGS_check_convergence returns a boolean indicating whether all requested, assessable parameters satisfy the enabled criteria. An explicitly empty monitor returns logical(0) rather than claiming convergence. When no parameters remain available (empty sample columns and no structural parameters) or the selection is empty, the function returns TRUE: there is nothing assessable, which is treated as vacuously satisfied. The diagnostics attribute contains one row per available parameter and classifies it as "assessable", "structural_constant", "not_assessable", "not_requested", or "not_checked" when fail_fast = TRUE stops before reaching that parameter. Its Rhat column is the larger of the point estimate and the upper confidence limit of the potential scale reduction factor (see max_Rhat). The errors attribute carries the failed checks: one message per failed criterion, with the most extreme value, as in BayesTools 0.3.0 (for example "R-hat 1.234 is larger than the set target (1.05)."), followed by the sentence "Failing parameters: 'mu', 'tau'." that names the failing parameters (the first five and the number of the others), and one message for every parameter whose enabled diagnostics are not assessable.

See Also

JAGS_fit() parameter_coordinates()

Examples

```
## Not run:
# simulate data
set.seed(1)
data <- list(
  x = rnorm(10),
  N = 10
)
data$x

# define priors
priors_list <- list(mu = prior("normal", list(0, 1)))

# define likelihood for the data
model_syntax <-
  "model{
    for(i in 1:N){
      x[i] ~ dnorm(mu, 1)
    }
  }"

# fit the models
fit <- JAGS_fit(model_syntax, data, priors_list)
JAGS_check_convergence(fit)

## End(Not run)
```

JAGS_deterministic_nodes *Generated deterministic nodes*

Description

JAGS_deterministic_nodes() lists the deterministic nodes that BayesTools generates in the JAGS model of a fit (other than the internal intermediates of random-effect blocks, which the random-effect reconstruction recomputes), with the family that defines them, the coordinates they produce, and the coordinates they are computed from. JAGS_evaluate_deterministic() recomputes their values from draws of those coordinates. JAGS_deterministic_evaluator() resolves the nodes of a fit once and returns a function that evaluates them on draws, for repeated evaluations (e.g. one draw at a time).

Usage

```
JAGS_deterministic_nodes(fit)

JAGS_evaluate_deterministic(fit, draws = NULL, nodes = NULL)

JAGS_deterministic_evaluator(fit, nodes = NULL)
```

Arguments

fit	a model fitted with JAGS_fit().
draws	draws to evaluate the nodes on: a numeric matrix with one column per coordinate (for example posterior draws with some columns replaced, or prior draws from transform_prior_samples() with formula_scale = list()), an mcmc or mcmc.list object, or a named numeric vector holding a single draw. Coordinates that have a point prior do not need a column. Defaults to the posterior draws of fit.
nodes	optional character vector of node names (the node column of JAGS_deterministic_nodes()) to evaluate. By default, every node whose dependencies are available in draws is evaluated.

Details

Each generated node belongs to one family, which defines both the JAGS syntax of the node and the R evaluator used everywhere BayesTools needs its value (prior draws, parameter-catalog quantities, bridge sampling, marginal-likelihood parameters, prediction, marginal posteriors of formula parameters, random-effect unscaling, and convergence roles):

"random_sd" random-effect SDs derived from a variance allocation (random_variance_allocation()): the source SD (the allocation's own SD or an external scalar SD source) times the chain of allocation factors from the root allocation, sqrt(w[i]) (total variance) or sqrt(K * w[i]) (mean variance) of the Dirichlet weights and the inclusion gates. The family also contains the parent allocation components that a child allocation splits further.

`"random_rho"` scalar correlations of structured random-effect blocks sampled on the Fisher-z (`rho = tanh(z)`) or logit scale (`rho = lower + (upper - lower) * plogis(z)`).

`"lkj"` the Cholesky factor `L`, the correlation matrix $R = LL^T$ (with an exact unit diagonal), and the monitored partial correlations `cpc = 2 u - 1` of an LKJ-Cholesky block, computed from its primitives `u` with the kernel of the BayesTools JAGS module.

`"omega"` the publication weights of a weight-function prior (alone, as the selection of `prior_bias()`, or as branches of a publication-bias mixture) on the one-sided global p-value bins, with the bins of two-sided weight functions mirrored. The dependencies are the free coordinates of the weights (`eta`, the monitored `omega[2]` of a binary cumulative weight function, or the independent `omega` or `log_omega`); mixture branches are read from their own component nodes and the `bias_indicator`.

`"phacking"` the nodes of the p-hacking branches of a `prior_phacking()`, `prior_bias()`, or publication-bias mixture prior (experimental): the form code `phack_kind`, the null masses `pi_null = alpha * a` and `beta_null = alpha * b` with the constants of the form, the `z` geometry `phack_z_source` and `phack_z_dest`, and the `sel_vector_rule` of the selection kernel. A publication-bias mixture composes them from its branches (zeros for a branch without p-hacking) and the `bias_indicator`, and adds the severity `alpha` composed from the component severities `alpha_component_k`. `pi_null` and `beta_null` are computed from the coordinate `alpha`, so a fit that monitors it needs no component severities.

`"prior_mixture"` parameters with spike-and-slab (`p = p_variable * p_indicator`) or mixture priors (`p = sum_k p_component_k * (p_indicator == k)`), and the PET and PEESE terms of publication-bias mixtures (`PET <- PET_1 * equals(bias_indicator, k)`), per coefficient of factor priors. Only the components that are active in some draw need to be available; literal point components are constants. Multi-slice ordered spike totals use indexed slab sources and one shared inclusion state.

`"prior_dirichlet"` the coordinates of an ordinary Dirichlet prior, `theta[i] = eta[i] / sum(eta)`, from its gamma auxiliaries `eta`. The Dirichlet weights of a variance allocation belong to the allocation family.

`"prior_mt"` the coordinates of a multivariate-t prior (also of an orthonormal or meandif multivariate-t factor, or of a component of a mixture), `theta[i] = z[i] / sqrt(s) + location`, from the normal coordinates `z` and the scale coordinate `s`. A location that is an expression leaves the node unavailable to the evaluator.

`"ordered_allocation"` normalized Gamma coordinates of an ordered allocation. Gamma values take precedence over supplied normalized weights. Replay accepts nonnegative Gamma values with a positive row sum; bridge stochastic-source checks retain their positive-coordinate rule.

`"ordered_coefficient"` each ordered increment from its declared total slice and factor allocations. Shared allocation keys have one node.

`"point_expression"` scalar expression-valued formula point coefficients and their declared expression-point parents. Replay uses the retained location AST, owned data, and available declared parents. It replaces stale coefficient monitors before dependent formula replay. A coefficient monitor can supply a formula when its parents are absent; that coefficient is then unavailable for explicit recomputation.

`"linear_predictor"` the linear predictor of a formula parameter on the fitted rows: the intercept (or its logarithm), the continuous and factor terms with their `multiply_by` multipliers, the `expression()` terms, and the contributions of the sampled random-effect blocks, evaluated

with JAGS_evaluate_formula(). It is evaluated only by JAGS_evaluate_deterministic(), and only when the standardized latent effects of the sampled blocks are among the draws. A block whose SDs come from a row-shaped external source (random_sd_source(..., shape = "row")) depends on the source rows source[1], ..., source[N], or, when the source's values function reconstructs them, on the inputs that function declares (parameter_source(inputs =)), and on the Dirichlet weights and inclusion gates of its allocation.

Nodes are evaluated with the arithmetic of the R evaluator, which reproduces the JAGS monitors exactly or to the last bits of floating-point rounding. Successful parent outputs replace stale derived columns before dependent nodes are evaluated, for every family. Point coefficient replay uses the bounded formula-expression arithmetic subset, requires one finite scalar per draw, and does not bind the loop index i. Unsupported replay syntax retains valid fitting and monitored-value routes, and reports BayesTools_formula_point_unavailable when replay is requested. Missing affected declaration ownership requires refitting.

Value

JAGS_deterministic_nodes() returns a data frame with one row per node and the columns node (node name), family, parameter (the formula parameter or prior name the node belongs to), block (the random-effect block, NA otherwise), coordinates and dependencies (list columns of coordinate names), and monitored (whether all coordinates of the node are monitored in fit).

JAGS_evaluate_deterministic() returns a numeric matrix with one row per draw and one column per coordinate of the evaluated nodes.

JAGS_deterministic_evaluator() returns a function with one argument, draws (in the forms accepted by JAGS_evaluate_deterministic(); a named numeric vector for one draw), that returns the matrix JAGS_evaluate_deterministic(fit, draws, nodes) returns, with the nodes of fit resolved when the evaluator is created. JAGS_evaluate_deterministic() is this evaluator applied once.

See Also

JAGS_fit() parameter_coordinates() transform_prior_samples()

JAGS_diagnostics *Plot diagnostics of a 'JAGS' model*

Description

Creates density plots, trace plots, and autocorrelation plots for a given parameter of a JAGS model.

Usage

```
JAGS_diagnostics(
  fit,
  parameter,
  type,
  plot_type = "base",
```

```
  xlim = NULL,
  ylim = NULL,
  lags = 30,
  n_points = 1000,
  transformations = NULL,
  transform_factors = FALSE,
  short_name = FALSE,
  parameter_names = FALSE,
  formula_prefix = TRUE,
  ...
)

JAGS_diagnostics_density(
  fit,
  parameter,
  plot_type = "base",
  xlim = NULL,
  n_points = 1000,
  transformations = NULL,
  transform_factors = FALSE,
  short_name = FALSE,
  parameter_names = FALSE,
  formula_prefix = TRUE,
  ...
)

JAGS_diagnostics_trace(
  fit,
  parameter,
  plot_type = "base",
  ylim = NULL,
  transformations = NULL,
  transform_factors = FALSE,
  short_name = FALSE,
  parameter_names = FALSE,
  formula_prefix = TRUE,
  ...
)

JAGS_diagnostics_autocorrelation(
  fit,
  parameter,
  plot_type = "base",
  lags = 30,
  transformations = NULL,
  transform_factors = FALSE,
  short_name = FALSE,
  parameter_names = FALSE,
```

```
  formula_prefix = TRUE,
  ...
)
```

Arguments

fit	a JAGS model fitted via JAGS_fit() or a derived-draw view from JAGS_with_draws(). Diagnostics describe the supplied view draws.
parameter	parameter to be plotted
type	what type of model diagnostic should be plotted. The available options are "density", "trace", and "autocorrelation"
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
xlim	x plotting range
ylim	y plotting range
lags	number of lags to be shown for the autocorrelation plot. Defaults to 30.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
transformations	named list of transformations to be applied to specific parameters
transform_factors	whether factors with orthonormal/meandif prior distribution should be transformed to differences from the grand mean, and factors with prior_ordered() prior distribution to their level effects (without the reference level of a "cumulative" contrast, which is fixed at zero). Untransformed ordered factors show the sampled parameters: runjags_estimates_table() reports the total effect (<term>{total}) and the allocation shares of the random allocations (<term>[share: <level>], the share of the total added when reaching <level>; fixed allocations have no rows). ensemble_estimates_table() reports retained fitted totals and the union of allocation shares sampled by any declared ordered model. Fixed ordered allocations fill that union with their declared constants. Models without the ordered parameterization have undefined (NA) shares, with their defined fraction reported in a footnote. Shares remain defined on zero-total and excluded spike draws of an ordered model. Legacy mixed objects without retained sources must be recreated from source fits.
short_name	whether prior distribution names should be shorted
parameter_names	whether parameter names should be printed
formula_prefix	whether the parameter prefix from formula should be printed. Defaults to TRUE.
...	additional arguments

Details

The trace shows the retained MCMC iteration of every draw on its x axis (thinned chains keep their original positions). Autocorrelation uses the retained original iteration positions. Requested lags without finite observation pairs, beyond the retained chain, or with nonfinite autocorrelation are retained as NA. One warning per parameter and chain has classes BayesTools_autocorrelation_unavailable

and BayesTools_plot_condition, with parameter, chain, unavailable_lags, pair_count and reason fields. Computed x/y objects retain full-length pair_count and unavailability_reason attributes; both renderers display finite bars only. No iteration compression, replacement estimate or confidence band is introduced.

Coordinates whose convergence role is "structural" (see JAGS_check_convergence()) are constants of the prior: the weight bins that the prior fixes (for example by wf_fixed()) or the coefficients of an ordered prior with a point total and fixed allocation. The density and autocorrelation diagnostics omit them when their draws are constant, and when nothing else remains the call returns NULL with a message, as for a spike prior; the trace is produced as for any other parameter. A parameter with a prior_none() prior is not sampled: the call returns NULL with a message for every type.

Value

diagnostics returns either NULL if plot_type = "base" or an object/list of objects (depending on the number of parameters to be plotted) of class 'ggplot' if plot_type = "ggplot".

See Also

JAGS_fit() JAGS_check_convergence()

JAGS_draw_geometry *JAGS draw geometry and deterministic materialization*

Description

JAGS_draw_geometry() returns the retained iteration and chain geometry stored from the actual chains returned by JAGS. Geometry is independent of which coordinates are public.

JAGS_materialize_draws() reconstructs requested coordinates from the fitted parameter map. Sampled coordinates are selected from the fitted chains and structural coordinates are filled with their exact fixed_value. Internal coordinates, including a private backend anchor, are excluded by default.

JAGS_with_draws() returns a derived-draw view with supplied draws and matching geometry, plus the complete untouched fit in original_fit. Views have class BayesTools_draws_view and BayesTools_fit, with only original_fit and mcmc slots. Nonruntime analysis attributes are copied; view-local changes do not modify the original. Replacing a view retains its original fit and current local analysis attributes without nesting views. Fit-level draw metadata and stored posterior densities/ordinates are cleared; attach fresh estimates with posterior_metadata() after replacement. A view supports descriptive estimates, diagnostic plots, mixed-posterior extraction and marginal inference from its supplied draws. It does not support original-model convergence, bridge sampling or sampling extension: use fit$original_fit and regenerate the view after extending that fit. These refusals have common class BayesTools_draws_view_unavailable, with leaves BayesTools_draws_view_sampling_unavailable (also BayesTools_sampling_unavailable) and BayesTools_draws_view_inference_unavailable. Missing requested sampled coordinates signal BayesTools_draws_view_coordinate_unavailable, with a missing field. Views need not contain every original sampled coordinate; prior-draw transformations retain their existing

intersection-of-columns policy. `coda::as.mcmc.list()` preserves chain timing. `coda::as.mcmc()` pools in chain-major order with indices starting at one and thinning one; pooled indices do not describe a continuous MCMC trajectory.

Usage

```
JAGS_draw_geometry(fit)

JAGS_draw_geometry_schema()

JAGS_materialize_draws(fit, parameters = NULL, include_internal = FALSE)

JAGS_with_draws(fit, draws)

## S3 method for class 'BayesTools_draws_view'
as.mcmc.list(x, ...)

## S3 method for class 'BayesTools_draws_view'
as.mcmc(x, ...)

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

Arguments

`fit`	fitted object created by `JAGS_fit()`.
`parameters`	optional exact vector of `coordinate_name` values. `NULL` selects every available public coordinate in map order.
`include_internal`	whether internal coordinates may be returned.
`draws`	replacement draws coercible to a `coda::mcmc.list`.
`x`	a derived-draw view.
`...`	additional arguments.

Value

`JAGS_draw_geometry()` returns a `BayesTools_draw_geometry` list. `JAGS_draw_geometry_schema()` returns field descriptions. `JAGS_materialize_draws()` returns a `coda::mcmc.list`, including a valid zero-column list when the fit has no public coordinate. `JAGS_with_draws()` returns a `BayesTools_draws_view`.

JAGS_evaluate_formula *Evaluate JAGS formula using posterior samples*

Description

Evaluates a JAGS formula on a posterior distribution obtained from a fitted model, through the formula design that JAGS_fit() stores for the formula parameter (the processed formula, factor levels and contrasts, and the standardization of predictors). Posterior or prior draws without a fit are evaluated through the same design, built by JAGS_formula_draws(). Fits created by BayesTools 0.3.0, which store no design, must be refitted: they stop with an error of class BayesTools_refit_required (see JAGS_validate_fit_contract()). Formula random effects can be evaluated for existing grouping levels when either standardized latent random effects and covariance hyperparameters were monitored via random_monitor(latent = TRUE), or the group-level coefficients were monitored via random_monitor(coefficients = TRUE); otherwise, and for random-effect blocks compiled as marginalized, conditional evaluation stops with an error of class BayesTools_refit_monitoring (also BayesTools_refit_required). In a random-effect block of a scalar-correlation structure (CS, HCS, AR1, HAR, or CAR) compiled group by group, the coefficients of index levels that a group does not have in the fitted data are drawn from their exact Gaussian conditional law given the observed levels. This draw is stochastic and follows set.seed(). Row-indexed external random-effect SD sources, such as random_sd_source("tau", shape = "row"), are evaluated from latent random effects only. Model generation automatically monitors the required latent effects for these blocks. The posterior samples must either contain source columns named tau[1], ..., tau[N]. When prediction data are supplied, fitted_rows must explicitly map each prediction row to its fitted observation index. Alternatively, the source must provide a parameter_source() values function for reconstructing row-wise source values from the posterior samples and supplied prediction data. An unavailable new-row evaluation of a posterior-indexed row source raises BayesTools_prediction_new_rows_unavailable (parent BayesTools_prediction_unavaila with reason = "posterior_indexed_row_source", block, the displayed source label, affected prediction rows, and caller context. It applies where the source of the new rows is needed, under new_levels = "sample"; with new_levels = "zero" the new rows contribute zero and need no source. Missing or malformed mappings, invalid source values, and missing monitored columns retain their own errors. Literal expression() terms use the replayable subset documented by JAGS_formula() and are evaluated against prediction data using JAGS-style row indexing through i. Array expressions such as X[i, column[i]] pair the coordinates pointwise. Each index must contain finite positive integers within its array dimension; scalar indices repeat across a common nonzero vector length. One-index expressions retain ordinary vector indexing. Sampled parameter references are reconstructed separately for every posterior draw. Replaying a fitted formula restores the stored expression syntax, dependency classification, and fitted data snapshot. Supply an explicit expression-free formula to evaluate a selected subset without those offsets. Inline transformations, offsets, dot expansion, and arbitrary calls are not supported. Create transformed predictors as explicit columns in data. Selected existing terms retain fitted factor coding and variable order, including interactions requested with reversed variable order. Predictors outside the selected terms are not required. Expression-valued scalar point coefficients use valid monitors first, otherwise their retained declaration and owned model data. New prediction rows do not replace that owned data. Unsupported scalar replay, missing parents, cycles, and malformed values raise BayesTools_formula_point_unavailable; absent affected owners require refitting. This replay

does not extend bridge prior-expression eligibility or establish a structural point law from constant draws.

Usage

```
JAGS_evaluate_formula(
  fit,
  formula = NULL,
  parameter,
  data = NULL,
  prior_list = NULL,
  formula_target = NULL,
  blocks = NULL,
  new_levels = NULL,
  fitted_rows = NULL
)
```

Arguments

fit	model fitted with JAGS_fit() with a formula for parameter, or posterior draws carrying its formula design (JAGS_formula_draws()).
formula	formula specifying the right hand side of the assignment (the left hand side is ignored). If NULL, the fitted formula stored in formula_design metadata is used. Whether the intercept values are log-transformed before computing the linear predictor is a property of the fitted formula: the "log(intercept)" flag stored in formula_design applies to every formula, also to an explicit one without the "log(intercept)" attribute. An explicit "log(intercept)" attribute that disagrees with the stored flag (TRUE for a fitted formula without a log intercept, or FALSE for one with it) stops with a BayesTools_formula_transform_unavailable error (reason "log_intercept_mismatch", fields explicit and stored) naming both values: omit the attribute or set it to the stored value.
parameter	name of the parameter created with the formula
data	data.frame containing predictors included in the formula. If NULL, versioned original-scale fitted source data from formula_design metadata are used.
prior_list	named list of prior distribution of parameters specified within the formula. If NULL, fitted priors from formula_design metadata are used.
formula_target	formula prediction target. "fixed" evaluates only the fixed formula contribution. "conditional" evaluates fixed effects plus fitted or explicitly generated random-effect contributions. NULL (the default) evaluates the fixed formula of fits without random effects and stops for fits with random effects, which need an explicit target.
blocks	optional random-effect block names used with formula_target = "conditional". When formula has random-effect terms, the default is the blocks of those terms and supplied blocks must be among them.
new_levels	optional new-level policy used only with formula_target = "conditional". Use a random_new_levels() object or one of "error", "zero", or "sample". Grouping levels declared in the fitted grouping factor but without rows in the

fitting data are predicted as new levels, except in blocks with a known group covariance; see `prior_random()`.

`fitted_rows` optional integer vector mapping supplied prediction rows to fitted observation indices. It is required whenever `data` is supplied and a selected random-effect block uses a posterior-indexed row source. Reordering and duplicate indices are supported. Callback-computed row sources do not use this mapping. The entries of rows of new random-effect levels predicted with `new_levels = "zero"` are ignored, because those rows contribute zero; they must still be valid fitted observation indices.

Value

`JAGS_evaluate_formula` returns a matrix of the evaluated posterior samples on the supplied data.

See Also

`JAGS_fit()` `JAGS_formula()`

`JAGS_fit` *Fits a 'JAGS' model*

Description

A wrapper around run.jags that simplifies fitting 'JAGS' models with usage with pre-specified model part of the 'JAGS' syntax, data and list of prior distributions.

Usage

```
JAGS_fit(
  model_syntax,
  data = NULL,
  prior_list = NULL,
  formula_list = NULL,
  formula_data_list = NULL,
  formula_prior_list = NULL,
  formula_scale_list = NULL,
  chains = 4,
  adapt = 500,
  burnin = 1000,
  sample = 4000,
  thin = 1,
  autofit = FALSE,
 autofit_control = list(max_Rhat = 1.05, min_ESS = 500, max_error = 0.01, max_SD_error =
  0.05, max_time = list(time = 60, unit = "mins"), sample_extend = 1000, restarts = 10,
  max_extend = 10, check_indicators = FALSE, monitor = NULL, allow_not_assessable =
    FALSE),
 parallel = FALSE,
```

```
  cores = chains,
  silent = TRUE,
  seed = NULL,
  add_parameters = NULL,
  required_packages = NULL,
  formula_random_prior_list = NULL,
  formula_random_effects_compile_list = NULL,
  jags_modules = NULL,
  runtime_setup = NULL,
  runtime_cache = NULL,
  worker_output = NULL,
  ...
)

JAGS_extend(
  fit,
 autofit_control = list(max_Rhat = 1.05, min_ESS = 500, max_error = 0.01, max_SD_error =
   0.05, max_time = list(time = 60, unit = "mins"), sample_extend = 1000, restarts = 10,
   max_extend = 10, check_indicators = FALSE, monitor = NULL, allow_not_assessable =
     FALSE),
  parallel = FALSE,
  cores = NULL,
  silent = TRUE,
  runtime_setup = attr(fit, "runtime_setup", exact = TRUE),
  runtime_cache = attr(fit, "runtime_cache", exact = TRUE),
  worker_output = NULL
)
```

Arguments

model_syntax	jags syntax for the model part
data	list containing data to fit the model (not including data for the formulas)
prior_list	named list of prior distribution (names correspond to the parameter names) of parameters not specified within the formula_list. Factor priors must carry their levels, set with prior_factor_levels().
formula_list	named list of formulas to be added to the model (names correspond to the parameter name created by each of the formula)
formula_data_list	named list of data frames containing data for each formula (names of the lists correspond to the parameter name created by each of the formula)
formula_prior_list	named list of named lists of prior distributions (names of the lists correspond to the parameter name created by each of the formula and the names of the prior distribution correspond to the parameter names) of parameters specified within the formula
formula_scale_list	named list of named lists for standardizing continuous predictors (names of the lists correspond to the parameter name created by each of the formula). Each

entry should be a named list where continuous predictors with TRUE values will be standardized. Defaults to NULL (no standardization).

chains number of chains to be run, defaults to 4

adapt number of samples used for adapting the MCMC chains, defaults to 500

burnin number of burnin iterations of the MCMC chains, defaults to 1000

sample number of sampling iterations of the MCMC chains, defaults to 4000

thin thinning interval for the MCMC samples, defaults to 1

autofit whether the models should be refitted until convergence criteria specified in autofit_control. Defaults to FALSE.

autofit_control

a list of arguments controlling the autofit function. Convergence thresholds and max_time$time must be finite and non-missing. Set a convergence threshold or max_time to NULL to disable that check. Possible options are:

max_Rhat maximum R-hat error for the autofit function. Defaults to 1.05. R-hat is the larger of the point estimate and the upper confidence limit of the potential scale reduction factor, whereas the summary tables show the point estimate (see JAGS_check_convergence()).

min_ESS minimum effective sample size. Defaults to 500.

max_error maximum MCMC error. Defaults to 0.01.

max_SD_error maximum MCMC error as the proportion of standard deviation of the parameters. Defaults to 0.05.

max_time list with fields time and unit, specifying when the automatic fitting function is stopped. The unit value corresponds to the units argument of difftime.

max_extend number of times after which the automatic fitting function is stopped.

sample_extend number of samples between each convergence check. Defaults to 1000.

restarts maximum number of fitting attempts when the model fails to initialize; every failed attempt but the last is retried with new initial values (restarts = 1 fits once, without a retry). A failed attempt that is retried is reported as a warning of the fit. Defaults to 10.

check_indicators whether model indicator variables should be included in convergence checks, including when monitor selects a narrower parameter set. Auxiliary inclusion-probability coordinates remain excluded unless requested explicitly. Defaults to FALSE.

monitor optional character vector selecting parameters for convergence checks. Base names select all indexed elements. Requests are resolved against the monitored nodes that the summary tables show, including the derived and auxiliary ones that the default selection excludes. Base names that are not monitored are rejected before sampling (error class BayesTools_parameter_not_found); an indexed element such as "mu[2]" is resolved against the fitted columns after the first run and fails then when it does not exist. Defaults to NULL, which checks the default selection of JAGS_check_convergence(): the

sampled parameters of the priors, add_parameters that depend on a stochastic node, and the stochastic nodes that formulas monitor (such as latent random effects and LKJ primitives). Deterministic nodes that formulas monitor (such as random-effect correlation matrices and their Cholesky factors) are derived and not checked by default. The role of every fitted coordinate is stored in the convergence_role column of parameter_coordinates().

allow_not_assessable whether undefined diagnostics for requested sampled parameters may be ignored. Defaults to FALSE.

parallel whether the chains should be run in parallel FALSE

cores number of cores used for multithreading if parallel = TRUE, defaults to chains

silent whether the function should proceed silently, defaults to TRUE

seed seed for random number generation by JAGS_fit(): an integer between -.Machine$integer.max and .Machine$integer.max. Draws are reproducible for the same seed, chains, and R random-number generator settings (RNGkind()), and, with parallel = TRUE, the same cores. Parallel draws need not equal serial draws with the same seed, because the backend passes each chain's initial state to its worker in a serialized text form. Initial values are drawn from the priors after set.seed(seed), and each chain's JAGS random-number seed is drawn from the same seed with sample.int(.Machine$integer.max, chains) (see JAGS_get_inits). A chain's JAGS seed therefore does not depend on the number of chains, but the JAGS generator does: with more than four chains it changes from "base::Super-Duper" to "lecuyer::RngStream", so the draws of a chain differ between fits with up to four and with more chains. The chains of adjacent seeds (e.g., seed and seed + 1) do not share random-number streams. Automatic restarts after a failed initialization (see restarts in autofit_control) draw their seeds from a separate stream of the same seed (the first L'Ecuyer-CMRG substream), so the i-th restart does not repeat the fit with seed + i. A seeded fit restores the caller's random-number state (.Random.seed and RNGkind()) when it returns. With seed = NULL, the initial values are drawn from one seed taken from the caller's random-number stream (see JAGS_get_inits), which the fit then continues to use.

add_parameters vector of additional parameter names that should be monitored but were not specified in the prior_list. Automatic fitting checks their convergence like that of the other parameters. Fully observed data and a node that the model syntax defines deterministically (<- or =) from data and constants only are structural constants and do not block convergence. A node that is stochastic (~) or depends on a stochastic node is checked, and its constant draws are not assessable (see autofit_control$allow_not_assessable). A name that repeats a parameter of prior_list is refused. A parameter that a formula needs monitored, such as the scalar SD source of a random-effect block (random_sd_source("tau")), is added automatically and, when prior_list already holds its prior, is monitored through that prior.

required_packages

character vector specifying list of packages containing JAGS models required for sampling (in case that the function is run in parallel or in detached R session). Defaults to NULL. Parallel workers must load the same versions and native library

builds as the parent session. When developing packages with `load_all()`, install matching builds into a library and set `R_LIBS_USER` before starting R and its workers.

`formula_random_prior_list`
: optional named list of `prior_random()` objects for random effects in `formula_list`. Required for any formula that contains random effects.

`formula_random_effects_compile_list`
: optional named list of `random_effects_compile()` objects controlling which formula random-effect blocks are sampled and which are compiled as marginalized structural blocks. Defaults to `NULL`, which preserves the current all-sampled behavior.

`jags_modules`
: character vector specifying JAGS modules required by the generated model syntax. Defaults to `NULL`.

`runtime_setup`
: optional function accepting one `context` list. After required packages and JAGS modules load, it is called with `phase = "start"` in the calling process, then each parallel worker. The context contains `role` (`"local"`, `"coordinator"`, or `"worker"`), `chains` (total chains), `processes` (sampling processes), `process_id` (one-based worker index, or zero for the coordinator), `process_chains` (chains assigned to this process; zero for the coordinator), and `parallel`. Chains are divided round-robin among at most `min(cores, chains)` workers. On exit, including fitting failures, workers are stopped before the calling process receives `phase = "finish"` with its original topology context. Worker processes do not receive a finish callback. If stopping workers fails, a warning is issued and the finish callback is not run; cleanup still attempts every remaining worker and closes failed connections. This permits releasing coordinator resources before parallel work and restoring them afterward. Return values are ignored. The callback must be idempotent, must not change random-number generator state, and should capture only small immutable settings. Packages it uses must be included in `required_packages`. The callback is stored with the fit; `JAGS_extend()` reuses it with the current chain and worker topology. Supplying `NULL` explicitly disables replay.

`runtime_cache`
: optional function accepting `context` and `state = NULL` for optional computational caches. It must not change inference targets or random-number state and must capture only small settings. After runtime setup, a fit or extension calls it with `context$phase = "restore"` and a list of prior process shards assigned to this process (empty for new fits or workers without an assigned shard). Each prior shard is sent to at most one current process, in round-robin order; shards are not duplicated when worker count increases. Before workers stop, a completed fit calls it with `context$phase = "capture"` and `state = NULL`. Capture returns one serializable shard, or `NULL` to retain no state. The fitted object's `runtime_state` attribute stores process shards separately from its small `runtime_cache` callback. Backend extension calls receive neither attribute. `JAGS_extend()` restores old state independently of whether its callback elects to capture fresh state. Callback warnings and errors are reported without discarding valid draws; a failed capture callback leaves a `NULL` shard for its process. If a worker connection fails during capture, no shard is retained from any process for that fit. Capture warnings, including such failures, are also recorded in the fit's `warnings` attribute. Error and interrupt cleanup does not attempt

capture; ordinary runtime cleanup still runs. Retained R snapshots, serialization buffers, and the caller's old fitted object consume memory in addition to active cache storage. Callback packages must be listed in `required_packages`. Passing `NULL` disables capture and restore.

`worker_output` optional file path for parallel worker stdout and stderr. The parent directory must exist. Workers append to the same file, so messages can interleave. `NULL` retains the backend default of discarding worker output. This setting is call-specific and is not retained with the fit; supply it again to `JAGS_extend()` when needed. Connection failures stop automatic fitting retries and preserve the original backend error.

`...` additional hidden arguments

`fit` a 'BayesTools_fit' object (created by `JAGS_fit()` function) to be extended. A derived-draw view signals `BayesTools_draws_view_sampling_unavailable`, also `BayesTools_sampling_unavailable` and `BayesTools_draws_view_unavailable`, before runtime arguments are forced; extend its `original_fit` and regenerate it with `JAGS_with_draws()`. Fits without the parameter map and fit contract of this version of BayesTools (such as fits created by BayesTools 0.3.0) must be refitted: they stop with an error of class `BayesTools_refit_required` (see `JAGS_validate_fit_contract()`).

Value

`JAGS_fit` returns an object of class 'runjags' and 'BayesTools_fit'. `JAGS_extend` continues the backend random-number generator state of the existing chains; it does not reseed them. Stored parameter and RNG states initialize reconstruction; recompilation can reset sampler tuning and follows runjags' adaptation policy. The model is always recompiled from these stored states, never continued from a compiled model left in the session, so extending the same object twice, or a saved and reloaded copy of it, gives identical draws. Neither function keeps runjags' compiled rjags model (`method.options$rjags`) in the returned object; `runjags::extend.jags()` recompiles a model without it, as after reloading.

See Also

`JAGS_check_convergence()`

Examples

```
## Not run:
# simulate data
set.seed(1)
data <- list(
  x = rnorm(10),
  N = 10
)
data$x

# define priors
priors_list <- list(mu = prior("normal", list(0, 1)))
```

```
# define likelihood for the data
model_syntax <-
  "model{
    for(i in 1:N){
      x[i] ~ dnorm(mu, 1)
    }
  }"

# fit the models
fit <- JAGS_fit(model_syntax, data, priors_list)

## End(Not run)
```

JAGS_fit_contract *JAGS formula name map and fitted-object contract*

Description

JAGS_formula_name_map() returns the persisted mapping between opaque JAGS base names and semantic formula coordinates (kind, formula parameter, term and role), with an injective encoded identifier of those fields. The identifier does not replace the backend column name stored in a fit. Downstream code should use this map or parameter_coordinates() rather than parsing a JAGS name.

JAGS_fit_contract() returns the schema contract attached to a fit. JAGS_validate_fit_contract() checks the components named in requires and requires unsupported objects to be refitted rather than adapted.

Every error that asks to refit the model has class BayesTools_refit_required: these validators, and every function that reads fitted metadata, stop with it on a fit (or stored metadata or draw geometry) created by another version of BayesTools, such as BayesTools 0.3.0, or lacking fitted metadata the request requires. A fit whose posterior samples lack coordinates the request needs because of its monitoring or sampling settings (a quantity whose source coordinates were not monitored in parameter_draws(), random effects without random_monitor(latent = TRUE) or random_monitor(coefficients = TRUE) in JAGS_evaluate_formula() and JAGS_bridgesampling(), or a marginalized random-effect block) stops with the child class BayesTools_refit_monitoring (also BayesTools_refit_required): refitting with this version alone does not resolve it, and the message names the settings to refit with. Callers match these classes, never the message.

Usage

```
JAGS_formula_name_map(fit, parameter = NULL)

JAGS_fit_contract(fit)

JAGS_validate_fit_contract(fit, requires = character())

JAGS_fit_contract_schema()
```

Arguments

fit	fitted object created by JAGS_fit().
parameter	optional formula parameter selecting one name map.
requires	character vector naming required contract components.

Value

JAGS_fit_contract_schema() returns a data frame. JAGS_formula_name_map() returns a BayesTools_formula_name_ma data frame (or a named list of them). JAGS_fit_contract() returns a named list, and the validator returns it invisibly.

JAGS_formula *Create JAGS formula syntax and data object*

Description

Creates a JAGS formula syntax, prepares data input, and returns modified prior list for further processing in the JAGS_fit function.

Usage

```
JAGS_formula(
  formula,
  parameter,
  data,
  prior_list,
  formula_scale = NULL,
  prior_random = NULL,
  random_effects_compile = NULL
)
```

Arguments

formula	formula specifying the right hand side of the assignment (the left hand side is ignored), or a BayesTools_random_effects object returned by random_effects_formula(). If the formula suppresses the intercept with 0 + or -1, it is converted to include an intercept with a spike(0) prior. The formula can also have a "log(intercept)" attribute set to TRUE to generate syntax of the form log(intercept) + sum(beta_i * x_i), which is useful for parameters that must be positive (e.g., standard deviation). In that case, the intercept prior must have strictly positive support.
parameter	valid unindexed JAGS node name of the parameter to be created with the formula
data	data.frame containing predictors included in the formula. A logical predictor is coded as a two-level indicator; an interaction of a logical predictor with a factor in which model.matrix() would code it by indicators (e.g., flag:g without g) is unavailable, in a fixed formula and in a random-effect term alike, unless

formula_scale standardizes the predictor, which makes it numeric. Otherwise convert the predictor to a factor or to numeric 0/1.

prior_list named list of prior distribution of parameters specified within the formula. The name "intercept" is reserved for the intercept: no other predictor or prior name may contain the substring "intercept". When suppressing the intercept in the formula, an "intercept" prior can be explicitly specified; otherwise, prior("spike", list(0)) is automatically added, or prior("spike", list(1)) when the formula uses the "log(intercept)" attribute. The list can also include two special entries:

"__default_continuous" A prior to use for any continuous predictors (including the intercept) that are not explicitly specified in the prior list. This can also be a zero-argument function returning a prior object; in that case it is evaluated only if a missing continuous term needs the default.

"__default_factor" A prior to use for any factor predictors (including interactions involving factors) that are not explicitly specified in the prior list. This can also be a zero-argument function returning a prior object; in that case it is evaluated only if a missing factor term needs the default.

These default priors allow for more concise specification when many predictors share the same prior distribution. For continuous formula terms, prior_none() is canonicalized to a point prior at zero. The prior of a formula term can carry a "multiply_by" attribute (a number or the name of another parameter) that multiplies the term's contribution to the linear predictor; the intercept prior cannot be scaled and a "multiply_by" attribute on it is an error. For a mixture or spike-and-slab prior, set the attribute on that prior itself; a component carrying one is an error.

formula_scale named list specifying whether to standardize continuous predictors. If NULL (default), no standardization is applied. If a named list is provided, continuous predictors with TRUE values will be standardized (mean-centered and scaled by standard deviation). The intercept is never standardized. Scaled predictors are centered in every fixed and random term, including terms without a free intercept, so the model is specified on the standardized scale; see Details.

prior_random optional prior_random() object defining random-effect standard-deviation, covariance, monitoring, and prediction policies. Required when formula contains random effects.

random_effects_compile optional random_effects_compile() object specifying which resolved random-effect blocks should be compiled as sampled random effects and which should be compiled as structural marginalized blocks.

Details

When a formula suppresses the intercept with 0 + or -1, the function adds an intercept back to the compiled formula and includes a point prior that contributes zero on the formula scale: spike(0) ordinarily and spike(1) for a log-transformed intercept.

Scaling with formula_scale replaces a continuous predictor x by $(x - m)/s$ in every term, where m and s are its mean and standard deviation in the fitting data. Centering also applies to terms without a free intercept, which is intended: the model is specified on the standardized scale. For

example, `~ 0 + x` with a scaled `x` fits $\mu = b(x - m)/s$, a line through $(m, 0)$ rather than through the origin; on the original scale its slope is b/s and its implied intercept is $-bm/s$. Original-scale summaries (`transform_scaled = TRUE`) report that implied intercept although the standardized intercept is fixed at zero. Likewise, a random-effect block without a random intercept, such as `(0 + x | g)`, fits group contributions $u(x - m)/s$. It implies an original-scale random intercept $-um/s$ (SD $\tau|m|/s$ for a scaled-slope SD τ), perfectly correlated with the original-scale slope u/s; random-effect summaries report the block's own coefficients, such as `sd(x)` $= \tau/s$, and not this implied intercept. Leave a predictor unscaled when a line or group slope through the origin of x is intended.

Independence of random-effect coefficients is likewise specified on the centered scale. An independent block such as `(1 + x || g)` (or a `diag()` block) with a scaled `x` has independent group coefficients u_0 and u_1 with SDs τ_0 and τ_1. On the original scale, the group intercept $u_0 - u_1 m/s$ and slope u_1/s are correlated with

$$\rho = -\frac{\tau_1 m/s}{\sqrt{\tau_0^2 + (\tau_1 m/s)^2}},$$

for example $\rho \approx -0.93$ for equal SDs and $m/s = 2.5$. The original-scale summaries of such a block report its SDs, $\sqrt{\tau_0^2 + (\tau_1 m/s)^2}$ and τ_1/s, but no correlation, because the block has no correlation parameter. Use a correlated block, such as `(1 + x | g)`, to estimate the correlation, or leave `x` unscaled for independence on the original scale.

A homogeneous-SD block (`id()`, `random(covariance = "id")`, or `diag(..., hom = TRUE)`) shares one SD among its columns, which cannot carry the different original-scale SDs of the columns. A homogeneous block with more than one column that contains a column of a scaled predictor is therefore refused by `JAGS_formula()`, whether its shared SD is sampled or fixed by a point prior: use a heterogeneous block, or leave the predictor unscaled. Single-column homogeneous blocks, such as `id(0 + x | g)`, are not affected.

Factor contrasts are owned by `prior_factor()` and stored formula metadata, not inferred from whether an intercept is present. Thus `~ 0 + group` with an independent factor prior means a structural zero intercept plus one coefficient per group level. With treatment or mean-difference contrasts, the same no-intercept expression preserves that selected basis. An interaction without one of its lower-order terms, such as `g:x` in `~ g + g:x`, codes that factor by level indicators, as `stats::model.matrix()` does, and has one coefficient per level (e.g., one slope per level of `g`). Such a term does not use the contrast of that factor, so its prior may use a different contrast than the factor's terms that code it by its contrast (e.g., a mean-difference `g` with an independent `g:x`). When no term codes the factor by its contrast (e.g., `~ g:x + g:z`), its indicator-coded terms must agree on the contrast instead. Mean-difference and orthonormal priors, other than a point mass at zero, are unavailable for such a term; use a treatment or independent prior for per-level coefficients, or add the lower-order term (here `~ g * x`) to keep the contrast.

When using default priors (`"__default_continuous"` or `"__default_factor"`), explicitly specified priors for individual terms take precedence over the defaults. The defaults are only applied to terms that are not already in the prior list.

Formula random effects require `prior_random`. Random-effect SD priors in `prior_list` using `"term|group"` names are no longer supported. Continuous fixed-effect terms must expand to one design-matrix column. Matrix-valued continuous predictors are not currently supported. Fixed formulas support literal data-column names and standard formula operators, including interaction expansion such as `(x + z)^2` with literal whole degrees from 2 through `.Machine$integer.max`. This is formula expansion, not arithmetic powers; `I(x^2)` remains unsupported. Degree 0 or 1

and symbolic degrees retain the base-R formula refusal. Dot expansion, `offset()`, inline transformations, and arbitrary calls are rejected; create explicit data columns for transformed predictors. The BayesTools `expression(...)` facility accepts finite numeric constants, formula or model data, JAGS-style `i` indexing, sampled scalar or one-dimensional indexed parameters, arithmetic operators, and `abs()`, `exp()`, `log()`, or `sqrt()`. Expressions retain literal JAGS indexing: users must write `x` or `x[i]` as required by the JAGS data shape. Their parsed syntax and dependencies are persisted and replayed draw by draw during prediction and marginal- likelihood reconstruction. Parameter dependencies must be owned by a prior or declared through `add_parameters`; opaque deterministic nodes and formula-output dependencies are rejected by `JAGS_fit()` because their defining JAGS graph is unavailable during replay. Random-effect predictors likewise support literal data-column names and formula operators, but not inline transformations or arbitrary calls. Create transformed random slopes as explicit data columns. Grouping terms support variables, `:` interactions, and `/` nesting.

Random-effect structures have two left-side grammars. `id()`, `diag()`, and `us()` / `un()` use a coefficient formula, so `1`, `0`, and `-1` control the random intercept and slopes or interactions generate coefficient columns. Plain bars default to `us()` and double bars to `diag()`. A factor's concrete random basis is inherited from resolved contrast metadata by default and can be set for one block with `random_block(contrasts = ...)`.

`cs()` / `hcs()`, `ar1()` / `ar()` / `har()`, and `car()` instead use a structure-owned index specification. They reject intercept controls and block contrast overrides. See `random_effects_formula()` and `prior_random()` for their index and covariance contracts. Categorical predictor and grouping levels must not contain BayesTools' reserved internal tokens, such as `__xXx__`. The predictor name `intercept` is reserved for the formula intercept.

Value

`JAGS_formula` returns a list containing the formula JAGS syntax, JAGS data object, modified prior_list, and (if standardization was applied) a `formula_scale` list with standardization information for back-transformation.

Conditions

Refusals that callers may need to tell apart are errors of class `BayesTools_formula_transform_unavailable` (parents `error` and `condition`); their `reason` field names the cause:

- `"single_level_factor"`: a factor with a single level in the data has no contrasts to code (the field `predictor` names it). The index of a `cs()`, `hcs()`, `ar1()`, or `har()` block is coded by one indicator per level and may have a single level.
- `"logical_factor_interaction"`: a logical predictor in an interaction with a factor that `model.matrix()` would code by indicators, in a fixed formula or a random-effect term (see the `data` argument; the fields `predictors` and `terms` name them).
- `"unsupported_random_factor_contrast"`: a factor used only in a random-effect block that carries a contrasts matrix or an unsupported contrast name (the field `predictor` names it; set the coding with `random_block(contrasts = )`).
- `"homogeneous_scaled_block"`: a homogeneous-SD random block with a standardized slope (see Details; the fields `parameter` and `coordinates` name the formula parameter and the shared SD).

- "basis_dependent_factor_prior": the prior of an interaction term of two or more factors, of which at least one is coded by a mean-difference or orthonormal contrast, depends on the basis of that contrast and would not treat the factor levels exchangeably. Only mean-difference and orthonormal priors, and treatment or independent priors built from zero-mean untruncated normal and zero spike priors, are accepted on such a term; when every coded factor has two levels, treatment and independent priors symmetric about zero are accepted as well. Other treatment and independent priors and every ordered prior are refused (the fields terms and predictors name the term and the coded factors; a refused ordered prior also has class BayesTools_ordered_unavailable).

Other invalid inputs stop with ordinary errors.

See Also

JAGS_fit()

Examples

```
# simulate data
set.seed(1)
df <- data.frame(
  y      = rnorm(60),
  x_cont = rnorm(60),
  x_bin  = rbinom(60, 1, .5),
  x_fac3 = factor(rep(c("A", "B", "C"), 20), levels = c("A", "B", "C")),
  x_fac4 = factor(rep(c("A", "B", "C", "D"), 15), levels = c("A", "B", "C", "D"))
)

# specify priors with intercept
prior_list <- list(
"intercept"     = prior("normal", list(0, 1)),
"x_cont"        = prior("normal", list(0, .5)),
"x_fac3"        = prior_factor("normal",  list(0, 1),  contrast = "treatment"),
"x_fac4"        = prior_factor("mnormal", list(0, 1),  contrast = "orthonormal"),
"x_fac3:x_fac4" = prior_factor("mnormal", list(0, .5), contrast = "orthonormal")
)

# create the formula object
formula_obj <- JAGS_formula(
  formula = ~ x_cont + x_fac3 * x_fac4,
  parameter = "mu", data = df, prior_list = prior_list)

# using -1 notation (automatically adds spike(0) intercept)
prior_list_no_intercept <- list(
  "x_fac3" = prior_factor("normal", list(0, 1), contrast = "treatment")
)
formula_no_intercept <- JAGS_formula(
  formula = ~ x_fac3 - 1,
  parameter = "mu", data = df, prior_list = prior_list_no_intercept)
# Equivalent to specifying intercept = prior("spike", list(0))

# using default priors for simpler specification
```

```
prior_list_defaults <- list(
  "__default_continuous" = prior("normal", list(0, 1)),
  "__default_factor"     = prior_factor("normal", list(0, 0.5), contrast = "treatment")
)
formula_defaults <- JAGS_formula(
  formula = ~ x_cont + x_fac3,
  parameter = "mu", data = df, prior_list = prior_list_defaults)
# intercept and x_cont get the default continuous prior
# x_fac3 gets the default factor prior
```

`JAGS_formula_coefficient_route` *Original-scale route of a formula coefficient*

Description

Describes how a coefficient of a fitted formula is related to the fitted coordinates on the original scale of the predictors, from the fitted coefficient transform (`JAGS_formula_coefficient_transform()`): the type of the map, its support, and the weights of the coefficient on the fitted coordinates. Fit-level hypothesis tests use it to evaluate hypotheses on original-scale coefficients (`hypothesis_BF()`).

Usage

```
JAGS_formula_coefficient_route(
  fit,
  parameter,
  target,
  caller = "hypothesis_BF()",
  refit_class = NULL
)
```

Arguments

`fit`	a model fitted with `JAGS_fit()`.
`parameter`	the formula parameter (for example `"mu"`).
`target`	the name of the fitted coefficient (for example `"mu_x"`).
`caller`	the name of the function reported in the reason of a coefficient with an unsupported map.
`refit_class`	condition classes that precede `BayesTools_refit_required` when the fitted metadata are outdated.

Value

A list with the `formula_parameter`, the `target`, its position `target_i` in the `transform`, the `type` of the map (`"identity"`, `"affine"`, `"exp_affine"`, `"state_dependent"`, or `"unsupported"`), the `weights` on the fitted coordinates, the `support` of the coefficient, and for an unsupported map the `reason` and its `cause` (`"metadata"`, `"fixed"`, or `"nonlinear"`).

See Also

JAGS_formula_coefficient_transform(), hypothesis_BF()

JAGS_formula_coefficient_transform
Formula coefficient transformations and induced prior densities

---

Description

JAGS_formula_coefficient_transform() exposes the declared fixed-coefficient transformation used by transform_scale_samples(). It records fitted sources, original-scale targets, named source/output transforms, exact nonzero dependencies, and parameter-map structural metadata.

JAGS_formula_prior_density() constructs the induced marginal prior measure for one target, or for a weighted combination of targets, through BayesTools' deterministic prior-density context and linear-density algebra. The result can be passed directly to prior_density_ordinate(). Fitted coefficient priors are raw; multiply_by belongs to their compiled formula contribution. Original raw coefficients use $z'_j = \sum_k A_{jk} z_k m_k / m_j$ on one immutable fitted state. Identical canonical multipliers cancel before division, including zero; declared zero contributions require no irrelevant state. A different needed zero denominator refuses the requested numeric vector, without row repair. The numerical grid of a weighted combination resolves its narrowest continuous component (64 knots per robust scale of its prior times its absolute weight), except that a simple component whose demanded spacing is below .Machine$double.eps times the largest absolute coordinate of the combination's range cannot be represented on that grid. Such a component, e.g. the weight of a centered predictor equal to its mean, is a point at its mean and does not set the grid spacing; a component whose spacing the grid can resolve does.

Finite matrix rows are static raw maps. Dynamic rows are entirely NA and retain finite basis_matrix, canonical multipliers, true state dependencies, and a raw_affine, contribution_affine, or unavailable prior recipe. Compatible weighted requests share one context; declared point siblings may add a constant offset. Incompatible recipes refuse with "incompatible_prior_recipes"; unsupported stochastic denominators refuse with "state_dependent_map". Unsupported dependence/products remain explicit. Uncanceled subnormal static ratio weights retain their static classification, but draw and declared-point evaluation use the original value-aware arithmetic. Continuous laws whose reused scale cannot retain that map have reason numerical_scale_unavailable; usable draws and certified point laws remain.

JAGS_formula_internal_coordinate_priors() returns exact scalar priors for stochastic formula coordinates that are intentionally absent from the ordinary fitted prior_list. Currently these are the independent beta primitives used by compiled LKJ random-effect blocks.

Usage

```
JAGS_formula_internal_coordinate_priors(fit)

JAGS_formula_coefficient_transform(fit, parameter, target_scale = "original")

JAGS_formula_coefficient_transform_schema()
```

```
JAGS_formula_prior_density(
  fit,
  parameter,
  target = NULL,
  target_scale = "original",
  context = NULL,
  weights = NULL
)
```

Arguments

fit	fitted object created by JAGS_fit().
parameter	scalar formula parameter name.
target_scale	requested coefficient scale. The current schema supports only "original".
target	exact target coordinate from the transformation object. Supply exactly one of target and weights.
context	optional BayesTools prior-density context. This is useful for product-space or conditional prior mixtures; when NULL, the fitted prior_list is used.
weights	named numeric vector of weights over target_names of the transformation object; the density is that of the weighted combination of those targets. Supply exactly one of target and weights.

Value

JAGS_formula_coefficient_transform() returns a BayesTools_formula_coefficient_transform list (schema version 4). Its targets data frame has one row per original-scale target with the structural_status, fixed_value and reason of the target, its map_type ("identity": the target is its own fitted source, including a log-transformed source with an exp output; "affine": a linear combination of fitted sources; "exp_affine": exp of a linear combination of identity and log-transformed sources; "state_dependent": a map requiring fitted multiplier states; "unsupported": any other map), and the support of the map (a list column of c(lower, upper): c(0, Inf) for maps with an exp output and c(-Inf, Inf) otherwise; the prior support of a target can be narrower). JAGS_formula_coefficient_transform_schema() returns field descriptions. JAGS_formula_prior_density() returns a prior_linear_density accepted by prior_density_ordinate(). An unavailable density stops with an error of class BayesTools_formula_prior_density_unavailable (also BayesTools_formula_transfo whose field reason names the cause, e.g. "unknown_target", "missing_source_coordinates", or "nonlinear_map" for a weighted combination with a target whose map is not linear in the fitted coefficients. Declared measure limitations additionally inherit BayesTools_formula_measure_unavailable; generic context/program failures and numeric transformation failures retain their strict boundary. Numeric refusal reasons include "zero_multiplier_denominator", "missing_multiplier_state", and "nonfinite_transform". Formula-design 7, unscale-design 2, and transform 4 require refitting older stored formats; readers never reconstruct missing fitted declaration ownership. A requested parameter without a persisted formula design stops with BayesTools_formula_transform_unavailable and reason "missing_formula_design", also when requested through either prior-density route. JAGS_formula_internal_coordinate_priors() returns a uniquely named list of scalar prior() objects keyed by concrete fitted coordinate.

JAGS_formula_coordinate_dependencies *Fitted-formula coordinate dependencies*

Description

JAGS_formula_coordinate_dependencies() reports every fitted formula predictor that structurally depends on the supplied posterior coordinates. It uses the persisted formula designs and parameter map, including direct coefficients, formula expressions, coefficient multipliers, the parents of expression-point coefficients and multipliers ("point_expression"), the coordinates a random block samples itself ("random_effect": SDs, latent deviations, group coefficients, correlations, and SD allocations and inclusions), and external random-effect SD sources ("random_sd_source"). A row-source callback is reported as opaque for every queried coordinate: its declared inputs are not used to narrow the dependencies, so a callback source is conservatively treated as depending on every coordinate.

Usage

```
JAGS_formula_coordinate_dependencies(fit, coordinates)
```

Arguments

fit	fitted object created by JAGS_fit().
coordinates	unique fitted coordinate names.

Value

A data frame with coordinate_name, formula_parameter, and dependency_type columns. An empty data frame means that none of the fitted formula predictors depends on the supplied coordinates.

See Also

JAGS_formula_design() parameter_map()

JAGS_formula_design *Extract Fitted JAGS Formula Design Metadata*

Description

Returns the fitted formula design metadata stored by JAGS_fit(). The design contains the processed formula, fitted model frame, exact model matrix used for JAGS data construction, replayable expression syntax and dependencies, JAGS-safe coefficient names, contrast and factor- level metadata, rank diagnostics, prior metadata, and formula-scale information.

Usage

```
JAGS_formula_design(fit, parameter = NULL)
```

Arguments

fit	a fitted object returned by JAGS_fit().
parameter	optional formula parameter name. If NULL, all stored formula designs are returned.

Value

A named list of formula designs, or one formula design when parameter is supplied. Returns NULL when no formula design metadata is stored on fit.

See Also

JAGS_fit() JAGS_formula()

JAGS_formula_draws *Formula designs for draws without a fit*

Description

Builds the formula design of a formula parameter exactly as JAGS_fit() builds it from its formula_list, formula_data_list, formula_prior_list, and the related arguments (through JAGS_formula()), with its model data as model_data, and attaches it to posterior or prior draws that do not come from such a fit, for example the prior draws of a model that was not fitted. JAGS_evaluate_formula() and JAGS_predict_formula() evaluate the returned draws through this design as they evaluate a fit: on data when their data is NULL, and on new prediction data with the levels, contrasts, and standardization of data.

Usage

```
JAGS_formula_draws(
  draws,
  formula,
  parameter,
  data,
  prior_list,
  formula_scale = NULL,
  prior_random = NULL,
  random_effects_compile = NULL,
  model_data = NULL
)
```

Arguments

draws	posterior or prior draws of the model coordinates: a numeric matrix or data frame with one column per coordinate, named as the fitted coordinates (e.g. `mu_intercept`, `mu_x`, `mu_g[1]`), or an `mcmc` or `mcmc.list` object (chains are merged). Draws returned by `JAGS_formula_draws()` keep their designs, so that the designs of several formula parameters can be attached one after another.
formula	formula specifying the right hand side of the assignment (the left hand side is ignored), or a `BayesTools_random_effects` object returned by `random_effects_formula()`. If the formula suppresses the intercept with `0 +` or `-1`, it is converted to include an intercept with a spike(0) prior. The formula can also have a `"log(intercept)"` attribute set to `TRUE` to generate syntax of the form `log(intercept) + sum(beta_i * x_i)`, which is useful for parameters that must be positive (e.g., standard deviation). In that case, the intercept prior must have strictly positive support.
parameter	valid unindexed JAGS node name of the parameter to be created with the formula
data	data.frame the formula is fitted to: it determines the factor levels and the standardization of the design.
prior_list	named list of prior distribution of parameters specified within the `formula`. The name `"intercept"` is reserved for the intercept: no other predictor or prior name may contain the substring `"intercept"`. When suppressing the intercept in the formula, an "intercept" prior can be explicitly specified; otherwise, `prior("spike", list(0))` is automatically added, or `prior("spike", list(1))` when the formula uses the `"log(intercept)"` attribute. The list can also include two special entries: `"__default_continuous"` A prior to use for any continuous predictors (including the intercept) that are not explicitly specified in the prior list. This can also be a zero-argument function returning a prior object; in that case it is evaluated only if a missing continuous term needs the default. `"__default_factor"` A prior to use for any factor predictors (including interactions involving factors) that are not explicitly specified in the prior list. This can also be a zero-argument function returning a prior object; in that case it is evaluated only if a missing factor term needs the default. These default priors allow for more concise specification when many predictors share the same prior distribution. For continuous formula terms, `prior_none()` is canonicalized to a point prior at zero. The prior of a formula term can carry a `"multiply_by"` attribute (a number or the name of another parameter) that multiplies the term's contribution to the linear predictor; the intercept prior cannot be scaled and a `"multiply_by"` attribute on it is an error. For a mixture or spike-and-slab prior, set the attribute on that prior itself; a component carrying one is an error.
formula_scale	named list specifying whether to standardize continuous predictors. If `NULL` (default), no standardization is applied. If a named list is provided, continuous predictors with `TRUE` values will be standardized (mean-centered and scaled by standard deviation). The intercept is never standardized. Scaled predictors are centered in every fixed and random term, including terms without a free intercept, so the model is specified on the standardized scale; see Details.

`prior_random` optional `prior_random()` object defining random-effect standard-deviation, covariance, monitoring, and prediction policies. Required when `formula` contains random effects.

`random_effects_compile` optional `random_effects_compile()` object specifying which resolved random-effect blocks should be compiled as sampled random effects and which should be compiled as structural marginalized blocks.

`model_data` optional named list of the JAGS model data of the fit, as the `data` argument of `JAGS_fit()`. `expression()` terms of `formula` may read these data besides `data`; the design keeps the data they read, as the fitted design does. The data are also retained for expression-point coefficient dependencies and scalar multiplier/SD sources. Owned point-expression arrays retain their original dimensions independently of prediction-row counts.

Value

An `mcmc` object of the draws carrying the formula design of `parameter` (see `JAGS_formula_design()`) and, when predictors are standardized, its `formula_scale`, which `JAGS_evaluate_formula()` and `JAGS_predict_formula()` accept as `fit`.

See Also

`JAGS_formula()` `JAGS_evaluate_formula()` `JAGS_formula_design()`

JAGS_formula_point_finalize

Finalize formula designs built outside JAGS_fit()

Description

Completes the `formula_design` and `formula_scale` returned by `JAGS_formula()` in the way `JAGS_fit()` does before it stores them in a fit. The functions let a caller rebuild the formula design of a fitted model outside `JAGS_fit()`, for example to replace the design stored with a fit.

Usage

```
JAGS_formula_point_finalize(
  design,
  prior_list,
  formula_data,
  model_data = NULL,
  parameter_names = character()
)

JAGS_formula_scale_finalize(
  formula_scale,
  design = NULL,
```

```
  prior_list = list(),
  model_data = NULL,
  owner_scope = "fit"
)
```

Arguments

design	the formula_design element returned by JAGS_formula(). For JAGS_formula_scale_finalize(), NULL or the finalized design of the parameter, which supplies the parameter name and the random-effect standard deviations whose scalar states are resolved.
prior_list	a named list of the priors of the model: the prior_list returned by JAGS_formula() combined with the other priors of the model, as JAGS_fit() combines them. Expression-point coefficients can read the priors in this list.
formula_data	NULL, a data frame, or a named list: the data the formula was evaluated on (the element of formula_data_list of JAGS_fit() for the parameter).
model_data	NULL or a named list: the JAGS data of the model (JAGS_fit() argument data), which expressions and scalar states can read.
parameter_names	a character vector of the names of the model coordinates (for example JAGS_parameter_names() of a fit) that expressions can read.
formula_scale	the formula_scale element returned by JAGS_formula() (NULL is returned unchanged).
owner_scope	the owner of the completed scale: "fit" for the scale of a fitted model, as JAGS_fit() stores it, or "prior_only" for a design used for prior densities without a fit.

Details

JAGS_formula() returns a fresh design and scale. JAGS_fit() completes them for each formula parameter in this order: first JAGS_formula_point_finalize() and the finalization of the expression() terms of the design, then the JAGS data generated by all the formulas are merged into the model data, and, in a second loop over the formula parameters, JAGS_formula_scale_finalize() completes the scale with that merged model data. The completed scale is stored both as the formula_scale of the parameter and as the formula_scale element of its finalized design.

The finalization of the expression() terms and the merge of the formula-generated data are not exported. A design rebuilt with these two functions alone does not carry the finalization of its expression() terms, and the scale is completed with the model_data that is supplied, without the data generated by the formulas.

JAGS_formula_point_finalize() retains, for each fixed coefficient with an expression() point prior and for each expression() point prior named by a multiply_by attribute of the design, the expression, the parent priors it reads, and the data it reads, in the point_expression_owner element of the design. A design without such a coefficient or multiplier is returned unchanged. Malformed or contradictory declarations stop with an error of class BayesTools_formula_point_unavailable (also BayesTools_formula_transform_unavailable) whose field reason names the cause.

JAGS_formula_scale_finalize() resolves the scalar states that the multipliers and standard deviations of the design need from the point coefficients of the scale, the point priors in prior_list, the

scalars in model_data, and the constants the scale already holds, and marks the scale as complete for owner_scope. Contradictory sources of one state stop with an error of class BayesTools_formula_transform_unavaila (field reason "contradictory_state_constants"), and a scale without its declarations stops with an error of class BayesTools_refit_required.

Value

JAGS_formula_point_finalize() returns design, with the point_expression_owner element when it has expression-point coefficients.

JAGS_formula_scale_finalize() returns formula_scale with the completed unscale_design attribute (the resolved state_constants, owner_scope, and complete = TRUE), or NULL for a NULL scale.

See Also

JAGS_formula() JAGS_fit() JAGS_formula_design()

Examples

```
data <- data.frame(x = c(-1, 0, 1, 2, 4))
prior_list <- list(
  intercept = prior("normal", list(0, 1)),
  x         = prior("normal", list(0, 1))
)
formula <- JAGS_formula(~ x, "mu", data, prior_list, formula_scale = TRUE)

design <- JAGS_formula_point_finalize(
  formula$formula_design, formula$prior_list, data
)
scale <- JAGS_formula_scale_finalize(
  formula$formula_scale, design, formula$prior_list
)
attr(scale, "unscale_design")$complete
```

JAGS_formula_predictor_basis

Exact fitted-formula predictor update basis

Description

JAGS_formula_predictor_basis() uses the fitted parameter map and persisted formula design to construct the exact observation-level change in a fixed linear predictor for a supplied direction through fitted coefficient coordinates. The result is conditional on all coordinates outside that direction, including fitted random effects.

The accessor reports "affine" only when the stored metadata proves that the selected update is additive on the fitted predictor scale. Coordinates reused by formula expressions, coefficient multipliers, expression-point coefficients, or random-effect scale sources are reported as non-affine or unsupported so callers can retain a generic evaluator.

A logged intercept is additive in its own logarithm rather than in the fitted coordinate: the predictor adds `log(new) - log(current)` times the intercept's model column. Such a direction is reported as `"affine"` with `coordinate = "log"` when the intercept is the only moving coordinate, and the caller must form the update from the logarithms. Every other affine result carries `coordinate = "identity"`. A direction that moves a logged intercept together with ordinary coefficients mixes the two coordinates and remains non-affine.

Usage

```
JAGS_formula_predictor_basis(fit, directions, posterior_samples = NULL)
```

Arguments

`fit`	fitted object created by `JAGS_fit()`.
`directions`	a fully named numeric vector, or a numeric matrix with unique coefficient-coordinate column names. Matrix rows define separate posterior-row update directions.
`posterior_samples`	optional posterior sample matrix aligned with the rows of `directions`. It is required when a selected formula term has a parameter-valued `multiply_by` attribute that is not a scalar constant declared by the fitted design; when omitted, draws are obtained from `fit`. A declared constant is used in preference to a column of the draws.

Value

A `BayesTools_formula_predictor_basis` list with `status`, `reason`, formula `parameter`, selected `coordinates`, the `coordinate` the update is additive in (`"identity"` or `"log"`), and an `S` by `N` numeric `basis`. `basis` is present only when `status` is `"affine"`.

See Also

`JAGS_formula_design()` `parameter_map()`

JAGS_formula_random_marginal_covariance

Compile formula random effects as marginal JAGS covariance nodes

---

Description

Compiles sampled or marginalized formula random-effect metadata into deterministic JAGS nodes containing observation-level covariance contributions. The compiler is intentionally likelihood agnostic: callers provide dependency-preserving row blocks and choose which terms contribute by subsetting the formula design. Sampling latent coordinates does not change their population covariance.

Usage

```
JAGS_formula_random_marginal_covariance(
  formula_design,
  row_blocks,
  prefix = "random_marginal_covariance",
  representation = c("dense", "auto")
)
```

Arguments

formula_design	a BayesTools_formula_design object created by JAGS_formula().
row_blocks	list of integer vectors partitioning the fitted rows.
prefix	valid JAGS node prefix used for generated nodes and data.
representation	covariance representation to compile. "dense" preserves the lower-triangle interface. "auto" returns the complete structurally certified diagonal-plus-factor representation when available and otherwise uses the dense representation.

Value

A list of class BayesTools_JAGS_random_marginal_covariance with syntax (the JAGS model syntax as one string), data (the named list of data the syntax reads), row_blocks (the validated integer row vectors), representation ("dense", or the diagonal-plus-factor representation that "auto" returns when it is structurally certified), term_names (the compiled random-effect block names), and the node names per row block. Dense results contain one lower-triangle covariance node name per row block in lower_names and NA in diagonal_names and loading_names. Diagonal-plus-factor results instead contain diagonal_names, loading_names (NA for a block without loading columns), and structural loading_ranks. A formula design without random effects returns empty syntax and data.

JAGS_get_inits *Create initial values for 'JAGS' model*

Description

Creates initial values for priors in a 'JAGS' model.

Usage

```
JAGS_get_inits(prior_list, chains, seed)
```

Arguments

prior_list	named list of prior distribution (names correspond to the parameter names)
chains	number of chains

seed — seed for random number generation: a number between -.Machine$integer.max and .Machine$integer.max. The initial values are drawn after set.seed(seed); each chain's .RNG.seed is drawn from the same seed with sample.int(.Machine$integer.max, chains), so the seed of a chain does not depend on the number of chains. The caller's random-number state (.Random.seed and RNGkind()) is restored afterwards. With seed = NULL, the seed is first drawn from the caller's random-number stream with sample(666666, 1), which advances that stream by this one draw.

Value

JAGS_get_inits returns a list with one element per chain, containing the chain's JAGS initial values and its .RNG.name and .RNG.seed. The generator is "base::Super-Duper" for up to four chains and "lecuyer::RngStream" for more chains, so the streams of a fit change when it is run with more than four chains. With an empty prior_list, each element contains only .RNG.name and .RNG.seed, so that fits of models whose priors are specified only in the model syntax remain reproducible.

JAGS_lkj_corr_cholesky

Generate a JAGS LKJ-Cholesky correlation module

Description

Creates JAGS syntax for a Cholesky factor of a correlation matrix with the same target distribution as Stan's lkj_corr_cholesky(eta), using the BayesTools compiled JAGS module.

Usage

```
JAGS_lkj_corr_cholesky(
  name,
  K,
  eta = 1,
  include_correlation = TRUE,
  include_primitives = FALSE
)
```

Arguments

name — character scalar. Prefix used for generated JAGS nodes.

K — integer scalar. Dimension of the correlation matrix.

eta — positive numeric scalar. LKJ concentration parameter.

include_correlation — logical scalar. Whether to generate the deterministic correlation matrix name_R.

include_primitives — logical scalar. Whether primitive beta/CPC nodes should be included in the returned monitor vector.

Value

A list with JAGS syntax, monitor names, primitive bridge coordinate names and bounds, and metadata for the generated LKJ-Cholesky block.

`JAGS_marglik_parameters_formula` *Extract parameters for 'JAGS' priors*

Description

Extracts transformed parameters from the prior part of a 'JAGS' model inside of a 'bridgesampling' function (returns them as a named list)

Usage

```
JAGS_marglik_parameters_formula(
  samples,
  formula_list,
  formula_data_list,
  formula_prior_list,
  prior_list_parameters,
  formula_design_list = NULL,
  model_data = NULL
)

JAGS_marglik_parameters(samples, prior_list)
```

Arguments

`samples`	samples provided by bridgesampling function
`formula_list`	named list of formulas to be added to the model (names correspond to the parameter name created by each of the formula). For `BayesTools_fit` objects with stored formula-design metadata, formula inputs can be omitted; if supplied, they are rebuilt only to check exact consistency with the fitted design and never replace fitted replay metadata. Fits without versioned formula-design metadata must be refitted: they stop with an error of class `BayesTools_refit_required` (see `JAGS_validate_fit_contract()`).
`formula_data_list`	named list of data frames containing data for each formula (names of the lists correspond to the parameter name created by each formula). When supplied for a fitted formula, these data must exactly match the fitted original-scale formula source snapshot.
`formula_prior_list`	named list of named lists of prior distributions (names of the lists correspond to the parameter name created by each of the formula and the names of the prior distribution correspond to the parameter names) of parameters specified within the `formula`

prior_list_parameters
: named list of the posterior values of the model parameters that are not specified within the formula but that might scale the formula parameters (the value returned by JAGS_marglik_parameters() for the same samples)

formula_design_list
: named list of the formula-design metadata (the formula_design element returned by JAGS_formula()) of each formula parameter. JAGS_marglik_parameters_formula() reconstructs the linear predictors, including formula random effects, from these fitted designs and stops when a design is missing.

model_data
: optional data passed to row-wise external parameter source reconstruction functions during formula random-effect bridge sampling.

prior_list
: named list of prior distributions (names correspond to the parameter names) of parameters not specified within the formula_list. For BayesTools_fit objects, stored non-formula priors are used when prior_list = NULL; a supplied prior must specify the same prior as the stored prior of the same name (distribution, parameters, truncation, multiply_by, and mixture components and weights; an error of class BayesTools_refit_required otherwise), while stored priors omitted from prior_list stay allowed. Formula priors are taken from the stored formula metadata: a formula prior supplied here is ignored when it specifies the same prior as the fitted formula prior (distribution, parameters, truncation, contrast, multiply_by, and mixture components and weights) and is an error otherwise.

Value

JAGS_marglik_parameters returns a named list of (transformed) posterior samples. Samples that lack monitored coordinates the priors require stop with an error of class BayesTools_missing_monitored_columns (also BayesTools_marglik_input); callers match the class, not the message. This includes ordinary scalar and vector priors. A named coordinate whose value is NA is present; literal point priors need no monitored coordinate. Formula reconstruction combines decoded ordinary and formula-prior values, genuine additional sampled sources, and retained scalar data before fixed coefficients, multipliers, and random-effect SDs are reconstructed. Natural owned state takes precedence over transformed aliases; malformed supplied owners refuse instead of falling back to a sample column. Expression-point replay is available through a fitted formula owner; generic prior readers without that owner report BayesTools_formula_point_unavailable. A state outside the support of the model (for example a negative row-indexed external random-effect SD value) stops JAGS_marglik_parameters_formula() with an error of class BayesTools_marglik_out_of_support, which JAGS_bridgesampling() treats as zero density.

JAGS_marglik_priors *Compute marginal likelihood for 'JAGS' priors*

Description

Computes marginal likelihood for the prior part of a 'JAGS' model within 'bridgesampling' function

Usage

```
JAGS_marglik_priors(samples, prior_list)

JAGS_marglik_priors_rows(samples, prior_list)

JAGS_marglik_priors_rows_evaluator(prior_list)

JAGS_marglik_priors_formula(samples, formula_prior_list, prior_list = NULL)
```

Arguments

samples	samples provided by the bridgesampling function. Supply one named posterior row to JAGS_marglik_priors() and a named matrix or data frame to JAGS_marglik_priors_rows(). Row-wise samples must have real numeric columns.
prior_list	named list of model-level prior distributions. For JAGS_marglik_priors_formula(), optional model priors whose shared ordered allocations have already been included by JAGS_marglik_priors().
formula_prior_list	named list of named lists of prior distributions (names of the lists correspond to the parameter name created by each of the formula and the names of the prior distribution correspond to the parameter names) of parameters specified within the formula

Details

JAGS_marglik_priors_rows() preserves joint prior boundaries within each posterior row. In particular, auxiliary-gamma contributions for vector and Dirichlet priors are summed separately within each row.

When model and formula priors share an ordered Dirichlet allocation, supply the model priors as prior_list to JAGS_marglik_priors_formula(). Its contribution then excludes allocations already included by JAGS_marglik_priors(samples, prior_list), so the two log prior contributions can be added without counting shared allocations twice. Evaluate the combined contribution as JAGS_marglik_priors(samples, prior_list) + JAGS_marglik_priors_formula(samples, formula_prior_list, prior_list).

Value

JAGS_marglik_priors returns a numeric value of the log prior density evaluated at the current posterior sample. JAGS_marglik_priors_rows returns one numeric value for every row of posterior samples. JAGS_marglik_priors_rows_evaluator returns a function that applies the same row-wise evaluation without recompiling the prior structure on each call. Samples that lack monitored coordinates the priors require stop with an error of class BayesTools_missing_monitored_columns (also BayesTools_marglik_input).

JAGS_ordered_density_kernel *Batched numerical density kernel for ordered primitives*

Description

Compiles the active ordered totals and distinct allocation coordinates into a function of a numeric draws matrix.

Usage

```
JAGS_ordered_density_kernel(prior_list, allocation_chart = NULL)
```

Arguments

prior_list	named list of original bound ordered priors.
allocation_chart	optional named list keyed by allocation key. Each entry has kind = "group" and a nonempty proper index subset J, or kind = "pair" and two distinct indices. Its gamma factors are replaced by the corresponding Beta density of the group or pair share.

Details

Total mixture components are localized independently for each row; multi-slice spike totals share one inclusion state. Each total slice and allocation key contributes once. Mixture-state probabilities, inactive component sources, and nuisance-only chart factors are omitted because they are constant along the conditional chart. This is a numerical conditional kernel, not a bridge density. Expression totals raise BayesTools_ordered_expression_unavailable, missing coordinates raise BayesTools_ordered_coordinates_unavailable, and invalid indicators or missing (NA) total values of a sampled total raise BayesTools_ordered_invalid_state; all inherit BayesTools_ordered_unavailable. Ordinary values outside numeric support return -Inf. Refit when bound ordered metadata are unavailable.

Value

A function of samples, a numeric matrix with named coordinate columns, returning one log density or log probability per row.

JAGS_ordered_parameter_spec *Fitted ordered parameter specifications*

Description

Reads the persisted total, allocation and coefficient recipes of ordered terms. The recipes are backend specifications; the totals and the shares of random allocations they describe are catalog quantities of the fit (`parameter_catalog()`).

Usage

```
JAGS_ordered_parameter_spec(
  fit,
  parameters = NULL,
  weights = NULL,
  draws = NULL
)
```

Arguments

`fit`	a model fitted with `JAGS_fit()`.
`parameters`	optional names of ordered terms; `NULL` selects all.
`weights`	optional named numeric fitted-coordinate projection weights.
`draws`	optional numeric source draws for sampled parameters or a projection.

Details

Missing fitted numeric provenance raises `BayesTools_ordered_metadata_unavailable` and `BayesTools_refit_require` refit with this version. Missing source draws raise `BayesTools_ordered_coordinates_unavailable`; invalid component states raise `BayesTools_ordered_invalid_state`. Continuous total families remain structurally continuous with expression parameters. Bernoulli totals retain their declared support even with an expression probability. Expression point locations can supply fitted snapshot values but have an unavailable scalar measure with a `BayesTools_ordered_expression_unavailable` reason. Without a registered certified ancestor recipe, expression totals cannot supply numerical charts or correct replay after changing an ancestor. These coordinate/expression conditions inherit `BayesTools_ordered_unavailable`. Missing or inconsistent fitted provenance reports: "Fitted ordered numeric provenance is unavailable. Refit the model with this version of BayesTools." Missing projection totals report: "Ordered total sources for '<term>' are unavailable in 'draws'. Include its declared source coordinates." Invalid total states report: "Ordered total indicator '<coordinate>' does not select a declared component."

Value

With no `weights`, a named list of term specifications containing the bound `prior`, coefficient names and grid, slice indices, total names and prior, deterministic total recipe, allocation records

(key, factor, slice, id, normalized and gamma coordinates, alpha or fixed weights), and label parts. `source_coordinates` combines total monitors, total primitive dependencies, and Gamma monitors; `total_auxiliary_coordinates` lists the total's other monitored coordinates, including indicators, inclusion probabilities and slab sources. Some total-component dependencies are optional unmonitored sources; the monitored total is their fitted snapshot alternative. No supplied draws means no values field. With `weights = NULL` and supplied `draws`, each term also contains `sampled_parameters`: `values` is the fitted total/share matrix (its columns keep the backend names of the sources), `label_parts` holds the rendered labels (`f{total}`, `f[share: mid]`, see `parameter_labels()`), and `allocation_names` identifies its sampled share columns. Fixed totals materialize every slice; fixed allocations produce no sampled share columns.

An empty named list is returned when no ordered terms are present. With `weights`, the result is a `BayesTools_ordered_projection` containing bound `specs`, fitted `weights`, original `requested_weights`, static tensor `contractions`, and `ordinary_weights`. With resolved draws it also contains row-aligned `values`, `atom` (point location, otherwise `NA`), `state` (`"point"`, `"continuous"`, or `"unavailable"`), and `exact` (a full-simplex identity). `total_contraction_weights` holds the per-term matrices of total slopes, and `allocation_contractions` holds complete coordinate contractions per allocation key, including totals and every retained other factor. `reason` is a typed condition when the scalar measure is structurally unavailable. Total-coordinate weights expand to the corresponding complete coefficient slice, so a total target uses its declared full-simplex identity. Available ordered semantic values in mixed posteriors, factor levels and marginal views use these primitive projections, including continuous cumulative effects. Declared view transformations and undefined rows are kept.

`JAGS_package_builds` *Build fingerprints of loaded packages*

Description

Identifies the builds of packages as loaded in the running session. Two processes run the same build of a package when its fingerprints are identical; parallel BayesTools computation (e.g., `JAGS_fit()` and `JAGS_runtime_cluster()`) requires this of every worker, and packages can use it to key caches by build.

Usage

```
JAGS_package_builds(packages)
```

Arguments

`packages`	character vector of package names.

Value

A list named by `packages`. An element is `NULL` when the package cannot be loaded, and otherwise a list with the package `version`, `r_code` (a hash of the functions and immutable constants defined in the loaded namespace, so an installed package and an equivalent development load agree), and `dll` (MD5 sums of the package's loaded native libraries).

See Also

JAGS_runtime_cluster()

Examples

```
JAGS_package_builds("stats")$stats$version
```

JAGS_predict_formula *Formula prediction with explicit random-effect target*

Description

Returns formula-scale predictions from fitted BayesTools formula metadata. Unlike JAGS_evaluate_formula(), this helper can return structured prediction output for marginal random-effect targets, including marginal random-effect covariance or simulated random-effect draws. Unavailable new rows for a posterior-indexed row source raise BayesTools_prediction_new_rows_unavailable (parent BayesTools_prediction_unavailable), with reason = "posterior_indexed_row_source", block, displayed source label, affected prediction rows, and caller context. This applies to conditional and both marginal targets where the source of the new rows is needed, under new_levels = "sample"; with new_levels = "zero" the new rows contribute zero and need no source. Mapping, source-value, and monitoring errors retain their existing conditions.

Usage

```
JAGS_predict_formula(
  fit,
  parameter,
  formula = NULL,
  data = NULL,
  prior_list = NULL,
  formula_target = c("conditional", "fixed", "marginal"),
  blocks = NULL,
  new_levels = NULL,
  marginal_method = c("covariance", "sample"),
  seed = NULL,
  components = FALSE,
  fitted_rows = NULL
)
```

Arguments

fit	model fitted with JAGS_fit(), or posterior draws carrying the formula design of parameter (JAGS_formula_draws()).
parameter	formula parameter name.

formula	optional formula. If NULL, the fitted formula for parameter is used. For formula_target = "marginal", random-effect terms in formula must match fitted blocks and select the blocks whose marginal covariance or draws are included; without such terms, all fitted blocks are included.
data	optional prediction data. If NULL, fitted source data are used.
prior_list	optional named prior list. If NULL, fitted priors are used.
formula_target	prediction target: "conditional", "fixed", or "marginal". The default includes fitted random-effect contributions.
blocks	optional random-effect block names. For formula_target = "conditional" or "marginal" with random-effect terms in formula, the default is the blocks of those terms, and supplied blocks must be among those blocks.
new_levels	new-level policy for conditional or marginal random-effect prediction. Use a random_new_levels() object or one of "error", "zero", or "sample". Grouping levels declared in the fitted grouping factor but without rows in the fitting data are predicted as new levels, except in blocks with a known group covariance; see prior_random().
marginal_method	marginal representation used only with formula_target = "marginal": "covariance" returns fixed means plus marginal random-effect covariance; "sample" returns materialized random-effect draws. For known group covariance, "sample" jointly draws the requested fitted levels from the corresponding covariance submatrix; noncentered multi-column ID, DIAG, and US blocks jointly retain both the group kernel and the draw-specific coefficient covariance. New levels remain unsupported.
seed	optional random seed used when random-effect draws are simulated.
components	whether to include component matrices where available.
fitted_rows	optional integer vector mapping supplied prediction rows to fitted observation indices. It is required for selected posterior-indexed row sources. Reordering and duplicate indices are supported; callback-computed row sources do not use it. The entries of rows of new random-effect levels predicted with new_levels = "zero" are ignored, because those rows contribute zero; they must still be valid fitted observation indices.

Value

A list of class BayesTools_formula_prediction with fields value, mean, random, vcov, components, and metadata.

See Also

JAGS_evaluate_formula() random_effects_marginal_vcov()

JAGS_runtime_cluster *Worker clusters with the running package builds*

Description

JAGS_runtime_cluster() starts a PSOCK cluster for parallel computation whose workers run the package builds of the calling session, as the workers of parallel JAGS_fit() do. Packages built on BayesTools use it for their own parallel computation. JAGS_runtime_cluster_stop() stops such a cluster.

Usage

```
JAGS_runtime_cluster(
  cores,
  packages = "BayesTools",
  runtime_setup = NULL,
  options = list(),
  operation = "Parallel computation"
)

JAGS_runtime_cluster_stop(cl)
```

Arguments

cores	number of worker processes.
packages	packages that every worker loads. Each worker must load the same build (version, R code, and native libraries; see JAGS_package_builds()) as the calling session.
runtime_setup	NULL or a function of one context argument, as in JAGS_fit(). It runs in the calling process and on every worker when the cluster starts (context$phase "start"), and in the calling process after the workers stopped ("finish"). The context counts one chain per worker (context$chains equals cores).
options	named list of options set on every worker, in addition to all current BayesTools.* options of the calling session; an entry of options takes precedence over a current option of the same name.
operation	description of the computation used in error messages.
cl	cluster returned by JAGS_runtime_cluster().

Details

Workers start with Rscript --vanilla, so no user or site startup file runs, and with the library paths of the calling session. Setting the options precedes loading the packages. If the start fails, the workers are stopped (and a started runtime setup finished) before the error is signalled. JAGS_runtime_cluster_stop() stops the workers first and then finishes the runtime setup in the calling process; further calls have no effect.

Value

JAGS_runtime_cluster() returns the cluster (as created by parallel::makePSOCKcluster()), for use with the functions of the **parallel** package. JAGS_runtime_cluster_stop() returns NULL invisibly.

See Also

JAGS_fit() JAGS_package_builds()

Examples

```
## Not run:
cl <- JAGS_runtime_cluster(2, packages = "BayesTools")
on.exit(JAGS_runtime_cluster_stop(cl), add = TRUE)
parallel::parLapply(cl, 1:2, function(i) i^2)

## End(Not run)
```

JAGS_to_monitor *Create list of monitored parameters for 'JAGS' model*

Description

Creates a vector of parameter names to be monitored in a 'JAGS' model.

Usage

```
JAGS_to_monitor(prior_list)
```

Arguments

prior_list	named list of prior distribution (names correspond to the parameter names)

Value

JAGS_to_monitor returns a character vector of parameter names.

kitchen_rolls *Kitchen Rolls data from Wagenmakers et al. (2015) replication study.*

Description

The data set contains mean NEO PI-R scores for two groups of students. Each of them filled a personality questionnaire while rotating a kitchen roll either clock or counter-clock wise. See Wagenmakers et al. (2015) for more details about the replication study and the https://osf.io/uszvx/ for the original data.

Usage

```
kitchen_rolls
```

Format

A data.frame with 2 columns and 102 observations.

Value

a data.frame.

References

Wagenmakers E, Beek TF, Rotteveel M, Gierholz A, Matzke D, Steingroever H, Ly A, Verhagen J, Selker R, Sasiadek A, others (2015). "Turning the hands of time again: a purely confirmatory replication study and a Bayesian analysis." *Frontiers in Psychology*, **6**, 1–6. doi:10.3389/fpsyg.2015.00494.

lines.prior *Add prior object to a plot*

Description

Add prior object to a plot

Usage

```
## S3 method for class 'prior'
lines(
  x,
  xlim = NULL,
  x_seq = NULL,
  x_range_quant = NULL,
  n_points = 1000,
  n_samples = 10000,
```

```
  force_samples = FALSE,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  show_parameter = if (individual) 1 else NULL,
  individual = FALSE,
  rescale_x = FALSE,
  scale_y2 = NULL,
  ...
)
```

Arguments

x	a prior
xlim	plotting range of the prior
x_seq	sequence of x coordinates
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation , or a list containing the transformation function fun, inverse transformation function inv, and derivative of the transformation jac, evaluated on the original support. A custom transformation can also declare output_support = c(lower, upper). Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.
transformation_arguments	a list with named arguments for the transformation

transformation_settings
: boolean indicating whether the explicitly supplied x_seq or x_range is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

show_parameter
: which parameter should be returned in case of multiple parameters per prior. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions, k selects the publication weight omega of the k-th p-value interval in ascending order, as show_figures in plot.prior() and the order of the summary tables; 1 is the reference weight fixed at 1.

individual
: should individual densities be returned (e.g., in case of weightfunction)

rescale_x
: allows to rescale x-axis in case a weightfunction is plotted.

scale_y2
: scaling factor for a secondary axis. The default NULL reuses the probability mapping from the active mixed plot when one is available.

...
: additional arguments

Details

For ordered and Dirichlet simplex priors, show_parameter is a single figure between 1 and the number of figures; other values stop with an error. The default draws the default selection of plot.prior().

Value

lines.prior returns NULL.

See Also

plot.prior() geom_prior()

---

lines_prior_list *Add list of prior objects to a plot*

Description

Add list of prior objects to a plot

Usage

```
lines_prior_list(
  prior_list,
  xlim = NULL,
  x_seq = NULL,
  x_range_quant = NULL,
  n_points = 500,
  n_samples = 10000,
```

```
  force_samples = FALSE,
  individual = FALSE,
  show_figures = if (individual) 1 else NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  rescale_x = FALSE,
  scale_y2 = NULL,
  prior_list_mu = NULL,
  effect_direction = "positive",
  ...
)
```

Arguments

prior_list	list of prior distributions
xlim	x plotting range
x_seq	sequence of x coordinates
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
individual	should individual densities be returned (e.g., in case of weightfunction)
show_figures	which figures should be returned in case of multiple plots are generated. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions plotted individually by plot.prior() and plot_posterior(), figure k shows the publication weight omega of the k-th p-value interval in ascending order, the order in which summary tables (including RoBMA's) print the omegas; figure 1 is the reference weight of the most significant interval. In plot.prior(), a negative k omits figure -k (the default -1 omits the reference weight fixed at 1). For a two-sided weight function prior, the intervals are its two-sided p-value bins; tables on the one-sided p-value scale print these weights first and then repeat them for the mirrored intervals.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations:

lin linear transformation in form of `a + b*x`

tanh hyperbolic tangent transformation

exp exponential transformation

, or a list containing the transformation function `fun`, inverse transformation function `inv`, and derivative of the transformation `jac`, evaluated on the original support. A custom transformation can also declare `output_support = c(lower, upper)`. Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

`transformation_arguments`
: a list with named arguments for the `transformation`

`transformation_settings`
: boolean indicating whether the explicitly supplied `x_seq` or `x_range` is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

`rescale_x`
: allows to rescale x-axis in case a weightfunction is plotted.

`scale_y2`
: scaling factor for a secondary axis

`prior_list_mu`
: list of priors for the mu parameter, required for the PET-PEESE regression bands (`individual = FALSE`) and refused otherwise: an individual plot shows the density of one coefficient and does not use it

`effect_direction`
: direction of the effect for PET-PEESE regression. Use `"positive"` (default) for `mu + PET*se + PEESE*se^2` or `"negative"` for `mu - PET*se - PEESE*se^2`.

`...`
: additional arguments

Details

For base plots, the default `scale_y2 = NULL` reuses the probability mapping established by the initial mixed-distribution plot. This keeps separately added prior and posterior point masses comparable.

Value

`lines_prior_list` returns `NULL`.

See Also

`plot_prior_list()` `geom_prior_list()`

`marginal_inference` *Model-average marginal posterior distributions and marginal Bayes factors*

Description

Creates marginal model-averaged and conditional posterior distributions based on a list of models, vector of parameters, formula, and a list of indicators of the null or alternative hypothesis models for each parameter. Computes inclusion Bayes factors for each marginal estimate via a Savage-Dickey density approximation.

Usage

```
marginal_inference(
  model_list,
  marginal_parameters,
  parameters,
  is_null_list,
  formula,
  null_hypothesis = 0,
  normal_approximation = FALSE,
  n_samples = 10000,
  seed = NULL,
  silent = FALSE,
  density_method = "KDE"
)
```

Arguments

`model_list`	list of models, each of which contains marginal likelihood estimated with bridge sampling `marglik` and prior model odds `prior_weights`
`marginal_parameters`	parameters for which the the marginal summary should be created
`parameters`	all parameters included in the model_list that are relevant for the formula (all of which need to have specification of `is_null_list`) Parameter keys must be unique, nonmissing and nonempty. Unnamed `is_null_list` entries bind positionally; complete unique names bind by key. Null masks are normalized before conditional model filtering.
`is_null_list`	list with entries for each parameter carrying either logical vector of indicators specifying whether the model corresponds to the null or alternative hypothesis (or an integer vector indexing models corresponding to the null hypothesis; use `0` or `integer(0)` when no models are null) Exact matching unique list names bind by parameter name; other ensemble lists retain positional order. Marginal inference requires unnamed lists or unique complete names matching its requested parameters.
`formula`	model formula (needs to be specified if `parameter` was part of a formula)

`null_hypothesis`
: point null hypothesis to test. Defaults to `0`

`normal_approximation`
: whether the height of the posterior density at the null should be approximated via a normal distribution (rather than kernel density). The prior height is always the structural prior-density ordinate. Defaults to `FALSE`.

`n_samples`
: controls the numerical grid used for model-averaged prior densities (a grid has at least 16 points). It is not used when the samples carry a stored prior-density context. For a simple parameter, an explicitly attached `prior_density` is authoritative and is propagated instead of being reconstructed from `prior_list`.

`seed`
: seed for random number generation. The caller's random-number state (`.Random.seed` and `RNGkind()`) is restored afterwards. With `NULL`, each model-averaging step draws its sampling seed from the caller's random-number stream.

`silent`
: whether warnings should be returned silently. Defaults to `FALSE`

`density_method`
: posterior density method used for Savage-Dickey Bayes factors. Currently only `"KDE"` is supported by `marginal_inference()` and `as_marginal_inference()` because generic model-averaged marginal posteriors do not have a well-defined precomputed density source.

Details

A parameter or level whose conditional marginal posterior has a declared point mass at the null hypothesis (e.g., the reference level of a treatment-coded factor, fixed at zero) gets an `NA` inclusion Bayes factor with the reason in its `"warnings"` attribute, which summary tables report; the other parameters and levels are computed. A parameter (scalar marginal) or level whose posterior ordinate is numerically unavailable (for example, a kernel estimate from identical draws) gets an `NA` inclusion Bayes factor with the reason in the same way.

Value

`marginal_inference` returns an object of class 'marginal_inference'.

See Also

ensemble_inference mix_posteriors BayesTools_ensemble_tables

`marginal_posterior` *Model-average marginal posterior distributions*

Description

Creates marginal model-averages posterior distributions for a given parameter based on model-averaged posterior samples and parameter name (and formula with at specification).

Usage

```
marginal_posterior(
  samples,
  parameter,
  formula = NULL,
  at = NULL,
  prior_samples = FALSE,
  use_formula = TRUE,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  n_samples = 10000,
  ...
)
```

Arguments

samples	model-averaged posterior samples created by `mix_posteriors()`. Mixed factor posteriors that lack the factor metadata of this version of BayesTools (such as those created by BayesTools 0.3.0) must be recreated from models fitted with this version: they stop with an error of class `BayesTools_refit_required` (see `JAGS_validate_fit_contract()`). With `use_formula = TRUE` and no attached formula state, orthonormal or mean-difference factor posteriors whose stored contrast basis differs from the current contrasts stop with the same error class; `use_formula = FALSE` keeps using the stored design.
parameter	parameter of interest
formula	model formula (needs to be specified if `parameter` was part of a formula)
at	named list with predictor levels of the formula for which marginalization should be performed. Continuous values use fitted/SD units for fitted posterior views and original data units for views created with `transform_scaled = TRUE`. A manipulated continuous predictor is evaluated at the existing grid `c(-1, 0, 1)` in those same view units. If a predictor level is missing, `0` is used for continuous predictors, the baseline factor level is used for factors with `contrast = "treatment"` prior distributions, and the parameter is completely omitted for factors with `contrast = "meandif"`, `contrast = "orthonormal"`, `contrast = "independent"`, and ordered-factor levels. A predictor without its own main-effect term (e.g., `x` in `~ g + g:x`) is handled by the type, levels, and contrast recorded for it in the interaction terms that contain it.
prior_samples	whether marginal prior distributions should be generated
use_formula	whether the parameter should be evaluated as a part of supplied formula
transformation, transformation_arguments	optional transformation of the marginal posterior, applied with `posterior_transform()` to the untransformed marginal posterior and its metadata. The transformation is validated before the marginal posterior is built.
transformation_settings, ...	not used by `marginal_posterior()`, which evaluates no plotting grid; `transformation_settings` is validated as a logical value and `...` is ignored.

n_samples controls the numerical grid used for model-averaged prior densities (a grid has at least 16 points). It is not used when the samples carry a stored prior-density context. For a simple parameter, an explicitly attached prior_density is authoritative and is propagated instead of being reconstructed from prior_list.

Details

When the mixed posterior samples carry deterministic posterior_density, posterior_ordinate, or support metadata (posterior_metadata()), marginal_posterior() propagates matching metadata to the returned marginal posterior. Matching uses the parameter name, list/level names such as theta[A], and conditional metadata when present. Exact support metadata is propagated even when prior_samples = FALSE; requesting prior samples adds prior-density metadata but does not replace already attached posterior support. A transformation is applied by posterior_transform() to the marginal posterior with all its metadata (supports, atoms, prior densities, stored posterior densities and ordinates, and component supports); the mixed posterior samples themselves must be untransformed (samples transformed with posterior_transform() whose prior is their prior_list are refused). If support metadata is absent, support is inferred from prior metadata only when the posterior samples are on the raw, unconditioned prior scale; otherwise the deterministic prior-density context is used so formula-scale transformations and conditional model restrictions are respected. Marginal posteriors with prior samples also record each draw's mixture component and each component's exact support (components metadata): the model of mix_posteriors() ensembles, or the combination of the mixture and spike-and-slab component indicators of a single fit (as_mixed_posteriors()). Savage_Dickey_BF() uses them when the components' supports differ. With log(intercept) formula scaling, the unscaled intercept (transform_scaled samples) is the exp of a linear combination of the fitted coefficients with the log of the fitted intercept: its simple marginal posterior (use_formula = FALSE) takes its prior density and support from the exp of that combination, and the linear predictors of a formula marginal posterior are linear in the same coefficients. Ordered formula projections reuse retained fitted coefficients, totals and allocations, with aligned model and draw rows, from the contributing formula prefixes. Every provided source of a contributing prefix is used; unrelated prefixes are excluded. Scaling is determined separately for each prefix; an unscaled prefix keeps its fitted draws when another prefix is scaled. Scaled mixed posteriors that lack the fitted ordinary coefficients needed by a formula stop with class BayesTools_ordered_coordinates_unavailable (parent BayesTools_ordered_unavailable); recreate the mixed posteriors from the source fits with this version of BayesTools.

Value

marginal_posterior returns a named list of mixed marginal posterior distributions (either vectors or matrices).

---

mean.prior *Prior mean*

Description

Computes mean of a prior distribution. (In case of orthonormal prior distributions for factors, the mean of for the deviations from intercept is returned.)

Usage

```
## S3 method for class 'prior'
mean(x, ...)
```

Arguments

x	a prior
...	unused

Details

The mean of a truncated non-normal prior is the integral of its truncated quantile function over the probability interval (0, 1). It is refused with `BayesTools_numerical_unavailable` when the quantile function is unavailable or the integral's estimated error exceeds 1e-6 of the interquartile range. Untruncated priors use closed forms and truncated normal priors their closed form.

Value

a mean of an object of class 'prior'.

See Also

`prior()`

Examples

```
# create a standard normal prior distribution
p1 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1))

# compute mean of the prior distribution
mean(p1)
```

---

mix_posteriors *Model-average posterior distributions*

Description

Model-averages posterior distributions based on a list of models, vector of parameters, and a list of indicators of the null or alternative hypothesis models for each parameter.

Usage

```
mix_posteriors(
  model_list,
  parameters,
  is_null_list,
  conditional = FALSE,
  seed = NULL,
  n_samples = 10000,
  on_failure = c("error", "drop", "zero")
)
```

Arguments

model_list	list of models, each of which contains marginal likelihood estimated with bridge sampling `marglik` and prior model odds `prior_weights`
parameters	vector of parameters names for which inference should be drawn
is_null_list	list with entries for each parameter carrying either logical vector of indicators specifying whether the model corresponds to the null or alternative hypothesis (or an integer vector indexing models corresponding to the null hypothesis; use `0` or `integer(0)` when no models are null) Exact matching unique list names bind by parameter name; other ensemble lists retain positional order. Marginal inference requires unnamed lists or unique complete names matching its requested parameters.
conditional	whether prior and posterior model probabilities should be returned only for the non-null models. Defaults to `FALSE`. Inclusion Bayes factors stay the unconditional inclusion odds; after conditioning, posterior inclusion is 1, so recomputing the Bayes factor from the renormalized probabilities would be `Inf`.
seed	integer specifying seed for sampling posteriors for model averaging. The caller's random-number state (`.Random.seed` and `RNGkind()`) is restored afterwards. Defaults to `NULL`, which draws the shared sampling seed from the caller's random-number stream (one draw).
n_samples	number of samples to be drawn for the model-averaged posterior distribution
on_failure	policy for missing (`NA`) marginal likelihoods in models with positive prior probability. `"error"` aborts (the default), `"drop"` removes failed models from the prior model space and renormalizes, and `"zero"` explicitly assigns the failed models zero evidence. Non-default policies emit a warning and return audit metadata. Only `NA` is treated as a computation failure under this policy. In model lists, store a failed computation as `bridgesampling_object(NA)`, a result flagged as failed whose marginal likelihood is read as `NA`. A finite failure must be reported as `NA`; by contrast, `-Inf` is an intentional zero-evidence shortcut (for example via `bridgesampling_object()`) and quietly receives posterior model probability zero without consulting `on_failure`.

Details

For simplex parameters such as Dirichlet priors, absent model parameters are not filled with an implicit spike at zero because the all-zero vector is off the simplex. Every model with positive

prior or posterior probability must provide an explicit compatible simplex prior or point prior on the simplex.

Mixture draws are aligned across parameters by restarting from one shared sampling seed and requiring identical posterior model probabilities for every requested parameter. That holds for unconditional averaging even when `is_null_list` differs by parameter. Conditional averaging (`conditional = TRUE`) with different null indicators per parameter produces different `post_probs` and therefore different mixture allocations; `mix_posteriors()` hard-errors in that case. Mix each conditional parameter in a separate call instead.

With `on_failure = "drop"` or `"zero"`, one warning is emitted for the model list, and every mixed parameter carries the `marglik_failure` attribute that `ensemble_inference()` returns for that parameter.

For an ordered total with a declared inclusion event, conditional averaging also conditions within each model. Posterior model probabilities are multiplied by the event's fraction of fitted draws; prior probabilities are multiplied by its declared prior probability. Resampling uses eligible original draw indices, and conditional prior contexts preserve the original fitted source specifications. A parameterized total without an inclusion event, including a Bernoulli total, retains every state. A compound event must be applied once to full aligned sources; separately conditioned ordered parameters cannot be combined as joint draws by this function.

Positive model declarations remain active when their visible probability rounds to zero. Numeric mixed draws remain usable. A prior law or atom law requiring model probabilities unavailable at full precision is explicitly refused with `BayesTools_formula_measure_unavailable` and the cause `numerical_model_probability_unavailable`; structured model/log diagnostics are retained in `posterior_metadata()`. Descriptive `marginal_posterior(use_formula = FALSE, prior_samples = FALSE)` remains available. An ordered projection with no allocated rows for a positive model requires a complete declared atom certificate; otherwise that measure has cause `structural_target_law_unavailable`. Independently known support and valid sibling measures remain available.

Value

`mix_posteriors` returns a named list of mixed posterior distributions (either a vector of matrix).

See Also

ensemble_inference BayesTools_ensemble_tables as_mixed_posteriors

---

`mpoint` *Multivariate point mass distribution*

Description

Density, distribution function, quantile function and random generation for multivariate point distribution.

Usage

```
dmpoint(x, location, log = FALSE)

rmpoint(n, location)

pmpoint(q, location, lower.tail = TRUE, log.p = FALSE)

qmpoint(p, location, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector or matrix of quantiles.
location	vector of locations corresponding to the location of individual points. Alternatively, a matrix with rows corresponding to the location of individual samples and columns correspond to the location of individual points.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
n	number of observations.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
p	vector of probabilities.

Value

dpoint gives the density, ppoint gives the distribution function, qpoint gives the quantile function, and rpoint generates random deviates.

Examples

```
# draw samples from a multivariate point distribution
rmpoint(10, location = c(0, 1))
```

---

parameter_catalog *Semantic parameter catalogs and deferred draw extraction*

Description

parameter_catalog() returns the semantic quantities and aliases from the fitted object's versioned parameter_map(). The catalog contains only metadata: selectable quantities, exact aliases, and serializable extraction keys. Square brackets after a factor term always hold a level label, never a coordinate position. Fixed factor terms, including ordinary factor priors whose levels are labelled 1, ..., K, expose every level or interaction cell as a quantity named <parameter>[<level>], so term[level], <parameter>[level], and term plus component = "level" all select the level with that label. Treatment and independent cells and the first ordered coordinate are structurally the fitted coordinate; reference cells are structural zeroes; and the remaining cells are reconstructed from the persisted term-only design matrix. A coordinate that is not a level cell (a mean-difference or

orthonormal coefficient, or a later ordered increment) is coefficient `j` of the contrast coding, named `<parameter>{j}` and displayed as `term{j}`. The total of an ordered term (`prior_ordered()`) is the quantity `<parameter>_ordered_total` displayed as `term{total}` (`term{total}:g{1}` for the slice of an ordinary factor; the inclusion indicator of a mixture total is displayed as `term{total} (inclusion)`), and the labels of earlier versions (`term_ordered_total`) remain aliases. Each share of a random allocation of an ordered term is a derived quantity, `<parameter>_ordered_allocation[<level>]` displayed as `term[share: level]` (support [0, 1], the Beta marginal of the Dirichlet prior); its earlier labels (`term_ordered_allocation[level]`) are aliases. The level names of transformed summaries, `<parameter>[dif: level]`, and the row labels those summaries display, `(formula) term[dif: level]` and `term[dif: level]`, are aliases of the level quantity. JAGS coordinate names such as `mu_g[1]` remain backend columns, used by `JAGS_materialize_draws()`, and do not select factor quantities. Ordinary level labels remain unchanged; syntax-sensitive characters (including curly braces) are percent-escaped and ambiguous interaction tokens are quoted so that every component remains hypothesis-safe and injective. These are coefficient-level quantities, distinct from estimated marginal means based on full predictions. Continuous coefficients retain their ordinary, canonical and fitted backend selectors; their backend names are not factor-level labels, so selectors such as `x[mu_x]` and `intercept[mu_intercept]` are unavailable. Parameter-map schema 10 requires refitting older maps rather than retaining their aliases. `parameter_coordinates()` is the linked concrete posterior-coordinate view; the catalog is the semantic view of the same fitted map. Random-effect canonical names follow `(formula) owner: quantity(arguments)`, with `owner:` omitted for a bare or unnamed one-entry random formula, retained for an explicitly named one-entry list, and required when multiple blocks need disambiguation. Parentheses contain parameter or coefficient names, while square brackets inside an argument contain its factor or index level. Examples include `(mu) study: sd(intercept)`, `(mu) study: cor(group[sensitivity],group[specificity])`, and `(mu) var_prop(study)`. A mean-difference or orthonormal factor term of a block is one exchangeable unit (see `prior_random()`) with one SD quantity, `sd(f)`; its correlations are reported by level, `cor(x,f[a])`, together with the multiple correlation `cor_total(x,f)` (support `[0, 1]`), and its coefficient coordinates `f{j}` appear only in raw tables: a selector such as `sd(f{1})` stops with class `BayesTools_selector_unavailable`, naming the unit's quantity. Formula-prefix omission is accepted as an alias. Additional aliases are limited to genuine semantic equivalences, such as a CS/HCS pairwise correlation referring to its shared `cor`. Random-effect quantities also carry centrally generated simplified aliases. These remove a sole intercept argument and may omit a redundant owner, but are considered by resolvers only when `simplify_names = TRUE`. For a log-intercept formula, `exp(intercept)` is an alias of the fitted coefficient quantity. Selecting that alias does not itself unscale the coefficient. RoBMA's `standardized_coefficients` setting selects the display scale independently of the alias.

Identity random-effect summaries reuse their sampled or structural coordinate. Summaries requiring a scale or covariance transformation expose only the transformed quantity; their fitted-scale inputs and all other private implementation coordinates remain dependencies rather than public catalog rows. A public coordinate that a variance allocation reads through `random_sd_source()` (a model parameter such as `tau`) is not a private coordinate of the allocation: it stays its own catalog quantity. Raw estimates tables (`JAGS_estimates_table(random_effects_summary = "raw")`) show these backend coordinates for inspection; their row labels are not catalog aliases. Each random extraction key records whether its source mapping is an identity, one-to-one transform, or composite, together with the source coordinate, prior, and transform when those are defined. Constructing or resolving the catalog never accesses posterior draws.

`parameter_catalog_extend()` adds plain-data quantities and aliases owned by another provider; the `"BayesTools"` provider name is reserved for native quantities. `parameter_catalog_resolve()`

applies optional namespace and component filters and returns a versioned selection only when the match is unique. Otherwise it stops with an error of class BayesTools_parameter_not_found or BayesTools_parameter_ambiguous (both also BayesTools_parameter_resolution_error; the condition names the matching quantities). A rendered label that is also the label of another quantity (e.g. the transformed label g[dif: b] of level b and the label of a level named dif: b) is ambiguous, while a rendered label never replaces the canonical name of another quantity. A contrast-coefficient selector term{j} whose coordinate is a level (the coordinates of treatment and independent contrasts and the first ordered coordinate have no coefficient quantity) stops with class BayesTools_selector_unavailable (also BayesTools_hypothesis_target and BayesTools_parameter_resolution_ whose message and field level name the level in the same form, e.g. g[10] for g{1} of a treatment factor with levels 5, 10, and 20. A selection that is malformed, stale, or does not belong to the catalog it is used with (for example one resolved from another fit) stops with an error of class BayesTools_parameter_resolution_error; resolve the parameter again from the fit's catalog.

parameter_catalog(fit, transform_scaled = FALSE) is the catalog of the fitted scale of the predictors. It has the quantities, identifiers, names, and aliases of the default catalog (transform_scaled = TRUE), with the fitted-coordinate support, status, fixed_value, source_type, and extraction_key of the random-effect SDs, variances, and correlations of blocks with scaled predictors (a bounded SD prior keeps its bounds, an SD that the original scale combines from several fitted coefficients is the fitted coefficient itself, and an SD fixed on the fitted scale is structural). It is the default catalog for a fit without scaled predictors and for quantities that the scales do not affect. A selection resolved from either catalog is accepted by parameter_draws(), parameter_prior_density(), parameter_transform(), parameter_gate_states(), random_effects_marginal_update_plan(), and parameter_mixed_posterior(), which use the matching scale: a selection of the fitted-scale catalog is extracted from the fit without its predictor scales, a selection of the default catalog from the fit as it is (so blanking the formula_scale of the fit is not needed to extract the fitted scale). A selection that belongs to neither catalog stops as described above, and parameter_mixed_posterior() stops for a selection of the fitted-scale catalog with transform_scaled = TRUE.

parameter_draws() is the deferred extraction boundary. The BayesTools fit method reads only the coordinates declared by the selected extraction key; a quantity whose source coordinates were not monitored (catalog status "unavailable") stops with an error of class BayesTools_refit_monitoring (also BayesTools_refit_required). Downstream packages can provide methods for package-owned derived quantities. For gated total-variance allocations, realized sd_total and var_total draws include the all-off zero branch. var_prop(...) draws are normalized over active components and are NA on draws where the realized allocation total is zero. Random-effect correlations (cor(...) of LKJ blocks; original-scale correlations in a block with scaled columns) are NA on draws where the correlation is undefined, i.e. where one of the SDs of its pair is zero; a correlation whose SD draws are not available (a row-indexed or unmonitored external SD source) cannot be masked and keeps its fitted draws. Each catalog quantity declares its definedness ("always", or the reason of possibly undefined draws: "correlation", "multiple_correlation" for a cor_total, or "allocation_active" for the variance shares of gated total-variance allocations) and its exact support from the prior provenance of its source coordinates (NULL when it is not derivable). A quantity with point masses in its prior density (an SD, variance, total, or variance share of an allocation with inclusion gates, or whose scale prior has a point component) lists exactly the locations of those atoms as the points of a "mixed" support; where the atoms cannot be derived (an external SD source without a BayesTools prior, a composite SD outside its allocation component), its hull is declared inexact (exact = FALSE). The mcmc.list returned by parameter_draws() carries these as draw metadata (posterior_metadata()): support, keyed by canonical name, and undefined_draws, a character vector named by the canonical names of the possibly undefined

quantities with their reasons. Summaries such as `ensemble_estimates_table()` accept missing draws only for columns carrying this declaration: callers that extract a numeric vector keep it by copying the element with `posterior_metadata(x, "undefined_draws") <- .`

`parameter_prior_density()` constructs a deterministic `prior_linear_density` for supported map-defined quantities: fitted coordinates and factor levels (the weighted sum of their fitted coordinates under the fitted prior, whose `multiply_by` scales enter only the linear predictor; structural levels are point masses at their fixed value), one-to-one transformations, Dirichlet allocation marginals, and allocation-derived component SDs. Variance proportions are conditional on positive allocation variance. A shared parent inclusion gate cancels from that conditional law, leaving the Dirichlet marginal `Beta(alpha_i, sum(alpha) - alpha_i)`. Independently gated components keep the mixed measure over the realized active set: atoms at 0 and 1, and a Beta mixture over nonempty sets of other active components. It returns `NULL` when the fitted map does not declare a supported deterministic prior composition; for fitted coordinates and factor levels, only when no fitted prior owns their coordinates (an owning prior-list entry that is not a BayesTools prior stops) or when an owning prior has parameters given as `expression()`s (JAGS-only support: the prior has no R-side law, and `parameter_mixed_posterior()` returns the draws and atoms without a prior density). The density records the source prior and transform it was built from, so its heights and region probabilities are evaluated on that prior's structural route (bounded-logit correlations on the refined numerical grid). Allocated SDs are the scale prior T times an independent multiplier of the fitted model, whose Dirichlet weights w span all components of the allocation: a component SD is T sqrt(k w_i) (k the number of components of a mean-variance allocation, 1 otherwise) while the gates of its allocation chain are on and 0 otherwise, and an allocation total (`sd_total`, `var_total`) is T sqrt(sum of w over the active components), 0 without an active component and T with all of them, times the allocation chain of its parents (their gates and Dirichlet shares) for a nested allocation. Their densities (and those of the variances) are exact: the atoms at zero, and the continuous parts as mixtures over the gate configurations of the scale prior and of the one-dimensional integrals over the Beta share margins (e.g., the total of an ungated allocation that splits the component of a gate-only allocation is the scale prior times the gate). SD components and totals of nested allocations with two or more allocation shares are products of density grids without such provenance, returned for plotting only (heights, region probabilities and point hypotheses are unavailable, and their ordinates name the reason). The SD of a gate-only allocation (one term with an inclusion gate) is the scale prior times its gate, and an inclusion indicator (of an allocation gate, or `inclusion(...)` of a component of a spike-and-slab or mixture SD prior) has the Bernoulli prior of its marginal inclusion probability (points at 0 and 1; the prior weight of the component, or the expected inclusion probability of a spike-and-slab prior) and the exact support {0, 1}. A pairwise correlation of an unstructured random-effect block with K columns and an LKJ(eta) prior has the exact LKJ marginal: `2 B - 1` with `B ~ Beta(eta - 1 + K/2, eta - 1 + K/2)` (Lewandowski et al. 2009). This is its prior on the fitted scale, also when the SDs of the block have inclusion gates or point masses (the correlation matrix is a priori independent of them), and on the original scale of scaled predictors when each coefficient of the pair is one rescaled fitted coefficient (e.g. two scaled slopes; the correlation is then the fitted one wherever both SDs are positive). An original-scale correlation that mixes fitted coefficients, such as that of the intercept and a slope of a centred predictor, combines the correlation with the SDs and returns `NULL`.

`parameter_transform()` returns the one-to-one map from the selected source coordinate to its public semantic quantity when that map exists. Composite quantities return `NULL`. The forward, inverse, and Jacobian helpers are the authoritative evaluators for these serializable descriptors, so downstream consumers do not need to reproduce backend correlation, allocation, or formula-scale transformations.

Usage

```
parameter_catalog(object, ...)

## S3 method for class 'BayesTools_fit'
parameter_catalog(object, transform_scaled = TRUE, ...)

parameter_catalog_schema()

parameter_catalog_extend(catalog, quantities, aliases, provider)

parameter_catalog_resolve(
  catalog,
  alias,
  namespace = NULL,
  component = NULL,
  simplify_names = FALSE
)

parameter_draws(object, selection, ...)

parameter_prior_density(object, selection, ...)

## S3 method for class 'BayesTools_fit'
parameter_draws(object, selection, model_samples = NULL, ...)

## S3 method for class 'BayesTools_fit'
parameter_prior_density(
  object,
  selection,
  n_grid = .prior_linear_density_default_grid(),
  tail_prob = .prior_linear_density_tail_prob(),
  ...
)

parameter_transform(object, selection)

parameter_transform_forward(values, transform)

parameter_transform_inverse(values, transform)

parameter_transform_jacobian(values, transform)
```

Arguments

object	fitted object.
...	arguments for methods.
transform_scaled	for the catalog of a fitted model, whether the random-effect quantities of blocks

	with scaled predictors are those of the original scale (TRUE, the default) or of the fitted scale of the standardized predictors (FALSE), as for parameter_mixed_posterior().
catalog	a BayesTools_parameter_catalog object.
quantities	quantity rows matching the quantity schema.
aliases	alias rows matching the alias schema.
provider	scalar provider name owning every added quantity.
alias	scalar exact canonical name or alias.
namespace	optional exact namespace filter.
component	optional exact component filter.
simplify_names	whether to accept centrally generated simplified random-effect aliases. Defaults to FALSE.
selection	a BayesTools_parameter_selection returned by parameter_catalog_resolve().
model_samples	optional numeric matrix containing the declared fitted source coordinates. This lets downstream summaries evaluate a selected semantic quantity on an already materialized posterior sample. A single coda::mcmc matrix retains its recorded sampling timing; plain matrices use the default timing. Ordered quantities also require the total, Gamma allocation and component indicator sources listed by JAGS_ordered_parameter_spec() in each term's source_coordinates; increment-only columns do not establish their semantic values or structural measure. Include those primitive sources, or omit model_samples to read the complete fitted draws. Missing supplied ordered sources raise BayesTools_ordered_coordinates_unavail
n_grid	number of grid points used for deterministic induced prior densities.
tail_prob	probability omitted from each continuous source-prior tail when constructing a finite numerical grid.
values	numeric values on the source scale for parameter_transform_forward() and parameter_transform_jacobian(), or on the public quantity scale for parameter_transform_invers
transform	a serializable transform descriptor returned by parameter_transform().

Value

parameter_catalog() and parameter_catalog_extend() return a BayesTools_parameter_catalog. parameter_catalog_schema() returns schema descriptions. parameter_catalog_resolve() returns a BayesTools_parameter_selection. parameter_draws() returns a coda::mcmc.list for BayesTools-owned quantities. parameter_prior_density() returns a prior_linear_density or NULL. parameter_transform() returns a serializable transform descriptor or NULL; the transform helpers return numeric values.

References

Lewandowski D, Kurowicka D, Joe H (2009). "Generating random correlation matrices based on vines and extended onion method." *Journal of Multivariate Analysis*, **100**(9), 1989–2001. doi:10.1016/j.jmva.2009.04.008.

parameter_gate_states *Per-draw gate and point states of a catalog quantity*

Description

parameter_gate_states() returns, for each draw, the state of the structural gates and point components that put one catalog quantity on its atoms: the inclusion gates of random-effect variance allocations (allocated SDs and variances, allocation totals, variance proportions, and inclusion indicators), and the component indicator of a mixture or spike-and-slab prior with point components (coefficients, factor levels, and random-effect SDs and variances of such priors). These are the states from which parameter_mixed_posterior() declares the posterior atoms.

Usage

```
parameter_gate_states(fit, selection, draws = NULL)
```

Arguments

fit	a model fitted with JAGS_fit().
selection	a selection of one catalog quantity, created with parameter_catalog_resolve().
draws	optional draws of the fitted coordinates to read the states from: a numeric matrix with one column per coordinate and one row per draw, an mcmc or mcmc.list object, or a named numeric vector holding one draw (as for JAGS_evaluate_deterministic()). It must contain the gate and indicator coordinates of the quantity. Defaults to the posterior draws of fit.

Value

NULL for a quantity without inclusion gates or point components. Otherwise a list with the per-draw elements

atom the atom of the quantity in the draw, NA where the quantity is on its continuous part or undefined. When known is FALSE, it holds only the gate atoms.

continuous whether the quantity is on its continuous part (defined and not on an atom; NA where that is not determined).

defined whether the quantity is defined in the draw (a variance proportion is undefined without an active component).

event whether the draw is in the inclusion event of the quantity (see parameter_mixed_posterior()), or NULL when the quantity has no such event.

and the elements

known a scalar: FALSE when the atom states are not determined by the draws (the component indicator of a prior with point components is not monitored).

prior_atoms for quantities whose atoms are the point components of a mixture, spike-and-slab, or selection prior, a data frame with the atom locations x and their prior masses mass from the prior's component weights (inclusion probabilities of spike-and-slab priors); NULL for the inclusion gates of variance allocations, whose prior masses are the point masses of parameter_prior_density().

See Also

parameter_mixed_posterior(), parameter_catalog()

---

parameter_labels *Parameter labels*

Description

Renders the labels of parameter quantities from their structured label parts: catalog quantities (the label_parts column of parameter_catalog()) and the columns of mixed and marginal posterior draws (the quantities draw metadata, see posterior_metadata()). The same renderer produces the catalog aliases, the rows of the summary tables, the column names of mixed posteriors, plot legends, and warning text, so a label shown by BayesTools is always a label of the same quantity everywhere.

Usage

```
parameter_labels(
  x,
  style = c("selector", "table", "plot", "warning"),
  formula_prefix = TRUE,
  simplify = FALSE,
  vocabulary = NULL
)
```

Arguments

x	a parameter catalog (or its quantities or aliases table), a parameter selection returned by parameter_catalog_resolve(), a data frame with a label_parts column (e.g. the quantities attribute of an estimates table), or a list of label parts.
style	one of "selector" (the exact selector, i.e., the canonical name of a catalog quantity; also for the quantities of an extension provider that have no label parts), "table" (the summary-table row label), "plot" (the plot-legend label: the level text of a level cell), or "warning" (the row label used in warnings and footnotes).
formula_prefix	whether table and warning labels show the formula parameter, as in (mu) x. Defaults to TRUE.
simplify	whether random-effect labels use their simplified display arguments (for example, sd for a sole random intercept). Defaults to FALSE.
vocabulary	optional named character vector renaming the random-effect quantities in the labels, e.g. c(sd = "tau", cor = "rho") renders (mu) study: tau(intercept) for (mu) study: sd(intercept). Quantities without an entry keep their names. Defaults to NULL.

Details

Square brackets after a factor term hold a level label (`g[b]`); interaction cells name the level of every factor (`g[b]:h[v]`, `g[b]:x`); contrast coefficients that are not level cells are written with curly braces (`g{1}`). Selectors percent-escape syntax-sensitive level characters and quote interaction tokens as the catalog does; table, plot, and warning labels show the level labels as they are. The quantities of an ordered term (`prior_ordered()`) follow the same grammar: its total is a coefficient of the term that is not a level, `f{total}` (`f:x{total}` for a term with a continuous predictor, written on the last component that is not an ordinary factor), and the share of the total reached at a level is a per-level quantity of the ordered factor, `f[share: mid]` (`f[share: mid]:x`, with the tag on the component of its ordered factor). When the ordinary factors of the term index several slices, the slice follows the last ordinary factor: `f{total}:g{1}`, `f[share: mid]:g{1}`, `g{1}:f{total}`. The inclusion indicator of a mixture total reads `f{total} (inclusion)`. The label parts of catalog aliases render the aliases with `style = "table"`; under a `vocabulary`, they render the aliases of the renamed random-effect quantities (including the pairwise aliases of the shared correlation of `cs()` and `hcs()` blocks).

Value

a character vector of labels.

See Also

`parameter_catalog()`

`parameter_map` *Fitted parameter map and coordinate view*

Description

`parameter_map()` returns the single versioned parameter map stored by `JAGS_fit()`. It contains three linked tables: concrete fitted `coordinates`, public semantic `quantities`, and exact `aliases`.

`parameter_coordinates()` returns the coordinate table from that map. Each row describes one sampled, structural, or unavailable fitted coordinate. Coordinate names are backend identities, not additional public aliases.

`parameter_map_schema()` documents all three tables, while `parameter_coordinates_schema()` documents the coordinate view alone.

Usage

```
parameter_map(object, ...)

## S3 method for class 'BayesTools_fit'
parameter_map(object, ...)

parameter_coordinates(object, ...)
```

```
## S3 method for class 'BayesTools_fit'
parameter_coordinates(object, ...)

parameter_coordinates_schema()

parameter_map_schema()
```

Arguments

object	a fitted object.
...	arguments for methods.

Value

`parameter_map()` returns a `BayesTools_parameter_map`. `parameter_coordinates()` returns a `BayesTools_parameter_coordinates` data frame. The schema helpers return data frames or a named list of them.

`parameter_map_cache` *Cache a value derived from a fitted parameter map*

Description

Stores one value per provider against a fitted parameter map, for the lifetime of the R session. It exists so that packages building on BayesTools can avoid recomputing map-derived metadata without inventing their own storage inside fitted objects.

The cache is keyed by the map *and* by `key`, a value naming everything else the cached result was derived from. Whenever `key` stops being `identical()` to the stored one the value is recomputed, so a result that also depends on data or priors stays correct when those change. Passing a `key` that does not cover every input is the one way to use this incorrectly.

Entries never travel with a saved fit, and BayesTools discards every provider's entries whenever the map's own tables are replaced.

Usage

```
parameter_map_cache(map, provider, key, compute)
```

Arguments

map	a parameter map, as returned by `parameter_map()`.
provider	name of the calling package.
key	a value identifying every input other than `map` that `compute` depends on. Compared with `identical()`.
compute	a function of no arguments returning the value to cache.

Value

The cached or freshly computed value of `compute()`.

See Also

`parameter_map()`

`parameter_mixed_posterior` *Mixed posterior of one catalog quantity*

Description

`parameter_mixed_posterior()` returns the posterior draws of one semantic quantity of the `parameter_catalog()` as a mixed posterior (the element form returned by `as_mixed_posteriors()`) together with the draw metadata that inference and plots need (`posterior_metadata()`):

`support` the exact support declared by the catalog.

`components` ordinary mixture/spike component indices and declared supports decoded from the same original fitted indicator rows before public coordinate filtering. Conditioned draws keep their original row identities; component supports never come from observed draw ranges.

`prior_density` the canonical prior density, `parameter_prior_density()` (conditional on the inclusion event with `conditional = TRUE`).

`atoms` the declared posterior point masses, from the structure of the quantity, never inferred from the draw values: the atoms on which the per-draw states of its inclusion gates and point components put it (`parameter_gate_states()`: the gates of a variance allocation, the mixture indicator of a scale prior with a point component, or the component indicator of a mixture or spike-and-slab prior of its fitted coordinates), with masses the shares of those draws and locations that match the point masses of the prior density when it is available. A quantity whose prior density has no point mass declares no atoms, and so does a quantity without a prior density none of whose fitted coordinates can take a point mass: each is owned by a continuous prior (including the Dirichlet weights of a variance allocation), is a continuous generated primitive (e.g. the correlations of an LKJ block), or is a generated random-effect SD or correlation node (`JAGS_deterministic_nodes()`) all of whose dependencies are such coordinates (e.g. the SDs of a variance allocation without inclusion gates whose scale prior has no point component, so the original-scale correlations of an allocated `us()` block declare no atoms). A random-effect correlation is scale-invariant, so a coordinate that enters its block only as a common factor of every SD (the scale source of the allocation that splits the block into its SDs, or an inclusion gate of the whole block, also with point masses) puts no atom on it: where the factor is 0, every SD is 0 and the correlation is undefined. The original-scale correlations of a block allocated from a gated or spike-and-slab scale therefore declare no atoms on their defined draws. Atoms remain undeclared (and posterior plots and Savage-Dickey ratios stop) when the point states are not in the draws (an unmonitored component indicator), when the quantity combines several coordinates of which some can take a point mass (e.g. original-scale random-effect SDs of a block with spike-and-slab SD priors, and

original-scale correlations of a block whose SDs have their own spike-and-slab priors or inclusion gates, which can be -1 or 1 where one SD is 0), and when the point structure of a fitted coordinate is not classified: coordinates of priors that are neither continuous, point, mixture, spike-and-slab, nor selection priors, and coordinates without a prior other than LKJ primitives, standardized random effects, generated random-effect SD and correlation nodes, and selection coordinates (e.g. `add_parameters`, such as a formula's linear predictor, which is constant on rows whose design is zero, and the `_indicator`, `_inclusion`, and `_variable` coordinates of mixture priors). The coordinates of a weight-function, publication-bias, or publication-bias mixture prior (the weights `omega`, `PET`, `PEESE`, and the p-hacking `alpha`, `pi_null`, `beta_null`, and `phack_kind`) are declared from the value each branch of the prior gives them: a constant weight (the reference bin, fixed weights, or 1 in a branch without a selection), 0 for `PET` and `PEESE` in branches without them and for the p-hacking parameters in branches without p-hacking, and the form code of `phack_kind`, with masses the shares of the draws whose branch (the `bias_indicator` of a mixture) gives the constant.

`undefined_draws` for quantities that are undefined on some fitted draws (the catalog `definedness`, e.g. variance proportions when no allocation component is active), the reason; those draws are omitted, and the prior density and atom masses are conditional on definedness.

`condition` the conditioning of the draws: unconditional (`averaged = TRUE`), or the inclusion event of the quantity.

`quantities` the catalog quantity of the draws: its id and label parts, from which `parameter_labels()` and the summaries render its labels.

Usage

```
parameter_mixed_posterior(
  fit,
  selection,
  conditional = FALSE,
  transform_scaled = TRUE
)
```

Arguments

`fit`	a model fitted with `JAGS_fit()`.
`selection`	a selection of one catalog quantity, created with `parameter_catalog_resolve()` from the catalog of the fit (`parameter_catalog()`) or, with `transform_scaled = FALSE`, from the catalog of the fitted scale (`parameter_catalog(fit, transform_scaled = FALSE)`); a selection of the latter with `transform_scaled = TRUE` stops.
`conditional`	whether to condition on the inclusion event of the quantity: the gates of its allocation chain on for an allocation-derived component SD or variance, and its own inclusion gate on for a variance proportion; for a quantity of the fitted coordinates of one mixture or spike-and-slab prior with point components (a coefficient, a factor level, a random-effect SD or variance, or the scalar correlation of a structured block on any `cor_scale`), the component indicator of that prior selecting a continuous component (the `"alternative"` components of a prior whose components are `"null"` and `"alternative"`, the event of `JAGS_estimates_table(conditional = TRUE)` and of `as_mixed_posteriors(conditional = <param`

An allocation total (sd_total, var_total) is conditioned on its parent-allocation (binding-factor) gates on when some component has no inclusion gate (some component is then always active), and on at least one component active when every component has an inclusion gate and the total has no parent gates. A total whose components all have inclusion gates and which also has parent gates combines the two kinds of gates, which are not one conjunction or disjunction, and is refused ("combines parent-allocation and component inclusion gates"). Only the draws in the event are kept, and the prior density is restricted to the event and renormalized (point masses inside the event, e.g. a variance proportion of one, keep their renormalized masses; the prior of a point-component quantity is restricted to its continuous components, with their prior probabilities renormalized to 1, on the scale of the quantity, and declares no atom, the point components being the complement of the event). The declared condition names the gates (the component indicator) of the event, and the declared support is that of the event: the catalog support under the restricted prior for a point-component quantity, and the catalog support without the points the event excludes (the atom at 0 of a gate that is off) for a gated allocation quantity. Quantities without an inclusion gate cannot be conditioned, nor can a total of an allocation without parent gates that has an ungated component, the coordinates of a selection prior (weight function, PET/PEESE, p-hacking), the inclusion indicator of a component, a quantity whose prior has other components than "null" and "alternative" or a point "alternative" component, one whose component indicator is not monitored, and a quantity that combines several coordinates of which some can take a point mass (e.g. original-scale SDs of a block with spike-and-slab SD priors); each stops with a message naming the reason.

transform_scaled

whether the random-effect quantities (SDs, variances, and correlations) of a fit with scaled predictors are reported on the original scale (TRUE, the default) or on the fitted scale of the standardized predictors (FALSE), as for JAGS_estimates_table(). The draws, the declared support and atoms, and the prior density share the scale. Fits without scaled predictors, and quantities that the scaling does not affect, are unchanged.

Value

A numeric vector of class mixed_posteriors, mixed_posteriors.simple, marginal_posterior.simple, and marginal_posterior (attribute parameter: the canonical name) with the draw metadata described above.

See Also

parameter_catalog(), parameter_prior_density(), posterior_metadata(), posterior_atoms_free()

---

parameter_names *JAGS parameter-name helpers*

Description

Format formula-derived parameter names and extract exact indexed JAGS parameter columns.

Usage

```
format_parameter_names(
  parameters,
  formula_parameters = NULL,
  formula_random = NULL,
  formula_prefix = TRUE,
  formula_scale = NULL
)

JAGS_parameter_names(parameters, formula_parameter = NULL)

JAGS_regex_escape(x)

JAGS_indexed_parameter_columns(columns, parameter)

JAGS_indexed_parameter_matrix(samples, parameter, drop_missing = TRUE)

JAGS_indexed_parameter_vector(row, parameter)
```

Arguments

parameters	a vector of parameter names
formula_parameters	a vector of formula parameter prefix names
formula_random	a vector of random effects grouping factors
formula_prefix	whether the formula_parameters names should be kept. Defaults to TRUE.
formula_scale	optional nested list containing scaling info. When provided, intercepts from parameters with log_intercept = TRUE attribute will be renamed to exp(intercept).
formula_parameter	a formula parameter prefix name
x	character vector to escape for regular-expression matching.
columns	character vector of column names.
parameter	exact unindexed JAGS parameter name.
samples	matrix, data frame, mcmc, or mcmc.list object.
drop_missing	whether missing indexed columns should return NULL. If FALSE, a zero-column matrix with the input row count is returned.
row	named row vector, list, or one-row data frame.

Value

format_parameter_names(), JAGS_parameter_names(), and JAGS_regex_escape() return character vectors. JAGS_indexed_parameter_columns() returns a logical vector. JAGS_indexed_parameter_matrix() returns a matrix of exact indexed parameter columns sorted by numeric index, or NULL when none are present and drop_missing = TRUE. JAGS_indexed_parameter_vector() returns indexed row values sorted by numeric index.

Examples

```
format_parameter_names(c("mu_x_cont", "mu_x_fac3t", "mu_x_fac3t__xXx__x_cont"),
                       formula_parameters = "mu")
```

---

parameter_source *External JAGS parameter source*

Description

parameter_source() describes an existing JAGS node that another generated model component may reference without creating or owning a prior for it.

random_sd_source() marks a parameter source as an external random-effect SD source. It is intended for variance allocations that should use an already-defined JAGS node, such as scalar tau or row-shaped tau, instead of creating a prior-owned total SD parameter.

Usage

```
parameter_source(
  name,
  shape = c("scalar", "row"),
  values = NULL,
  inputs = NULL
)

random_sd_source(source, shape = c("scalar", "row"))
```

Arguments

name	JAGS node name.
shape	source shape. "scalar" references one existing JAGS node; "row" references a vector node aligned to the formula rows. Generated row-shaped random-effect syntax indexes the node inside the observation loop.
values	optional function for R-side reconstruction of row-shaped sources. When supplied with shape = "row", it must accept named arguments parameters, data, and n_rows, or include ..., and return a numeric vector of length n_rows. Prediction and bridge-sampling reconstruction evaluate a supplied values function first; without one they use the posterior columns named name[1], ..., name[N]

when present. Supply `values` when those columns are absent or when prediction needs source values for arbitrary new rows (missing `values` then errors). For formula bridge reconstruction, `data` contains raw row-aligned formula/model data rather than standardized design matrices. Scalar sources do not accept `values`.

`inputs` — character vector declaring the posterior coordinates (column names such as `"tau_scale"` or `"beta[2]"`) that `values` reads from `parameters`; `character()` for a function of `data` alone. Required with `values` and supported only with it. Declared inputs are the dependencies of the source in `JAGS_deterministic_nodes()`, and `values` receives only these coordinates: reading any other parameter, or evaluating draws that lack a declared input, stops with an error.

`source` — JAGS node name or `parameter_source()` object. When `source` is a `parameter_source()` object, `shape` must not be supplied separately; the shape is inherited from `source`.

Details

Access control for formula bridge/prediction callbacks only guards named lookup of forbidden formula parameters and undeclared inputs on the stamped `parameters` list (`$` / `[[` / `[`). It is not a sandbox: callbacks may still close over external state, and a raw node name without `values` remains valid whenever the corresponding posterior columns (or non-new-row fitted mapping) suffice.

A `values` function without `inputs` stops with an error of class `BayesTools_missing_source_inputs` (also `BayesTools_parameter_source`), also when a source object without declared inputs is used.

Value

A list-like `parameter_source` object.

A list-like `random_sd_source` object.

`phack_pi_null` *P-hacking calibration helpers*

Description

`phack_pi_null()` converts the sampled severity `alpha` to the amount of null probability mass depleted from the source interval. `phack_alpha_from_pi_null()` applies the inverse calibration. `phack_backend_constants()` returns the z-scale constants used by the backend.

Usage

```
phack_pi_null(alpha, form, source, destination, target = 0.025)

phack_alpha_from_pi_null(pi_null, form, source, destination, target = 0.025)

phack_backend_constants(form, source, destination, target = 0.025)
```

Arguments

alpha	p-hacking severity parameter.
form	power depletion form, either "linear" or "quadratic".
source	source p-value cut point.
destination	destination p-value cut point.
target	target p-value cut point.
pi_null	null probability mass depleted from the source interval.

Value

Numeric vector for the calibration helpers and a named list for phack_backend_constants().

---

plot.prior *Plots a prior object*

Description

Plots a prior object

Usage

```
## S3 method for class 'prior'
plot(
  x,
  plot_type = "base",
  x_seq = NULL,
  xlim = NULL,
  x_range_quant = NULL,
  n_points = 1000,
  n_samples = 10000,
  force_samples = FALSE,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  show_figures = if (individual) -1 else NULL,
  individual = FALSE,
  rescale_x = FALSE,
  par_name = NULL,
  ...
)
```

Arguments

x	a prior
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
x_seq	sequence of x coordinates
xlim	x plotting range
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation , or a list containing the transformation function fun, inverse transformation function inv, and derivative of the transformation jac, evaluated on the original support. A custom transformation can also declare output_support = c(lower, upper). Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.
transformation_arguments	a list with named arguments for the transformation
transformation_settings	boolean indicating whether the explicitly supplied x_seq or x_range is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.
show_figures	which figures should be returned in case of multiple plots are generated. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions plotted individually by plot.prior() and plot_posterior(), figure k shows the publication weight omega of the k-th p-value interval in ascending order, the order in which summary tables (including RoBMA's) print

	the omegas; figure 1 is the reference weight of the most significant interval. In `plot.prior()`, a negative `k` omits figure `-k` (the default `-1` omits the reference weight fixed at 1). For a two-sided weight function prior, the intervals are its two-sided p-value bins; tables on the one-sided p-value scale print these weights first and then repeat them for the mirrored intervals.
`individual`	should individual densities be returned (e.g., in case of weightfunction)
`rescale_x`	allows to rescale x-axis in case a weightfunction is plotted.
`par_name`	a type of parameter for which the prior is specified. Only relevant if the prior corresponds to a mu parameter that needs to be transformed.
`...`	additional graphical arguments. For mixed continuous and point distributions, `ylim` controls the density axis, `ylim2` controls the probability-mass axis, and `ylab2` controls its label.

Details

Declared point locations whose transformation images are not representable stop with `BayesTools_transformation_image_` parent `BayesTools_transformation`. Its `call` is `NULL` and its `transformation`, `source_values` and `images` fields identify the matched failing finite entries; see `posterior_transform()`. Automatic prior-coordinate limits are selected on the source scale before transformation, regardless of `transformation_settings`. An endpoint with an infinite image uses the existing marginal tail convention described in `density.prior()`. Explicit `xlim` and `x_seq` retain their supplied source or transformed interpretation.

Ordered prior plots retain the continuous density of every non-reference level, including priors with an infinite density at an exact point. Only nonfinite plotting ordinates are omitted; finite ordinates and declared point masses are unchanged. An infinite density point is not a probability mass. This display rule also applies to `lines.prior()` and `geom_prior()` and leaves `density.prior()` and `prior_density_ordinate()` unchanged. Unknown or unsupported density routes keep their existing limitations. Simple priors, the marginals of simplex priors and individual omegas follow the same rule: nonfinite ordinates (such as the infinite density of a Gamma(0.5, 1) prior at zero) are not drawn, and the density axis ends at the largest finite ordinate. A curve without any finite ordinate stops with class `BayesTools_prior_curve_unavailable`; choose another `xlim`. Direct ordered plots and layers omit persisted zero-design reference levels from the default selection. Explicit selectors retain the original level numbering and can select a reference level. Genuine non-reference zero and point levels remain visible. Finite displayed peak heights depend on the evaluation grid and do not represent an infinite mathematical peak. A direct ordered curve with no finite plotting ordinate stops with class `BayesTools_prior_curve_unavailable` (also `BayesTools_plot_condition`); supply `x_seq` containing finite density ordinates. Probability-axis mapping and clipping checks use selected components only, including an explicitly selected reference.

Value

`plot.prior` returns either `NULL` or an object of class 'ggplot' if plot_type is `plot_type = "ggplot"`. Dirichlet simplex priors are plotted as one beta marginal per coordinate; the ggplot method returns a list unless a single figure is selected. Ordered ggplot results use the original figure indices: a multiple-figure selection returns a list through the largest selected index, with `NULL` entries for unselected figures, including the default reference level. A bare ggplot is returned only when exactly

one figure is selected; a selection of several figures always returns a positional list, even if one figure is visible. For an ordered level with mixed probability measure, the continuous density and exact probability-mass arrows are drawn together without rescaling either component. For ordered and Dirichlet simplex priors, `show_figures` holds non-zero integers within the number of figures that are all positive (select) or all negative (omit); other values stop with an error before any figure is drawn. Individual weight functions take a single figure.

See Also

`prior()` `lines.prior()` `geom_prior()`

Examples

```
# create some prior distributions
p0 <- prior(distribution = "point",  parameters = list(location = 0))
p1 <- prior(distribution = "normal", parameters = list(mean = 0, sd = 1))
p2 <- prior(distribution = "normal", parameters = list(mean = 0, sd = 1), truncation = list(0, Inf))

# a default plot
plot(p0)

# manipulate line thickness and color, change the parameter name
plot(p1, lwd = 2, col = "blue", par_name = bquote(mu))

# use ggplot
plot(p2, plot_type = "ggplot")

# utilize the ggplot prior geom
plot(p2, plot_type = "ggplot", xlim = c(-2, 2)) + geom_prior(p1, col = "red", lty = 2)

# apply transformation
plot(p1, transformation = "exp")
```

`plot.prior_linear_density` *Plot a prior linear density*

Description

Plots the prior density of a quantity that is a linear combination of fitted coordinates, as returned by `parameter_prior_density()` and `JAGS_formula_prior_density()`: the continuous part as a curve, and the point masses as arrows on a secondary probability axis.

Usage

```
## S3 method for class 'prior_linear_density'
plot(
  x,
```

```
  plot_type = "base",
  n_points = 1000,
  xlim = NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  par_name = NULL,
  ...
)
```

Arguments

x	a prior_linear_density object.
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
n_points	number of plotting points. A density that needs numerical integration (a Gaussian convolution, a scale mixture, or a sum of two non-normal terms) is plotted at no more than 200 equally spaced points plus its support bounds, jumps, peaks and atoms.
xlim	optional plotting range of length two, on the untransformed scale (on the transformed scale when transformation_settings = TRUE).
transformation	optional transformation applied to the density: one of the prepared transformations of density.prior() ("lin", "tanh", "exp", "exp_lin"), or a list with the functions fun, inv, and jac.
transformation_arguments	optional list with named arguments for the transformation.
transformation_settings	whether xlim is supplied on the transformed scale.
par_name	optional label of the x-axis.
...	additional graphical arguments, as in plot.prior(). For mixed continuous and point densities, ylim controls the density axis, ylim2 the probability axis, ylab2 its label, and scale_y2 the scaling factor of the probability axis. add = TRUE adds the density to the current base plot, which keeps the scaling of its probability axis, or, for plot_type = "ggplot", returns an overlay that is added to an existing prior plot with +.

Details

Continuous parts are drawn as curves and point masses as arrows on the probability axis, as for the priors of plot.prior(). A continuous part without an exact route that cannot be resolved is omitted with a warning of class BayesTools_prior_curve_unavailable (also BayesTools_plot_condition), and the declared point masses are still drawn. A density with neither a curve nor a point mass in the plotted range stops with an error.

Value

plot.prior_linear_density() returns an object of class 'ggplot' if plot_type = "ggplot" (an overlay of class BayesTools_prior_overlay if add = TRUE), and otherwise draws the base plot and returns NULL invisibly.

See Also

plot.prior() parameter_prior_density() JAGS_formula_prior_density() prior_density_ordinate()

---

plot_marginal *Plot samples from the marginal posterior distributions*

Description

Plot samples from the marginal posterior distributions

Usage

```
plot_marginal(
  samples,
  parameter,
  plot_type = "base",
  prior = FALSE,
  n_points = 1000,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  rescale_x = FALSE,
  par_name = NULL,
  dots_prior = list(),
  legend = TRUE,
  legend_title = NULL,
  legend_labels = NULL,
  legend_position = NULL,
  ...,
  density_method = c("KDE", "precomputed")
)
```

Arguments

samples	samples from a posterior distribution for a parameter generated by marginal_inference.
parameter	parameter name to be plotted.
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
prior	whether prior distribution should be added to the figure. When samples were prepared with as_mixed_posteriors(..., transform_scaled = TRUE), the transformed prior samples are automatically used.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation

exp exponential transformation

, or a list containing the transformation function `fun`, inverse transformation function `inv`, and derivative of the transformation `jac`, evaluated on the original support. A custom transformation can also declare `output_support = c(lower, upper)`. Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

`transformation_arguments`
: a list with named arguments for the `transformation`

`transformation_settings`
: boolean indicating whether the explicitly supplied `x_seq` or `x_range` is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

`rescale_x`
: allows to rescale x-axis in case a weightfunction is plotted.

`par_name`
: a type of parameter for which the prior is specified. Only relevant if the prior corresponds to a mu parameter that needs to be transformed.

`dots_prior`
: additional arguments for the prior distribution plot

`legend`
: whether factor legends should be drawn.

`legend_title`
: optional title for factor legends.

`legend_labels`
: optional labels for factor legend levels.

`legend_position`
: optional legend position for factor legends.

`...`
: additional arguments

`density_method`
: density source for continuous posterior curves. `"KDE"` computes a kernel density estimate. Plot methods with exact finite prior-support bounds apply boundary reflection. `"precomputed"` uses valid `posterior_density` metadata when matching the selected parameter and condition; a missing or unusable continuous curve stops with a remedy to supply matching metadata or use `"KDE"`. Pure-atom targets need no continuous curve. A precomputed density is the continuous part of the posterior; atoms are drawn from the declared posterior atoms with either method.

Details

Marginal posterior vectors may carry `posterior_density` metadata (`posterior_metadata()`) with `x` and `y` coordinates. These densities are used only when `density_method = "precomputed"`, which requires a matching valid curve for each selected continuous leaf. Pure-atom leaves need no continuous curve. Marginal plots with `density_method = "KDE"` use a standard KDE because marginal prior-density grids are numerical density ranges rather than true support metadata; this can intentionally differ from the support-reflected ordinate used by `Savage_Dickey_BF` when exact posterior-support metadata is available. A stored density describes the continuous part of the posterior; point masses are drawn from the declared posterior atoms (the `atoms` metadata, see `posterior_atom_attribute()`) with either density method. A declared continuous component

that received fewer than two distinct continuous draws has its curve omitted and its point masses drawn, with a warning of class `BayesTools_posterior_curve_unavailable` (also `BayesTools_plot_condition`), as in `plot_posterior()`. A prior curve without an exact route whose numerical grid cannot resolve a heavy-tailed product term is omitted with a warning of class `BayesTools_prior_curve_unavailable`, as in `plot_posterior()`, and so is the prior curve of draws that declare no prior (a `prior_list` of `prior_none()`) and carry no prior density. Other draws without a prior density (e.g. from `marginal_posterior()` without prior samples) stop. Direct ordered factor marginal plots omit persisted zero-design reference rows and retain true non-reference point levels. They use the retained level color/label list and matching dashed prior-overlay defaults of `plot_posterior()`. Every non-reference continuous prior retains its finite plotting ordinates, including priors with an exact infinite density point. Unresolved raw numerical evaluations warn once per continuous route with the classes above, distinguishing partial from whole curve unavailability and retaining available points and atoms. The warning's `unresolved_values` field contains evaluation coordinates on the source scale, before any plotting transformation. Use `prior = FALSE` to draw the posterior alone. Exact infinite density and zero-density points remain valid, and finite displayed peak heights depend on the evaluation grid.

Value

`plot_marginal` returns either `NULL` or an object of class 'ggplot' if plot_type is `plot_type = "ggplot"`.

See Also

`prior()` `marginal_inference()` `plot_posterior()`

`plot_models` *Plot estimates from models*

Description

Plot estimates from models

Usage

```
plot_models(
  model_list,
  samples,
  inference,
  parameter,
  plot_type = "base",
  prior = FALSE,
  conditional = FALSE,
  order = NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
```

```
  par_name = NULL,
  formula_prefix = TRUE,
  ...
)
```

Arguments

model_list	list of models, each of which contains marginal likelihood estimated with bridge sampling marglik and prior model odds prior_weights
samples	samples from a posterior distribution for a parameter generated by mix_posteriors or as_mixed_posteriors.
inference	object created by ensemble_inference function
parameter	parameter name to be plotted. Does not support PET-PEESE and weightfunction.
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
prior	whether prior distribution should be added to the figure. When samples were prepared with as_mixed_posteriors(..., transform_scaled = TRUE), the transformed prior samples are automatically used.
conditional	whether conditional models should be displayed
order	list specifying ordering of the models. The first element describes whether the ordering should be "increasing" or "decreasing" and the second element describes whether the ordering should be based "model" order, "estimate" size, posterior "probability", or the inclusion "BF".
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x **tanh** hyperbolic tangent transformation **exp** exponential transformation , or a list containing the transformation function fun, inverse transformation function inv, and derivative of the transformation jac, evaluated on the original support. A custom transformation can also declare output_support = c(lower, upper). Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.
transformation_arguments	a list with named arguments for the transformation
transformation_settings	boolean indicating whether the explicitly supplied x_seq or x_range is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.
par_name	a type of parameter for which the prior is specified. Only relevant if the prior corresponds to a mu parameter that needs to be transformed.

formula_prefix whether the formula_parameters names should be kept. Defaults to TRUE.

... additional arguments. E.g.:

"show_updating" whether Bayes factors and change from prior to posterior odds should be shown on the secondary y-axis

"show_estimates" whether posterior estimates and 95% CI should be shown on the secondary y-axis

"y_axis2" whether the secondary y-axis should be shown

Details

Plots prior and posterior estimates of the same parameter across multiple models (prior distributions with orthonormal/meandif contrast are always plotted as differences from the grand mean). Ordered factors use the retained semantic level effects, omitting their zero-design reference rows, and per-model transformed level summaries. The model-averaged estimate uses the selected scalar level draws. With prior = FALSE, no prior means or intervals are computed or transformed; model probabilities and Bayes factors remain available for updating labels. With prior = TRUE, ordered level means use the total mean and exact allocation expectations with the persisted design. Their equal-tail 95\ intervals use structural prior probabilities and generalized quantiles, including exact jumps at point masses. No sampled prior means or quantiles are substituted. An undefined mean, unresolved expression, unsupported structural probability route, or unavailable numerical diagnostics raises BayesTools_prior_interval_unavailable (also BayesTools_plot_condition), identifying the model and level; set prior = FALSE to plot posterior estimates.

Value

plot_models returns either NULL or an object of class 'ggplot' if plot_type is plot_type = "ggplot".

See Also

prior() lines_prior_list() geom_prior_list()

---

plot_posterior *Plot samples from the mixed posterior distributions*

Description

Plot samples from the mixed posterior distributions

Usage

```
plot_posterior(
  samples,
  parameter,
  plot_type = "base",
  prior = FALSE,
  n_points = 1000,
  n_samples = 10000,
```

```
  force_samples = FALSE,
  individual = FALSE,
  show_figures = NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  rescale_x = FALSE,
  par_name = NULL,
  effect_direction = "positive",
  dots_prior = list(),
  data = NULL,
  show_data = FALSE,
  dots_data = list(),
  legend = TRUE,
  legend_title = NULL,
  legend_labels = NULL,
  legend_position = NULL,
  ...,
  density_method = c("KDE", "precomputed")
)
```

Arguments

samples	samples from a posterior distribution for a parameter generated by mix_posteriors or as_mixed_posteriors.
parameter	parameter name to be plotted. Use "PETPEESE" for PET-PEESE plot with parameters "PET" and "PEESE", and "weightfunction" for plotting a weightfunction with parameters "omega". The combined "PETPEESE" curve requires individual = FALSE; use "PET" or "PEESE" for individual coefficient densities.
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
prior	whether prior distribution should be added to the figure. When samples were prepared with as_mixed_posteriors(..., transform_scaled = TRUE), the transformed prior samples are automatically used.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
individual	should individual densities be returned (e.g., in case of weightfunction)
show_figures	which figures should be returned in case of multiple plots are generated. Useful when priors for the omega parameter are plotted and individual = TRUE. For

weight functions plotted individually by `plot.prior()` and `plot_posterior()`, figure `k` shows the publication weight omega of the `k`-th p-value interval in ascending order, the order in which summary tables (including RoBMA's) print the omegas; figure 1 is the reference weight of the most significant interval. In `plot.prior()`, a negative `k` omits figure `-k` (the default `-1` omits the reference weight fixed at 1). For a two-sided weight function prior, the intervals are its two-sided p-value bins; tables on the one-sided p-value scale print these weights first and then repeat them for the mirrored intervals.

`transformation` transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations:

lin linear transformation in form of `a + b*x`

tanh hyperbolic tangent transformation

exp exponential transformation

, or a list containing the transformation function `fun`, inverse transformation function `inv`, and derivative of the transformation `jac`, evaluated on the original support. A custom transformation can also declare `output_support = c(lower, upper)`. Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

`transformation_arguments` a list with named arguments for the `transformation`

`transformation_settings` boolean indicating whether the explicitly supplied `x_seq` or `x_range` is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

`rescale_x` allows to rescale x-axis in case a weightfunction is plotted.

`par_name` a type of parameter for which the prior is specified. Only relevant if the prior corresponds to a mu parameter that needs to be transformed.

`effect_direction` direction of the effect for PET-PEESE regression. Use `"positive"` (default) for `mu + PET*se + PEESE*se^2` or `"negative"` for `mu - PET*se - PEESE*se^2`.

`dots_prior` additional arguments for the prior distribution plot

`data` optional numeric vector of observed p-values in `[0, 1]`, or a data frame with a `p` column. Used only for weightfunction plots.

`show_data` whether observed p-values should be shown as rug marks on the weightfunction x-axis.

`dots_data` additional styling arguments for observed p-value rug marks. Supports `col`/`color`, `alpha`, `lwd`/`linewidth`, `side`/`rug_side`, and `height`/`rug_height`.

`legend` whether factor legends should be drawn.

`legend_title` optional title for factor legends.

`legend_labels` optional labels for factor legend levels.

`legend_position`
: optional legend position for factor legends.

`...`
: additional graphical arguments. For mixed continuous and point distributions, `ylim` controls the density axis, `ylim2` controls the probability-mass axis, and `ylab2` controls its label.

`density_method`
: density source for continuous posterior curves. `"KDE"` computes a kernel density estimate. Plot methods with exact finite prior-support bounds apply boundary reflection. `"precomputed"` uses valid `posterior_density` metadata when matching the selected parameter and condition; a missing or unusable continuous curve stops with a remedy to supply matching metadata or use `"KDE"`. Pure-atom targets need no continuous curve. A precomputed density is the continuous part of the posterior; atoms are drawn from the declared posterior atoms with either method.

Details

When using scaled predictors (via `formula_scale_list` in JAGS_fit), you can plot posteriors on the original (unscaled) scale by preparing samples with `as_mixed_posteriors(..., transform_scaled = TRUE)`. The function automatically detects this and uses the pre-computed transformed prior samples when `prior = TRUE`.

Posterior sample vectors may carry `posterior_density` metadata (`posterior_metadata()`) with `x` and `y` coordinates. These densities are used only when `density_method = "precomputed"`. A stored density describes the continuous part of the posterior; point masses are drawn from the declared posterior atoms (the `atoms` metadata, see `posterior_atom_attribute()`) with either density method. A declared continuous component (or the continuous part of a factor level) that received fewer than two distinct continuous draws has no kernel density, as for a model of posterior probability near or below one over `n_samples` in `mix_posteriors()`: its curve is omitted and the declared point masses are drawn, with a warning of class `BayesTools_posterior_curve_unavailable` (also `BayesTools_plot_condition`) that reports the declared continuous mass and the number of continuous draws; use a larger `n_samples` in `mix_posteriors()`. Without declared point masses to draw, the plot stops. `plot_marginal()` does likewise.

Prior-density curves are evaluated on their exact routes where they exist. A combination without an exact route is drawn from its numerical grid; when that grid cannot resolve the scale of a heavy-tailed product term (e.g., an intercept plus a level of an ordered factor with a Cauchy total), the prior curve is omitted with a warning of class `BayesTools_prior_curve_unavailable` (also `BayesTools_plot_condition`) and the rest of the plot is drawn. Posterior draws that declare no prior (a `prior_list` of `prior_none()`, as `parameter_mixed_posterior()` draws of a quantity without a prior density) and carry no prior density have no prior curve: with `prior = TRUE`, the posterior is drawn alone with a warning of the same class. The same holds for a prior with parameters given as `expression()`s, which only JAGS evaluates and which has no R-side prior law (unless the atoms or the support of the draws are unavailable, which `posterior_measure_check()` reports). Likewise, when the automatic display range of a transformed prior is unavailable (the marginal-quantile endpoints of a very wide prior have no finite image under `transformation = "exp"`, see `density.prior()`), the prior curve is omitted with the same warning, which names `xlim`, and the posterior is drawn alone; an explicit `xlim` avoids the omission. Raw numerical plotting evaluations that remain unresolved warn once per continuous route with the same classes. The warning distinguishes a partly available curve from a wholly unavailable one; every available curve point and declared atom is retained. Its `unresolved_values` field contains the evaluation coordinates on the

source scale, before any plotting transformation. Use `prior = FALSE` to draw the posterior alone. Exact infinite density points and exact zero densities do not trigger this warning. Finite displayed peak heights depend on the evaluation grid.

Ordered factors are plotted as semantic level effects. Their persisted zero-design reference rows are omitted for both raw and already transformed inputs; genuine non-reference zero and point levels remain. Both finite and singular non-reference prior curves retain their actual finite ordinates. Colors and `legend_labels` refer to the full retained non-reference level list. Visible curves and point masses determine the axes. Ordered prior overlays default to the corresponding posterior colors and dashed lines; explicit `dots_prior` `col`, `lty`, or `linetype` override these defaults. The posterior owns the legend.

For base plots, the initial call establishes the density-to-probability mapping. Calls with `add = TRUE` reuse that mapping so point masses from separate prior or posterior objects remain comparable. Point masses outside the active `ylim2` are clipped with a warning; redraw the initial plot with wider limits to display them.

Value

`plot_posterior` returns either `NULL` or an object of class 'ggplot' if plot_type is `plot_type = "ggplot"`.

See Also

`prior()` `lines_prior_list()` `geom_prior_list()`

`plot_prior_list` *Plot a list of prior distributions*

Description

Plot a list of prior distributions

Usage

```
plot_prior_list(
  prior_list,
  plot_type = "base",
  x_seq = NULL,
  xlim = NULL,
  x_range_quant = NULL,
  n_points = 500,
  n_samples = 10000,
  force_samples = FALSE,
  individual = FALSE,
  show_figures = if (individual) 1 else NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
```

```
  rescale_x = FALSE,
  par_name = NULL,
  prior_list_mu = NULL,
  effect_direction = "positive",
  legend = TRUE,
  legend_title = NULL,
  legend_labels = NULL,
  legend_position = NULL,
  ...
)
```

Arguments

prior_list	list of prior distributions
plot_type	whether to use a base plot "base" or ggplot2 "ggplot" for plotting.
x_seq	sequence of x coordinates
xlim	x plotting range
x_range_quant	quantile used for automatically obtaining x_range if both x_range and x_seq are unspecified. A number in the open interval (0, 0.5); other values are refused. Defaults to 0.005 for all but Cauchy, Student-t, Gamma, inverse-gamma, moment, and inverse-moment distributions that use 0.010.
n_points	number of equally spaced points in the x_range if x_seq is unspecified
n_samples	number of samples from the prior distribution if the density cannot be obtained analytically (or if samples are forced with force_samples = TRUE)
force_samples	should prior be sampled instead of obtaining analytic solution whenever possible. For supported ordered mixed measures, exact analytic curves and atoms are retained and the requested samples are attached to each component. Supported scalar ordered literal point totals and mixtures whose total components are all literal points do not sample, store draws, or advance the RNG when force_samples = FALSE. With force_samples = TRUE, their original seeded draws are attached without replacing the analytic measure. Other ordered families retain their existing sampling fallback.
individual	should individual densities be returned (e.g., in case of weightfunction)
show_figures	which figures should be returned in case of multiple plots are generated. Useful when priors for the omega parameter are plotted and individual = TRUE. For weight functions plotted individually by plot.prior() and plot_posterior(), figure k shows the publication weight omega of the k-th p-value interval in ascending order, the order in which summary tables (including RoBMA's) print the omegas; figure 1 is the reference weight of the most significant interval. In plot.prior(), a negative k omits figure -k (the default -1 omits the reference weight fixed at 1). For a two-sided weight function prior, the intervals are its two-sided p-value bins; tables on the one-sided p-value scale print these weights first and then repeat them for the mirrored intervals.
transformation	transformation to be applied to the prior distribution. Either a character specifying one of the prepared transformations: **lin** linear transformation in form of a + b*x

tanh hyperbolic tangent transformation

exp exponential transformation

, or a list containing the transformation function `fun`, inverse transformation function `inv`, and derivative of the transformation `jac`, evaluated on the original support. A custom transformation can also declare `output_support = c(lower, upper)`. Display grids on the transformed scale are evaluated only within these bounds; a wider plotting range is retained without extending the density. Infinite inverse values at support boundaries are omitted from continuous density grids. Point masses retain their transformed locations.

`transformation_arguments`
: a list with named arguments for the `transformation`

`transformation_settings`
: boolean indicating whether the explicitly supplied `x_seq` or `x_range` is on the transformed scale. Automatically selected prior-coordinate ranges are always obtained on the source scale before the density family's existing transformation handling.

`rescale_x`
: allows to rescale x-axis in case a weightfunction is plotted.

`par_name`
: a type of parameter for which the prior is specified. Only relevant if the prior corresponds to a mu parameter that needs to be transformed.

`prior_list_mu`
: list of priors for the mu parameter, required for the PET-PEESE regression bands (`individual = FALSE`) and refused otherwise: an individual plot shows the density of one coefficient and does not use it

`effect_direction`
: direction of the effect for PET-PEESE regression. Use `"positive"` (default) for `mu + PET*se + PEESE*se^2` or `"negative"` for `mu - PET*se - PEESE*se^2`.

`legend`
: whether factor legends should be drawn.

`legend_title`
: optional title for factor legends.

`legend_labels`
: optional labels for factor legend levels.

`legend_position`
: optional legend position for factor legends.

`...`
: additional graphical arguments. For mixed continuous and point distributions, `ylim` controls the density axis, `ylim2` controls the probability-mass axis, and `ylab2` controls its label.

Details

Automatic prior-coordinate ranges use a common source grid before transformation, regardless of `transformation_settings`. An endpoint with an infinite image uses the union of participating component marginal tail quantiles, as described in `density.prior()`. This display convention keeps exact atoms, mixture weights and prior support; explicit coordinates retain their supplied interpretation. The nonindividual PET/PEESE and weightfunction axes keep their standard-error and p-value meanings.

Value

`plot_prior_list` returns either `NULL` or an object of class 'ggplot' if plot_type is `plot_type = "ggplot"`.

See Also

prior() lines_prior_list() geom_prior_list()

---

plot_transformed_prior *Plot Transformed Prior Densities*

Description

Plots a prior density after applying the same deterministic coefficient transformation used for formula-scale back-transforms. The helper supports continuous densities and point-mass priors in the same plot. The density is that of the monitored coefficient: a prior's `multiply_by` scaling applies only to the linear predictor and is not applied here.

Usage

```
plot_transformed_prior(
  prior_list,
  column_names,
  formula_scale = NULL,
  parameter,
  n_points = 1000,
  x_range = NULL,
  transformation = NULL,
  transformation_arguments = NULL,
  transformation_settings = FALSE,
  plot_type = c("base", "ggplot"),
  par_name = NULL,
  ...
)
```

Arguments

`prior_list`	named list of prior distributions.
`column_names`	character vector of coefficient column names defining the fitted parameter space.
`formula_scale`	optional nested formula-scale metadata: the `formula_scale` attribute of the fitted model (`attr(fit, "formula_scale")`), which carries the fitted design of each formula. Standardization information without the fitted design stops with a `BayesTools_formula_transform_unavailable` error.
`parameter`	coefficient name to plot.
`n_points`	number of plotting points. A prior density that needs numerical integration (a Gaussian convolution, a scale mixture, or a sum of two non-normal terms) is plotted at no more than 200 equally spaced points plus its support bounds, jumps, peaks and atoms.
`x_range`	optional plotting range on the untransformed scale.

transformation	optional output transformation passed to the prior plotting machinery.
transformation_arguments	optional list of transformation arguments.
transformation_settings	whether x_range is supplied on the transformed scale.
plot_type	either "base" or "ggplot".
par_name	optional plotting label.
...	additional plotting arguments.

Value

For plot_type = "base", invisibly returns base-plot metadata. For plot_type = "ggplot", returns a ggplot object. Returns NULL when the requested parameter is an identity transform and no output transformation was requested. A parameter without a prior in prior_list stops with an error. A prior density that is unavailable stops with an error of class BayesTools_formula_measure_unavailable; in particular, the standard deviation of a random-effect block with a scaled column has no density on this route (it needs the fitted design of the block) and is refused rather than returned as an identity.

See Also

plot_prior_list() transform_scale_samples()

---

point *Point mass distribution*

Description

Density, distribution function, quantile function and random generation for point distribution.

Usage

```
dpoint(x, location, log = FALSE)

rpoint(n, location)

ppoint(q, location, lower.tail = TRUE, log.p = FALSE)

qpoint(p, location, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector or matrix of quantiles.
location	vector of locations.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
n	number of observations.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
p	vector of probabilities.

Value

dpoint gives the density, ppoint gives the distribution function, qpoint gives the quantile function, and rpoint generates random deviates.

Examples

```
# draw samples from a point distribution
rpoint(10, location = 1)
```

---

posterior_atoms_free *Whether posterior draws are declared atom-free*

Description

Reads the declared posterior atoms of BayesTools posterior draws (the atoms metadata, see posterior_metadata()) and reports whether the draws are structurally atom-free.

Usage

```
posterior_atoms_free(x)
```

Arguments

x	BayesTools posterior draws, e.g. an element returned by as_mixed_posteriors(), marginal_posterior(), or parameter_mixed_posterior().

Value

TRUE when atoms are declared and all declared scalar marginals and the joint measure have no positive atomic mass; FALSE when the draws declare a point mass or do not declare their atom status. Plain numeric draws, which carry no metadata, are an error. An explicitly unavailable joint law signals its retained typed condition, even when one or more scalar marginals have valid certificates.

Examples

```
draws <- structure(stats::rnorm(10), class = c("marginal_posterior.simple", "marginal_posterior"))
posterior_atoms_free(draws)
posterior_metadata(draws, "atoms") <- posterior_atom_attribute()
posterior_atoms_free(draws)
```

posterior_atom_attribute

Declare posterior point-mass metadata

Description

Creates exact posterior measure metadata used to distinguish a continuous posterior from one containing atoms. Supply NULL to explicitly declare that the posterior has no point masses.

Usage

```
posterior_atom_attribute(point_masses = NULL, source = "user")
```

Arguments

point_masses	optional data frame or list with x and mass entries.
source	short label describing the source of the declaration.

Value

A posterior atom metadata object for the atoms metadata of posterior draws (posterior_metadata(x, "atoms") <-). Posterior point masses are read only from these metadata: precomputed posterior densities (posterior_density_attribute()) describe the continuous part and carry no point masses. BayesTools producers can additionally declare positional scalar marginals, named in the full draw-column order. Scalar extraction prefers these declarations; a continuous joint measure can have a point marginal, for example the last level of an ordered point-total prior. When a producer can certify only scalar marginals, the object records joint_declared = FALSE and retains its typed joint refusal in joint_unavailable. Its empty joint table is dimensional storage. It does not declare an atom-free joint law. Scalar selection uses only certified marginals; whole-law inference retains the joint refusal.

posterior_density_attribute

Posterior density and ordinate attributes

Description

Construct and inspect posterior-density and posterior-ordinate attributes in the schema consumed by Savage_Dickey_BF and hypothesis_BF.

Usage

```
posterior_density_attribute(
  x,
  y,
  method,
  density_method,
  diagnostics = NULL,
  support = NULL,
  ...
)

posterior_ordinate_attribute(
  value,
  ordinate,
  method,
  density_method,
  diagnostics = NULL,
  ...
)

posterior_ordinate_append(existing, ordinate)

posterior_ordinate_supports_bf(ordinate, validator = NULL)

posterior_ordinate_has_value(ordinate, value)
```

Arguments

x	numeric density grid locations. Heights at repeated locations are averaged when the attribute is read.
y	numeric density grid heights.
method	estimator label stored in the attribute.
density_method	public density method label stored in the attribute.
diagnostics	optional estimator diagnostics.
support	optional exact support metadata for the density scale, created with posterior_support_attribute(). KDE boundary reflection uses only interval-capable support; support with exact = FALSE (plotting or integration limits rather than true support boundaries) is not used to exclude a null hypothesis.
...	additional named metadata fields, for example parameter, conditional, or conditional_rule.
value	numeric null-hypothesis value or values.
ordinate	for posterior_ordinate_attribute(), numeric posterior ordinate height or heights; for posterior_ordinate_append(), a posterior-ordinate attribute to append.
existing	existing posterior-ordinate attribute or NULL.

`validator` optional function for stricter method-specific BF support checks. It is called only after the generic BayesTools schema is valid.

Details

Additional metadata must be named and cannot replace structural schema fields. Posterior ordinate values are unique so a point-null lookup maps to at most one stored ordinate. When a validator is supplied to `posterior_ordinate_supports_bf()`, it is applied to matching ordinate entries rather than to the multi-ordinate container as a whole. The `support` field is used to check whether exact support excludes a requested point-null value and, when a kernel-density fallback must be computed from posterior samples, to supply boundary reflection. It must describe the true posterior support on the same scale as `x`, not the finite range of an estimator grid.

The objects returned by these constructors are the only accepted posterior density and ordinate metadata: other objects set as `posterior_density` or `posterior_ordinate` metadata of posterior draws (`posterior_metadata()`) are rejected with an error. An unclassed list of such objects, named by parameter or level, can hold the metadata of several parameters or levels. The fields used to match the metadata to posterior samples are `parameter`, `conditional`, `conditional_rule`, and `condition_key`. Multi-ordinate matching validates the original attributes and filters each single attribute by its own parameter and condition metadata before exact-null lookup. Container metadata do not replace incompatible child metadata. Compatible entries at other null values remain relevant metadata and retain the existing refusal when the requested null is absent.

A stored density describes only the continuous part of the posterior: for a posterior with point masses, its heights integrate to the continuous mass. The point masses are the posterior atoms of the draws, which are declared only in their `atoms` metadata with `posterior_atom_attribute()`; densities do not carry point masses.

Value

`posterior_density_attribute()` returns a list suitable for the `posterior_density` metadata of posterior draws (`posterior_metadata()`).

`posterior_ordinate_attribute()` returns a list suitable for the `posterior_ordinate` metadata of posterior draws (`posterior_metadata()`).

`posterior_ordinate_append()` returns a posterior-ordinate attribute container that may hold multiple ordinates.

`posterior_ordinate_supports_bf()` and `posterior_ordinate_has_value()` return scalar logical values.

See Also

`posterior_log_ordinate_attribute()`

posterior_density_method_match *Posterior density method helpers*

Description

Helpers for normalizing public posterior-density method arguments and identifying the method that uses precomputed posterior density or ordinate metadata. Only `"precomputed"` does; packages with their own estimators (e.g., qCMDE or IWMDE) map their method names to `"precomputed"` themselves.

Usage

```
posterior_density_method_match(
  method,
  allowed = c("KDE", "precomputed"),
  name = "method"
)

posterior_density_method_uses_precomputed(method)
```

Arguments

`method`	density method. Method names match exactly: the match is case-sensitive and the whole name is required (no partial matching).
`allowed`	character vector of allowed density methods.
`name`	argument name used in error messages.

Value

`posterior_density_method_match()` returns a scalar character value; the full default vector `allowed` selects its first method. `posterior_density_method_uses_precomputed()` returns a scalar logical value.

posterior_formula_measure_check *Metadata checks of the linear targets of posterior draws*

Description

Checks of the metadata that the draws of a formula coefficient or a factor level carry about the linear map of the fitted coordinates they are a function of. The fit-level hypothesis tests (`hypothesis_BF()`) and the density backends (`hypothesis_density_backend()`) use them to describe a target.

`posterior_formula_measure_check()` stops with the declared refusal (`BayesTools_formula_measure_unavailable`) when the draws declare the `measure` (`"prior_density"` or `"atoms"`) unavailable. `posterior_linear_weight_space()` returns the coordinate space of the linear weights of the draws (`"coefficient"` or `"formula_contribution"`, `NULL` without weights) and stops with `BayesTools_refit_required` (preceded by `refit_class`) for outdated metadata. `posterior_fitted_linear_weights()` returns the weights on the fitted coordinates, with the declared constant multipliers of formula contributions folded in; stochastic multipliers have no raw fitted linear projection and stop with the classes `BayesTools_fitted_linear_weights_unavailabl` and its parent `BayesTools_formula_transform_unavailable` (and no class of a hypothesis refusal, so that a caller outside of `hypothesis_BF()` gives it its own). The condition has the fields `reason` (`"stochastic_multiplier"`) and `cause` (`"original_scale"`, the cause for which the density backends of a hypothesis refuse the target); its message says that the ordinate density methods are unavailable.

Usage

```
posterior_formula_measure_check(sample, measure)

posterior_linear_weight_space(sample, build = "BayesTools", refit_class = NULL)

posterior_fitted_linear_weights(
  sample,
  weights,
  ordinate_label = "Ordinate density methods",
  build = "BayesTools",
  refit_class = NULL
)
```

Arguments

`sample`	posterior draws with metadata (`posterior_metadata()`).
`measure`	the measure to check.
`build`	the name of the build in the request to refit a model with outdated metadata ("Recreate the posterior with the current <build> build.").
`refit_class`	condition classes that precede `BayesTools_refit_required` in that request, or `NULL`.
`weights`	a named numeric vector of weights on the fitted coordinates.
`ordinate_label`	how the message names the density methods that compute posterior ordinates.

See Also

`posterior_measure_check()`, `hypothesis_BF()`

posterior_log_ordinate_attribute
Posterior ordinate attributes from log heights

Description

Constructs a posterior-ordinate attribute from the natural logarithm of the posterior heights. It is the log-height counterpart of `posterior_ordinate_attribute()`, for posterior ordinates whose natural-scale height underflows: `posterior_ordinate_attribute()` requires finite positive heights, while a log height only has to be finite.

Usage

```
posterior_log_ordinate_attribute(
  value,
  log_ordinate,
  method,
  density_method,
  diagnostics = NULL,
  ...
)
```

Arguments

`value`	numeric null-hypothesis value or values.
`log_ordinate`	numeric natural logarithm of the posterior ordinate height or heights, one per `value`.
`method`	estimator label stored in the attribute.
`density_method`	public density method label stored in the attribute.
`diagnostics`	optional estimator diagnostics.
`...`	additional named metadata fields, for example `parameter`, `conditional`, or `conditional_rule`.

Details

The attribute stores the fields `status` (`"ok"`), `value`, `ordinate` (equal to `exp(log_ordinate)`, so it can underflow to `0` while `log_ordinate` stays finite), `log_ordinate`, `method`, `density_method`, `diagnostics`, and the additional metadata. Consumers that evaluate a Bayes factor from a stored ordinate (e.g., `Savage_Dickey_BF()`) use the log height when it is present, and `posterior_transform()` maps it with the Jacobian of the transformation.

The values are unique and finite and there is one log height per value. Additional metadata must be named with unique names and cannot replace the fields `status`, `value`, `ordinate`, `log_ordinate`, `method`, `density_method`, `diagnostics`, `x`, `y`, `null_hypothesis`, `height`, `posterior_height`, `ordinates`, `posterior_ordinate`, `posterior_ordinates`, and `estimator`.

Value

`posterior_log_ordinate_attribute()` returns a list of class `c("BayesTools_posterior_log_ordinate", "BayesTools_posterior_ordinate","list")` with the fields listed in the details, suitable for the `posterior_ordinate` metadata of posterior draws (`posterior_metadata()`).

See Also

`posterior_ordinate_attribute()` `posterior_metadata()`

Examples

```
# a height of exp(-1000) underflows to zero on the natural scale
attribute <- posterior_log_ordinate_attribute(
  value          = 0,
  log_ordinate   = -1000,
  method         = "example",
  density_method = "precomputed"
)
attribute$ordinate
attribute$log_ordinate
```

`posterior_measure_check` *Check that a measure of posterior draws is available*

Description

Stops with the declared typed refusal when the metadata of posterior draws mark a measure of a column as unavailable. The refusals are declared in the `measure_unavailable` field of the draws (`posterior_metadata()`), for example for the prior density of a quantity whose prior law cannot be certified. Calling it before the `prior_density`, `atoms`, or `support` metadata are read separates a refused measure from a measure that was never declared.

Usage

```
posterior_measure_check(x, measure, column = NULL)
```

Arguments

`x`	posterior draws with metadata, such as the results of `as_mixed_posteriors()`, `mix_posteriors()`, or `marginal_posterior()`.
`measure`	the measure that is about to be read: `"prior_density"`, `"atoms"`, or `"support"`.
`column`	`NULL`, or a character vector of the columns of `x` to check. `NULL` checks the column names of matrix draws, otherwise the column of the single declared quantity or the `parameter` attribute.

Details

Draws without a `measure_unavailable` declaration, and declarations of other measures or other columns, pass. When several entries match, the first one in the declaration table is raised. Stale or invalid draw metadata are refused by the metadata reader.

The refusal is an error condition with a `NULL` call, the `message`, and the fields `target` (the column), `reason` (the declared `cause` when the entry has one, otherwise the reason text), `detail` (the reason text of the entry), and `diagnostics` (the compact model probability diagnostics of the entry, or `NULL`). Its classes depend on the measure and end with `"error"` and `"condition"`:

- `"prior_density"`: `BayesTools_formula_prior_density_unavailable`, `BayesTools_formula_measure_unavail` (only when `reason` is an enumerated measure cause of the `measure_unavailable` metadata), and `BayesTools_formula_transform_unavailable`.
- `"atoms"`: `BayesTools_formula_atoms_unavailable` and, under the same condition, `BayesTools_formula_measure`
- `"support"`: `BayesTools_formula_support_unavailable` and, under the same condition, `BayesTools_formula_measure_unavailable`.

Callers should match these classes and the field `reason` rather than the message.

Value

`posterior_measure_check()` returns `invisible(TRUE)` when `posterior_metadata(x, "measure_unavailable")` has no entry for `measure` in `column`, and otherwise raises the refusal described in the details.

See Also

`posterior_metadata()`

Examples

```
draws <- c(0.1, 0.2, 0.3)
attr(draws, "parameter") <- "theta"
posterior_metadata(draws, "measure_unavailable") <- data.frame(
  column  = "theta",
  measure = "atoms",
  reason  = "the atoms are not declared"
)

posterior_measure_check(draws, "support")
tryCatch(
  posterior_measure_check(draws, "atoms"),
  error = function(e) class(e)
)
```

posterior_metadata *Metadata of BayesTools posterior draws*

Description

Reads and sets the metadata that BayesTools attaches to posterior draws: the elements and lists returned by mix_posteriors(), as_mixed_posteriors(), marginal_posterior(), and random_effects_summary_poste and the draws of parameter_draws(). All metadata are stored in one attribute and validated when they are set; use this accessor instead of setting attributes.

Usage

```
posterior_metadata(x, field)

posterior_metadata(x, field) <- value
```

Arguments

x	posterior draws (an element or a list of them).
field	metadata field:

"support" exact support, created with posterior_support_attribute() (or a list of such objects keyed by column).

"atoms" declared posterior point masses, created with posterior_atom_attribute(); point masses are read only from this field.

"undefined_draws" a character vector naming quantities whose draws may be NA because the quantity is undefined in those draws (e.g. "correlation"), as returned by parameter_draws().

"prior_density" the prior density of the quantity.

"prior_densities" a list of prior densities keyed by parameter, attached to the list returned by as_mixed_posteriors() with transform_scaled = TRUE for the requested parameters.

"prior_context" the joint prior-density context of the draws.

"posterior_density", "posterior_densities" precomputed posterior densities created with posterior_density_attribute().

"posterior_ordinate", "posterior_ordinates" precomputed posterior ordinates created with posterior_ordinate_attribute() or posterior_ordinate_append().

"condition" the conditioning of the draws: a list with conditional, conditional_rule, condition_key, condition_event, resolved_condition_event, averaged (TRUE when the draws are not conditioned on any event, i.e., the unconditional model-averaged posterior; FALSE otherwise), and, for levels whose conditioning was resolved per level, effective_conditional and effective_conditional_rule. Read averaged instead of comparing condition_key with a literal key.

"measure_unavailable" a plain data frame with exact columns column, measure, and reason, with an optional declared cause enum; absent causes use an already enumerated reason. An optional diagnostics list column retains

compact model probability diagnostics. Each non-NULL cell is a plain list with `model_indices`, `log_prior_probabilities`, `log_posterior_probabilities`, and `stage`; supplied log vectors align with positive unique integer model indices. A missing log vector is NULL. Stage identifies prior, posterior, conditional prior/posterior, event, or component probabilities. A cell may alternatively contain the existing typed `BayesTools_ordered_expression_unavailable` condition, with a NULL call, retaining that ordered-expression refusal. Other condition objects are not accepted as diagnostics. The cause `numerical_model_probability_unava` identifies a declared law whose active model probabilities cannot be represented at full precision. The cause `missing_source_context` identifies a prior density that a transformation could not rebuild because the source has neither a declared prior context nor priors (formula marginals of several models keep none); the transformed values and other metadata remain. The cause `random_sd_prior_law_unavailable` identifies the prior density of a random-effect SD on the original predictor scale that combines several fitted SDs of its block (e.g., an intercept SD with a non-centered scaled slope), and `random_effect_design_unavailable` one whose block has no fitted design in the prior context; the SD of a block that rescales one fitted SD has the fitted law rescaled. The cause `expression_prior_law_unavailable` identifies the prior density of a quantity whose prior has parameters given as `expression()`s (an expression point or an expression container, including the components of a mixture or spike-and-slab prior): such priors are evaluated only by JAGS and have no R-side prior law. The draws are not affected, and the atoms and the support of a quantity keep their declarations, except on the routes that build them from the contributing priors (formula marginals, `transform_scaled`), where the atoms and the support carry their own `unsupported_contribution_measure` marks. Invalid metadata requires recomputation/refitting. Human reason text alone does not classify a measure refusal. Tables use unique column/measure pairs. The measures are `prior_density`, `atoms`, or `support`. Entries prevent stale prior/density/support fallback while numeric draws remain usable. Atom/support refusals use `BayesTools_formula_atoms_unavailable` and `BayesTools_formula_support_unavailable`, inheriting `BayesTools_formula_measure_unavai`

`"linear_weight_space"` `"coefficient"` for coefficient recipes, or `"formula_contribution"` for fitted design rows and compiled multipliers. Contribution weights never pass through coefficient C/A again. Old ambiguous stored contexts must be recreated.

`"linear_weights"` the weights of the fitted coordinates that form a level of a `marginal_posterior()` (a named finite numeric vector, or a finite matrix: one row per draw for coefficient weights, one row per fitted design row for the formula contributions of a formula marginal, where the rows are not draws). Removing weights also removes their companion weight-space declaration. Unavailable dynamic recipes remain in the producer's separate transformation/refusal metadata.

`"ordered_source"` internal retained fitted totals, normalized allocations, model provenance and source-row indices. Producers validate its row count and preserve the primitive values independently of effect transformations. Recreate missing sources from the fitted models; shares are never reconstructed

from increments divided by totals.

`"quantities"` the quantity of every column of mixed and marginal posterior draws: a data frame with one row per column (one row for vector draws) and the columns `column` (the column name), `quantity_id` (the `parameter_catalog()` quantity id, or `""` for columns that are not catalog quantities), `dependencies` and `weights` (the fitted coordinates the column is a linear function of), and `label_parts` (the label parts rendered by `parameter_labels()`). Set by `as_mixed_posteriors()`, `mix_posteriors()`, `transform_factor_samples()`, and `marginal_posterior()` (whose estimated marginal means are predictions and declare no fitted coordinates); summaries map columns to fitted coordinates and render their labels from it.

`"output_transformations"` the transformations applied to the values of the draws by `posterior_transform()` (and `marginal_posterior()` with `transformation`), in the order they were applied: `"lin"`, `"exp_lin"`, `"tanh"`, `"exp"`, or `"custom"` for a transformation given as functions; absent for untransformed draws.

`value` the new value of the field; `NULL` removes it.

Details

Arithmetic and mathematical functions of posterior draws (`Ops` and `Math` group generics), `c()`, `as.numeric()`, and subsetting return plain numeric draws without metadata, because supports, atoms, and prior densities do not follow the transformed values. Functions that need the metadata (e.g. `Savage_Dickey_BF()`, `plot_posterior()`, `marginal_posterior()`) stop on such draws; `posterior_transform()` (and `marginal_posterior()` with `transformation`) transforms the draws together with their metadata.

Operations that replace values but keep the attributes of the draws (`x[] <-`, `x[i] <-`, `pmin()`, `pmax()`) leave metadata that no longer describe the draws. The metadata record a fingerprint of the values they describe (their number, the number of missing values, and two sums of the observed values), and reading or setting the metadata of such draws, here or in any function that uses them, stops with an error of class `BayesTools_stale_metadata` (parent class `BayesTools_metadata`). Subsetting a list of mixed posteriors with `[` keeps the list's metadata, with the prior densities of the omitted parameters removed.

The fingerprint cannot detect every change: the two sums are compared within their rounding-error bound, so changes that keep both sums (e.g., adding d, -2d, and d to three consecutive draws), changes of single draws smaller than about $5.6 \cdot 10^{-17} n^2$ times the draws' mean absolute value for n draws (about $6 \cdot 10^{-5}$ for 10^6 draws), and reorderings that move draws by only a few positions can keep the metadata. The bound grows with the number of draws: such changes are detected at some positions or for fewer draws (e.g., swapping two adjacent standard-normal draws is typically detected for 10^5 draws but not for 10^6).

Value

`posterior_metadata()` returns the value of the field or `NULL`; the replacement form returns `x` with the field set.

Examples

```
draws <- stats::rnorm(100)
posterior_metadata(draws, "atoms") <- posterior_atom_attribute()
posterior_metadata(draws, "atoms")
```

`posterior_support_attribute` *Posterior support metadata*

Description

Constructs exact support metadata of a scalar posterior distribution, for the `support` metadata of posterior samples (`posterior_metadata()`) and the `support` of `posterior_density_attribute()`.

Usage

```
posterior_support_attribute(
  bounds,
  points = NULL,
  type = NULL,
  exact = TRUE,
  source = NULL
)
```

Arguments

`bounds`	numeric vector `c(lower, upper)` with `lower <= upper`; bounds may be infinite.
`points`	optional finite numeric vector of point-support locations.
`type`	support type: `"interval"`, `"points"`, or `"mixed"`. If `NULL`, `"points"` is used when the finite `bounds` are exactly (not up to a tolerance) the range of `points` and `"interval"` otherwise.
`exact`	whether `bounds` and `points` describe the true support. Set `exact = FALSE` for plotting or integration limits; non-exact support is not used to exclude a null hypothesis value or to reflect a kernel density estimate.
`source`	optional label of the source of the support metadata.

Details

Posterior support metadata must be created with this constructor or by a BayesTools posterior producer; other objects stored as support metadata are rejected with an error. An unclassed list of such objects, named by column, can hold the support of several columns.

Value

A `BayesTools_posterior_support` object.

See Also

posterior_density_attribute()

Examples

```
posterior_support_attribute(c(0, Inf))
posterior_support_attribute(c(0, 1), points = c(0, 1), type = "points")
```

---

posterior_transform *Transform posterior draws with their metadata*

Description

Applies a strictly monotone transformation to BayesTools posterior draws (an element returned by marginal_posterior(), parameter_mixed_posterior(), as_mixed_posteriors(), or mix_posteriors(), or a list of them) and transforms the draw metadata (posterior_metadata()) with the values, so that inference, plots, and summaries of the transformed draws remain valid. Arithmetic on draws returns plain numeric draws without metadata; this function is the route for transformed posterior distributions.

Usage

```
posterior_transform(x, transformation, transformation_arguments = NULL)
```

Arguments

x	BayesTools posterior draws with metadata, or a list of them.
transformation	a transformation in the form accepted by marginal_posterior(): "lin" (a + b * x), "exp_lin" (exp(a + b * log(x))), "exp", "tanh", or a list of functions fun, inv, and jac (the transformation, its inverse, and its derivative), optionally with output_support.
transformation_arguments	optional named list of arguments of the transformation (a and b of "lin" and "exp_lin").

Details

The transformation must be strictly monotone and invertible. "lin" and "exp_lin" with b = 0 are constant, and a transformation given as functions must have a finite, nonzero derivative of one sign at every draw and atom; otherwise the function stops with an error of class BayesTools_nonmonotone_transformation. Draws that the transformation maps to non-finite values (for example, negative draws of "exp_lin") stop with class BayesTools_transformation_domain. Both classes have the parent class BayesTools_transformation. Finite builtin images that round to mathematical limits (exponential zero, tanh endpoints, or positive-power zero from a strictly positive source) stop with BayesTools_transformation_image_unavailable, also inheriting BayesTools_transformation, error and condition. Its call is NULL; transformation

names the builtin or "custom", and source_values and images contain matched failing finite entries. Declared point and support mapping uses the same condition for nonrepresentable images. True source zero under a positive power and infinite mathematical support limits remain valid. No clipping or probability-mass repair is performed.

The metadata are transformed as follows: A missing canonical prior law is first built from the complete declared source context and recipe, preserving its conditioning and coefficient or formula-contribution space. It is then transformed once. An attached law on the current output scale takes precedence over the original source context in plots. Known unsupported or numerical prior-law routes retain usable posterior draws and declare the prior density unavailable; requesting a prior overlay raises BayesTools_formula_prior_density_unavailable. Transformed publication-bias scalar prior overlays require a supported current-scale law and otherwise have the same explicit limitation.

support the bounds and points are mapped, and the bounds of decreasing transformations swapped. The support of a transformation given as functions is unknown and removed.

atoms the locations are mapped; the masses are kept.

prior_density, prior_densities change of variables through the transformation machinery of the prior densities: a density built from a prior-density context or a linear combination of priors is rebuilt with the transformation as its output transformation (the density marginal_posterior() returns with transformation), a density that already has an output transformation or is an allocation product is transformed on top of its recorded provenance, and a density grid without provenance stays a grid for plotting only.

posterior_density, posterior_ordinate the locations are mapped and the heights divided by the absolute derivative of the transformation; a density or ordinate whose transformed heights are not finite and positive (e.g. at a point where the derivative vanishes) is removed. Diagnostics are kept; their null-value entries (bf_value, value, null_hypothesis) are mapped.

components the per-component supports are transformed.

linear_weights "lin" scales the weights by b and adds a to the offset (linear_offset); any other transformation removes them and records its name in the joint prior transformation.

quantities the name of the transformation ("custom" for functions) is appended to the transformation of the label parts; the columns are no longer the catalog quantities (the quantity id is "") and declare fitted coordinates only for "lin" with a = 0, whose weights are scaled by b. The rendered labels are unchanged.

The conditioning (condition), undefined draws, and the other metadata are kept, the applied transformation is appended to the output_transformations metadata (whether or not the draws have a quantities table), and the metadata record the fingerprint of the transformed values. The prior_list attribute of mixed posteriors describes their untransformed values: marginal_posterior() refuses transformed mixed posteriors whose prior it would build from it; use its transformation argument instead.

Value

x with transformed values and metadata.

See Also

marginal_posterior(), posterior_metadata()

Examples

```
draws <- structure(stats::rnorm(100), class = c("marginal_posterior.simple", "marginal_posterior"))
posterior_metadata(draws, "support") <- posterior_support_attribute(c(-Inf, Inf))
posterior_metadata(draws, "atoms") <- posterior_atom_attribute()
transformed <- posterior_transform(draws, "exp")
posterior_metadata(transformed, "support")
```

---

print.BayesTools_table

Print a BayesTools table

Description

Print a BayesTools table

Usage

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

Arguments

x	a BayesTools_values_tables
...	additional arguments.

Details

Finite positive Bayes-factor bounds that would round to zero at three decimals are printed as reciprocals (for example `<1/10000`), or as `exp()` of their logarithm when the reciprocal overflows. Log-scale bounds that would round to zero keep three significant digits, whichever sign they have (for example `>-1e-05`). This changes only their presentation; numeric values, BF direction and bound operators are retained. A validated finite-log endpoint whose natural value in the displayed orientation is not representable (`0` or `Inf`) is printed as `exp(log_endpoint)`.

Value

`print.BayesTools_table` returns `NULL`.

print.prior *Prints a prior object*

Description

Prints a prior object

Usage

```
## S3 method for class 'prior'
print(
  x,
  short_name = FALSE,
  parameter_names = FALSE,
  plot = FALSE,
  digits_estimates = 2,
  silent = FALSE,
  ...
)
```

Arguments

x	a prior
short_name	whether prior distribution names should be shorted
parameter_names	whether parameter names should be printed
plot	to return bquote formatted prior name for plotting.
digits_estimates	number of decimals to be displayed for printed parameters.
silent	to silently return the print message.
...	additional arguments

Details

A weightfunction prior, or a publication-bias prior with a weightfunction component, prints the specification of its `selection_model` after the prior name only when that specification differs from the default `selection_model()`. The branches of a bias-prior mixture are numbered by their position.

Value

`print.prior` invisibly returns the print statement.

See Also

`prior()`

Examples

```
# create some prior distributions
p0 <- prior(distribution = "point",  parameters = list(location = 0))
p1 <- prior(distribution = "normal", parameters = list(mean = 0, sd = 1))

# print them
p0
p1

# use short names
print(p1, short_name = TRUE)

# print parameter names
print(p1, parameter_names = TRUE)

# generate bquote plotting syntax
plot(0, main = print(p1, plot = TRUE))
```

`prior` *Creates a prior distribution*

Description

`prior` creates a prior distribution. The prior can be visualized by the `plot` function.

Usage

```
prior(
  distribution,
  parameters,
  truncation = list(lower = -Inf, upper = Inf),
  prior_weights = 1
)

prior_none(prior_weights = 1)
```

Arguments

`distribution`	name of the prior distribution. The possible options are `"point"` for a point density characterized by a `location` parameter. `"normal"` for a normal distribution characterized by a `mean` and `sd` parameters. `"lognormal"` for a lognormal distribution characterized by a `meanlog` and `sdlog` parameters. `"cauchy"` for a Cauchy distribution characterized by a `location` and `scale` parameters. Internally converted into a generalized t-distribution with `df = 1`.

"t" for a generalized t-distribution characterized by a location, scale, and df parameters.

"gamma" for a gamma distribution characterized by either shape and rate, or shape and scale parameters. The latter is internally converted to the shape and rate parametrization

"invgamma" for an inverse-gamma distribution characterized by a shape and scale parameters. The JAGS part uses the BayesTools module distribution dbt_invgamma(shape, scale) on the natural parameter.

"beta" for a beta distribution characterized by an alpha and beta parameters.

"exp" for an exponential distribution characterized by either rate or scale parameter. The later is internally converted to rate.

"uniform" for a uniform distribution defined on a range from a to b

"moment" for a Johnson-Rossell moment prior characterized by exactly one of mode or tau, an optional order that defaults to 1, and an optional location that defaults to 0.

"invmoment" for a Johnson-Rossell inverse-moment prior characterized by exactly one of mode or tau, a df, an optional order that defaults to 1, and an optional location that defaults to 0.

"dirichlet" for a Dirichlet distribution over a simplex, whose finite nonnegative coordinates must have a finite total and roundoff bound and sum to one within that bound, characterized by a concentration vector alpha. Numeric entries must be finite and at least 0.01; this is a supported-input minimum, not a numerical guarantee. Positive expression-valued concentrations retain their existing JAGS-only support: expression containers, including literal elements, are not statically evaluated or validated, and R random generation refuses them. Cumulative weight functions and ordered allocations require numeric concentrations. Accepted sampling can still be unavailable; see prior_functions. Two-bin cumulative JAGS initialization near the minimum can repeatedly refuse unrepresentable interior ratios. Density and plot methods show one beta marginal per simplex coordinate.

`parameters` list of appropriate parameters for a given distribution.

`truncation` list with two elements, lower and upper, that define the lower and upper truncation of the distribution. Defaults to list(lower = -Inf, upper = Inf). The truncation is automatically set to the bounds of the support.

`prior_weights` prior odds associated with a given distribution. The value is passed into the model fitting function, which creates models corresponding to all combinations of prior distributions for each of the model parameters and sets the model priors odds to the product of its prior distributions.

Details

Moment and inverse-moment priors are symmetric nonlocal priors with zero density at location. The mode parameter is the positive distance from location to each symmetric mode; supplied negative values are converted to their absolute value. Positional parameters are interpreted as mode for "moment" and mode, df for "invmoment". Use named parameters for tau, order, location, or the inverse-moment nu alias. Positional input always treats the first parameter as mode. For moment priors, tau = mode^2 / (2 * order). For inverse-moment priors, tau = mode^2 * ((df +

1) / (2 * order))^(1 / order). Aliases "pmom", "pimom", and "inversemoment" are accepted, and inverse-moment df may also be supplied as nu.

Native inverse-gamma, moment and inverse-moment calculations distinguish certified finite-input tails outside logarithmic range from unavailable arithmetic. Direct density and log-density evaluations signal BayesTools_numerical_range_limit; unresolved calculations signal BayesTools_numerical_unavailable. Both inherit BayesTools_numerical_condition and retain the original input indices, operation, family and requested scale. Truncated evaluations warn only for points inside the declared support and truncation. Numerical tail zeros do not declare zero probability mass or point atoms. Nonempty intervals whose two endpoints both lose tail-log range can have unavailable normalization. Finite-required fitting and bridge consumers refuse these range results; natural integration may consume certified negative-infinite log tails only after checking normalization and the required kernel and Jacobian. Certification requires the inspected MinGW-w64 11.x x86-64 log backend, documented Intel or AMD instruction contracts, checked floating-point modes and a checked extended logarithmic constant. Other backends, overflowing original nonlocal distances and uncertified coordinates remain unavailable in this FAR regime. Constructor parameter domains and ordinary arithmetic are unchanged.

Value

prior and prior_none return an object of class 'prior'. A named list containing the distribution name, parameters, and prior weights.

See Also

plot.prior(), Normal, Lognormal, Cauchy, Beta, Exponential, LocationScaleT, InvGamma.

Examples

```
# create a standard normal prior distribution
p1 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1))

# create a half-normal standard normal prior distribution
p2 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1),
truncation = list(lower = 0, upper = Inf))

# the prior distribution can be visualized using the plot function
# (see ?plot.prior for all options)
plot(p1)

# create nonlocal priors centered on zero
p3 <- prior("moment", list(mode = 0.5))
p4 <- prior("invmoment", list(mode = 0.5, df = 3))

# tau can be specified directly when named
p5 <- prior("pmom", list(tau = 0.125))
p6 <- prior("pimom", list(tau = 0.5, nu = 3))
```

prior_bias *Creates a composed publication-bias prior*

Description

`prior_bias()` composes a step-selection `prior_weightfunction()` and/or a `prior_phacking()` object. It does not introduce new math; the backend compiler treats the composition as the product of both kernels.

Usage

```
prior_bias(selection = NULL, phacking = NULL, prior_weights = 1)
```

Arguments

`selection`	optional `prior_weightfunction()` object.
`phacking`	optional `prior_phacking()` object.
`prior_weights`	prior odds associated with a given distribution.

Value

`prior_bias()` returns an object of class `"prior"`.

prior_density_has_provenance *Whether a prior density has deterministic provenance*

Description

`prior_density_has_provenance()` tells whether the prior measure of `x` is available from deterministic provenance, so that its density heights, ordinate values and region probabilities are evaluated by BayesTools (from structural routes, or from numerical grids refined under error control). It is the public signal for densities without such provenance, whose region probabilities a caller has to obtain in another way (e.g. from prior draws).

Usage

```
prior_density_has_provenance(x)
```

Arguments

`x`	A BayesTools prior or `prior_linear_density` object.

Details

A BayesTools prior object is its own provenance. A `prior_linear_density` has provenance when its builder recorded the prior measure it describes (every density built by the BayesTools prior-density builders, e.g. `JAGS_formula_prior_density()`, `parameter_prior_density()` and the prior densities of `marginal_posterior()`), unless that measure contains a `multiply_by` product without a structural density route. It has none when it is a numerical density grid without that record (e.g. the standard-deviation components of nested variance allocations) or such a product: both are used for plotting only, and their heights and region probabilities stop. A density of point masses only is exact without a record.

Provenance does not make an ordinate exact: a combination without a structural route (e.g. three non-normal terms) has provenance and refined grid region probabilities, while its ordinates are `"unknown"`, so point hypotheses need `prior_ordinate_status()`. The `method` field of `prior_density_ordinate()` does not separate the cases: it is `"unsupported_provenance"` for a grid without provenance and for a combination without a structural route alike, and `"named_transform"` for such a combination under an output transformation.

Value

`TRUE` or `FALSE`.

See Also

`prior_density_ordinate()`, `prior_ordinate_status()`

Examples

```
prior_density_has_provenance(prior("normal", list(mean = 0, sd = 1)))
```

`prior_density_ordinate` *Classify a prior-density ordinate*

Description

`prior_density_ordinate()` classifies the mathematical behavior of a prior at one exact value. The classification uses the prior definition and deterministic provenance recorded on linear-combination prior densities. It does not use samples, kernel density estimates, numerical grids, or nearby probe values to establish the behavior. Declared atom producers refuse nonrepresentable finite-source transformation images with `BayesTools_transformation_image_unavailable` (parent `BayesTools_transformation`); fields `transformation`, `source_values` and `images` identify the failing entries, with `call = NULL`. Conservative provenance certification remains unknown when an image cannot be established, without manufacturing a boundary atom.

Usage

```
prior_density_ordinate(x, value)
```

Arguments

x	A BayesTools prior object or a prior_linear_density object produced by BayesTools' deterministic prior-density builders.
value	One finite, non-missing numeric value at which to classify the prior ordinate.

Value

A named list of class prior_density_ordinate with fields:

- schema_version: the stable result-schema version, currently "1";
- value: the requested value;
- behavior: one of "regular", "zero", "infinite", "point_mass", "undefined", or "unknown";
- log_density: the continuous log-density ordinate, when available;
- point_mass: discrete probability at exactly value;
- exact: whether behavior follows from deterministic prior provenance and, for a "regular" ordinate, log_density is available (in closed form or from a quadrature accepted by its diagnostics). A regular ordinate without a value (a quadrature rejected by its diagnostics, or a boundary limit without a structural value) has exact = FALSE, log_density = NA and the failure in reason; so has an ordinate whose evaluation point, or a value computed from value on the way to it, is not representable at full double precision (e.g. a subnormal value or distance from an offset), with behavior = "regular" where the route has established it (inside the support) and "unknown" otherwise (e.g. an inverse affine value that is not representable). exact = TRUE never comes with a missing regular log_density;
- method: a machine-readable classification method;
- reason: NULL for an ordinary regular ordinate and a concise diagnostic otherwise;
- provenance: compact deterministic information used for classification. When behavior is "point_mass" (and only then), it contains continuous_behavior: the behavior of the continuous part of the prior at value, i.e., of the measure without its point masses, as one of "regular", "zero", "infinite", "undefined", or "unknown", with the same meaning as behavior. It is "zero" when the prior has no continuous part there (e.g., a point prior) and "unknown" when the continuous part has no structural classification. log_density is then the log density of that continuous part (not renormalized by the continuous probability), which for an "unknown" continuous part may be a numerical grid value or NA.

A regular classification is structural. Consequently, log_density may be -Inf when an otherwise positive finite density underflows in ordinary floating-point evaluation. General numerical convolutions, products, and arbitrary user transformations are reported as unknown unless exact point mass establishes the requested behavior. Named monotone transformations retain their normalized arguments and declared source support and atoms through composition. Two-term convolutions retain their independent primitive definitions and signed support hull. Missing structural support gives unknown, while a known invalid transformation domain gives undefined. Finite endpoints or atoms whose images round to an exponential or hyperbolic-tangent limit do not certify exact provenance. Constant affine routes declare a point even without a source definition. A stored point-only measure has exact zero continuous density away from its atoms, including when no adaptive provenance is attached. Positive continuous grids and deferred measures without a declared continuous mass remain unknown. For exp_lin, positive powers are defined on nonnegative sources

and map a zero atom to zero with its original mass. The continuous density retains its original mixture weight; atoms have no Jacobian density contribution. Negative powers still refuse zero atoms, and negative source domains remain undefined. Supported conditional-normal mixtures are structurally regular because they include an independent positive-variance Gaussian term. A product term (`multiply_by`) without an additive normal term is a pure scale mixture: away from its deterministic offset it is regular, and at the offset its density is $\phi(b_m/b_s)E[1/|s|]/b_s$ for a multiplied $N(b_m, b_s)$ term and multiplier s, finite exactly when the multiplier's declared density vanishes at zero and infinite otherwise. Closed inverse moments retain their logarithms even when the natural moment overflows. Provenance records availability, its reason and a floating-resolution diagnostic; the diagnostic is not a rigorous special-function error bound. Unsupported mapped-Beta shifts, differences of log normalizers and unresolved offset arithmetic give a structurally regular inexact ordinate with no log density. For example, a standard Normal multiplied by a Lognormal with meanlog zero and sdlog 40 has offset log density $800 - \log(2\pi)/2$, which remains available. Levels of ordered priors are the ordered total times their Dirichlet allocation share and are classified in the same way; a non-normal total or multiplied term uses the analogous `"scale_mixture"` quadrature. A level of an interaction takes the share of its slice (the coordinates of the ordinary factors, whose totals and allocations are independent). A level of several ordered factors, a combination across slices, or a combination of increments that is not one cumulative share is a total (or several independent totals) times a multilinear function of independent Beta coordinates of the allocations: for a single total with a simple continuous prior, or normal totals, the ordinate and region probabilities integrate the closed-form conditional law over these coordinates with a tensor Gauss-Jacobi rule, doubled until two rules agree (method `"scale_mixture"`, provenance `ordered_product`); a total with a finite support bound is integrated with the closed-form breakpoints where the argument of its density crosses a bound (at most three coordinates; a density that is infinite at a bound is smoothed there with its closed-form exponent, and refused when the exponent is unknown or below 0.5). Its offset density is classified for a single total times a product of shares, and a Beta exponent below 0.1 rejects the quadrature. Other combinations (several totals that are not normal, point totals that make the level a nonlinear function of several shares) are `unknown` and refused. Their ordinates use quadrature over the multiplier's declared support, split at its bounds, at quantiles of its declared prior (spaced out next to a bound with infinite density) and around the location peak of the conditional normal density (kept next to such a bound) when the multiplied standard deviation is at most half of the absolute value of its mean (always for a Gaussian convolution), and retain integration errors and evaluation budgets in `provenance`. Every independent quadrature (each piece of a split integral, each distinct design row, model or conditional mixture component, and spike-and-slab or mixture leaf) receives the full evaluation budget of the density (its grid size) and its own convergence check. `provenance` reports the evaluations summed over all of them; its `budget` is the grid size per split integral (the budget of each of its pieces), summed over rows, components and leaves. Split integrals also keep each piece's evaluations and absolute error, and row mixtures each row's diagnostics. An accepted quadrature has `exact = TRUE` (the structural classification), while `provenance$integration$exact = FALSE` describes the numerical ordinate. A quadrature rejected by its diagnostics keeps the behavior `"regular"` with `exact = FALSE`, `log_density = NA`, the rejection in `reason` and its diagnostics (`converged = FALSE`) in `provenance`; requesting a density height then rejects the result.

Mixture ordinates (model-averaged and conditional mixtures, and linear combinations of mixture or spike-and-slab priors) are the probability- weighted sums of their component ordinates, each classified by its own method; a component with a truncated prior at its support bound uses the one-sided limit inside the support. The continuous behavior of the mixture follows the precedence `"undefined"` over `"infinite"` over `"unknown"`: one undefined component makes the mixture un-

defined, else one infinite component makes it infinite (even beside an unknown one); otherwise, if any component remains unknown, the whole mixture is unknown, and else it is "regular" (some component regular) or "zero" (all zero). In any linear combination (inside or outside a mixture), a Gaussian term plus one other continuous scalar term is a Gaussian convolution evaluated by the same quadrature over that term's declared support. When that term is a truncated normal, the density is the closed form (method "truncated_normal_convolution") $\phi(x; m, s)[\Phi(\beta(x)) - \Phi(\alpha(x))]/[\Phi(B) - \Phi(A)]$: m and s^2 are the sums of the means and variances of the untruncated terms, A and B the standardized truncation bounds, and $\alpha(x)$ and $\beta(x)$ the truncation bounds standardized by the normal distribution of the truncated term given the sum x. The exponential of a log-transformed positive term X plus a Gaussian part G (e.g. the unscaled intercept of a log(intercept) formula scaling) is the scale product of X and the lognormal $\exp(G)$ (method "scale_mixture"), and the log-scale sum $\log(X) + G$ is the log image of that product, with density $f(e^z)e^z$ (method "scale_mixture", provenance kind "log_scale_product"). A linear combination $a'X$ of a multivariate t vector prior ("mt", and "mcauchy" with one degree of freedom; e.g. levels and contrasts of mean-difference and orthonormal factor priors) with location μ, scale s (the scale matrix $s^2 I$) and ν degrees of freedom is the univariate t with location $\mu \sum_j a_j$, scale $s\|a\|$ and ν degrees of freedom; it enters the combination as that scalar term, recorded in provenance$multivariate_t (alone it is exact with method "scalar_affine"). Sums of Cauchy terms are Cauchy (method "scalar_affine"), and two other continuous scalar terms without a Gaussian part are a "convolution" quadrature over one term's declared support; where finite support bounds of both terms meet at value, the ordinate is classified from the terms' declared behavior at those bounds (the positive finite limit at an end of the support where the density exponents sum to one, e.g. two arcsine terms, is unknown). Sums of three or more non-normal terms, or of a Gaussian part and two or more non-normal terms, are unknown.

Examples

```
normal_prior <- prior("normal", list(mean = 0, sd = 1))
prior_density_ordinate(normal_prior, 0)

gamma_prior <- prior("gamma", list(shape = 0.5, rate = 1))
prior_density_ordinate(gamma_prior, 0)

mixture_prior <- prior_mixture(
  list(
    prior("point", list(location = 0), prior_weights = 1),
    prior("normal", list(mean = 0, sd = 1), prior_weights = 3)
  ),
  is_null = c(TRUE, FALSE)
)
prior_density_ordinate(mixture_prior, 0)
```

---

prior_factor *Creates a prior distribution for factors*

Description

`prior_factor` creates a prior distribution for fitting models with factor predictors. (Note that results across different operating systems might vary due to differences in JAGS numerical precision.)

Usage

```
prior_factor(
  distribution,
  parameters,
  truncation = list(lower = -Inf, upper = Inf),
  prior_weights = 1,
  contrast = "meandif"
)
```

Arguments

`distribution` name of the prior distribution. The possible options are

- `"point"` for a point density characterized by a `location` parameter.
- `"normal"` for a normal distribution characterized by a `mean` and `sd` parameters.
- `"lognormal"` for a lognormal distribution characterized by a `meanlog` and `sdlog` parameters.
- `"cauchy"` for a Cauchy distribution characterized by a `location` and `scale` parameters. Internally converted into a generalized t-distribution with `df = 1`.
- `"t"` for a generalized t-distribution characterized by a `location`, `scale`, and `df` parameters.
- `"gamma"` for a gamma distribution characterized by either `shape` and `rate`, or `shape` and `scale` parameters. The latter is internally converted to the `shape` and `rate` parametrization
- `"invgamma"` for an inverse-gamma distribution characterized by a `shape` and `scale` parameters. The JAGS part uses the BayesTools module distribution `dbt_invgamma(shape, scale)` on the natural parameter.
- `"beta"` for a beta distribution characterized by an `alpha` and `beta` parameters.
- `"exp"` for an exponential distribution characterized by either `rate` or `scale` parameter. The later is internally converted to `rate`.
- `"uniform"` for a uniform distribution defined on a range from `a` to `b`
- `"moment"` for a Johnson-Rossell moment prior characterized by exactly one of `mode` or `tau`, an optional `order` that defaults to 1, and an optional `location` that defaults to 0.
- `"invmoment"` for a Johnson-Rossell inverse-moment prior characterized by exactly one of `mode` or `tau`, a `df`, an optional `order` that defaults to 1, and an optional `location` that defaults to 0.
- `"dirichlet"` for a Dirichlet distribution over a simplex, whose finite nonnegative coordinates must have a finite total and roundoff bound and sum to one within that bound, characterized by a concentration vector `alpha`. Numeric entries must be finite and at least `0.01`; this is a supported-input minimum,

not a numerical guarantee. Positive expression-valued concentrations retain their existing JAGS-only support: expression containers, including literal elements, are not statically evaluated or validated, and R random generation refuses them. Cumulative weight functions and ordered allocations require numeric concentrations. Accepted sampling can still be unavailable; see prior_functions. Two-bin cumulative JAGS initialization near the minimum can repeatedly refuse unrepresentable interior ratios. Density and plot methods show one beta marginal per simplex coordinate.

`parameters` list of appropriate parameters for a given `distribution`.

`truncation` list with two elements, `lower` and `upper`, that define the lower and upper truncation of the distribution. Defaults to `list(lower = -Inf, upper = Inf)`. The truncation is automatically set to the bounds of the support.

`prior_weights` prior odds associated with a given distribution. The value is passed into the model fitting function, which creates models corresponding to all combinations of prior distributions for each of the model parameters and sets the model priors odds to the product of its prior distributions.

`contrast` type of contrast for the prior distribution. The possible options are

`"meandif"` for contrast centered around the grand mean with equal marginal distributions, making the prior distribution exchangeable across factor levels. In contrast to `"orthonormal"`, the marginal distributions are identical regardless of the number of factor levels and the specified prior distribution corresponds to the difference from grand mean for each factor level. Only supports `distribution = "mnormal"` and `distribution = "mt"` which generates the corresponding multivariate normal/t distributions. The distribution must be centered exactly at zero.

`"orthonormal"` for contrast centered around the grand mean with equal marginal distributions, making the prior distribution exchangeable across factor levels. Only supports `distribution = "mnormal"` and `distribution = "mt"` which generates the corresponding multivariate normal/t distributions. The distribution must be centered exactly at zero.

`"treatment"` for contrasts using the first level as a comparison group and setting equal prior distribution on differences between the individual factor levels and the comparison level.

`"independent"` for one independently regularized coefficient per factor level. With a free intercept, this coefficient design is likelihood-rank-deficient and its separation is prior-dependent. In a BayesTools formula with `0 + factor` or `factor - 1`, the intercept is instead fixed structurally at zero, so the level coefficients are identified directly.

Details

The factor prior owns the contrast family. BayesTools does not change that family merely because the formula includes or removes an intercept. In particular, `~ 0 + factor` represents a point-zero intercept and the contrast selected here; it requests one coefficient per level only when `contrast = "independent"`.

A formula interaction of two or more factors in which a factor is coded by a mean-difference or orthonormal contrast accepts only priors that do not depend on the basis of that contrast: mean-

difference and orthonormal priors, and treatment or independent priors that are zero-mean untruncated normal or zero spike priors (priors symmetric about zero when every coded factor of the term has two levels). Other treatment and independent priors, and ordered priors, are refused (see the conditions of JAGS_formula()).

Value

return an object of class 'prior'.

See Also

prior(), JAGS_formula()

Examples

```
# create an orthonormal prior distribution
p1 <- prior_factor(distribution = "mnormal", contrast = "orthonormal",
                   parameters = list(mean = 0, sd = 1))
```

---

prior_factor_levels *Levels of a factor prior distribution*

Description

Sets the factor levels of a factor prior distribution used outside a formula, e.g., in the prior_list of JAGS_fit(). The prior receives the complete factor metadata that the priors of formula factor terms carry: the number and names of the levels, the contrast, and the design that maps its coefficients to the levels. JAGS_fit() requires this metadata of every factor prior outside a formula and stops without it.

Usage

```
prior_factor_levels(prior, levels)
```

Arguments

prior	a factor prior distribution (prior_factor() or prior_ordered()), or a mixture or spike-and-slab prior of factor prior distributions.
levels	the number of factor levels, or a character vector with the level names.

Details

The contrast of the prior determines the number of coefficients: one per level for `contrast = "independent"` and ordered priors with `contrast = "cumulative_levels"`, and one per level after the first otherwise. Levels given by their number are named `"1"`, `"2"`, and so on. The factor is named after the parameter under which the prior is fitted. Bound mixture and spike-and-slab components use the declared design's raw coefficient and cell dimensions. Scalar component families draw independent raw coordinates; multivariate families retain their joint law, including a shared scale for multivariate t draws. Each row shares one component or inclusion gate across its coordinates. Standalone sampling and initialization retain their existing contracts.

Value

The prior distribution with its factor levels.

See Also

`prior_factor()` `prior_ordered()` `JAGS_fit()`

Examples

```
p <- prior_factor("mnormal", list(mean = 0, sd = 1), contrast = "meandif")
p <- prior_factor_levels(p, c("low", "medium", "high"))
```

`prior_functions` *Elementary prior related functions*

Description

Density (pdf / lpdf), distribution function (cdf / ccdf), quantile function (quant), random generation (rng), mean, standard deviation (sd), and marginal variants of the functions (mpdf, mlpf, mcdf, mccdf, mquant) for prior distributions. Joint quantiles are unavailable for vector factor priors; use `mquant()` for their marginal quantiles. The marginal methods and `var()`/`sd()` of a bound orthonormal or mean-difference `mnormal`/`mt` factor prior describe its continuous levels: they take the scale of each level from the declared factor design, a level whose design row vanishes identically (the reference level of a treatment factor inside an interaction) is the atom at zero that the cell declares and does not enter the common scale, and levels whose non-zero rows differ in norm have no common marginal and refuse. Missing evaluation values `NA_real_` and `NaN` propagate distinctly through density, distribution and quantile methods for moment, inverse-moment and inverse-gamma scalar priors, including their marginal methods. Generic scalar CDF and complementary CDF methods also retain this distinction under truncation. This statement does not cover truncated Normal quantiles. Continuous truncation uses logarithmic interval masses, including separate signed radial intervals for nonlocal priors. An unresolved positive normalizer stops with `BayesTools_numerical_unavailable` before a cached density evaluator or bridge calculation is constructed. Existing Normal-only rounding adjustments are retained; other truncated quantiles never clip an unavailable result into their bounds. Ordinary Gamma normalization is unavailable for a numeric subnormal shape when truncation cuts its positive support: finite base R log tails

can be wrong in this floating-point regime. Constructors retain their supported parameter domain and whole-positive-support mass remains exactly one. Raw untruncated base R Gamma extreme calculations are not certified by this normalization guard.

Valid interior numerical failures give `NaN` with one `BayesTools_numerical_unavailable` warning per vector. Established inverse-gamma or nonlocal natural density or quantile range limits retain their raw zero or infinite result with `BayesTools_numerical_range_limit`. Ordinary family density range results retain their existing behavior when their log density is available. Both numerical classes inherit `BayesTools_numerical_condition`, `warning`, and `condition`. Conditions record `operation`, `family`, `requested_scale`, `indices`, and `reason`, with `call = NULL`. Exact support endpoints and rounded natural probabilities with available log tails do not warn. Sampling failures are errors additionally inheriting `BayesTools_prior_rng_unavailable`; all original uniforms are consumed before signaling. Finite-required consumers and JAGS refuse unavailable draws, density values, and normalization, without redraw or model repair.

Usage

```
## S3 method for class 'prior'
rng(x, n, ...)

## S3 method for class 'prior'
cdf(x, q, ...)

## S3 method for class 'prior'
ccdf(x, q, ...)

## S3 method for class 'prior'
lpdf(x, y, ...)

## S3 method for class 'prior'
pdf(x, y, ...)

## S3 method for class 'prior'
quant(x, p, ...)

## S3 method for class 'prior'
mcdf(x, q, ...)

## S3 method for class 'prior'
mccdf(x, q, ...)

## S3 method for class 'prior'
mlpdf(x, y, ...)

## S3 method for class 'prior'
mpdf(x, y, ...)

## S3 method for class 'prior'
mquant(x, p, ...)
```

Arguments

x	prior distribution
n	number of observations
...	additional arguments of rng(); the other functions ignore them. transform_factor_samples (default TRUE): whether the samples of orthonormal, mean-difference and ordered factor priors are transformed to the level scale (FALSE: the raw coefficients); sample_components (default FALSE): whether mixture and spike-and-slab priors return the sampled component (or inclusion indicator) instead of the values; quantity (ordered priors only, default "level"): the draws returned, one of "level" (the transformed level samples), "coefficient" and "increment" (the raw coefficients), "total" (the totals of the ordered sums) or "allocation" (the allocation shares).
q	vector or matrix of quantiles
y	vector of observations
p	vector of probabilities

Details

Numeric Dirichlet concentrations must be finite and at least 0.01. This supported-input minimum does not guarantee representable draws for every accepted specification. Dirichlet and cumulative weight-function rng() calls stop with class BayesTools_dirichlet_rng_unavailable, parent class BayesTools_prior_rng_unavailable, and classes error and condition if sampling produces non-finite entries, entries outside [0, 1], or a simplex row with zero total. The condition has call = NULL and message: "Dirichlet draws are unavailable because sampling produced non-finite or degenerate simplex draws. Use a less extreme 'alpha' specification." There is no retry or repair. Individual rounded zero or one entries are retained and do not represent new prior atoms. Ordinary expression-valued Dirichlet priors retain their JAGS-only support; R random generation refuses priors containing parameter expressions.

Value

pdf (mpdf) and lpdf (mlpdf) give the (marginal) density and the log of (marginal) density, cdf (mcdf) and ccdf (mccdf) give the (marginal) distribution and the complement of (marginal) distribution function, quant (mquant) give the (marginal) quantile function, and rng generates random deviates for an object of class 'prior'.

Examples

```
# create a standard normal prior distribution
p1 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1))

# generate a random sample from the prior
rng(p1, 10)

# compute cumulative density function
cdf(p1, 0)

# obtain quantile
```

```
quant(p1, .5)

# compute probability density
pdf(p1, c(0, 1, 2))
```

`prior_functions_methods` *Creates generics for common statistical functions*

Description

Density (pdf / lpdf), distribution function (cdf / ccdf), quantile function (quant), random generation (rng), and marginal variants of the functions (mpdf, mlpdf, mcdf, mccdf, mquant).

Usage

```
rng(x, ...)

cdf(x, ...)

ccdf(x, ...)

quant(x, ...)

lpdf(x, ...)

pdf(x, ...)

mcdf(x, ...)

mccdf(x, ...)

mquant(x, ...)

mlpdf(x, ...)

mpdf(x, ...)
```

Arguments

`x`	main argument
`...`	further arguments passed to the method; the methods for prior objects and their arguments are documented in `prior_functions`

Value

pdf (mpdf) and lpdf (mlpdf) give the (marginal) density and the log of (marginal) density, cdf (mcdf) and ccdf (mccdf) give the (marginal) distribution and the complement of (marginal) distribution function, quant (mquant) give the (marginal) quantile function, and rng generates random deviates for an object of class 'prior'.

The pdf function proceeds to PDF graphics device if x is a character.

`prior_informed` *Creates an informed prior distribution based on research*

Description

prior_informed creates an informed prior distribution based on past research. The prior can be visualized by the plot function.

Usage

```
prior_informed(name, parameter = NULL, type = "smd")
```

Arguments

name	name of the prior distribution. There are many options based on prior psychological or medical research. For psychology, the possible options are "van Erp" for an informed prior distribution for the heterogeneity parameter tau of meta-analytic effect size estimates based on standardized mean differences (van Erp et al. 2017), "Oosterwijk" for an informed prior distribution for the effect sizes expected in social psychology based on prior elicitation with dr. Oosterwijk (Gronau et al. 2017). For medicine, the possible options are based on Bartoš et al. (2021) and Bartoš et al. (2023) who developed empirical prior distributions for the effect size and heterogeneity parameters of the continuous outcomes (standardized mean differences), dichotomous outcomes (logOR, logRR, and risk differences), and time to event outcomes (logHR) based on the Cochrane database of systematic reviews. Use "Cochrane" for a prior distribution based on the whole database or call print(prior_informed_medicine_names) to inspect the names of available subfields and set the appropriate parameter and type; availability of subfield-specific priors depends on the selected type.
parameter	parameter name describing what prior distribution is supposed to be produced in cases where the name corresponds to multiple prior distributions. Relevant only for the empirical medical prior distributions.
type	prior type describing what prior distribution is supposed to be produced in cases where the name and parameter correspond to multiple prior distributions. Relevant only for the empirical medical prior distributions with the following options "smd" for standardized mean differences

"logOR" for log odds ratios
"logRR" for log risk ratios
"RD" for risk differences
"logHR" for log hazard ratios

Value

prior_informed returns an object of class 'prior'.

References

Bartoš F, Gronau QF, Timmers B, Otte WM, Ly A, Wagenmakers E (2021). "Bayesian model-averaged meta-analysis in medicine." *Statistics in Medicine*, **40**(30), 6743–6761. doi:10.1002/sim.9170.

Bartoš F, Otte WM, Gronau QF, Timmers B, Ly A, Wagenmakers E (2023). "Empirical prior distributions for Bayesian meta-analyses of binary and time-to-event outcomes." doi:10.48550/arXiv.2306.11468, preprint at https://doi.org/10.48550/arXiv.2306.11468.

Gronau QF, Van Erp S, Heck DW, Cesario J, Jonas KJ, Wagenmakers E (2017). "A Bayesian model-averaged meta-analysis of the power pose effect with informed and default priors: The case of felt power." *Comprehensive Results in Social Psychology*, **2**(1), 123–138. doi:10.1080/23743603.2017.1326760.

van Erp S, Verhagen J, Grasman RP, Wagenmakers E (2017). "Estimates of between-study heterogeneity for 705 meta-analyses reported in Psychological Bulletin from 1990–2013." *Journal of Open Psychology Data*, **5**(1). doi:10.5334/jopd.33.

See Also

prior(), prior_informed_medicine_names

Examples

```
# prior distribution representing expected effect sizes in social psychology
# based on prior elicitation with dr. Oosterwijk
p1 <- prior_informed("Oosterwijk")

# the prior distribution can be visualized using the plot function
# (see ?plot.prior for all options)
plot(p1)


# empirical prior distribution for the standardized mean differences from the oral health
# medical subfield based on meta-analytic effect size estimates from the
# Cochrane database of systematic reviews
p2 <- prior_informed("Oral Health", parameter ="effect", type ="smd")
print(p2)
```

prior_informed_medicine_names *Names of medical subfields from the Cochrane database of systematic reviews*

Description

Contains the union of names identifying the individual subfields from the Cochrane database of systematic reviews. Type-specific availability depends on the `type` argument passed to `prior_informed()`.

Usage

```
prior_informed_medicine_names
```

Format

An object of class `character` of length 57.

Value

returns a character vector with names of medical subfields from Cochrane database of systematic reviews.

See Also

`prior_informed()`

Examples

```
print(prior_informed_medicine_names)
```

prior_mixture *Creates a mixture of prior distributions*

Description

`prior_mixture` creates a mixture of prior distributions. This is a more generic version of the `prior_spike_and_slab` function.

Usage

```
prior_mixture(
  prior_list,
  is_null = rep(FALSE, length(prior_list)),
  components = NULL
)
```

Arguments

prior_list	a list of prior distributions to be mixed.
is_null	a logical vector indicating which of the prior distributions should be considered as a null distribution. Defaults to rep(FALSE, length(prior_list)).
components	a character vector indicating which of the prior distributions belong to the same mixture component (this is an alternative specification to the is_null argument). Defaults to NULL (i.e., is_null is used.

Value

A prior object of class prior.mixture (with the classes of a factor, ordered or simple mixture where the components have them) that holds the component priors with their prior_weights and components.

See Also

prior()

---

prior_ordered *Creates a prior distribution for ordered factor effects*

Description

prior_ordered creates a factor prior for ordered predictors by separating a scalar total effect from an allocation of that effect across ordered-level increments.

Usage

```
prior_ordered(
  total,
  allocation = NULL,
  contrast = "cumulative",
  id = NULL,
  prior_weights = 1
)
```

Arguments

total	scalar prior for the first-to-last effect with "cumulative", or the last level's effect from the zero baseline with "cumulative_levels".
allocation	allocation prior or fixed split. NULL creates a late-bound flat Dirichlet allocation with dimension determined by the formula term. Numeric vectors specify fixed splits of finite nonnegative weights with a finite total and roundoff bound, summing to one within that bound. Dirichlet priors specify random allocations. For terms with multiple ordered factors, use a named list. A list is available in a JAGS_formula() term only: sampling or plotting a standalone prior, or setting its levels with prior_factor_levels(), raises BayesTools_ordered_unavailable.

contrast	ordered contrast. "cumulative" uses nlevels(f) - 1 increments and sets the first level effect to zero. "cumulative_levels" uses nlevels(f) increments and gives the first level a non-zero allocation share.
id	optional allocation-sharing id. Sharing uses the id and factor together. For terms with multiple ordered factors, use a named character vector; partial named maps may be reused across terms.
prior_weights	prior odds associated with a given distribution. The value is passed into the model fitting function, which creates models corresponding to all combinations of prior distributions for each of the model parameters and sets the model priors odds to the product of its prior distributions.

Details

Spike-and-slab or mixture behavior belongs on total; e.g., prior_ordered(prior_spike_and_slab(...)). Ordered density methods preserve point masses and continuous probability separately for each level; point masses are not included in a kernel density estimate. Bridge sampling is available only when total is a nonexpression, nondiscrete simple scalar prior and allocation is fixed or a simple Dirichlet allocation. Mixtures of bound ordered prior containers (see prior_factor_levels()) can be constructed and sampled, but their binding in JAGS_formula() is unavailable. Put the mixture on total instead. General mixture totals require a single slice; multi-slice interactions support simple scalar, point, and spike-and-slab totals. Unsupported binding raises BayesTools_ordered_unavailable. Factors use their bound levels() order, including character predictors converted to factors by the formula. Without id, each ordinary-factor slice has an independent allocation. With id, one allocation is shared by all slices and terms with the same id and factor. Joint prior sampling uses the first owner's allocation draws; separate standalone sampling calls remain independent.

Estimates tables with transform_factors = TRUE report the level effects (the first level of a "cumulative" contrast is the reference and has no row). With transform_factors = FALSE, they report the sampled parameters: the total effect, labelled f{total}, and, for random allocations, the allocation share of each increment, labelled by the level the increment reaches, f[share: mid] (see runjags_estimates_table(), ensemble_estimates_table(), and parameter_labels() for the labels of terms with several slices or predictors). The total (one per slice) and each share of a random allocation are quantities of the parameter catalog (parameter_catalog()), also under their earlier names (f_ordered_total, f_ordered_allocation[mid]), and hypothesis_BF() tests them (for example "f{total} = 0" or "f[share: mid] > 0.5"): the total has the prior law of total, a share the Beta marginal of its Dirichlet allocation. A spike-and-slab or mixture total needs conditional = TRUE for point tests at its null. Totals and primitive shares remain on the fitted scale when interactions prevent an original-scale factorization. Inclusion conditioning uses the declared total event; a discrete total alone does not create such an event. Literal numeric arguments are emitted at round-trip precision and persisted with the bound recipe. Fits without that provenance require refitting.

Value

return an object of class 'prior'.

See Also

prior() prior_factor()

prior_ordered_total_name

Name of the total node of an ordered prior

Description

Returns the name of the node that holds the total effect of an ordered prior (`prior_ordered()`) for a parameter in the JAGS model: `<parameter>_ordered_total`, without any index. When the total is a mixture (for example a spike and slab), its inclusion indicator is named by appending `_indicator` to this name.

Usage

```
prior_ordered_total_name(parameter_name)
```

Arguments

`parameter_name`	a character vector of parameter names with at least one element: non-missing and not empty.

Value

`prior_ordered_total_name()` returns a character vector with `<parameter_name>_ordered_total` for each element of `parameter_name`.

See Also

`prior_ordered()`

Examples

```
prior_ordered_total_name("mu_f")
```

prior_ordinate_status *Point-hypothesis eligibility of prior ordinates*

Description

`prior_ordinate_status()` applies the exactness rule of point hypotheses (`hypothesis_BF()`, `Savage_Dickey_BF()`) to a prior density at each requested value and reports the outcome instead of stopping. A value is eligible for a Savage-Dickey point hypothesis when its prior ordinate is `"regular"`, classified exactly by `prior_density_ordinate()`, and has an available log density. Otherwise the row names the condition class and the message with which a point hypothesis at that value stops.

Usage

```
prior_ordinate_status(prior_density, values, labels = NULL)
```

Arguments

prior_density	a BayesTools prior or a prior_linear_density, as accepted by prior_density_ordinate().
values	finite numeric values of the point hypotheses (at least one).
labels	optional labels of the point hypotheses used in the messages, one per value; defaults to the values.

Value

A data frame with one row per value and the columns

value the value.

eligible whether a point hypothesis at the value has an exact regular prior ordinate.

condition the class of the error a point hypothesis at the value stops with: "BayesTools_point_mass_at_null", "BayesTools_infinite_ordinate", "BayesTools_zero_ordinate", "BayesTools_undefined_ordinate", or "BayesTools_inexact_ordinate" (all also of class BayesTools_hypothesis_ordinate); NA when eligible.

reason the message of that error; NA when eligible.

continuous_behavior the behavior of the prior without its point masses at the value ("regular", "zero", "infinite", "undefined", or "unknown"): the ordinate's provenance$continuous_behavior at a point mass and its behavior otherwise.

See Also

prior_density_ordinate(), hypothesis_BF()

Examples

```
spike_and_slab <- prior_mixture(
  list(
    prior("point", list(location = 0), prior_weights = 1),
    prior("normal", list(mean = 0, sd = 1), prior_weights = 1)
  ),
  is_null = c(TRUE, FALSE)
)
prior_ordinate_status(spike_and_slab, c(0, 0.5))
```

prior_phacking *Creates a prior distribution for a p-hacking selection kernel*

Description

prior_phacking() creates a prior distribution for a mass-preserving p-hacking redistribution kernel. The initial backend supports linear and quadratic power depletion between source and target, with depleted probability mass redistributed between target and destination.

This path is experimental and is not used by a released consumer yet. Mixed source/destination cuts across branches cannot be expressed; segments does not supply that geometry. Mixed or missing q currently falls back to a static linear bin rather than encoding mixed forms.

Usage

```
prior_phacking(
  side = "one-sided",
  target = 0.025,
  source = 0.25,
  destination = 0.005,
  form = c("linear", "quadratic"),
  alpha = prior("beta", list(1, 1)),
  report_scale = "pi_null",
  prior_weights = 1
)
```

Arguments

side	side geometry. Currently only "one-sided" is supported.
target	target p-value cut point.
source	source p-value cut point. Must be larger than target.
destination	destination p-value cut point. Must be smaller than target.
form	power depletion form, either "linear" or "quadratic".
alpha	prior distribution for the p-hacking severity parameter.
report_scale	reporting scale for deterministic summaries.
prior_weights	prior odds associated with a given distribution.

Value

prior_phacking() returns an object of class "prior".

prior_PP *Creates a prior distribution for PET or PEESE models*

Description

prior creates a prior distribution for fitting a PET or PEESE style models in RoBMA. The prior distribution can be visualized by the plot function.

Usage

```
prior_PET(
  distribution,
  parameters,
  truncation = list(lower = 0, upper = Inf),
  prior_weights = 1
)

prior_PEESE(
  distribution,
  parameters,
  truncation = list(lower = 0, upper = Inf),
  prior_weights = 1
)
```

Arguments

distribution name of the prior distribution. The possible options are

- "point" for a point density characterized by a location parameter.
- "normal" for a normal distribution characterized by a mean and sd parameters.
- "lognormal" for a lognormal distribution characterized by a meanlog and sdlog parameters.
- "cauchy" for a Cauchy distribution characterized by a location and scale parameters. Internally converted into a generalized t-distribution with df = 1.
- "t" for a generalized t-distribution characterized by a location, scale, and df parameters.
- "gamma" for a gamma distribution characterized by either shape and rate, or shape and scale parameters. The latter is internally converted to the shape and rate parametrization
- "invgamma" for an inverse-gamma distribution characterized by a shape and scale parameters. The JAGS part uses the BayesTools module distribution dbt_invgamma(shape, scale) on the natural parameter.
- "beta" for a beta distribution characterized by an alpha and beta parameters.
- "exp" for an exponential distribution characterized by either rate or scale parameter. The later is internally converted to rate.

"uniform" for a uniform distribution defined on a range from a to b

"moment" for a Johnson-Rossell moment prior characterized by exactly one of mode or tau, an optional order that defaults to 1, and an optional location that defaults to 0.

"invmoment" for a Johnson-Rossell inverse-moment prior characterized by exactly one of mode or tau, a df, an optional order that defaults to 1, and an optional location that defaults to 0.

"dirichlet" for a Dirichlet distribution over a simplex, whose finite nonnegative coordinates must have a finite total and roundoff bound and sum to one within that bound, characterized by a concentration vector alpha. Numeric entries must be finite and at least 0.01; this is a supported-input minimum, not a numerical guarantee. Positive expression-valued concentrations retain their existing JAGS-only support: expression containers, including literal elements, are not statically evaluated or validated, and R random generation refuses them. Cumulative weight functions and ordered allocations require numeric concentrations. Accepted sampling can still be unavailable; see prior_functions. Two-bin cumulative JAGS initialization near the minimum can repeatedly refuse unrepresentable interior ratios. Density and plot methods show one beta marginal per simplex coordinate.

parameters — list of appropriate parameters for a given distribution.

truncation — list with two elements, lower and upper, that define the lower and upper truncation of the distribution. Defaults to list(lower = -Inf, upper = Inf). The truncation is automatically set to the bounds of the support.

prior_weights — prior odds associated with a given distribution. The value is passed into the model fitting function, which creates models corresponding to all combinations of prior distributions for each of the model parameters and sets the model priors odds to the product of its prior distributions.

Value

prior_PET and prior_PEESE return an object of class 'prior'.

See Also

plot.prior(), prior()

Examples

```
# create a half-Cauchy prior distribution
# (PET and PEESE specific functions automatically set lower truncation at 0)
p1 <- prior_PET(distribution = "Cauchy", parameters = list(location = 0, scale = 1))

plot(p1)
```

prior_random *Random-effect prior specification*

Description

prior_random() defines priors and policies for formula random effects. Random-effect formulas own the grouping variables, slopes, intercepts, and covariance structure. prior_random() supplies priors and monitoring policy for the already parsed random-effect blocks. Random-effect grouping expressions must be grouping variables, explicit : interactions, or explicit / nesting. Derived grouping variables should be materialized as columns in data before they are referenced by a formula.

Usage

```
prior_random(
  ...,
  sd = NULL,
  covariance = NULL,
  cor = NULL,
  monitor = random_monitor(),
  new_levels = random_new_levels(),
  allocation = NULL,
  parameterization = "noncentered"
)

random_block(
  sd = NULL,
  covariance = NULL,
  cor = NULL,
  monitor = NULL,
  new_levels = NULL,
  terms = NULL,
  contrasts = NULL,
  sd_source = NULL,
  parameterization = NULL
)

random_variance_allocation(
  name,
  terms = NULL,
  sd = NULL,
  weights = NULL,
  parent = NULL,
  target = c("block", "sd_component"),
  scale = c("total_variance", "mean_variance"),
  sd_source = NULL,
  inclusion = NULL,
  display_name = name,
```

```
  component_names = NULL
)

allocation_ref(allocation_name, component)

random_covariance(
  structure = NULL,
  sd = NULL,
  cor = NULL,
  eta = NULL,
  cor_scale = c("fisher_z", "cor", "logit")
)

prior_lkj(eta = 1, include_correlation = TRUE, include_primitives = FALSE)

random_monitor(
  latent = TRUE,
  coefficients = FALSE,
  correlation = TRUE,
  lkj_primitives = FALSE
)

random_new_levels(method = c("error", "zero", "sample"))
```

Arguments

`...`	named `random_block()` specifications and, optionally, top-level `random_variance_allocation()` specifications.
`sd`	prior distribution for random-effect standard deviations. In `random_block()`, it is the default for the terms that `terms` does not override, and may be omitted when `terms` covers every SD of the block (see the standard-deviation priors section).
`covariance`	optional `random_covariance()` specification.
`cor`	optional correlation prior. Use `prior_lkj()` for unstructured correlation matrices and an ordinary scalar prior for structured correlation models.
`monitor`	`random_monitor()` object.
`new_levels`	`random_new_levels()` object controlling prediction for previously unseen grouping levels.
`allocation`	optional `random_variance_allocation()` specification, or a list of such specifications, defining aggregate-SD plus Dirichlet variance allocation across named random-effect blocks.
`parameterization`	requested sampled random-effect parameterization: `"noncentered"`, `"centered"`, or `"auto"`. The top-level value is inherited by blocks without an explicit `random_block()` override. A named block can additionally request `"mean_centered"`; see details.

terms — for random_block(), an optional named list of term-specific SD overrides. Each entry must be either a prior object or random_block(sd = ...), and names must match resolved model terms such as "intercept" or a slope term label (the index term for HCS and HAR). Terms not listed use sd; when every term is listed, sd may be omitted. For random_variance_allocation(), optional random-effect block names targeted by the shared allocation. If NULL, the allocation targets all remaining unallocated random-effect blocks at resolution time.

contrasts — for random_block(), an optional named character vector mapping factor predictors in that block to contrast families. Accepted values are "treatment" (also spelled "dummy"), "independent", "orthonormal", "meandif", "cumulative", and "cumulative_levels" (with or without the contr. prefix). Random-block contrasts are resolved independently of the fixed-effect design. Correlation structures with a structure-defined level basis do not accept this argument.

sd_source — for random_block(), optional external random-effect SD source created with random_sd_source(). A block-local sd_source replaces inherited top-level SD priors and inherited covariance SD priors, but conflicts with block-local sd, block-local covariance$sd, and term-specific SD overrides. For random_variance_allocation(), root allocations must specify exactly one of sd or sd_source; child allocations inherit their SD budget from parent.

name — required stable allocation identifier used by the backend and by allocation_ref().

weights — optional Dirichlet simplex prior over allocation weights.

parent — optional allocation_ref() object selecting a component of an earlier named allocation. Child allocations inherit that component's SD budget and must not specify sd or sd_source.

target — allocation target type. "block" allocates across named random-effect blocks or symbolic intermediate components. "sd_component" expands one selected heterogeneous block to its resolved SD components (one per exchangeable unit of a mean-difference or orthonormal factor term).

scale — variance normalization. "total_variance" uses sd_child = sd_parent * sqrt(w). "mean_variance" uses sd_child = sd_parent * sqrt(K * w) and is intended for target = "sd_component" heterogeneity tests. Summaries report the realized, gate-adjusted variance proportion for "total_variance" and K * w as a variance multiplier to the average SD-component variance for "mean_variance".

inclusion — optional named list of scalar probability priors, keyed by allocation component label. For target = "block", each listed component receives an independent Bernoulli gate and its SD contribution is multiplied by that gate. Public total-variance aggregates include these zero branches, while variance proportions condition on at least one active component. Gate names must match resolved allocation component labels. A block allocation with exactly one term is permitted only as a gate-only allocation: weights must be NULL and inclusion must name that term. Conditioning a gated total on presence uses the binding-factor (parent) gates that define a positive realized source, not every nested component gate; a total whose components all have inclusion gates and which also has parent gates cannot be conditioned (see parameter_mixed_posterior()). inclusion is not supported with target = "sd_component".

display_name — public semantic owner used as the parameter-name prefix. It defaults to name; use "" when a single model component needs no disambiguating prefix.

component_names
: optional public names for the allocation components, in the same order as terms. Internal component identifiers remain the (sanitized) names of terms. A sole "" suppresses the redundant public component argument for a one-component allocation.

allocation_name
: allocation name referenced by allocation_ref().

component
: component label inside the referenced allocation. Named terms supply component labels; unnamed terms use sanitized term names.

structure
: covariance structure label. Accepted labels are "ID", "DIAG", "CS", "HCS", "AR" / "AR1", "HAR", "CAR", and "UN" / "US"; matching is case-insensitive.

eta
: LKJ concentration parameter used when cor is omitted for unstructured covariance. Values above one regularize correlations toward zero; eta = 1 is uniform over correlation matrices.

cor_scale
: scale for an explicitly supplied scalar correlation prior. "fisher_z" is the default, "logit" uses a logit transform of the valid raw-correlation interval, and "cor" uses the raw correlation parameter. Coordinates beyond about 19 (Fisher-z) or 36.7 (logit) in absolute value map to a correlation of exactly +-1 in double precision (see prior_random()). For multi-column scalar-correlation structures, omitted cor resolves to a uniform prior on the raw correlation scale; explicitly requesting another scale without supplying cor is an error. US/UN structures use an LKJ prior instead. ID/DIAG structures and single-column blocks have no applicable scalar correlation.

include_correlation
: whether to monitor the correlation matrix for prior_lkj().

include_primitives
: whether to retain LKJ primitive beta coordinates for backend inspection. They remain internal and are not added to the public semantic parameter catalog.

latent
: whether to monitor standardized latent random effects. Required for bridge sampling and coefficient reconstruction.

coefficients
: whether to monitor realized group-level coefficients. This is convenient for inspection but can be memory intensive.

correlation
: whether correlation matrices are part of the semantic monitored output. Scalar CS/HCS/AR1/HAR/CAR backends retain only an internal rho coordinate during sampling and derive the requested matrix afterward; this avoids a quadratic JAGS graph without changing the public cor target.

lkj_primitives
: whether to retain LKJ primitive beta coordinates for backend inspection. They remain internal rather than semantic catalog quantities. For formula us random effects, raw LKJ primitive u coordinates are always retained internally because bridge sampling evaluates the LKJ prior on those coordinates.

method
: new-level prediction method: "error" disallows new levels, "zero" fixes their random-effect contribution at zero, and "sample" treats them as draws from the fitted random-effect distribution.

Details

Formula random effects in JAGS_formula() and JAGS_fit() require an explicit prior_random() object. The top-level arguments are defaults used by every random-effect block unless a named random_block() override is supplied in .... A top-level cor prior applies only to blocks whose resolved structure and dimension have a correlation parameter; it is not applied to id(), diag(), or single-column blocks. A block override with no covariance subcall inherits the shared covariance. A supplied random_covariance() or cor shorthand replaces the whole covariance object and uses its constructor defaults for omitted fields. In particular, random_covariance(cor = NULL) resets its correlation prior and scale to the formula defaults. A replacement also removes any shared covariance SD; if that was the only SD, supply a local or top-level SD prior. A block SD supplied through either sd or covariance = random_covariance(sd = ...) replaces the top-level SD supplied through either argument.

Each override name must match a stable random-effect block label generated from the formula parser or supplied by the user through named random terms. This keeps prior specification separate from formula syntax. For example, if the formula contains a block named "study", then prior_random(study = random_block(...)) overrides only that block.

Value

A list-like S3 object describing random-effect priors, covariance settings, allocation settings, and monitoring policy.

Standard-deviation priors

sd must be a BayesTools prior object with nonnegative support, such as a truncated normal, gamma, point prior at zero, or mixture prior. When a structure needs multiple marginal standard deviations, a single sd prior is replicated over those SD components unless random_block(terms = ...) overrides the prior of individual terms; sd remains the default for the terms that are not overridden. A block with one SD per term (DIAG, US/UN, HCS, and HAR) may omit sd, both in the block and at the top level, when terms supplies a prior for every resolved term: the coefficient columns such as "intercept" or a slope label, or the index term whose per-level SDs HCS and HAR share one prior. Coverage is checked when the formula is resolved, and uncovered terms are named in the error. Structures with one shared SD (ID, diag(..., hom = TRUE), CS, AR1, and CAR) take no term overrides and always require sd. Non-point multivariate (vector) SD priors are unsupported; use nonnegative scalar SD priors. Deterministic nonnegative point vectors and supported independent factor priors retain their usual behavior.

For ordinary estimation, use continuous positive SD priors. Use point or mixture priors at zero only when a random-effect component is intentionally fixed or model-averaged.

Covariance structures

random_covariance() stores structure-specific covariance settings. The formula owns the structure and its design. ID, DIAG, and US/UN operate on generated random-coefficient columns; CS/HCS, AR1/HAR, and CAR operate on structure-owned indices. Supported structure labels are:

- "ID": independent columns with one shared SD.
- "DIAG": independent columns with one SD per generated column (one per exchangeable unit of a mean-difference or orthonormal factor term).

- "CS" / "HCS": compound symmetry over one or more index variables, with one scalar correlation; "HCS" has one SD per index level. Multiple index variables in cs(index1 + index2 | group) are combined with interaction(..., drop = TRUE).
- "AR" / "AR1" / "HAR": discrete autoregressive structure over a single ordered index variable, with one scalar correlation; "HAR" has one SD per index level.
- "CAR": continuous-time AR(1), currently for car(time | group), using cor ^ abs(t_i - t_j) with 0 <= cor < 1.
- "UN" / "US": unstructured covariance with SD priors and prior_lkj().

ID/DIAG/US formulas use random-intercept controls on their left side. Factor coding is resolved separately and can be set explicitly with random_block(contrasts = ...). The structured index families reject that contrast override and reject explicit 1, 0, or -1 terms. HCS is not an alias for US: HCS has one common pairwise correlation, whereas US estimates an unrestricted correlation matrix among coefficient columns.

A mean-difference or orthonormal factor term (also f:x, and f:g when every factor is so coded) is one exchangeable unit in a DIAG or US/UN block: its contrast columns form an arbitrary basis of the level deviations, so they share one SD (reported as sd(f), with one scalar prior) and are uncorrelated, which makes the model of the level deviations independent of the basis. A term that crosses such a factor with a treatment, independent, or ordered factor has one unit per column of that partner (e.g., sd(f:h[b])). Under US/UN a unit keeps its correlations with the other columns: the LKJ prior is restricted to an identity unit block (density det(R)^(eta - 1)), and the summaries report the correlation with each level of the factor, cor(x, f[a]), and the multiple correlation with the whole term, cor_total(x, f), instead of correlations of contrast columns. A US/UN block with more than one such term of three or more levels is refused (class BayesTools_random_effects_exchangeable_units_unavailable, reason "several_correlated_units"), as is a factor or vector SD prior on a unit (reason "unit_sd_prior"); a US/UN block that is one such term is its diagonal block, and a two-level factor has one column and is an ordinary column of the LKJ prior. Use treatment or independent contrasts for one SD per level.

The formula parser owns the actual structure used by a block. Supplying a conflicting structure in random_covariance() is rejected when the formula is resolved.

Correlation priors

Use prior_lkj(eta = ...) for full unstructured correlation matrices. When cor is omitted, an unstructured block receives prior_lkj(eta = 1) after its dimension is known. LKJ priors use the BayesTools compiled JAGS module.

Use an ordinary scalar prior in cor for scalar-correlation structures ("CS", "HCS", "AR", "HAR", and "CAR"). When cor is omitted, BayesTools places a uniform prior directly on the complete valid raw-correlation interval: (-1 / (K - 1), 1) for CS/HCS, (-1, 1) for AR/HAR, and [0, 1) for CAR. Here K is the resolved number of index levels. The CS/HCS lower bound -1/(K-1) is a singular kernel point and is open; the stored default Uniform parameters name the closed JAGS dunif hull. Uniform as a family is not an open-interval distribution. User-supplied Uniforms that include the CS bound are retained, and density there remains -Inf / a native error. AR/HAR |rho| = 1 is likewise a kernel singularity, not a Uniform policy. CAR includes rho = 0. Thus the default does not silently exclude negative correlations wherever the structure permits them. cor_scale controls the scale of an explicitly supplied scalar cor prior:

- "fisher_z" places the prior on atanh(cor) and is the default for an explicit scalar cor prior. A point prior at zero corresponds to independence. For one-sided structures such as "CAR", the Fisher-z prior is truncated to the valid raw-correlation interval.
- "logit" places the prior on a logit transform of the valid raw-correlation interval.
- "cor" places the prior directly on the raw correlation; the prior is truncated to the structure-specific valid interval.

In double precision, Fisher-z coordinates beyond about 19 in absolute value (logit coordinates beyond about 36.7) map to a correlation of exactly ± 1. A scalar correlation prior with mass there gives draws on the boundary of the support: centered fits stop in JAGS with "Invalid parent values" and noncentered draws are refused when the correlation is reconstructed. A "CAR" correlation prior whose whole support lies there is refused when the model is built. A prior with less mass beyond these coordinates avoids the problem.

Variance allocation

random_variance_allocation() expresses one aggregate SD source plus a Dirichlet prior over variance proportions. A one-component block allocation is also supported when it supplies only an inclusion gate; this multiplies the component SD by the Bernoulli indicator without creating an artificial allocation weight. Root allocations must specify exactly one of a prior-owned sd prior or sd_source = random_sd_source(...). sd_source can reference an already-defined JAGS node such as scalar tau or row-shaped tau without creating a prior for that node. Multiple named allocations can be supplied in prior_random(); child allocations can inherit one component of an earlier allocation through parent = allocation_ref(...).

With scale = "total_variance", the ungated allocation preserves summed component variance. With optional independent component gates, the realized component SD and total variance are

$$\sigma_j = I_j \sigma_{\text{slab}} \sqrt{w_j}, \quad \sigma^2_{\text{total}} = \sigma^2_{\text{slab}} \sum_j I_j w_j, \quad w \sim \text{Dirichlet}(\alpha).$$

Its public aggregate names sd_total and var_total refer to the realized gated total. var_prop(j) is the realized share $I_j w_j / \sum_k I_k w_k$, conditional on positive realized total variance. It is undefined on all-off draws, which are omitted when the variance proportions are summarized. The positive slab scale and raw Dirichlet weights remain internal coordinates.

With scale = "mean_variance", the aggregate SD preserves mean component variance:

$$\sigma_j = \sigma_{\text{common}} \sqrt{K w_j}, \quad w \sim \text{Dirichlet}(\alpha).$$

Its public aggregate names are sd_common and var_common, and its components are exposed as var_mult(...) and sd_mult(...), where var_mult(j) = K * w[j] and sd_mult(j) = sqrt(K * w[j]). Thus sd_common is the shared variance-scale anchor before the component multipliers are applied; it is not a sum of SDs.

The fitted parameter_map() links concrete parameter_coordinates() to the public random-effect quantities in parameter_catalog(). Canonical names use (formula) owner: quantity(arguments), with owner: omitted for exactly one random block and retained when multiple blocks require disambiguation. Parentheses contain coefficient or parameter names and square brackets contain factor or index levels. Public correlations use cor; compact scalar rho and LKJ construction coordinates remain internal backend dependencies. Downstream packages can opt into centrally generated simplified aliases. A sole intercept argument is then omitted (sd(intercept) becomes sd), and an

owner-free shorthand is accepted only when it resolves uniquely. A non-intercept argument such as `sd(x)` is retained.

Allocation `name` is a required stable backend identifier. `display_name` independently controls an allocation-specific public owner, and `component_names` controls the public component arguments. An unnamed local SD-component allocation inherits its enclosing block's public owner under the same one-block/multiple-block rule. Total-variance allocations expose realized `sd_total`, `var_total`, and `var_prop(...)`; mean-variance allocations expose `sd_common`, `var_common`, `var_mult(...)`, and `sd_mult(...)`.

This is useful for nested or crossed random intercepts when the prior should control total heterogeneity separately from how that heterogeneity is allocated across components. The allocation targets named random-effect blocks through `terms`. If `terms = NULL`, a single root block allocation targets all random-effect blocks at resolution time and must resolve to at least two blocks. Generated JAGS syntax supports allocated blocks with one SD component (such as a block that is one mean-difference or orthonormal factor term, an exchangeable unit with one SD), homogeneous structures where one SD controls the block, and explicit `target = "sd_component"` allocations that split the resolved SD components of one heterogeneous block. Row-shaped external sources are applied inside the observation-level random-effect contribution, including child allocations and SD-component allocations; they do not create scalar SD summary parameters. Prediction and bridge-sampling reconstruction for these sources require standardized latent random-effect monitors, row-source values from either posterior columns or a `parameter_source(..., values = ...)` callback, and any required Dirichlet allocation coordinates. Use `scale = "total_variance"` to preserve summed variance and `scale = "mean_variance"` to preserve average variance inside a heterogeneous block.

Monitoring and prediction

`random_monitor()` controls which random-effect quantities are saved from JAGS. The default monitors standardized latent random effects because they are required for bridge sampling and for reconstructing group-level coefficients in prediction. Turning `latent = FALSE` can reduce memory use, but bridge sampling and conditional prediction may require refitting.

`random_new_levels()` controls prediction for grouping levels that were not observed when the model was fitted. The default `method = "error"` disallows new levels. `method = "zero"` assigns zero random-effect contribution to new levels and `method = "sample"` draws new group-level coefficients from the fitted random-effect distribution during draw-valued prediction or integrates them through the block covariance for covariance-valued prediction. A level declared in a single grouping factor (for example, an unused factor level) but without rows in the fitting data remains a fitted group whose coefficients are informed only by the random-effect distribution; prediction treats it as a new level. Blocks with a known group covariance (`random_group_covariance()`) are the exception: the kernel links declared levels to the observed ones, so their declared levels are predicted from the fitted coefficients, and new levels are unsupported. Groupings of several variables contain only observed combinations.

Parameterization

`parameterization` controls how sampled random-effect blocks are represented by the computational backend. `"noncentered"` retains standardized latent effects, `"centered"` samples realized group-level coefficients, and `"auto"` lets the backend choose a supported representation from the

resolved block design. Its deterministic heuristic uses within-group replication and design conditioning; it does not inspect outcome-specific likelihood curvature. The resolved choice, reason, and tuning thresholds are stored in fitted random-effect metadata; those thresholds are computational tuning defaults, not statistical assumptions. The choice does not change the SD, allocation, or correlation priors. Block-specific values supplied through `random_block()` override the top-level default. Explicit `"centered"` compilation requires strictly positive backend-owned SD coordinates and therefore rejects point-mass-at-zero scales, inclusion-gated variance allocations, and external or row-indexed SD sources. A centered CAR block additionally requires one block-owned SD prior whose support is bounded away from zero and infinity, and a known between-group covariance with several random-effect columns is compiled only noncentered. `"auto"` falls back to `"noncentered"` when those contracts or its design-information checks do not support centering.

An explicitly named `random_block()` may request `"mean_centered"` for a sampled block. It samples the group effects around the compiled contribution of the fixed terms the block absorbs, preserving the original effect, SD, allocation, and correlation priors (hierarchical centering). A block absorbs every fixed term whose design columns its random-effect design spans (such as the intercept and the slope of `x` for `(1 + x | g)`, or the intercept and a factor term for `us(0 + f | g)`), every term constant within its groups when it has a random intercept (such as a group-level moderator), and the group locations of a mean-centered block whose groups contain its groups (nested blocks, such as `(1 | g/h)` with both blocks mean-centered). Each term is absorbed by the first block that can absorb it, coarser groupings first; other terms, including terms only partly spanned by a block design, remain in the predictor. The fixed effects need not have Normal priors; multipliers and a compiled `log(intercept)` contribution are also supported. A block that would absorb nothing and a rank-deficient block design are rejected. This mode requires the positive-SD contract above and excludes known between-group covariance and CAR structures. Selection-model callers must additionally require that the translated block is retained in the selection normalizer. Semantic random coefficients remain zero-mean deviations; prediction and bridge sampling retain their existing coefficient and standardized-latent conventions. Mean centering usually mixes better when the data inform the group effects well, and noncentering when they do not.

Examples

```
sd_prior <- prior(
  "normal", list(mean = 0, sd = 0.5),
  truncation = list(lower = 0, upper = Inf)
)
cor_prior <- prior("normal", list(mean = 0, sd = 0.5))

# One SD prior used for every random-effect block.
prior_random(sd = sd_prior)

# Override a named block while keeping a global default.
prior_random(
  sd = sd_prior,
  study = random_block(sd = prior("gamma", list(shape = 2, rate = 2)))
)

# Unstructured covariance: omitted correlation resolves to LKJ(1).
prior_random(
  study = random_block(sd = sd_prior)
)
```

```
# Scalar-correlation structure with a Fisher-z prior.
prior_random(
  study_time = random_block(
    covariance = random_covariance(
      structure = "AR",
      sd = sd_prior,
      cor = cor_prior,
      cor_scale = "fisher_z"
    )
  )
)

# Total heterogeneity plus Dirichlet variance allocation. The allocation
# has an internal name but no redundant public owner: sd_total,
# var_prop(study), and var_prop(outcome).
prior_random(
  random_variance_allocation(
    name = "heterogeneity",
    display_name = "",
    terms = c(study = "study", outcome = "outcome"),
    sd = sd_prior,
    weights = prior("dirichlet", list(alpha = c(1, 1)))
  )
)

# Allocate a block SD across its coefficient components. This exposes
# study_components: sd_common, study_components: var_mult(intercept), and
# study_components: sd_mult(intercept), with analogous names for slopes.
prior_random(
  random_variance_allocation(
    name = "study_components",
    terms = "study",
    target = "sd_component",
    scale = "mean_variance",
    sd = sd_prior,
    weights = prior("dirichlet", list(alpha = c(1, 1)))
  )
)

# A child allocation can split one inherited variance budget.
prior_random(
  random_variance_allocation(
    name = "total_re",
    terms = c(nested = "nested", simple = "study"),
    sd = sd_prior,
    weights = prior("dirichlet", list(alpha = c(1, 1)))
  ),
  random_variance_allocation(
    name = "nested_split",
    parent = allocation_ref("total_re", "nested"),
    terms = c("paper", "country"),
    weights = prior("dirichlet", list(alpha = c(3, 1)))
```

```
  )
)
```

`prior_spike_and_slab` *Creates a spike and slab prior distribution*

Description

`prior_spike_and_slab` creates a spike and slab prior distribution corresponding to the specification in Kuo and Mallick (1998) (see O'Hara and Sillanpää (2009) for further details). I.e., a prior distribution is multiplied by an independent indicator with values either zero or one.

Usage

```
prior_spike_and_slab(
  prior_parameter,
 prior_inclusion = prior(distribution = "spike", parameters = list(location = 0.5)),
  prior_weights = 1
)
```

Arguments

`prior_parameter`	a prior distribution for the parameter
`prior_inclusion`	a prior distribution for the inclusion probability. The inclusion probability must be bounded within 0 and 1 range. Defaults to `prior("spike", parameters = list(location = 0.5))` which corresponds to 1/2 prior probability of including the slab prior distribution (but other prior distributions, like beta etc can be also specified).
`prior_weights`	prior odds associated with a given distribution. The value is passed into the model fitting function, which creates models corresponding to all combinations of prior distributions for each of the model parameters and sets the model priors odds to the product of its prior distributions.

Value

return an object of class 'prior'.

See Also

`prior()`

Examples

```
# create a spike and slab prior distribution
p1 <- prior_spike_and_slab(
   prior(distribution = "normal", parameters = list(mean = 0, sd = 1)),
   prior_inclusion = prior(distribution = "beta", parameters = list(alpha = 1, beta = 1))
)
```

`prior_weightfunction` *Creates a prior distribution for a weight function*

Description

`prior_weightfunction` creates a prior distribution for fitting a RoBMA selection model. The `side` and `steps` arguments define the p-value bins, and the `weights` argument defines the prior on the publication weights in those bins.

Usage

```
prior_weightfunction(
  side = "one-sided",
  steps = c(0.025, 0.05),
  weights = wf_cumulative(),
  reference = "most_significant",
  prior_weights = 1,
  model = selection_model()
)

wf_cumulative(alpha = NULL)

wf_fixed(omega)

wf_independent(prior, scale = "omega")
```

Arguments

`side`	side geometry. Either `"one-sided"` or `"two-sided"`.
`steps`	increasing p-value cut points between 0 and 1.
`weights`	a weight-prior object created by `wf_cumulative()`, `wf_fixed()`, or `wf_independent()`.
`reference`	reference bin. Currently only `"most_significant"` is supported and fixes the most significant bin to `omega = 1`.
`prior_weights`	prior odds associated with a given distribution.
`model`	a fixed selection-model specification from `selection_model()`. Its choices do not add weight-prior parameters or change `prior_weights`.

alpha	finite numeric cumulative-Dirichlet concentration parameters, each at least `0.01`. If omitted, a flat Dirichlet prior is used with one concentration parameter per bin. The supported minimum is not a numerical guarantee: accepted R sampling can still be unavailable, and two-bin cumulative JAGS initialization near the minimum can repeatedly refuse ratios that are no longer representable strictly inside `(0, 1)`. Such failures are explicit; concentrations, draws and initialization settings are not repaired. See `prior_functions` for sampling conditions.
omega	fixed non-negative relative publication weights, one per bin. When assembled by `prior_weightfunction()`, the reference-bin weight must be exactly 1. Named and integer values are accepted; nearby values are refused without normalization.
prior	prior distribution for each non-reference weight.
scale	latent scale for independent weights. `"omega"` places the prior directly on the non-negative publication weight. `"log_omega"` places the prior on `log(omega)` and transforms with `omega = exp(log_omega)`, allowing weights above one whenever the log prior assigns mass above zero. `"log"` is accepted as an alias of `"log_omega"`.

Value

`prior_weightfunction` returns an object of class 'prior'.

See Also

`plot.prior()`

Examples

```
p1 <- prior_weightfunction(
  side = "one-sided",
  steps = c(.05, .10),
  weights = wf_cumulative(alpha = c(1, 1, 1))
)

p2 <- prior_weightfunction(
  side = "one-sided",
  steps = c(.05),
  weights = wf_independent(prior("beta", list(1, 1)))
)
```

---

random_effects_compile

Random-effect compilation policy

Description

`random_effects_compile()` specifies which resolved formula random-effect blocks are compiled as structural marginalized blocks. Blocks not listed in `marginalized` are compiled as ordinary sampled random effects. Marginalized blocks keep their SD, allocation, correlation, covariance-structure, and metadata, but do not generate latent group-level coefficients and do not contribute to the formula linear predictor. Callers that marginalize a block are responsible for using the stored metadata and generated hyperparameter nodes in their likelihood.

Usage

```
random_effects_compile(marginalized = NULL)
```

Arguments

`marginalized`	optional character vector of resolved random-effect block names that should be compiled structurally without latent group-level coefficients.

Value

A list-like S3 object containing the requested `marginalized` block names, used by `JAGS_formula()`, `JAGS_fit()`, and `JAGS_bridgesampling()`. Resolved formula-design metadata additionally records the complete `sampled`, `marginalized`, and named `mode` outputs.

Examples

```
random_effects_compile()
random_effects_compile(marginalized = "study_estimate")
```

---

random_effects_correlation_draws

Scalar-structured random-effect correlation draws

Description

Reconstructs dense correlation matrices for one scalar-structured formula random-effect block from its compiled metadata and posterior draws. The matrices are derived only when this function is called; fitted objects can otherwise retain the compact scalar correlation coordinate.

Usage

```
random_effects_correlation_draws(random_term, posterior_samples)
```

Arguments

random_term	compiled random-effect term metadata from a BayesTools_formula_design object.
posterior_samples	posterior sample matrix, data frame, mcmc, or mcmc.list. Samples must contain the compiled internal scalar-correlation coordinate unless it is fixed in random_term metadata. This coordinate is a backend dependency, not the public semantic cor quantity. A one-coefficient block needs no correlation coordinate and accepts a numeric draw-by-zero matrix.

Details

Supported structures are compound symmetry ("CS" and "HCS"), discrete autoregressive ("AR1" and "HAR"), and continuous-time autoregressive ("CAR"). Correlation coordinates are transformed and checked against the structure-specific support recorded in random_term. "CAR" matrices use the compact ordered time coordinates stored by the formula compiler.

The returned array is dense and therefore requires memory proportional to the number of posterior draws times the squared coefficient count.

Value

A dense numeric array with dimensions draw x coefficient x coefficient. Coefficient dimension names use the compiled random-effect column names when available.

See Also

prior_random() random_effects_marginal_vcov()

Examples

```
dat <- data.frame(
  study = factor(rep(c("s1", "s2"), each = 3)),
  time = factor(rep(1:3, 2))
)
generated <- JAGS_formula(
  ~ 1 + ar1(time | study),
  parameter = "mu",
  data = dat,
  prior_list = list(intercept = prior("normal", list(0, 1))),
  prior_random = prior_random(
    sd = prior("point", list(location = 1)),
    cor = prior("normal", list(0, 0.5))
  )
)
random_term <- generated$formula_design$random_effects[[1]]
posterior <- matrix(c(-0.2, 0.4), ncol = 1)
colnames(posterior) <- random_term$correlation$rho_name
random_effects_correlation_draws(random_term, posterior)
```

random_effects_dependency_matrix

Structural row dependencies of formula random effects

Description

Returns conservative row adjacency implied by compiled random-effect group maps, known group covariance, and the fixed coefficient supports of `id()` and `diag()` blocks. Disjoint independent coefficient supports do not connect rows. Other covariance structures retain conservative group dependencies. This is independent of posterior draws, sampled covariance, and compile mode. Downstream likelihoods can combine this matrix with their sampling dependencies before partitioning rows.

Usage

```
random_effects_dependency_matrix(random_effects, n_rows, blocks = NULL)
```

Arguments

`random_effects`	Compiled `random_effects` terms from a `BayesTools_formula_design`, or the `metadata$blocks` returned by `random_effects_marginal_vcov()` for prediction rows.
`n_rows`	Number of observation rows.
`blocks`	Optional character vector of block names to include. `NULL` includes every block; `character()` includes none.

Value

A symmetric logical `n_rows` by `n_rows` matrix with a true diagonal.

random_effects_formula

Normalize random-effect formula input

Description

Normalizes a formula or a list of formulas containing random-effect terms into a single formula and parsed BayesTools random-effect terms.

Usage

```
random_effects_formula(random, envir = parent.frame(), group_covariance = NULL)
```

Arguments

random	a formula or non-empty list of formulas.
envir	environment used for the returned formula.
group_covariance	optional known group covariance. Use one random_group_covariance() object or numeric matrix for a single random block, or a named list of objects/matrices for multiple random blocks. Names should match resolved random-effect block names, or grouping labels when the grouping label is unambiguous.

Details

List names define top-level random-effect components used by allocation and summary metadata. In lists with two or more entries, missing names are replaced by "component 1", "component 2", and so on, then sanitized for JAGS node names. An unnamed one-entry list is treated like its bare formula and receives no user-facing component prefix; an explicitly named entry retains its name. Unnamed random-effect terms inside a list entry are auto-named from the component label. Explicit name = arguments define the final random-effect block name and are the names targeted by prior_random() block overrides. The fitted parameter_map() links concrete parameter_coordinates() to public quantities declared by parameter_catalog(). Their canonical form is (formula) owner: quantity(arguments), with owner: omitted for exactly one random block and retained when multiple blocks require disambiguation. Parentheses contain coefficient or parameter names and square brackets contain factor or index levels. Public correlations use cor; compact scalar rho and LKJ construction coordinates remain internal.

Random-effect terms have two distinct left-side grammars:

- id(), diag(), and us() / un() use a coefficient formula. The 1, 0, and -1 terms control the random intercept; continuous slopes, factor slopes, and interactions are supported. Plain (expr | group) defaults to us(), and expr || group to diag().
- cs() / hcs(), ar1() / ar() / har(), and car() use a structure-owned index specification. They do not accept random_block(contrasts = ...). cs(), hcs(), ar1(), and har() reject explicit intercept controls; car() has no intercept, so it rejects an explicit 1 and accepts the redundant 0 and - 1.

Random-coefficient factor coding comes from the block's resolved contrast metadata. It reuses an already established fixed-factor contrast by default; random_block(contrasts = ...) is the explicit per-block override. Removing the random intercept does not by itself request one coefficient per factor level.

The discrete index structures cs(), hcs(), ar1(), and har() accept factor, character, numeric/integer, or logical columns. Existing factor levels are preserved; otherwise sorted unique values define levels and AR(1) order. CS/HCS can combine several columns with +; AR1/HAR require exactly one. CAR requires one finite numeric/integer column, or an ordered factor with numeric labels, and uses actual coordinate distances. AR1/HAR warns when character or unordered-factor numeric labels resolve to a non-increasing numeric order, such as 1, 10, 2. The order is retained; use numeric values for numeric ordering, or an ordered factor with explicit levels to confirm another intended order.

The random() / re() wrappers also accept hom = FALSE with covariance = "cs" or "ar1" as aliases for "hcs" or "har", respectively. An explicit random_covariance(structure = ...) prior

must match that resolved heterogeneous structure. Direct `cs()` / `ar1()` tags retain their homogeneous meaning; use `hcs()` / `har()` for heterogeneous SDs.

Random-effect predictors must be literal data-column names combined with standard formula operators. Inline transformations and arbitrary calls are rejected; create transformed predictors as explicit columns first. Grouping terms support variables, : interactions, and / nesting. Grouping interactions follow base-R/lme4 lexicographic tuple order, and the fitted tuple-to-index map is stored for replay. Categorical predictor and grouping levels must not contain BayesTools' reserved internal tokens, such as `__xXx__`. Predictor names must not contain the substring `intercept`, which is reserved for the intercept.

Only the random-effect terms of the input are returned. Other parts of the right-hand side, such as `offset()` or the `0 +` / `- 1` intercept controls of the fixed part (`~ 0 + (1 | g)` returns the term `us(1 | g)`), are dropped; the fixed part belongs to the model formula passed to `JAGS_formula()`.

Value

A `BayesTools_random_effects` object with `formula`, `terms`, `components`, and `group_covariance` fields.

`random_effects_level_roles`

Grouping-level roles of formula random effects

Description

Identifies the sole declared estimate-level term from the compiled grouping maps. This classification selects a source term; it does not assert independent observations or alter its covariance representation.

Usage

```
random_effects_level_roles(random_effects, n_rows)
```

Arguments

`random_effects`	Compiled `random_effects` terms from a `BayesTools_formula_design`.
`n_rows`	Number of retained observation rows.

Details

A term is `"estimate"` when its `group_map` has a distinct grouping level for every retained row. There may be zero or one such term; multiple qualifying terms are an error. All remaining terms are `"other"`, including terms with independent or mixed coefficient supports within repeated groups.

A qualifying term keeps the estimate role even when a known group covariance correlates its distinct levels. Consumers must preserve that full covariance and its row dependencies. Roles do not depend on SD values, coefficient supports, posterior draws, or sampled versus marginalized compile modes. `random_effects_source_roles()` separately describes covariance geometry.

Value

A named character vector, keyed by block name, containing `"estimate"` or `"other"` for every term.

See Also

`random_effects_dependency_matrix()`

`random_effects_marginal_diagonal_factor`
Diagonal-plus-factor random-effect marginal representation

Description

Reduces structurally eligible random-effect covariance factor states to a diagonal contribution and structurally certified loading columns for every supplied row block. Eligibility and factor dimension are determined from compiled metadata, never from evaluated covariance values. Known group covariance uses its fixed Cholesky factor while retaining the complete coefficient covariance. Sampled zero SDs do not alter the structural supports.

Usage

```
random_effects_marginal_diagonal_factor(factors, cache = NULL)
```

Arguments

`factors`	A `BayesTools_random_effects_marginal_factor_states` object, or a single bridge-likelihood `factor_state` value carrying `factor_plans`, `factor_states`, and `row_blocks`.
`cache`	Optional environment used to retain the invariant reduction plan across repeated evaluations of one bridge-likelihood factor-state contract. Changing numerical factor states are still validated on every call. The reduction is keyed on the identity of the compiled contract, so the cache serves factor states that share it: calls of `random_effects_marginal_factor_states()` that pass one common `cache`, or the states of one bridge likelihood. States of a separately compiled contract signal a changed contract.

Details

Structurally unavailable representations signal `BayesTools_random_effects_marginal_factor_unavailable`, inheriting from `BayesTools_random_effects_error`, `error`, and `condition`. Its stable `reason` field accompanies `message` and `call`. The shared parent also includes invalid allocation support errors; it alone does not identify structural unavailability.

Value

A list with a numeric `diagonal` matrix, one loading array and structural rank per row block, and `row_blocks`. The `loading_supports` list contains one logical row-by-loading matrix per block, derived from compiled metadata independently of the evaluated loading values. The logical `diagonal_support` vector marks rows with a structurally declared diagonal contribution, including when its evaluated SD is zero.

See Also

`random_effects_marginal_factor_states()`

`random_effects_marginal_factor_diagonal`
Random-effect marginal covariance diagonals from factor states

---

Description

Evaluates the observation-level marginal variance represented by `random_effects_marginal_factor_states()` without reconstructing dense covariance matrices or recompiling the random-effect formula metadata.

Usage

```
random_effects_marginal_factor_diagonal(factors, by_block = FALSE)
```

Arguments

`factors`	a `BayesTools_random_effects_marginal_factor_states` object.
`by_block`	whether to return one variance matrix per random-effect block.

Value

A numeric matrix with one posterior draw per row and one fitted observation per column when `by_block = FALSE`; otherwise a named list of such matrices in fitted random-block order.

See Also

`random_effects_marginal_factor_states()` `random_effects_marginal_factor_product()`

random_effects_marginal_factor_product

Multiply random-effect marginal covariance factors by row vectors

Description

Applies the covariance represented by `random_effects_marginal_factor_states()` to one row vector per posterior draw without constructing a dense `draw x row x row` covariance array.

Usage

```
random_effects_marginal_factor_product(factors, vectors, by_block = FALSE)
```

Arguments

`factors`	a `BayesTools_random_effects_marginal_factor_states` object.
`vectors`	numeric matrix with one posterior draw per row and one fitted observation per column.
`by_block`	whether to return one product matrix per random-effect block.

Value

A numeric matrix when `by_block = FALSE`; otherwise a named list of numeric matrices in fitted random-block order.

See Also

`random_effects_marginal_factor_states()`

random_effects_marginal_factor_states

Random-effect marginal covariance factor states

Description

Compiles the same structural covariance representation used by bridge sampling and evaluates its draw-varying factor states for fitted rows. This avoids constructing dense `draw x row x row` arrays when a downstream likelihood can consume covariance factors directly.

Usage

```
random_effects_marginal_factor_states(
  fit,
  parameter = NULL,
  posterior_samples = NULL,
  prior_list = NULL,
  blocks = NULL,
  row_blocks,
  cache = NULL,
  ...
)
```

Arguments

`fit`, `parameter`, `posterior_samples`, `prior_list`, `blocks`	See `random_effects_marginal_vcov()`.
`row_blocks`	a list of integer row-index vectors that partitions the fitted rows without separating any structurally nonzero covariance contribution.
`cache`	optional environment used to retain the compiled contract – the design, its priors, the selected blocks and the evaluator – across repeated calls that differ only in their posterior draws. The key is compared by value, so a different contract recompiles.
`...`	reserved for future extensions. Unused arguments are rejected.

Value

A list of class `BayesTools_random_effects_marginal_factor_states` with invariant `factor_plans`, one `factor_states` list per posterior draw, `row_blocks`, a `contract_id` identifying the compiled contract, and structural `metadata`.

See Also

`random_effects_marginal_vcov()` `JAGS_bridgesampling()`

---

random_effects_marginal_factor_vcov

Materialize random-effect marginal covariance factors

Description

Constructs dense covariance draws from compiled factor states, using the same group, row-scale, and coefficient-factor geometry as the diagonal and matrix-product helpers. Prefer those helpers when full covariance is not required by the consuming likelihood.

Usage

```
random_effects_marginal_factor_vcov(factors)
```

Arguments

factors	See random_effects_marginal_diagonal_factor().

Value

A numeric array with dimensions draw x row x row.

See Also

random_effects_marginal_factor_diagonal() random_effects_marginal_factor_product()

---

random_effects_marginal_update_grid

Random-effect marginal covariance update grid

Description

Compiles the draw- and candidate-varying coefficient state for an exact non-affine random-effect covariance update. The output remains compact in the number of posterior draws and grid values; it does not materialize one random-effect state for every draw-by-grid combination.

Usage

```
random_effects_marginal_update_grid(
  fit,
  update,
  values,
  posterior_samples = NULL,
  prior_list = NULL
)
```

Arguments

fit	a fitted object carrying formula random-effect metadata.
update	a plan returned by random_effects_marginal_update_plan().
values	finite candidate values on the scale declared by update. Source-coordinate factor updates replace source_parameter in the posterior. Quantity-scale factor updates accept non-negative values of the selected SD or variance; variance values are converted to SDs using the plan's coefficient_scale_transform.
posterior_samples	optional posterior sample object. By default the posterior is obtained from fit.
prior_list	optional formula prior list used to reconstruct random SD bindings. By default it is obtained from fit.

Details

This accessor supports plans whose `family` is `"factor"` or `"markov"`. Every state is evaluated by the same compiled formula machinery used by `random_effects_marginal_factor_states()`. No covariance form is inferred from posterior samples or candidate covariance matrices.

Value

A list of class `BayesTools_random_effects_marginal_update_grid`. Factor updates contain posterior coefficient scales, correlation Cholesky factors, and the selected candidate scales. The candidate scale replaces the scale of column `component_index` only for a plan that holds the sibling SDs fixed (a direct per-column SD), and of every column in `driven_columns` when the direct SD is shared by several columns. For a plan with `held_fixed = "allocation_weights"` (an allocation-derived component SD) the grid also contains `column_factor` (values by posterior draws), the factor `candidate_scale / coefficient_scale[, component_index]` by which the scale of every driven column (`driven_columns`, all columns when it is `NULL`) is multiplied; it is `NA` for the draws where the current selected SD is zero, which no allocation scale maps to a non-zero (or an unchanged zero) candidate. Markov updates contain posterior coefficient scales plus candidate correlation Cholesky factors, transitions, and innovation variances.

See Also

`random_effects_marginal_update_plan()` `random_effects_marginal_factor_states()`

---

random_effects_marginal_update_plan

Random-effect marginal covariance update plan

Description

Resolves a selected public random-effect quantity to the exact scalar dependence of its marginal covariance contribution. The result is compiled exclusively from the fitted parameter map and formula random-effect metadata; posterior draws are neither inspected nor used to infer an update form. An unsupported plan can include a `conditional` entry containing `required_active` and `plan`. The nested plan applies only to posterior rows where every named inclusion indicator equals one. It does not declare an unconditional update for the selected public quantity.

Usage

```
random_effects_marginal_update_plan(fit, selection)
```

Arguments

`fit`	a fitted object carrying a BayesTools parameter map and formula design metadata.
`selection`	a one-quantity selection returned by `parameter_catalog_resolve()`.

Details

`family = "affine"` means that, conditional on every non-selected posterior coordinate, the row covariance has the exact form `A + h(value) * B`, where `value` is the coordinate declared by `coefficient_input` (`"source"` or `"quantity"`) and `h` is recorded in `coefficient_transform`. `family = "factor"` identifies an exact candidate-dependent covariance factor, while `family = "markov"` identifies a compiled structured Markov state. Allocation-derived component SDs are affine on every covariance structure when the allocation scale drives all columns of every block it reaches (`A + h(v) B` with `h` the square of the SD or the identity of the variance, `B` the summed covariance of `blocks` per unit of `h`); they are factor `column_scale` updates only when it drives part of the columns of the one block it reaches. All of them hold the allocation weights fixed and scale every driven column SD by the same factor (`held_fixed`, above), so the `column_scale` plan of an allocation-derived SD differs from that of a direct SD, which replaces the scale of its own columns only (one column, or every column in `driven_columns` for an SD shared by several columns). Their `coefficient_scale_transform` converts a selected variance to its square root and leaves a selected SD unchanged. These quantity-scale plans require the public SD to be on the fitted coefficient scale; after formula scaling that changes it, the plan is unsupported. Unsupported plans carry a structural reason. The accessor never estimates affineness from evaluated covariance matrices.

The invariant covariance of a block with several coefficient columns is a dense rows x rows matrix of the fitted data. Its estimated peak memory is checked against the `BayesTools.random_effects_memory_limit_bytes` option (see BayesTools) before it is formed; above the ceiling the accessor stops with an error of class `BayesTools_random_effects_memory_limit` (also `BayesTools_random_effects_error`).

The `conditional` entry of an unsupported plan is compiled only for a variance allocation with exactly two single-column block targets, a scalar total-variance source and inclusion gates on the blocks; an allocation with more than two targets has no conditional plan and stays unsupported.

Value

A list of class `BayesTools_random_effects_marginal_update_plan`. Field `family` is one of `"affine"`, `"factor"`, `"markov"`, or `"unsupported"`. An affine plan additionally records the scalar covariance coefficient transform and the affected random-effect blocks. For a plan in a source coordinate (`coefficient_input = "source"`), the blocks include every block whose SD binding uses that source, also through another variance allocation.

Fields shared by the supported families are `update` (the kind of update, below), `blocks`, `structure` and `component_index` (the covariance structure and the column of a per-column quantity), `coefficient_input`, `source_parameter`, `source_transform` and `source_transform_spec` (the fitted coordinate and its transform to the public quantity), `quantity_id`, `canonical_name`, `formula_parameter` and `dependencies`. Affine plans add `coefficient_transform` and, for the Dirichlet-weight updates of an allocation, `allocation` (`weight_name`, `index`, `n_targets`, and the allocation `scale` when the plan comes from the conditional route). `invariant_covariance` holds the covariance that does not depend on the selected coordinate when it is structurally available, and `NULL` otherwise. Factor plans of allocation-derived SDs add `coefficient_scale_transform`. An unsupported plan carries `reason` and `message` instead.

`update` is `"scale"` when the contribution of the affected blocks is scaled by one coefficient, `"column_scale"` when the SDs of some columns of a block change (`component_index`, the first column of the selected SD; the other column SDs of a direct per-column SD are held fixed, a direct SD shared by several columns lists them in `driven_columns`, and so do the driven columns of an allocation-derived

SD), "allocation" when a Dirichlet weight of a variance allocation changes, and "correlation" for a correlation coordinate. A direct per-column SD with unequal column SDs reports "column_scale".

The plan of an allocation-derived component SD or variance, and of an aggregate allocation SD or variance (sd_common, var_common, sd_total, var_total) that is a quantity-scale plan, has the field held_fixed = "allocation_weights": whatever the covariance structure, setting the selected SD to v holds the allocation weights and every non-allocation coordinate (correlations included) fixed and moves the allocation scale tau of the SD SD_k = tau * m_k(w) to v / m_k(w), where m_k(w) is the product of the square roots of the weights on the path from tau to the SD (sqrt(w_j) for a total-variance allocation, sqrt(K_j w_j) for a mean-variance one, over the nested allocations). This is the conditional of the variance prior declared on (tau, w), which are independent a priori. For an SD value v the prior of the moved coordinate is the tau prior evaluated at v / m_k(w) with the Jacobian 1 / m_k(w); for a variance value u the SD is sqrt(u) and the Jacobian is 1 / m_k(w)^2 on the variance parameterization of the scale (tau^2, equivalently 1 / (2 m_k(w) sqrt(u)) from tau). The scale reaches every block whose SD binding uses it, also through nested allocations (a split root with one <block>_components allocation per block) or a shared random_sd_source(): blocks lists all of them, the selected block first. Every column SD that derives from tau is multiplied by the same factor c = v / SD_k, so a reached block whose columns are all driven by tau has its covariance contribution multiplied by c^2 on every covariance structure: the plan is then the affine update = "scale" plan of the summed contribution of blocks. If tau drives only some columns of the selected block, which is the only block it reaches, the plan is the factor "column_scale" plan with driven_columns and the grid factor column_factor for them, when the selected column is among the driven ones (otherwise the plan is unsupported with the reason "selected_column_not_driven"); a scale that reaches several blocks and only part of their columns, or an aggregate whose scale drives only part of a block, has no exact scalar update and is unsupported with the reason "partial_scale_reach". An aggregate of a nested allocation moves the root scale of its allocation chain as well, so its blocks are those of every sibling allocation of the root, not only its own. Plans of direct per-column SDs have no held_fixed field.

See Also

parameter_catalog() random_effects_marginal_factor_states()

---

random_effects_marginal_variance_factors

Random-effect marginal variance factors

Description

Extracts row-aligned design multipliers for one-column formula random-effect blocks that are intended to be marginalized by a downstream likelihood. For a block with SD parameter τ, the marginal variance contribution is τ^2 times row_multiplier. With known group covariance this factor is built from the prepared group kernel, so the dense row-space covariance factor is ZKZ'.

Usage

```
random_effects_marginal_variance_factors(
  formula_design,
```

```
  blocks = NULL,
  require_diagonal = TRUE,
  require_one_to_one = FALSE
)
```

Arguments

formula_design	a BayesTools_formula_design object returned by JAGS_formula() or JAGS_formula_design().
blocks	optional character vector of random-effect block names. When NULL, only marginalized random-effect blocks are selected.
require_diagonal	whether to require the row-space covariance factor to be diagonal. This protects downstream likelihoods that can only consume row-wise variance multipliers.
require_one_to_one	whether each observation row must map to a distinct grouping level.

Details

Unsupported factor representations signal an error inheriting from BayesTools_random_effects_marginal_variance_una
The condition also inherits from BayesTools_random_effects_error, error, and condition, and carries message and call. The shared parent includes invalid allocation support errors and does not itself imply unavailability. The unavailability condition carries stable block_name and reason fields so downstream packages can distinguish structural unavailability from invalid inputs. reason is one of "row_indexed_sd", "multiple_columns", "non_diagonal_row_covariance", or "repeated_groups".

Value

A list of class BayesTools_random_effects_marginal_variance_factors with selected block metadata. Each block contains row_multiplier, group_map, group_levels, model_matrix, SD metadata, compile mode, and known group covariance metadata when present.

See Also

random_effects_marginal_vcov() random_effects_compile()

---

random_effects_marginal_vcov

Random-effect marginal variance-covariance samples

Description

Constructs posterior-draw marginal covariance matrices implied by BayesTools formula random effects. For each included random-effect block and posterior draw, the helper builds the observation-level contribution $Z_b G_b Z_b'$ and sums contributions across blocks.

Usage

```
random_effects_marginal_vcov(
  fit,
  parameter = NULL,
  data = NULL,
  posterior_samples = NULL,
  prior_list = NULL,
  blocks = NULL,
  new_levels = NULL,
  diagonal_only = FALSE,
  fitted_rows = NULL,
  ...
)
```

Arguments

fit	a fitted object returned by JAGS_fit(), an object carrying formula_design metadata, or a BayesTools_formula_design object.
parameter	formula parameter name. Required when fit stores more than one formula design.
data	optional data frame defining the observation rows. When NULL, the fitted formula design rows are used. New random-effect grouping levels require an explicit new_levels policy unless allowed by block metadata.
posterior_samples	optional posterior sample matrix, data frame, mcmc, or mcmc.list. When NULL, samples are extracted from fit.
prior_list	optional named prior list used to materialize fixed point hyperparameters. When NULL, priors are taken from fit or the formula design.
blocks	optional character vector of random-effect block names to include. The default includes all formula random-effect blocks.
new_levels	optional new-level policy. Use a random_new_levels() object or one of "error", "zero", or "sample". "zero" assigns zero covariance contribution to unseen grouping levels. "sample" treats unseen grouping levels as independent draws from the block covariance G. Grouping levels declared in the fitted grouping factor but without rows in the fitting data are unseen levels, except in blocks with a known group covariance; see prior_random().
diagonal_only	whether to return only observation-level marginal variances. This avoids allocating a draw x row x row array when callers need only diag(Z G Z'). CAR blocks still reconstruct a draw x column x column factor. Their admission estimate includes conservative factor and contraction work; diagonal output may not resolve a factor-dominated refusal. This estimate is not a measured peak or success guarantee: live inputs and uncollected garbage can require additional memory.
fitted_rows	optional integer vector mapping supplied prediction rows to fitted observation indices. It is required whenever data is supplied and a selected block uses a posterior-indexed row source. Reordering and duplicate indices are supported.

Callback-computed row sources do not use this mapping. The entries of rows of new random-effect levels with `new_levels = "zero"` are ignored, because those rows have zero covariance; they must still be valid fitted observation indices.

`...` reserved for future extensions. Unused arguments are rejected.

Details

Invalid supplied variance-allocation coordinates can signal `BayesTools_random_effect_allocation_out_of_support`, inheriting from `BayesTools_random_effects_error`, `error`, and `condition`, with `message` and `call` fields. This is an input support error, distinct from structural factor unavailability. `JAGS_bridgesampling()` converts this support leaf to `BayesTools_marglik_out_of_support` during likelihood reconstruction. Unavailable new rows for a posterior-indexed row source raise `BayesTools_prediction_new_rows_unavailable` (parent `BayesTools_prediction_unavailable`), with `reason = "posterior_indexed_row_source"`, `block`, displayed `source` label, affected prediction `rows`, and caller `context`. It applies where the source of the new rows is needed, under `new_levels = "sample"`; with `new_levels = "zero"` the new rows have zero covariance and need no source. Mapping, source value, and monitoring errors retain their existing conditions.

Value

A list of class `BayesTools_random_effects_marginal_vcov` with fields:

samples A dense array with dimensions `draw x row x row`, or a `draw x row` matrix when `diagonal_only = TRUE`.

metadata A list describing row ordering, included blocks, skipped blocks, structures, dimensions, and memory size. With `diagonal_only = TRUE`, it additionally identifies the `"diagonal"` variance representation, its sample dimensions, and the equivalent dense entry count. The default covariance metadata schema is unchanged.

See Also

`JAGS_formula_design()` `JAGS_fit()`

---

random_effects_source_roles

Structural source roles of formula random effects

Description

Classifies compiled random-effect terms as wholly independent estimate-level variation or contextual variation shared across observations. Classification uses the declared group map, coefficient structure, fixed design supports, and known group covariance, never draws or numerical factors.

Usage

```
random_effects_source_roles(random_effects, n_rows)
```

Arguments

`random_effects`	Compiled `random_effects` terms from a `BayesTools_formula_design`.
`n_rows`	Number of observation rows.

Details

A term with one observation per independent group remains estimate-level variation even with random slopes or multiple coefficients. Exact zeros in a compiled known group covariance can certify independence. For `id()` and `diag()`, disjoint coefficient supports can also certify an independent term. A contextual coefficient family remains contextual when some of its observed groups contain only one row. Terms mixing independent and contextual coefficient families have the role `"mixed"`. Integrating the complete term remains well defined; a downstream conditioning model must reject partial-term retention unless it explicitly supports it.

Roles do not depend on whether coefficients are sampled or marginalized, on their SDs, or on the realized covariance in a posterior row.

Value

A named character vector, keyed by block name, containing `"estimate"`, `"context"`, or `"mixed"` for every term.

See Also

`random_effects_dependency_matrix()`

`random_effects_summary_posterior`

Extract random-effect summary posterior distributions

Description

Extracts posterior draws for derived random-effect summary quantities with their canonical prior densities. For allocation summaries, the helper keeps raw Dirichlet allocation weights internal and exposes interpretable scalar summaries such as mean-variance SD-component variance multipliers. Gated total-variance proportions are normalized over active components and omit draws on which every component is excluded because a variance share is undefined there.

Each summary is the `parameter_mixed_posterior()` of its catalog quantity: its prior density is `parameter_prior_density()` (for gated allocations the mixed measure with its inclusion-gate atoms), and its posterior atoms are declared from its structure: none when the prior density has no point mass; the inclusion-gate atoms of gated total-variance proportions (at 0 and 1) and allocation totals (at 0), and the point components of a mixture scale prior whose indicator is monitored, with masses from the posterior inclusion- and mixture-indicator draws. Point masses without such a source leave the atom status undeclared, and posterior plots of such summaries stop instead of inferring point masses from the draws.

Usage

```
random_effects_summary_posterior(
  fit,
  summary = c("var_mult", "var_prop", "sd_mult", "sd_total", "var_total", "sd_common",
    "var_common"),
  allocation = NULL,
  component = NULL,
  formula_parameter = NULL,
  simplify_names = FALSE,
  n_prior_points = 4096
)
```

Arguments

fit	model fit created by JAGS_fit or a derived-draw view from JAGS_with_draws(). Summaries use its supplied draws and local analysis metadata.
summary	semantic quantity to extract: "var_mult", "var_prop", "sd_mult", "sd_total", "var_total", "sd_common", or "var_common". The default "var_mult" exists only for mean-variance SD-component allocations: for a fit with total-variance allocations it stops (request "var_prop" or a total instead).
allocation	optional allocation filter. It matches the stable allocation name given to random_variance_allocation() (the identifier the backend uses), not the public display_name.
component	optional allocation component label filter, matched against the component labels shown in the quantity names.
formula_parameter	optional formula parameter filter.
simplify_names	whether returned quantities should use centrally generated simplified display labels. Defaults to FALSE.
n_prior_points	grid size (n_grid of parameter_prior_density()) of the attached prior densities.

Value

A named list of posterior vectors with class mixed_posteriors and marginal_posterior. The list can be passed to plot_posterior, and the individual vectors carry their prior_density, support, and declared atoms as draw metadata (see posterior_metadata()).

---

random_group_component_names

Grouping variables and group keys of a compiled random-effect term

---

Description

random_group_component_names() returns the names of the grouping variables of a compiled random-effect term. random_group_prediction_keys() assigns each row of a data frame the key of its group, so that the rows of new data can be matched to the fitted groups of the term (or to groups that were not fitted) when random effects are predicted or marginalized.

Usage

```
random_group_component_names(term)

random_group_prediction_keys(term, data)
```

Arguments

term	a compiled random-effect term (a list), see the details.
data	a data frame with the grouping variables of term and one row per observation to be keyed.

Details

A compiled random-effect term is an element of the random_effects of a formula design: the design returned by JAGS_formula() (JAGS_formula(...)$formula_design$random_effects[[i]]) or the design stored with a fit or with draws (JAGS_formula_design(fit, parameter)$random_effects[[i]]). Pass the element unchanged; its fields are not part of the public interface.

An interaction (g1:g2) or nested (g1/g2) grouping expression has one grouping variable per component, returned in the order of the fitted groups.

The keys of random_group_prediction_keys() are character strings whose format is not part of the interface. Rows with the same grouping values share a key; a row whose grouping values identify a fitted group of the term has the key of that group, and a row with values that are not a fitted group has a key of its own that differs from the keys of all fitted groups. A grouping value is matched to the fitted levels by its exact value; a value that matches no fitted level but equals the label (or the displayed value) of exactly one fitted level takes that level, and a label shared by several fitted levels stops with an error. A term without its fitted grouping ownership (a design stored by an earlier version) stops with an error of class BayesTools_refit_required. A grouping variable that is missing in data or contains missing values stops with an error.

Value

random_group_component_names() returns a character vector of the names of the grouping variables of term.

random_group_prediction_keys() returns a character vector with one key per row of data: the key of the fitted group for rows in a fitted group of term, and a distinct key for rows in a new group.

See Also

JAGS_formula() JAGS_formula_design() random_effects_formula()

random_group_covariance

Known group covariance for random effects

Description

Creates a fixed group-level covariance kernel for use with random_effects_formula(). The kernel is a random-effect design property: it describes covariance across grouping levels and is separate from column-level random-effect covariance structures such as random_covariance(). The associated fitted block scale remains the public sd quantity (with var as its square). It multiplies the supplied kernel; it is not an sd_mult. Consequently, when an unscaled kernel has a non-unit diagonal, the fitted sd need not equal every grouping level's marginal standard deviation.

Before fitting, the kernel is restricted to the fitted grouping levels and transformed according to scale. Its positive definiteness must be resolvable at working precision. Validation uses correlation units and preserves the chosen scaled kernel; unresolved near-singular kernels are unavailable.

Usage

```
random_group_covariance(covariance, scale = c("cor", "none", "cor0", "cov0"))
```

Arguments

covariance	numeric square matrix with row and column names identifying grouping levels. Columns are aligned to the supplied row-name order before checking symmetry or completing a single supplied triangle. A partial triangle is interpreted in that row order: column permutations are allowed, but row permutations that scatter entries across both triangles can be refused. Complete symmetric matrices allow independent axis permutations.
scale	how to scale the kernel before fitting. "cor" converts the reordered matrix to a correlation matrix, "none" uses the matrix as supplied, "cor0" forms C = stats::cov2cor(K) and applies (C - min(C)) / (1 - min(C)), and "cov0" applies K - min(K).

Value

A random_group_covariance object.

See Also

random_effects_formula()

range.prior *Prior range*

Description

Computes range of a prior distribution (if the prior distribution is unbounded range from quantiles to 1 -quantiles) is returned.

Usage

```
## S3 method for class 'prior'
range(x, quantiles = NULL, ..., na.rm = FALSE)
```

Arguments

x	a prior
quantiles	quantile to be returned in case of unbounded distribution.
...	additional arguments
na.rm	unused

Value

range.prior returns a numeric vector of length with a plotting range of a prior distribution.

See Also

prior()

remove_column *Removes column to BayesTools table*

Description

Removes column to a BayesTools table while not breaking formatting, attributes, etc...

Usage

```
remove_column(table, column_position = NULL)
```

Arguments

table	BayesTools table
column_position	position of the to be removed column (defaults to NULL which removes the last column)

Value

returns an object of 'BayesTools_table' class.

`Savage_Dickey_BF` *Compute Savage-Dickey inclusion Bayes factors*

Description

Computes Savage-Dickey (density ratio) inclusion Bayes factors based on the change of height from prior to posterior distribution at the test value.

Usage

```
Savage_Dickey_BF(
  posterior,
  null_hypothesis = 0,
  normal_approximation = FALSE,
  silent = FALSE,
  density_method = c("KDE", "precomputed")
)
```

Arguments

`posterior`	marginal posterior distribution generated via the `marginal_posterior` function
`null_hypothesis`	point null hypothesis to test. Defaults to `0`
`normal_approximation`	whether the height of the posterior density at the null should be approximated via a normal distribution (rather than kernel density). The prior height is always the structural prior-density ordinate. Defaults to `FALSE`.
`silent`	whether warnings should be returned silently. Defaults to `FALSE`
`density_method`	density source for the posterior ordinate. `"KDE"` computes the Gaussian kernel density estimate at the null exactly (the kernel sum over the continuous draws, with bandwidth `stats::bw.nrd0()` of those draws; no evaluation grid or interpolation), using boundary reflection when exact posterior-support metadata is available. Finite sample ranges are not treated as exact support; Gaussian kernel tails are evaluated at finite null values. `"precomputed"` requires valid `posterior_ordinate` metadata when present, or otherwise valid `posterior_density` metadata (see `posterior_metadata()`). Posterior atom status must be declared through package-generated `atoms` metadata or `posterior_atom_attribute()`. An ordinary Savage-Dickey density ratio is rejected when atom status is unknown or when any positive atom is located exactly at the null. For a scalar marginal posterior, a positive atom at the null is an error; for a list of marginal posteriors (levels), such a level returns `NA` with the reason in its `"warnings"` attribute ("fixed at the null hypothesis value" for an atom of mass one, e.g., the reference level of a treatment-coded factor) and the other levels are computed.

Declared atoms at other locations are removed from the continuous posterior ordinate and the resulting density is scaled by the remaining continuous mass.

Details

Marginal posterior vectors may carry `posterior_ordinate` metadata with exact `value` and `ordinate` entries. When `density_method = "precomputed"` and `normal_approximation = FALSE`, a matching ordinate is used for the Savage-Dickey ratio. If no matching ordinate is available, a valid `posterior_density` grid with `x` and `y` coordinates is used. If neither valid precomputed source is available, an error is thrown. Exact support metadata is also checked before accepting precomputed posterior ordinates or densities; support is used to exclude the null only after it is validated against the posterior samples. A stale precomputed value is ignored when compatible exact support excludes the null; returned Bayes factors label this path as exact support exclusion rather than as a KDE estimate. Support is not inferred from prior-density grids, posterior-density grids, or plotting ranges, which may be finite numerical integration ranges rather than true support boundaries. Exact support metadata is created with `posterior_support_attribute()` (`exact = TRUE`); `type = "points"` support is not treated as a continuous interval for KDE boundary reflection. The same object may be supplied as `support` of `posterior_density` metadata. A prior density that is zero or infinite at the null makes the density ratio a 0/0 or singular limit that a kernel estimate of the posterior ordinate cannot estimate (e.g., with a log-singular prior the posterior carries the same singularity and the true ratio is a finite limit): a scalar call stops (see Value), and for a list of marginal posteriors (levels) such a level returns `NA` with the reason in its `"warnings"` attribute ("The prior density at the null hypothesis value is zero" or "infinite") and the other levels are computed.

Marginal posteriors created by `marginal_posterior()` record the mixture component of each draw and each component's exact support: the model for `mix_posteriors()` ensembles, and, for single fits with mixture or spike-and-slab priors (`as_mixed_posteriors()`), the combination of the component indicators of the terms entering the parameter or level. When continuous components have different exact supports (for example, a truncated prior in only some models or mixture components), the KDE posterior ordinate is estimated per component and mixed by the components' shares of the continuous draws: each component's ordinate uses its own support (zero when the support excludes the null, the one-sided limit on a support bound, as for the prior ordinate). The `"posterior_density_components"` attribute of the Bayes factor lists the components. When all continuous components share their support, the pooled KDE is used.

When the null hypothesis lies outside the continuous posterior draws (and is not an exact support bound), the KDE posterior density at the null is an extrapolation from Gaussian kernel tails. The finite Bayes factor is then returned with the warning that the density ratio is likely to be overestimated; for a model mixture, only the draws of models whose support contains the null count. Diagnostic messages are stored in the `"warnings"` attribute of each Bayes factor and, unless `silent = TRUE`, emitted once per parameter or level. The warnings of the conditions that are reported as `NA` (a posterior point mass at the null, a zero or infinite prior ordinate) are prefixed with the rendered label of the level, as in the rows of the marginal summary tables (for example `theta[A]` or, for a level of the formula parameter `mu`, `(mu) x[A]`); the warning table of `marginal_estimates_table()` prefixes the warnings that 0.3.0 reported with the fitted parameter instead.

The KDE posterior ordinate needs at least two finite continuous draws with positive spread (unless exact support excludes the null). With fewer, or with identical draws, a scalar call stops with `BayesTools_hypothesis_numerical_unavailable`, and a level of a list, and a parameter or level of marginal inference (a scalar marginal included), gets an `NA` Bayes factor with the reason. The same holds on the per-component route, for the pooled continuous draws and for each component

whose support contains the null and that has several identical draws (a component with a single draw uses the pooled bandwidth).

Affine expressions retain an unshifted linear numerator and a checked constant divisor. Point inference evaluates that numerator at the translated null; diagnostic density heights retain the requested quantity units. Numerical coefficient, boundary or log-ordinate resolution loss raises `BayesTools_hypothesis_numerical_` (ordinate parent) or `BayesTools_hypothesis_region_numerical_unavailable` (region parent), also `BayesTools_numerical_unavailable` and `BayesTools_numerical_condition`. The log-resolution screen is a conservative floating-point heuristic, not an estimator or backend error bound. Finite log values may correspond to displayed natural-scale zero or infinity. `raw_log_BF` and `numerical_diagnostics` retain aligned diagnostics; BF formatting uses the live column carrier, which replacement, arithmetic and Math operations invalidate.

Value

`Savage_Dickey_BF` returns a Bayes factor. The prior ordinate at the null follows the exactness rule of `hypothesis_BF()` point hypotheses, with the same condition classes: a prior point mass at the null value stops with an error of class `BayesTools_point_mass_at_null`, a zero, infinite or undefined ordinate with `BayesTools_zero_ordinate`, `BayesTools_infinite_ordinate` or `BayesTools_undefined_ordinate`, and an ordinate without an exact structural value (a prior-density combination evaluated only on a numerical grid, a quadrature rejected by its diagnostics, or a density grid without recorded provenance) with `BayesTools_inexact_ordinate`; each also has class `BayesTools_hypothesis_ordinate`. A declared posterior point mass at the null value of a scalar marginal posterior stops with an error of class `BayesTools_posterior_point_mass_at_null` (also `BayesTools_hypothesis_ordinate`). These conditions are errors of a direct scalar call only. For a list of marginal posteriors, and for the marginal posteriors of `marginal_inference()` and `as_marginal_inference()` (scalar marginals included), the Bayes factor of the affected parameter or level is `NA` instead, with the reason in its `"warnings"` attribute: for a posterior point mass at the null, for each of the five prior-ordinate conditions above (zero and infinite ordinates keep the reason text given in Details; the others, the message of their condition), and for a numerically unavailable posterior ordinate (`BayesTools_hypothesis_numerical_unavailable`, which the `"numerical_diagnostics"` attribute retains).

---

sd *Creates generic for sd function*

Description

Creates generic for sd function

Usage

```
sd(x, ...)
```

Arguments

`x`	main argument
`...`	additional arguments

Value

sd returns a standard deviation of the supplied object (if it is either a numeric vector or an object of class 'prior').

See Also

sd

`sd.prior` *Prior sd*

Description

Computes standard deviation of a prior distribution.

Usage

```
## S3 method for class 'prior'
sd(x, ...)
```

Arguments

x	a prior
...	unused arguments

Details

The standard deviation is the square root of `var()`; see `var()` for the precision check of truncated normal priors.

Value

a standard deviation of an object of class 'prior'.

See Also

`prior()`

Examples

```
# create a standard normal prior distribution
p1 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1))

# compute sd of the prior distribution
sd(p1)
```

`selection_backend_spec` *Compile selection priors for backend consumers*

Description

`selection_backend_spec()` compiles step-selection and p-hacking prior objects into active backend parameters. The returned object contains stable p/z geometry, JAGS prior/transform code, monitor names, initial values, and data constants. Set `include_init = FALSE` for deterministic metadata without drawing unused initial values.

Usage

```
selection_backend_spec(
  priors,
  backend = "jags",
  names = list(omega = "omega", alpha = "alpha"),
  global_breaks = NULL,
  include_init = TRUE
)
```

Arguments

`priors`	a selection prior, p-hacking prior, composed bias prior, `prior_none()`, `prior_mixture()`, or a list of those priors.
`backend`	backend target. Currently only `"jags"` is supported.
`names`	named list of backend parameter names for the public nodes (`omega`, `alpha`, `pi_null`, `beta_null`, `phack_kind`, `phack_z_source`, `phack_z_dest`); omitted entries keep their default names. The names are used in the generated JAGS code, monitors, and initial values of single-branch and mixture specifications alike, so they must be distinct valid JAGS node names that do not coincide with internal selection nodes. Custom `names` (and `global_breaks`) change only the syntax, monitors, and initial values that this function emits: the evaluators of the marginal-likelihood tools (`JAGS_marglik_parameters()`, `JAGS_marglik_priors()` and bridge sampling) read the default names and the grid that `JAGS_fit()` emits, so they do not evaluate the draws of a model emitted with customized names correctly (without a warning).
`global_breaks`	optional global p-value break grid. Like `names`, it changes only the emitted syntax; the evaluators of the marginal-likelihood tools read the default grid, so they do not evaluate the draws of a model emitted with a customized grid correctly.
`include_init`	whether to generate JAGS initial values. Defaults to `TRUE`; when `FALSE`, the returned `init` field is `NULL` and compilation does not change the random-number state.

Value

A list describing the compiled backend specification.

`selection_context_validate` *Selection context helpers*

Description

Validate and subset row-wise selection-kernel contexts and prepare small argument lists for native selected-normal backends.

Usage

```
selection_context_validate(context, n_samples = NULL, required = character())

selection_context_subset_rows(context, rows)

selection_context_subset_observations(context, idx)

selection_native_static_args(selection_spec)

selection_native_kernel_args(
  selection_spec,
  S,
  alpha = NULL,
  phack_kind = NULL,
  kernel_mode = NULL
)

selection_row_arg(x, n, name)
```

Arguments

`context`	selection context list.
`n_samples`	optional expected number of posterior/sample rows.
`required`	character vector of fields that must be present.
`rows`	row indices or logical row mask.
`idx`	observation indices.
`selection_spec`	selection backend specification or context list.
`S`	number of rows for row-wise native arguments.
`alpha`	optional p-hacking severity values.
`phack_kind`	optional p-hacking form indicators.
`kernel_mode`	optional selection-kernel mode indicators.

x	scalar or row-wise vector.
n	required row count.
name	argument name used in errors.

Details

Context validation checks row-wise posterior fields, observation fields, and compiled selection-bin metadata together. Custom row-wise fields may be listed in a character vector named row_fields; row subsetting rejects undeclared fields that otherwise look row-wise. The built-in vector_rule field accepts the integer codes 0 (product), 1 (best one-sided p-value), and 2 (best two-sided p-value), with one value or one value per posterior row. Native argument helpers accept either an augmented context or a bare selection_backend_spec() object, using compiled step, phacking, and data fallbacks where available. Spec-level kernel_mode is the union of branch kernels (a capability flag). Row routing uses branch_kernel_mode or an explicit per-row vector; the per-row kernel_mode (and phack_kind) of a context take precedence over the branch metadata, and an explicit argument over both. Branches with kernel mode 0 carry no selection kernel and never compete for a route, so a specification whose active branches all share one kernel routes on that kernel. When two or more *different* active kernels coexist, their bitwise union names a third kernel rather than a route, and passing it is an error unless bias_indicator can map each row to a branch. The built-in row fields, each of length one or one per posterior row, are omega (a numeric matrix of non-negative weights with one row per posterior row and one column per selection bin), alpha (the p-hacking severity, in [0, 1)), the integer codes phack_kind (0 to 2), kernel_mode (0 to 3), vector_rule (0 to 2), and bias_indicator (the index of the branch of each row, from 1 to the number of branches), and the logical use_normal. The observation fields share one length: obs_bin (the integer selection bin of each observation, from 1 to the number of bins), yi (finite effect sizes) and sei (positive standard errors). A fractional or missing integer code is an error, never rounded or truncated onto another code. Native caches are reused only when effective static geometry and complete row-routing inputs are unchanged. Explicit segments take precedence over p-value cuts; cuts are validated only when segments must be derived.

Value

selection_context_validate(), selection_context_subset_rows() and selection_context_subset_observatior return the context list with every field validated (integer codes as integer vectors, scalars expanded to one value per row) and its native cache reset by the subsetting functions; the native argument helpers return lists of prepared native arguments.

---

selection_event_support

Check Gaussian selection-event support

Description

Checks whether a Gaussian candidate law can reach a positive-weight event for almost every retained latent context. The complete candidate-plus-retained covariance must be positive definite. Retained covariance itself may be singular. This is a support check, not a numerical integration diagnostic.

Usage

```
selection_event_support(prior, candidate_basis)
```

Arguments

prior	a weightfunction prior specifying one publication group's rule.
candidate_basis	finite numeric matrix whose column space is exactly the candidate variation in that group. Zero columns are allowed, including a matrix with no columns. Use compiler-declared source geometry, not posterior covariance estimates or numerical rank truncation.

Details

Positive weights require no candidate variation. With hard-zero bins, the best-result rule needs at least one varying result. Two-sided product selection, and one-sided product selection with a positive least-significant weight, need every result to vary. One-sided product selection with a zero least-significant weight instead needs a strictly positive vector in the candidate column space.

The positive-direction check uses sign-definite columns, then base-R QR and optimization as candidate finders. A computed direction is accepted only when its positive entries exceed a standard dot-product roundoff bound. Failed searches return unknown rather than proving infeasibility.

The caller must check every candidate span having positive prior probability, including zero-SD and inclusion branches. When only a guaranteed subspace is supplied, only a positive result is conclusive; other results cannot classify the complete candidate law. This helper does not derive scale guarantees or change weights, priors, or source geometry.

Value

A list with `feasible` (`TRUE`, `FALSE`, or `NA`) and a stable character `reason`. `FALSE` proves that some positive-probability retained contexts have zero acceptance probability. `NA` means that a positive direction could not be certified. Reasons are `positive_weights`, `varying_best_result`, `varying_two_tails`, `positive_direction`, `deterministic_candidate`, `deterministic_candidate_rows`, `opposed_candidate_direction`, or `positive_direction_unavailable`.

---

selection_model *Specifies source conditioning in a selection model*

Description

`selection_model()` declares which sources a selection model conditions on or integrates out, and how estimate-level publication weights combine within a selection group. These are fixed model choices, not parameters with prior distributions.

Usage

```
selection_model(
  estimate_random_effects = "integrate",
  other_random_effects = "condition",
  known_sampling_variance = "integrate",
  weight_rule = "product",
  group = NULL
)

selection_model_spec(prior)

check_selection_model(model, name = "model")

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

Arguments

estimate_random_effects	either "integrate" (the default) or "condition" for the estimate-level random-effect term. In a random formula, this is the sole declared term whose grouping map has one distinct level per retained estimate. No such term is required; multiple qualifying terms are an error when the model binds its data.
other_random_effects	either "condition" (the default) or "integrate" for all remaining declared random-effect terms.
known_sampling_variance	either "integrate" (the default) or "condition" for the complete sampling-error vector with known covariance. This choice also applies to univariate models and independent sampling errors. Conditioning retains the full unknown realized error; integration averages its full distribution before selection normalization.
weight_rule	either "product" (the default), the product of estimate weights, or "best", the weight of the bin containing the smallest p-value within each publication group. The best-p-value rule does not select the largest publication weight.
group	publication-group column for weight_rule = "best", specified as a bare or backticked column name or a single character string. The default NULL requests automatic resolution by the consuming model; a best-rule branch requires resolved publication groups when its data are bound. Product selection is invariant to publication partitions, so under weight_rule = "product" no publication-group column is required or resolved and print() reports the group as unused. Supplying a group alongside the product rule is inert rather than an error, so that a caller can parameterize weight_rule and forward one group for both rules. References are captured without evaluating the column. Derived groups must first be stored as data columns.
prior	a single prior object or NULL. Mixtures must be examined one branch at a time.
model	a selection-model specification to validate.

name	argument name used in validation messages.
x	a selection-model specification.
...	additional arguments, currently unused.

Details

The three source choices are independent. Conditioning retains unknown latent effects during selection normalization; it does not make them observed or fixed. Integration averages the source before normalization. Sampling versus marginalizing a JAGS node is a separate computational choice. The consuming model binds source roles and, for the best-p-value rule, publication groups to its data, so a prior can be constructed before data exist. An ensemble needs publication groups only for its best-rule branches.

Publication groups describe how the best-p-value selection rule combines outcomes. They do not create random effects or define sampling covariance. Under the product rule, multiplying all estimate weights gives the same selection weight for every partition into groups. Its joint normalization therefore does not depend on `group`, including with correlated outcomes. Integration blocks instead follow dependencies from sampling covariance and random effects marked `"integrate"`; assigning singleton publication groups does not make correlated outcomes independent. A consuming model may impose additional covariance-support restrictions on best-rule publication groups.

Known sampling covariance V describes the complete error vector $e \sim N(0, V)$. With positive sampling standard deviations it can be written as $V = SRS$ and $e = Sz$, where S contains the standard deviations and $z \sim N(0, R)$. Conditioning retains all of e; there is no automatically integrated residual sampling component. Covariance factorizations are computational representations and do not define additional selection sources. P-values retain the original marginal sampling standard deviations under either choice.

If every nonzero source is conditioned on, the candidate result is deterministic. Positive selection weights then cancel in normalization, leaving the unselected distribution and no likelihood information about those weights. A retained context with zero acceptance probability has no normalized selected distribution and must not be silently discarded.

Estimate-level identification follows the declared grouping map, not numerical independence, coefficient supports, or posterior variances. A one-to-one term with correlated known group covariance remains estimate-level and retains that full covariance. Other terms are not split into coefficient families. See `random_effects_level_roles()` for the shared formula resolver.

Ordinary forwarding through wrapper arguments preserves the original column reference. Wrappers can also explicitly forward with `group = {{group}}`. To pass a column name stored in a variable, use `do.call(selection_model, list(group = column_name))`. A bare symbol always denotes a column, even if a vector with that name exists elsewhere.

Value

`selection_model()` returns an object of class `selection_model` containing only the four choices and a character column reference or `NULL`. `selection_model_spec()` returns the validated specification from a weightfunction prior or the selection child of a composed `prior_bias()`; it returns `NULL` for a prior without a selection component. `check_selection_model()` and `print()` invisibly return their validated input.

Examples

```
model <- selection_model(
  estimate_random_effects = "integrate",
  other_random_effects = "condition",
  known_sampling_variance = "integrate",
  weight_rule = "best",
  group = paper_id
)
p <- prior_weightfunction(steps = .05, model = model)
selection_model_spec(p)
```

---

selection_qmc_design *Fixed randomized quasi-Monte Carlo design*

Description

Constructs a deterministic shifted-Halton design in an explicitly supplied integration dimension. Random shifts are generated locally and do not alter R's random-number state. Returned points are strictly inside the open unit hypercube; a design that lands on a boundary is rejected rather than modified.

Usage

```
selection_qmc_design(dimensions, points, scrambles, seed = 1L)
```

Arguments

`dimensions`	positive integration dimension.
`points`	number of points per scramble.
`scrambles`	number of independently shifted designs.
`seed`	non-negative integer seed used only to construct the shifts.

Value

Numeric array with dimensions `scrambles`, `points`, and `dimensions`.

`simplify_as_mixed_posterior_bias` *Scalar PET, PEESE or omega posterior from a bias mixed posterior*

Description

Extracts the posterior of one component of a bias mixture from model-averaged posterior samples that hold the whole mixture as one `bias` element, so that the component can be handled like any other scalar parameter (for example plotted or summarized as a single parameter). The component keeps the model-averaged posterior over all bias branches: in the `prior_list` of the result, the branches without the component are replaced by a null prior (a point at zero for PET and PEESE, `prior_none()` for omega) that keeps their model weights.

Usage

```
simplify_as_mixed_posterior_bias(samples, parameter)
```

Arguments

`samples`	the output of `as_mixed_posteriors()` or `mix_posteriors()`: a list with an element `bias` of class `mixed_posteriors.bias`.
`parameter`	the component to extract: `"PET"`, `"PEESE"`, or `"omega"`.

Details

`samples` is the output of `as_mixed_posteriors()` or `mix_posteriors()` for a bias mixture (e.g., a mixture of `prior_PET()`, `prior_PEESE()`, `prior_weightfunction()` and `prior_none()` priors), whose `bias` element has the class `mixed_posteriors.bias`. For `"PET"` and `"PEESE"` the scalar column is the one of the bias element; a bias mixture without that column is represented by the point locations of its branches, which requires every active branch to be a point prior. The draw metadata (atoms, support, prior densities, quantities, and measure refusals) and the `prior_list` of the component are subset to it; the model weights of the branches are kept.

Value

`simplify_as_mixed_posterior_bias()` returns `samples` without the `bias` element and with the element `samples[[parameter]]`: the mixed posterior of the component (a single column for `"PET"` and `"PEESE"`, the omega columns for `"omega"`, the latter with the class `mixed_posteriors.weightfunction`) and its metadata. The other elements of `samples` are unchanged.

See Also

`as_mixed_posteriors()` `mix_posteriors()` `plot_posterior()`

transform_factor_samples

Transform factor posterior samples into differences from the mean

Description

Transforms posterior samples from model-averaged posterior distributions of factor priors: meandif and orthonormal samples into differences from the mean, and the samples of ordered factor priors from their contrast coefficients to the level scale.

Usage

```
transform_factor_samples(samples)
```

Arguments

samples	(a list) of mixed posterior distributions created with `mix_posteriors` function

Value

`transform_factor_samples` returns a named list of mixed posterior distributions (either a vector or a matrix).

See Also

mix_posteriors transform_meandif_samples transform_orthonormal_samples

transform_meandif_samples

Transform meandif posterior samples into differences from the mean

Description

Transforms posterior samples from model-averaged posterior distributions based on meandif prior distributions into differences from the mean.

Usage

```
transform_meandif_samples(samples)
```

Arguments

samples	(a list) of mixed posterior distributions created with `mix_posteriors` function

Value

`transform_meandif_samples` returns a named list of mixed posterior distributions (either a vector of matrix).

See Also

mix_posteriors contr.meandif

`transform_orthonormal_samples` *Transform orthonomal posterior samples into differences from the mean*

Description

Transforms posterior samples from model-averaged posterior distributions based on orthonormal prior distributions into differences from the mean.

Usage

```
transform_orthonormal_samples(samples)
```

Arguments

`samples`	(a list) of mixed posterior distributions created with `mix_posteriors` function

Value

`transform_orthonormal_samples` returns a named list of mixed posterior distributions (either a vector of matrix).

See Also

mix_posteriors contr.orthonormal

`transform_prior_samples` *Transform prior samples to original scale*

Description

Generate prior samples and transform them using the same matrix transformation as posterior samples. This is the correct approach for visualizing priors on the original (unscaled) scale, especially for the intercept which depends on contributions from multiple coefficient priors.

Usage

```
transform_prior_samples(
  fit,
  n_samples = 10000,
  seed = NULL,
  formula_scale = NULL
)
```

Arguments

`fit`	a fitted model object with `prior_list` and optionally `formula_scale` attributes
`n_samples`	number of samples to generate (default: 10000)
`seed`	random seed for reproducibility (optional). The caller's random-number state (`.Random.seed` and `RNGkind()`) is restored afterwards. With `NULL`, the caller's random-number stream is used.
`formula_scale`	optional nested list containing standardization information. If not provided, extracted from `fit` attribute. An entry must name every predictor that `fit` standardized for that parameter. The transformation is derived from the fitted structure stored in `fit`: the fixed-effect design, a log intercept, and the random-effect SD structure.

Details

When models use auto-scaling (standardizing predictors), the posterior samples are on the standardized scale. To correctly visualize priors on the original scale, we cannot simply apply a linear transformation to individual priors because the intercept on the original scale is a weighted sum of multiple priors:

$$\beta_0^{orig} = \beta_0^* - \sum_i \frac{\mu_i}{\sigma_i}\beta_i^*$$

This function generates samples from ALL priors simultaneously and applies the same matrix transformation used for posterior samples, which correctly handles the intercept and all other parameters.

Monitored random-effect nodes that the model defines deterministically from other nodes are computed from the prior draws of those nodes with the model's own definitions: standard deviations derived from a variance allocation (`random_variance_allocation()`), scalar correlations sampled

on the Fisher-z or logit scale, and LKJ Cholesky factors, correlation matrices, and partial correlations. Every parameter-catalog quantity that depends only on such nodes and on the priors can therefore be evaluated on the prior draws (see `parameter_draws()`). The auxiliary nodes of spike-and-slab and mixture priors are included from the components of their `rng()` draws: the component indicator (`<parameter>_indicator`) of both, and the inclusion probability (`<parameter>_inclusion`) and slab draws (`<parameter>_variable`) of spike-and-slab priors; the random-number stream of the other columns is unchanged. The totals of ordered priors (`<parameter>_ordered_total`) are included with the same auxiliary nodes of a spike-and-slab or mixture total; the slices of a spike-and-slab total of an interaction share its inclusion probability and indicator, as in the fitted model. Standardized latent random effects, nodes derived from them, such as realized group coefficients, the component nodes of mixture priors, and the auxiliary nodes of Dirichlet priors are not included, so catalog quantities that depend on them cannot be evaluated on the prior draws. Variance-allocation inclusion indicators are included, drawn from their inclusion probabilities. Valid LKJ concentrations do not guarantee representable primitive Beta draws. Nonfinite draws or draws outside the open interval `(0, 1)` stop before deterministic replay with `BayesTools_lkj_rng_unavailable` (parent `BayesTools_prior_rng_unavailable`). The condition records `block`, `primitive`, `K`, `eta`, original-row `failed_draws` and `n_failed`. Choose a concentration whose primitive draws remain representable; no retry, clamping, endpoint substitution or prior change is performed. Explicit scalar correlation transforms that numerically saturate a boundary retain their compiler refusal rather than changing the requested prior or scale.

Value

A matrix of prior samples on the original (unscaled) scale, with columns matching the structure of posterior samples. Columns that are not mapped by the scale transformation, such as the totals and allocation nodes of ordered priors, keep their fitted-scale values.

Conditions

The conditions of `transform_scale_samples()` apply (`BayesTools_formula_transform_unavailable` and `BayesTools_refit_required`). Expression-point priors without a certified persisted recipe stop with `BayesTools_formula_transform_unavailable` and reason `"uncertified_point_replay"`; non-representable LKJ primitive draws signal `BayesTools_lkj_rng_unavailable`.

See Also

`transform_scale_samples()` `plot_posterior()`

Examples

```
# With a fitted model that used formula_scale:
# prior_samples <- transform_prior_samples(fit, n_samples = 10000)
# This can then be used with density() or for custom plotting
```

`transform_scale_samples` *Transform standardized posterior samples back to original scale*

Description

Transforms posterior samples from standardized continuous predictors back to the original scale. This function is used when predictors were standardized during model fitting via the `formula_scale` parameter.

Usage

```
transform_scale_samples(fit, formula_scale = NULL)
```

Arguments

`fit`	a model fitted with `JAGS_fit()`. Matrices of posterior samples and other fit objects are not supported.
`formula_scale`	optional nested list containing standardization information keyed by parameter name, replacing the `formula_scale` attribute of `fit`. Each parameter entry contains scaling info (mean and sd) for each standardized predictor, e.g., `list(mu = list(mu_x1 = list(mean = 0, sd = 1)))`. An entry must name every predictor that `fit` standardized for that parameter. The transformation is derived from the fitted structure stored in `fit`: the fixed-effect design, a log intercept, and the random-effect SD structure.

Details

The function transforms regression coefficients and intercepts to account for predictor standardization, including interactions of any order.

Columns that are not coefficients of the fitted design or random-effect SDs and correlations of a scaled block, such as the totals and allocation nodes of ordered priors, keep their fitted-scale values.

Internal standardized random-effect coordinates (latent `xRE_Zx` columns and realized `xRE_COEFx` columns) are omitted from the returned matrix. They do not have the same transformation as fixed coefficients or random-effect covariance summaries and must not be presented as original-scale coefficients. This omission also applies when no scaling information is supplied.

For a k-way interaction between standardized predictors, the expansion of $\prod_i (x_i - \mu_i)/\sigma_i$ contributes to all lower-order terms. The contribution to a target term T from a source term S (where T is a subset of S's scaled components) is:

$$(-1)^{|extra|} \cdot \prod_{i \in extra} \mu_i / \prod_{i \in S_{scaled}} \sigma_i$$

where $extra = S_{scaled} \setminus T_{scaled}$. Terms with a `multiply_by` multiplier (see `JAGS_formula()`) additionally carry the ratio of the source to the target multiplier, m_S/m_T. The posterior must

contain every lower-order coefficient with a nonzero contribution from this expansion. If it does not, the function stops rather than returning an incomplete transformation.

For a fitted object, or formula-scale metadata created by `JAGS_formula()` or `JAGS_fit()`, the fixed-coefficient transformation is derived from the stored fixed-effect design (persisted factor levels and contrasts) and verified to reproduce the fitted linear predictor exactly, which also covers nested terms such as `~ f/x`. Formulas whose centered terms induce lower-order effects that the fitted formula does not contain (e.g., `~ x + x:f` with `x` standardized) have no original-scale representation, and the function stops.

The original-scale random-effect SDs and correlations of a block with a standardized slope are transformed the same way, from the fitted design of the block: when the standardized block columns equal the original-scale columns times a matrix A, the block coefficients transform as Au and their covariance as $A\Sigma A'$. A block whose original-scale design is not spanned by its own columns (a term that would have to be induced, such as the implied intercept of `(0 + x | g)`) has no such matrix. It keeps the pairing of coefficient names only when each retained contribution has identical fitted categorical coding and every omitted contribution is unrepresentable by the block's lower-order columns. Otherwise original-scale output is unavailable, with reason `"original_scale_not_representable"`; use fitted-scale output or include the needed lower-order terms in the block. A block that the fitted design does not hold (hand-built random-effect SD metadata) is transformed by the same pairing of names.

Value

A numeric matrix of posterior samples transformed back to the original predictor scale, with chains merged in their existing order. When no scaling information is supplied, the function returns the sample matrix without changing its values, rather than returning the fitted object.

Conditions

Invalid arguments, including a `formula_scale` entry for a formula parameter the model does not have (also for a model without formula parameters) or a standardized predictor that is missing from the posterior, stop with an ordinary error. A condition of class `BayesTools_formula_transform_unavailable` (parents `error` and `condition`; the `reason` field names the cause, for example `"incomplete_formula_scale"` (a `formula_scale` entry that omits a predictor standardized in the fitted model; the fields `parameter` and `predictors` name them), `"missing_fitted_design"`, `"original_scale_not_representable"`, `"original_scale_not_identified"`, `"homogeneous_scaled_block"` (hand-built metadata of a homogeneous-SD block with a scaled slope, which `JAGS_formula()` refuses), or `"nonfinite_transform"`) reports a transformation that is structurally or numerically unavailable. A condition of class `BayesTools_refit_required` reports a fit whose stored formula declarations are missing or disagree.

See Also

`JAGS_formula()` `JAGS_fit()`

`update.BayesTools_table` *Updates BayesTools table*

Description

Updates BayesTools table while not breaking formatting, attributes, etc...

Usage

```
## S3 method for class 'BayesTools_table'
update(
  object,
  title = NULL,
  footnotes = NULL,
  warnings = NULL,
  remove_parameters = NULL,
  logBF = FALSE,
  BF01 = FALSE,
  ...
)
```

Arguments

`object`	a BayesTools table
`title`	title of the table
`footnotes`	add footnotes to the table
`warnings`	add warnings of the table
`remove_parameters`	remove parameters from the table
`logBF`	whether to format Bayes factors as log(BF)
`BF01`	whether to format Bayes factors as 1/BF
`...`	additional arguments.

Value

returns an object of 'BayesTools_table' class.

var *Creates generic for var function*

Description

Creates generic for var function

Usage

```
var(x, ...)
```

Arguments

x	main argument
...	additional arguments

Value

var returns a variance of the supplied object (if it is either a numeric vector or an object of class 'prior').

See Also

cor

var.prior *Prior var*

Description

Computes variance of a prior distribution.

Usage

```
## S3 method for class 'prior'
var(x, ...)
```

Arguments

x	a prior
...	unused arguments

Details

The variance of a truncated normal prior is computed in closed form. Its terms cancel far in a tail and for narrow truncation intervals, so the variance is returned only when its estimated relative rounding error in double precision is at most 1e-6; otherwise the function stops with an error. For a one-sided truncation, the error is raised beyond about 33 prior standard deviations from the mean. The same applies to `sd()` and to spike-and-slab priors with such a slab. The variance can then be estimated from `rng()` draws of the prior, which sample the truncated distribution exactly, subject to Monte Carlo error. The variance of any other truncated prior is the integral of the squared deviation of its truncated quantile function from the mean over the probability interval (0, 1), non-negative by construction; it is refused with `BayesTools_numerical_unavailable` when the quantile function is unavailable or the integral's estimated error exceeds 1e-6 of the variance.

Value

a variance of an object of class 'prior'.

See Also

`prior()`

Examples

```
# create a standard normal prior distribution
p1 <- prior(distribution = "normal", parameters = list(mean = 1, sd = 1))

# compute variance of the prior distribution
var(p1)
```

---

weightfunctions *Weight functions*

Description

Marginal density, marginal distribution function, marginal quantile function and random generation for weight functions.

Usage

```
mdone.sided(x, alpha = NULL, alpha1 = NULL, alpha2 = NULL, log = FALSE)

mdtwo.sided(x, alpha, log = FALSE)

mdone.sided_fixed(x, omega, log = FALSE)

mdtwo.sided_fixed(x, omega, log = FALSE)
```

```
rone.sided(n, alpha = NULL, alpha1 = NULL, alpha2 = NULL)

rtwo.sided(n, alpha)

rone.sided_fixed(n, omega)

rtwo.sided_fixed(n, omega)

mpone.sided(
  q,
  alpha = NULL,
  alpha1 = NULL,
  alpha2 = NULL,
  lower.tail = TRUE,
  log.p = FALSE
)

mptwo.sided(q, alpha, lower.tail = TRUE, log.p = FALSE)

mpone.sided_fixed(q, omega, lower.tail = TRUE, log.p = FALSE)

mptwo.sided_fixed(q, omega, lower.tail = TRUE, log.p = FALSE)

mqone.sided(
  p,
  alpha = NULL,
  alpha1 = NULL,
  alpha2 = NULL,
  lower.tail = TRUE,
  log.p = FALSE
)

mqtwo.sided(p, alpha, lower.tail = TRUE, log.p = FALSE)

mqone.sided_fixed(p, omega, lower.tail = TRUE, log.p = FALSE)

mqtwo.sided_fixed(p, omega, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	numeric vector of quantiles; the parameter arguments (alpha, omega) may be matrices with one row per quantile.
alpha	vector or matrix with concentration parameters for the Dirichlet distribution for a monotonic one.sided or a two.sided weight function.
alpha1	vector or matrix with concentration parameters for the Dirichlet distribution for the expected direction of non-monotonic one.sided of weight function.
alpha2	vector or matrix with concentration parameters for the Dirichlet distribution for the unexpected direction of non-monotonic one.sided of weight function.

log, log.p	logical; if TRUE, probabilities p are given as log(p).
omega	vector or matrix of fixed non-negative relative weights for a one.sided or a two.sided weight function. The first/reference weight must be exactly 1.
n	number of observations.
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
p	vector of probabilities.

Details

The marginal density, distribution function, and quantile function are implemented for the monotonic parameterization (alpha) and for fixed weights (omega); the non-monotonic parameterization (alpha1, alpha2) supports random generation only. The same marginals of a prior_weightfunction() prior are available through mpdf(), mcdf(), mquant(), and rng().

The marginals of the monotonic parameterization are Beta distributions with the cumulative sums of alpha as shapes. Finite alpha whose cumulative sums exceed the largest double signal an error of class BayesTools_weightfunction_beta_shape_unavailable (parent BayesTools_weightfunction_unavailable) from the density, distribution, and quantile functions, and from mpdf(), mcdf(), and mquant() of a cumulative prior_weightfunction(). Degenerate Dirichlet draws of the random generation signal BayesTools_dirichlet_rng_unavailable.

Value

mdone.sided, mdtwo.sided, mdone.sided_fixed, and mdtwo.sided_fixed give the marginal density, mpone.sided, mptwo.sided, mpone.sided_fixed, and mptwo.sided_fixed give the marginal distribution function, mqone.sided, mqtwo.sided, mqone.sided_fixed, and mqtwo.sided_fixed give the marginal quantile function, and rone.sided, rtwo.sided, rone.sided_fixed, and rtwo.sided_fixed generate random deviates.

Examples

```
# draw samples from a two-sided weight function
rtwo.sided(10, alpha = c(1, 1))

# draw samples from a monotone one-sided weight function
rone.sided(10, alpha = c(1, 1, 1))

# draw samples from a non-monotone one-sided weight function
rone.sided(10, alpha1 = c(1, 1), alpha2 = c(1, 1))
```

---

weightfunctions_mapping

Create coefficient mapping between multiple weightfunctions

Description

Creates coefficients mapping between multiple weightfunctions.

Usage

```
weightfunctions_mapping(prior_list, cuts_only = FALSE, one_sided = FALSE)
```

Arguments

prior_list	list of prior distributions
cuts_only	whether only p-value cuts should be returned
one_sided	force one-sided output

Value

weightfunctions_mapping returns a list of indices mapping the publication weights omega from the individual weightfunctions into a joint weightfunction.

Index

∗ **datasets**
kitchen_rolls, 131
prior_informed_medicine_names, 211
∗ **package**
BayesTools, 9

add_column, 5
allocation_ref *(*prior_random*)*, 219
allocation_ref(), *221*
as.data.frame.BayesTools_hypothesis_BF *(*hypothesis_BF*)*, 46
as.mcmc.BayesTools_draws_view *(*JAGS_draw_geometry*)*, 93
as.mcmc.list.BayesTools_draws_view *(*JAGS_draw_geometry*)*, 93
as_marginal_inference, 6
as_mixed_posteriors, *8*, 8, *142*, *167*, *169*
as_mixed_posteriors(), *13*, *14*, *21*, *51*, *53*, *62*, *64*, *153*, *177*, *184*, *186*, *188*, *190*, *264*

BayesTools, 9, *243*
BayesTools-package *(*BayesTools*)*, 9
BayesTools_ensemble_tables, 10, *24*, *34*, *68*, *137*, *142*
BayesTools_load_JAGS_module, 15
BayesTools_model_tables, *14*, 15, *68*
Beta, *196*
BF_carrier_drop, 24
BF_carrier_drop(), *36*
bquote, *193*
bridge_sampler, *74*, *76–78*
bridgesampling_object, 25, *33*, *141*

Cauchy, *196*
ccdf *(*prior_functions_methods*)*, 208
ccdf.prior *(*prior_functions*)*, 205
cdf *(*prior_functions_methods*)*, 208
cdf.prior *(*prior_functions*)*, 205
check_bool *(*check_input*)*, 26
check_char *(*check_input*)*, 26
check_input, 26
check_int *(*check_input*)*, 26
check_list *(*check_input*)*, 26
check_real *(*check_input*)*, 26
check_selection_model *(*selection_model*)*, 260
compute_inference *(*ensemble_inference*)*, 32
compute_inference(), *67*
contr.BayesTools, 28
contr.independent *(*contr.BayesTools*)*, 28
contr.meandif, *266*
contr.meandif *(*contr.BayesTools*)*, 28
contr.ordered_cumulative *(*contr.BayesTools*)*, 28
contr.ordered_cumulative_levels *(*contr.BayesTools*)*, 28
contr.orthonormal, *266*
contr.orthonormal *(*contr.BayesTools*)*, 28
cor, *272*

density.prior, 29
density.prior(), *161*, *163*, *171*, *174*
difftime, *83*, *99*
dmpoint *(*mpoint*)*, 142
dpoint *(*point*)*, 176

ensemble_diagnostics_empty_table *(*BayesTools_ensemble_tables*)*, 10
ensemble_diagnostics_table *(*BayesTools_ensemble_tables*)*, 10
ensemble_estimates_empty_table *(*BayesTools_ensemble_tables*)*, 10
ensemble_estimates_table *(*BayesTools_ensemble_tables*)*, 10

ensemble_estimates_table(), *146*, *213*
ensemble_inference, *10*, *14*, 32, *68*, *137*, *142*, *167*
ensemble_inference(), *25*, *142*
ensemble_inference_empty_table (BayesTools_ensemble_tables), 10
ensemble_inference_table (BayesTools_ensemble_tables), 10
ensemble_summary_empty_table (BayesTools_ensemble_tables), 10
ensemble_summary_table (BayesTools_ensemble_tables), 10
Exponential, *196*

fit_backend_fingerprint, 34
format_BF, 35
format_BF(), *24*, *25*, *35*, *36*, *70*
format_BF_from_log, 35
format_BF_from_log(), *24*, *25*, *70*
format_parameter_names (parameter_names), 155
formula_add_intercept, 37

geom_prior, 37
geom_prior(), *133*, *161*, *162*
geom_prior_list, 39
geom_prior_list(), *135*, *168*, *172*, *175*

harrell_davis_quantile, 41
hypothesis_alias_label, 43
hypothesis_ast, 44
hypothesis_ast_schema (hypothesis_ast), 44
hypothesis_ast_schema(), *60*
hypothesis_BF, 46, *61*, *178*
hypothesis_BF(), *9*, *44*, *52–56*, *58*, *60*, *63*, *65*, *66*, *109*, *110*, *182*, *214*, *215*, *255*
hypothesis_BF_model, 52
hypothesis_BF_model(), *43*, *44*, *48*, *51*, *54*, *55*, *58*, *62*, *63*, *65–67*
hypothesis_cache_clear, 54
hypothesis_density_backend, 55
hypothesis_density_backend(), *51*, *52*, *54–56*, *65*, *182*
hypothesis_density_backends (hypothesis_density_backend), 55
hypothesis_density_backends(), *53*, *55*, *58*, *65*
hypothesis_linear_target, 59
hypothesis_linear_target(), *60*, *61*
hypothesis_linear_target_side, 60
hypothesis_normalize_level_references (hypothesis_parse_point_reference), 61
hypothesis_parse (hypothesis_ast), 44
hypothesis_parse(), *47*, *53*, *60*, *61*
hypothesis_parse_level_reference (hypothesis_parse_point_reference), 61
hypothesis_parse_point_reference, 61
hypothesis_provider, 62
hypothesis_provider(), *52*, *54*, *56*, *58*, *65*
hypothesis_quantities, 65
hypothesis_quantities(), *54*, *56*, *63*, *64*
hypothesis_render (hypothesis_ast), 44
hypothesis_resolve (hypothesis_ast), 44
hypothesis_resolve(), *53*
hypothesis_rewrite (hypothesis_ast), 44
hypothesis_stop, 66
hypothesis_symbols (hypothesis_ast), 44
hypothesis_unique_named_warnings (hypothesis_alias_label), 43

identical(), *152*
inclusion_BF, 67
inclusion_BF(), *34*
interpret, 68
interpret2 (interpret), 68
interpret_records, 69
interpret_tables (interpret_records), 69
invert_BF_bound_operator, 70
invert_BF_bound_operator(), *36*
InvGamma, *196*
is.prior, 71
is_prior_bias, 72
is_prior_phacking, 73

JAGS_add_priors, 73
JAGS_bridgesampling, 74
JAGS_bridgesampling(), *103*, *232*, *240*, *247*
JAGS_bridgesampling_posterior, 81
JAGS_check_and_list, 82

JAGS_check_and_list_autofit_settings (JAGS_check_and_list), 82
JAGS_check_and_list_fit_settings (JAGS_check_and_list), 82
JAGS_check_convergence, 84
JAGS_check_convergence(), *83*, *84*, *93*, *99*, *102*
JAGS_deterministic_evaluator (JAGS_deterministic_nodes), 88
JAGS_deterministic_nodes, 88
JAGS_deterministic_nodes(), *153*, *158*
JAGS_diagnostics, 90
JAGS_diagnostics_autocorrelation (JAGS_diagnostics), 90
JAGS_diagnostics_density (JAGS_diagnostics), 90
JAGS_diagnostics_trace (JAGS_diagnostics), 90
JAGS_draw_geometry, 93
JAGS_draw_geometry_schema (JAGS_draw_geometry), 93
JAGS_estimates_empty_table (BayesTools_model_tables), 15
JAGS_estimates_table (BayesTools_model_tables), 15
JAGS_estimates_table(), *48*, *51*, *53*, *155*
JAGS_evaluate_deterministic (JAGS_deterministic_nodes), 88
JAGS_evaluate_deterministic(), *149*
JAGS_evaluate_formula, 95
JAGS_evaluate_formula(), *90*, *103*, *113*, *115*, *127*, *128*
JAGS_extend (JAGS_fit), 97
JAGS_fit, *7*, *8*, *82*, 97, *171*, *249*
JAGS_fit(), *47*, *85*, *87*, *88*, *90*, *92–97*, *104*, *108*, *109*, *111–113*, *115–118*, *125–127*, *129*, *130*, *149*, *151*, *154*, *204*, *205*, *223*, *232*, *246*, *247*, *269*, *270*
JAGS_fit_contract, 103
JAGS_fit_contract_schema (JAGS_fit_contract), 103
JAGS_formula, 104
JAGS_formula(), *95*, *97*, *113*, *115–117*, *119*, *204*, *223*, *232*, *236*, *245*, *250*, *269*, *270*
JAGS_formula_coefficient_route, 109
JAGS_formula_coefficient_transform, 110
JAGS_formula_coefficient_transform(), *62*, *109*, *110*
JAGS_formula_coefficient_transform_schema (JAGS_formula_coefficient_transform), 110
JAGS_formula_coordinate_dependencies, 112
JAGS_formula_design, 112
JAGS_formula_design(), *112*, *115*, *117*, *118*, *245*, *247*, *250*
JAGS_formula_draws, 113
JAGS_formula_draws(), *95*, *96*, *127*
JAGS_formula_internal_coordinate_priors (JAGS_formula_coefficient_transform), 110
JAGS_formula_name_map (JAGS_fit_contract), 103
JAGS_formula_point_finalize, 115
JAGS_formula_predictor_basis, 117
JAGS_formula_prior_density (JAGS_formula_coefficient_transform), 110
JAGS_formula_prior_density(), *59*, *62*, *162*, *164*, *198*
JAGS_formula_random_marginal_covariance, 118
JAGS_formula_scale_finalize (JAGS_formula_point_finalize), 115
JAGS_get_inits, *83*, *100*, 119
JAGS_indexed_parameter_columns (parameter_names), 155
JAGS_indexed_parameter_matrix (parameter_names), 155
JAGS_indexed_parameter_vector (parameter_names), 155
JAGS_inference_empty_table (BayesTools_model_tables), 15
JAGS_inference_table (BayesTools_model_tables), 15
JAGS_lkj_corr_cholesky, 120
JAGS_marglik_parameters (JAGS_marglik_parameters_formula), 121
JAGS_marglik_parameters_formula, 121
JAGS_marglik_parameters_formula(), *80*
JAGS_marglik_priors, 122

JAGS_marglik_priors_formula (JAGS_marglik_priors), 122
JAGS_marglik_priors_rows (JAGS_marglik_priors), 122
JAGS_marglik_priors_rows_evaluator (JAGS_marglik_priors), 122
JAGS_materialize_draws (JAGS_draw_geometry), 93
JAGS_materialize_draws(), *144*
JAGS_ordered_density_kernel, 124
JAGS_ordered_parameter_spec, 125
JAGS_ordered_parameter_spec(), *148*
JAGS_package_builds, 126
JAGS_package_builds(), *129, 130*
JAGS_parameter_names (parameter_names), 155
JAGS_parameter_names(), *116*
JAGS_predict_formula, 127
JAGS_predict_formula(), *113*, *115*
JAGS_regex_escape (parameter_names), 155
JAGS_runtime_cluster, 129
JAGS_runtime_cluster(), *126, 127*
JAGS_runtime_cluster_stop (JAGS_runtime_cluster), 129
JAGS_summary_table (BayesTools_model_tables), 15
JAGS_to_monitor, 130
JAGS_validate_fit_contract (JAGS_fit_contract), 103
JAGS_validate_fit_contract(), *7, 8*, *75*, *85*, *95*, *102*, *121*, *138*
JAGS_with_draws (JAGS_draw_geometry), 93
JAGS_with_draws(), *19*, *92*, *102*, *249*

kitchen_rolls, 131

lines.prior, 131
lines.prior(), *39*, *161, 162*
lines_prior_list, 133
lines_prior_list(), *41*, *168*, *172*, *175*
LocationScaleT, *196*
Lognormal, *196*
lpdf (prior_functions_methods), 208
lpdf.prior (prior_functions), 205

marginal_estimates_table, *10*
marginal_estimates_table (BayesTools_ensemble_tables), 10
marginal_estimates_table(), *13*, *21*
marginal_inference, *8*, *10*, 136, *164*
marginal_inference(), *166*
marginal_posterior, 137
marginal_posterior(), *9*, *51*, *62*, *166*, *177*, *184*, *186–188*, *190, 191*, *198*
mccdf (prior_functions_methods), 208
mccdf.prior (prior_functions), 205
mcdf (prior_functions_methods), 208
mcdf(), *275*
mcdf.prior (prior_functions), 205
mdone.sided (weightfunctions), 273
mdone.sided_fixed (weightfunctions), 273
mdtwo.sided (weightfunctions), 273
mdtwo.sided_fixed (weightfunctions), 273
mean.prior, 139
mix_posteriors, *8–10*, *12*, *14*, *34*, *68*, *137*, 140, *167*, *169*, *265, 266*
mix_posteriors(), *13*, *21*, *25*, *171*, *184*, *186*, *188*, *190*, *264*
mlpdf (prior_functions_methods), 208
mlpdf.prior (prior_functions), 205
model_summary_empty_table (BayesTools_model_tables), 15
model_summary_table (BayesTools_model_tables), 15
models_inference, *10*, *14*
models_inference (ensemble_inference), 32
models_inference(), *25*
mpdf (prior_functions_methods), 208
mpdf(), *275*
mpdf.prior (prior_functions), 205
mpoint, 142
mpone.sided (weightfunctions), 273
mpone.sided_fixed (weightfunctions), 273
mptwo.sided (weightfunctions), 273
mptwo.sided_fixed (weightfunctions), 273
mqone.sided (weightfunctions), 273
mqone.sided_fixed (weightfunctions), 273
mqtwo.sided (weightfunctions), 273
mqtwo.sided_fixed (weightfunctions), 273
mquant (prior_functions_methods), 208
mquant(), *275*
mquant.prior (prior_functions), 205

Normal, *196*

parallel::makePSOCKcluster(), *130*

parameter_catalog, 143
parameter_catalog(), *14*, *24*, *51*, *62, 63*, *125*, *150, 151*, *153–155*, *188*, *213*, *225*, *235*, *244*
parameter_catalog_extend (parameter_catalog), 143
parameter_catalog_extend(), *63*
parameter_catalog_resolve (parameter_catalog), 143
parameter_catalog_resolve(), *22*, *44, 45*, *51*, *149, 150*, *154*, *242*
parameter_catalog_schema (parameter_catalog), 143
parameter_coordinates (parameter_map), 151
parameter_coordinates(), *84*, *86, 87*, *90*, *100*, *103*, *144*, *225*, *235*
parameter_coordinates_schema (parameter_map), 151
parameter_draws (parameter_catalog), 143
parameter_draws(), *12*, *103*, *186*, *268*
parameter_gate_states, 149
parameter_gate_states(), *153*
parameter_labels, 150
parameter_labels(), *14*, *24*, *59*, *126*, *154*, *188*, *213*
parameter_map, 151
parameter_map(), *112*, *118*, *143*, *152, 153*, *225*, *235*
parameter_map_cache, 152
parameter_map_schema (parameter_map), 151
parameter_mixed_posterior, 153
parameter_mixed_posterior(), *51*, *53*, *62*, *145, 146*, *148–150*, *171*, *177*, *190*, *221*, *248*
parameter_names, 155
parameter_prior_density (parameter_catalog), 143
parameter_prior_density(), *59*, *149*, *153*, *155*, *162*, *164*, *198*, *248, 249*
parameter_source, 157
parameter_transform (parameter_catalog), 143
parameter_transform_forward (parameter_catalog), 143
parameter_transform_inverse (parameter_catalog), 143
parameter_transform_jacobian (parameter_catalog), 143
pdf (prior_functions_methods), 208
pdf.prior (prior_functions), 205
phack_alpha_from_pi_null (phack_pi_null), 158
phack_backend_constants (phack_pi_null), 158
phack_pi_null, 158
plot.prior, 159
plot.prior(), *39*, *133*, *163, 164*, *196*, *218*, *231*
plot.prior_linear_density, 162
plot_marginal, 164
plot_marginal(), *171*
plot_models, 166
plot_posterior, *9*, 168, *249*
plot_posterior(), *9*, *31*, *166*, *188*, *264*, *268*
plot_prior_list, 172
plot_prior_list(), *41*, *135*, *176*
plot_transformed_prior, 175
pmpoint (mpoint), 142
point, 176
posterior_atom_attribute, 178
posterior_atom_attribute(), *165*, *171*, *180*, *186*
posterior_atoms_free, 177
posterior_atoms_free(), *155*
posterior_density_attribute, 178, *189*
posterior_density_attribute(), *178*, *186*, *190*
posterior_density_method_match, 181
posterior_density_method_uses_precomputed (posterior_density_method_match), 181
posterior_fitted_linear_weights (posterior_formula_measure_check), 181
posterior_formula_measure_check, 181
posterior_linear_weight_space (posterior_formula_measure_check), 181
posterior_log_ordinate_attribute, 183
posterior_log_ordinate_attribute(), *180*
posterior_measure_check, 184
posterior_measure_check(), *9*, *171*, *182*
posterior_metadata, 186

posterior_metadata(), *13, 14*, *21*, *47*, *57*, *59*, *93*, *139*, *142*, *145*, *150*, *153*, *155*, *165*, *171*, *177*, *180*, *182*, *184, 185*, *189–191*, *249*, *253*
posterior_metadata<- *(*posterior_metadata*)*, 186
posterior_ordinate_append *(*posterior_density_attribute*)*, 178
posterior_ordinate_append(), *57*, *186*
posterior_ordinate_attribute *(*posterior_density_attribute*)*, 178
posterior_ordinate_attribute(), *57*, *183, 184*, *186*
posterior_ordinate_has_value *(*posterior_density_attribute*)*, 178
posterior_ordinate_supports_bf *(*posterior_density_attribute*)*, 178
posterior_ordinate_supports_bf(), *58*
posterior_support_attribute, *179*, 189
posterior_support_attribute(), *186*, *254*
posterior_transform, 190
posterior_transform(), *13*, *21*, *138, 139*, *161*, *183*, *188*
ppoint *(*point*)*, 176
print.BayesTools_draws_view *(*JAGS_draw_geometry*)*, 93
print.BayesTools_table, 192
print.prior, 193
print.selection_model *(*selection_model*)*, 260
prior, 194
prior(), *32*, *111*, *140*, *162*, *166*, *168*, *172*, *175*, *193*, *204*, *210*, *212, 213*, *218*, *229*, *252*, *256*, *273*
prior_bias, 197
prior_bias(), *89*
prior_density_has_provenance, 197
prior_density_ordinate, 198
prior_density_ordinate(), *49, 50*, *110, 111*, *161*, *164*, *198*, *214, 215*
prior_factor, 201
prior_factor(), *106*, *204, 205*, *213*
prior_factor_levels, 204
prior_factor_levels(), *98*, *212, 213*
prior_functions, *195*, *203*, 205, *208*, *218*, *231*
prior_functions_methods, 208
prior_informed, 209
prior_informed(), *211*
prior_informed_medicine_names, *210*, 211
prior_lkj *(*prior_random*)*, 219
prior_mixture, *8*, 211
prior_none *(*prior*)*, 194
prior_none(), *93*, *166*, *171*, *264*
prior_ordered, 212
prior_ordered(), *12*, *20*, *92*, *144*, *151*, *204, 205*, *214*
prior_ordered_total_name, 214
prior_ordinate_status, 214
prior_ordinate_status(), *53, 54*, *198*
prior_PEESE *(*prior_PP*)*, 217
prior_PEESE(), *264*
prior_PET *(*prior_PP*)*, 217
prior_PET(), *264*
prior_phacking, 216
prior_phacking(), *89*
prior_PP, 217
prior_random, 219
prior_random(), *97*, *107*, *128*, *144*, *222*, *233*, *246*
prior_spike_and_slab, *8*, 229
prior_weightfunction, 230
prior_weightfunction(), *264*, *275*

qmpoint *(*mpoint*)*, 142
qpoint *(*point*)*, 176
quant *(*prior_functions_methods*)*, 208
quant.prior *(*prior_functions*)*, 205

random_block *(*prior_random*)*, 219
random_covariance *(*prior_random*)*, 219
random_covariance(), *251*
random_effects_compile, 231
random_effects_compile(), *245*
random_effects_correlation_draws, 232
random_effects_dependency_matrix, 234
random_effects_dependency_matrix(), *237*, *248*
random_effects_formula, 234
random_effects_formula(), *104*, *107*, *114*, *250, 251*
random_effects_level_roles, 236
random_effects_level_roles(), *262*

random_effects_marginal_diagonal_factor, 237
random_effects_marginal_diagonal_factor(), *241*
random_effects_marginal_factor_diagonal, 238
random_effects_marginal_factor_diagonal(), *241*
random_effects_marginal_factor_product, 239
random_effects_marginal_factor_product(), *238*, *241*
random_effects_marginal_factor_states, 239
random_effects_marginal_factor_states(), *237–239*, *242*, *244*
random_effects_marginal_factor_vcov, 240
random_effects_marginal_update_grid, 241
random_effects_marginal_update_plan, 242
random_effects_marginal_update_plan(), *241*, *242*
random_effects_marginal_variance_factors, 244
random_effects_marginal_vcov, 245
random_effects_marginal_vcov(), *128*, *233*, *234*, *240*, *245*
random_effects_source_roles, 247
random_effects_source_roles(), *236*
random_effects_summary_posterior, 248
random_effects_summary_posterior(), *186*
random_group_component_names, 249
random_group_covariance, 251
random_group_covariance(), *226*, *235*
random_group_prediction_keys *(*random_group_component_names*)*, 249
random_monitor *(*prior_random*)*, 219
random_new_levels *(*prior_random*)*, 219
random_sd_source *(*parameter_source*)*, 157
random_sd_source(), *244*
random_variance_allocation *(*prior_random*)*, 219
random_variance_allocation(), *88*, *267*
range.prior, 252
register_hypothesis_density_backend *(*hypothesis_density_backend*)*, 55
register_hypothesis_density_backend(), *55*
remove_column, 252
rjags-package, *74*
rmpoint *(*mpoint*)*, 142
rng *(*prior_functions_methods*)*, 208
rng(), *268*, *275*
rng.prior *(*prior_functions*)*, 205
rone.sided *(*weightfunctions*)*, 273
rone.sided_fixed *(*weightfunctions*)*, 273
rpoint *(*point*)*, 176
rtwo.sided *(*weightfunctions*)*, 273
rtwo.sided_fixed *(*weightfunctions*)*, 273
run.jags, *97*
runjags, *74*
runjags-package, *84*
runjags_estimates_empty_table *(*BayesTools_model_tables*)*, 15
runjags_estimates_table, *14*
runjags_estimates_table *(*BayesTools_model_tables*)*, 15
runjags_estimates_table(), *213*
runjags_inference_empty_table *(*BayesTools_model_tables*)*, 15
runjags_inference_table *(*BayesTools_model_tables*)*, 15

Savage_Dickey_BF, *46*, *48*, *165*, *178*, 253
Savage_Dickey_BF(), *183*, *188*, *214*
sd, 255, *256*
sd.prior, 256
selection_backend_spec, 257
selection_context_subset_observations *(*selection_context_validate*)*, 258
selection_context_subset_rows *(*selection_context_validate*)*, 258
selection_context_validate, 258
selection_event_support, 259
selection_model, *193*, 260
selection_model_spec *(*selection_model*)*, 260
selection_native_kernel_args *(*selection_context_validate*)*, 258

selection_native_static_args
(selection_context_validate), 258
selection_qmc_design, 263
selection_row_arg
(selection_context_validate), 258
simplify_as_mixed_posterior_bias, 264
stan_estimates_table
(BayesTools_model_tables), 15
stats::quantile(), *43*

transform_factor_samples, 265
transform_factor_samples(), *188*
transform_meandif_samples, *265*, 265
transform_orthonormal_samples, *265*, 266
transform_prior_samples, 267
transform_prior_samples(), *88*, *90*
transform_scale_samples, 269
transform_scale_samples(), *110*, *176*, *268*

unregister_hypothesis_density_backend
(hypothesis_density_backend), 55
unregister_hypothesis_density_backend(), *55*
update.BayesTools_table, 271
utils::modifyList(), *62*

var, 272
var(), *256*
var.prior, 272

weightfunctions, 273
weightfunctions_mapping, 275
wf_cumulative (prior_weightfunction), 230
wf_fixed (prior_weightfunction), 230
wf_fixed(), *93*
wf_independent (prior_weightfunction), 230