Package: SBC (via r-universe)

May 22, 2026

Title Simulation Based Calibration for Bayesian models

Version 0.5.0.0

Description SBC helps perform Simulation Based Calibration on Bayesian models. SBC lets you check for bugs in your model code and/or algorithm that fits the model. SBC focuses on models built with 'Stan' <https://mc-stan.org>, but can support other modelling languages as well.

License MIT + file LICENSE

URL https://hyunjimoon.github.io/SBC/, https://github.com/hyunjimoon/SBC/

BugReports https://github.com/hyunjimoon/SBC/issues

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports R6, posterior (>= 1.0.0), ggplot2, stats, stringi, abind, future.apply, dplyr, tidyr, tidyselect, purrr, memoise

Suggests testthat, cmdstanr (>= 0.4.0), rstan, knitr, rmarkdown, brms, MCMCpack, medicaldata, formula.tools, MASS, mvtnorm, patchwork, bridgesampling, bayesplot, reliabilitydiag, progressr

VignetteBuilder knitr

Language en-US

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-03-10 12:13:35 UTC

RemoteUrl https://github.com/hyunjimoon/SBC

RemoteRef v0.5.0

RemoteSha e40a320c335460851618f7019baf79d39bbedf29

Contents

`[.SBC_datasets` *Subset an* `SBC_datasets` *object.*

Description

Subset an `SBC_datasets` object.

Usage

```
## S3 method for class 'SBC_datasets'
x[indices]
```

[.SBC_results *Subset the results.*

Description

Subset the results.

Usage

```
## S3 method for class 'SBC_results'
x[indices]
```

Arguments

indices	integer indices or a binary vector of the same length as the number fits, selecting which fits to keep.

attribute_present *Get attribute presence for a vector of variable names.*

Description

Get attribute presence for a vector of variable names.

Usage

```
attribute_present(attribute, variable_names, var_attr)
```

Arguments

attribute	the attribute to test
variable_names	names of variable to test for the attribute
var_attr	an object defining the attributes, see variable-attributes.

Value

a logical vector the same length as variable_names

attribute_present_stats

Check an attribute in the `$stats` *element of the results for presence of an attribute.*

Description

Check an attribute in the `$stats` element of the results for presence of an attribute.

Usage

```
attribute_present_stats(attribute, x)
```

binary_calibration_from_bp

Binary prediction calibration computation and visualisation.

Description

Obtain estimate of binary prediction calibration and the associated uncertainty interval for further processing or plot it directly.

Usage

```
binary_calibration_from_bp(
  bp,
  type = c("reliabilitydiag", "calibrationband"),
  alpha = 0.05,
  ...,
  region.position = NULL
)

plot_binary_calibration_diff(
  x,
  type = c("reliabilitydiag", "calibrationband"),
  alpha = 0.05,
  ...,
  region.position = NULL,
  prob_histogram = TRUE
)

## S3 method for class 'SBC_results'
plot_binary_calibration_diff(res, ...)

## S3 method for class 'data.frame'
```

```
plot_binary_calibration_diff(
  bp,
  type = c("reliabilitydiag", "calibrationband"),
  alpha = 0.05,
  ...,
  region.position = NULL,
  prob_histogram = TRUE
)

plot_binary_calibration(
  x,
  type = c("reliabilitydiag", "calibrationband"),
  alpha = 0.05,
  ...,
  region.position = NULL,
  prob_histogram = TRUE
)

## S3 method for class 'SBC_results'
plot_binary_calibration(res, ...)

## S3 method for class 'data.frame'
plot_binary_calibration(
  bp,
  type = c("reliabilitydiag", "calibrationband"),
  ...,
  region.position = NULL,
  prob_histogram = TRUE
)
```

Arguments

bp	the binary probabilities — typically obtained with binary_probabilities_from_stats(). Can however be manually constructed, it needs to be a data.frame with columns variable, prob and simulated_value.
type	the type of calibration uncertainty bands to compute, see details.
alpha	the level associated with the confidence intervals reports
...	additional arguments passed to reliabilitydiag::reliabilitydiag() or calibrationband::calibr
region.position	for type ="reliabilitydiag" we may choose whether the uncertainty interval surrounds the estimate (region.position = "estimate") or the null distribution (region.position = "diagonal")
prob_histogram	Whether a histogram of the observed probabilities should be overlaid with the calibration curve.
res	An SBC_results object

Details

When type = "reliabilitydiag", the intervals are based on reliabilitydiag::reliabilitydiag() and depending on region.position can be centered on the null distribution of perfect calibration or on the estimated calibration. When type = "calibrationband" the intervals are around the estimated calibration using calibrationband::calibration_bands() — in our experience the calibrationband method has less sensitivity to detect miscalibration, but they require somewhat weaker assumptions.

Value

binary_calibration_from_bp returns a data.frame with columns variable, prob, estimate, low and high, for each variable, it contains an estimate + confidence interval across a range of probabilities (in equal steps). The plot_ methods return a ggplot2 object showing either the calibration curve or the difference between the calibration curve and perfect calibration (the diagonal)

---

binary_probabilities_from_stats

Extract binary probabilities from SBC stats.

Description

Takes all variables marked with binary_var_attribute() in the statistics (the $stats field of an SBC_results object) and transforms the reported CDF values into probabilities that the underlying binary variable is 1.

Usage

```
binary_probabilities_from_stats(stats)
```

Details

To support exact binary probabilities, a backend must implement SBC_posterior_cdf(), as exact probabilities cannot be derived from samples.

Value

A data.frame with at least the columns variable and prob.

bind_datasets *Combine multiple datasets together.*

Description

Combine multiple datasets together.

Usage

```
bind_datasets(...)
```

Arguments

...	datasets to bind

bind_derived_quantities *Combine two lists of derived quantities*

Description

Combine two lists of derived quantities

Usage

```
bind_derived_quantities(dq1, dq2)
```

bind_globals *Combine two sets globals for use in derived quantities or backend*

Description

Combine two sets globals for use in derived quantities or backend

Usage

```
bind_globals(globals1, globals2)
```

See Also

compute_SBC(), derived_quantities()

bind_results *Combine multiple SBC results together.*

Description

Primarily useful for iteratively adding more simulations to your SBC check.

Usage

```
bind_results(...)
```

Arguments

...	objects of type SBC_results to be combined.

Details

An example usage can be found in the small_model_workflow vignette.

brier_resampling_test *Binary calibration tests*

Description

Dimitriadis et al. propose several tests based on comparing actual predictions to predictions when the probabilities are calibrated. This yields several possible tests of correctly calibrated predictions (i.e. that the expected proportion of true values matches the predicted probability).

Usage

```
brier_resampling_test(x, y, alpha = 0.05, B = 10000)

brier_resampling_p(x, y, B = 10000)

binary_miscalibration(x, y)

miscalibration_resampling_p(x, y, B = 10000)

miscalibration_resampling_test(x, y, alpha = 0.05, B = 10000)
```

Arguments

x	the predicted success probabilities
y	the actual observed outcomes (just 0 or 1)
alpha	the type I error rate for the test
B	number of boostrap samples for the null distribution

Details

The brier_ functions represent a test based on brier score, while the miscalibration_ functions represent a test based on miscalibration. In both cases we evaluate the null distribution via bootstrapping.

Value

brier_resampling_test and miscalibration_resampling_test return an object of class htest, brier_resampling_p and miscalibration_resampling_p return just the p-value (for easier use with automated workflows). binary_miscalibration computes just the miscalibration component using the PAV (pool adjacent violators) algorithm.

References

T. Dimitriadis, T. Gneiting, & A.I. Jordan,

Stable reliability diagrams for probabilistic classifiers, Proc. Natl. Acad. Sci. U.S.A. 118 (8) e2016191118, https://doi.org/10.1073/pnas.2016191118 (2021).

---

cached_fit_filename *Provide a file where a cached result of a fit will be/was stored.*

Description

This is to allow you to directly access the cache, should you need to. For normal fits, the cache file will always be a list readable by readRDS and have element $fit for the actual fit. Other elements will contain the text output, messages and warnings emitted by the fit.

Usage

```
cached_fit_filename(cache_dir, backend, generated, suffix = "")
```

Details

This is also used internally to cache some extra data (notably bridgesampling results) using a non-empty suffix string.

calculate_prior_sd *Calculate prior standard deviation of a dataset*

Description

Calculate prior standard deviation of a dataset

Usage

```
calculate_prior_sd(datasets)
```

Value

a named vector of prior SDs

calculate_ranks_draws_matrix *Calculate ranks given variable values within a posterior distribution.*

Description

When there are ties (e.g. for discrete variables), the rank is currently drawn stochastically among the ties. Depending on `na_lowest`, NA is assumed to be equal to any value.

Usage

```
calculate_ranks_draws_matrix(variables, dm, params = NULL)
```

Arguments

`variables`	a vector of values to check
`dm`	draws_matrix of the fit (assumed to be already thinned if that was necessary)
`params`	DEPRECATED. Use `variables` instead.

`check_all_SBC_diagnostics` *Check diagnostics for an SBC results object.*

Description

Produces a message for each failed check.

Usage

```
check_all_SBC_diagnostics(results)
```

Value

TRUE if all checks are OK, FALSE otherwise.

`combine_all_variables` *Helper functions to be passed to ECDF-plots to combine variables in a single panel.*

Description

`combine_all_variables` will merge all variables in a single plot, while `combine_array_elements` will merge all elements of any array into a single panel of the plot

Usage

```
combine_all_variables(x)

combine_array_elements(x)
```

Arguments

`x`	parameter names

`combine_args` *Combine two named lists and overwrite elements with the same name using the value from args2*

Description

Combine two named lists and overwrite elements with the same name using the value from args2

Usage

```
combine_args(args1, args2)
```

combine_var_attributes *Combine multiple sets of variable attributes.*

Description

Combine multiple sets of variable attributes.

Usage

```
combine_var_attributes(...)
```

Arguments

...	the individual var_attributes() objects to combine.

Details

It is currently by design that multiple copies of an attribute are kept

compute_dquants *Compute derived quantities based on given data and posterior draws.*

Description

Compute derived quantities based on given data and posterior draws.

Usage

```
compute_dquants(draws, generated, dquants, gen_quants = NULL)
```

Arguments

gen_quants	Deprecated, use dquants

compute_SBC *Fit datasets and evaluate diagnostics and SBC metrics.*

Description

Performs the main SBC routine given datasets and a backend.

Usage

```
compute_SBC(
  datasets,
  backend,
  cores_per_fit = default_cores_per_fit(length(datasets)),
  keep_fits = TRUE,
  thin_ranks = SBC_backend_default_thin_ranks(backend),
  ensure_num_ranks_divisor = 2,
  chunk_size = default_chunk_size(length(datasets)),
  dquants = NULL,
  cache_mode = "none",
  cache_location = NULL,
  globals = list(),
  gen_quants = NULL
)
```

Arguments

datasets	an object of class SBC_datasets
backend	the model + sampling algorithm. The built-in backends can be constructed using SBC_backend_cmdstan_sample(), SBC_backend_cmdstan_variational(), SBC_backend_rstan_sam SBC_backend_rstan_optimizing() and SBC_backend_brms(). (more to come: issue 31, 38, 39). The backend is an S3 class supporting at least the SBC_fit(), SBC_fit_to_draws_matrix() methods.
cores_per_fit	how many cores should the backend be allowed to use for a single fit? Defaults to the maximum number that does not produce more parallel chains than you have cores. See default_cores_per_fit().
keep_fits	boolean, when FALSE full fits are discarded from memory - reduces memory consumption and increases speed (when processing in parallel), but prevents you from inspecting the fits and using recompute_SBC_statistics(). We recommend to set to TRUE in early phases of workflow, when you run just a few fits. Once the model is stable and you want to run a lot of iterations, we recommend setting to FALSE (even for quite a simple model, 1000 fits can easily exhaust 32GB of RAM).
thin_ranks	how much thinning should be applied to posterior draws before computing ranks for SBC. Should be large enough to avoid any noticeable autocorrelation of the thinned draws See details below.

ensure_num_ranks_divisor
: Potentially drop some posterior samples to ensure that this number divides the total number of SBC ranks (see Details).

chunk_size
: How many simulations within the `datasets` shall be processed in one batch by the same worker. Relevant only when using parallel processing. The larger the value, the smaller overhead there will be for parallel processing, but the work may be distributed less equally across workers. We recommend setting this high enough that a single batch takes at least several seconds, i.e. for small models, you can often reduce computation time noticeably by increasing this value. You can use `options(SBC.min_chunk_size = value)` to set a minimum chunk size globally. See documentation of `future.chunk.size` argument for `future.apply::future_lapply()` for more details.

dquants
: Derived quantities to include in SBC. Use `derived_quantities()` to construct them.

cache_mode
: Type of caching of results, currently the only supported modes are `"none"` (do not cache) and `"results"` where the whole results object is stored and recomputed only when the hash of the backend or dataset changes.

cache_location
: The filesystem location of cache. For `cache_mode = "results"` this should be a name of a single file. If the file name does not end with `.rds`, this extension is appended.

globals
: A list of names of objects that are defined in the global environment and need to present for the backend to work (if they are not already available in package). It is added to the `globals` argument to `future::future()`, to make those objects available on all workers.

gen_quants
: Deprecated, use dquants instead

Value

An object of class `SBC_results()`.

Paralellization

Parallel processing is supported via the `future` package, for most uses, it is most sensible to just call `plan(multisession)` once in your R session and all cores your computer will be used. For more details refer to the documentation of the `future` package. On some Linux systems, using future.mirai::mirai_multisession may give performance boosts.

Progress reporting

This function supports progress reporting with the `progressr` package. Run `progressr::handlers(global = TRUE)` to enable progress reporting globally, see the `progressr` docs for more options.

Thinning

When using backends based on MCMC, there are two possible moments when draws may need to be thinned. They can be thinned directly within the backend and they may be thinned only to compute the ranks for SBC as specified by the `thin_ranks` argument. The main reason those are separate is that computing the ranks requires no or negligible autocorrelation while some autocorrelation may

be easily tolerated for summarising the fit results or assessing convergence. In fact, thinning too aggressively in the backend may lead to overly noisy estimates of posterior means, quantiles and the `posterior::rhat()` and `posterior::ess_tail()` diagnostics. So for well-adapted Hamiltonian Monte-Carlo chains (e.g. Stan-based backends), we recommend no thinning in the backend and even value of `thin_ranks` between 6 and 10 is usually sufficient to remove the residual autocorrelation. For a backend based on Metropolis-Hastings, it might be sensible to thin quite aggressively already in the backend and then have some additional thinning via `thin_ranks`.

Backends that don't require thining should implement `SBC_backend_iid_draws()` or `SBC_backend_default_thin_ranks(` to avoid thinning by default.

Rank divisors

Some of the visualizations and post processing steps we use in the SBC package (e.g. `plot_rank_hist()`, `empirical_coverage()`) work best if the total number of possible SBC ranks is a "nice" number (lots of divisors). However, the number of ranks is one plus the number of posterior samples after thinning - therefore as long as the number of samples is a "nice" number, the number of ranks usually will not be. To remedy this, you can specify `ensure_num_ranks_divisor` - the method will drop at most `ensure_num_ranks_divisor - 1` samples to make the number of ranks divisible by `ensure_num_ranks_divisor`. The default 2 prevents the most annoying pathologies while discarding at most a single sample.

`data_for_ecdf_plots` *Maybe not export in the end? Useful for debugging*

Description

Maybe not export in the end? Useful for debugging

Usage

```
data_for_ecdf_plots(x, ..., prob = 0.95, gamma = NULL, K = NULL)
```

`default_chunk_size` *Determines the default chunk size.*

Description

By default will make every worker process a single chunk. You can set the `options(SBC.min_chunk_size = value)` to enforce a minimum chunk size globally (chunk size can still be larger if you have substantially more fits to run than workers.

Usage

```
default_chunk_size(n_fits, n_workers = future::nbrOfWorkers())
```

default_cores_per_fit *Determines the default cores per single fit.*

Description

When parallel processing is disabled, this just returns the number of available cores. Otherwise, it chooses the largest integer that keeps `cores_per_fit * (n_fits / chunk_size) <= total_cores`, i.e. it avoids running so many chains in parallel that there will be more chains than cores.

Usage

```
default_cores_per_fit(
  n_fits,
  total_cores = future::availableCores(),
  chunk_size = default_chunk_size(n_fits)
)
```

derived_quantities *Create a definition of derived quantities evaluated in R.*

Description

When the expression contains non-library functions/objects, and parallel processing is enabled, those must be named in the `.globals` parameter (hopefully we'll be able to detect those automatically in the future). Note that `recompute_SBC_statistics()` currently does not use parallel processing, so `.globals` don't need to be set.

Usage

```
derived_quantities(..., .var_attributes = NULL, .globals = list())
```

Arguments

`...`	named expressions representing the quantitites
`.var_attributes`	a `var_attributes()` object providing attributes for the derived quantities, if necessary
`.globals`	A list of names of objects that are defined in the global environment and need to present for the gen. quants. to evaluate. It is added to the `globals` argument to `future::future()`, to make those objects available on all workers.

Examples

```
# Derived quantity computing the total log likelihood of a normal distribution
# with known sd = 1
normal_lpdf <- function(y, mu, sigma) {
 sum(dnorm(y, mean = mu, sd = sigma, log = TRUE))
}

# Note the use of .globals to make the normal_lpdf function available
# within the expression
log_lik_dq <- derived_quantities(log_lik = normal_lpdf(y, mu, 1),
                                 .globals = "normal_lpdf" )
```

`dquants_var_attributes` *Get the* `var_attributes()` *object associated with the derived quantities*

Description

Get the `var_attributes()` object associated with the derived quantities

Usage

```
dquants_var_attributes(dquants)
```

`empirical_coverage` *Compute observed coverage of posterior credible intervals.*

Description

Uses ranks to compute coverage and surrounding uncertainty of posterior credible intervals. The uncertainty is only approximate (treating coverage for each interval as a set of independent Bernoulli trials, while in fact they are not independent), so for making claims on presence/ absence of detectable discrepancies we strongly recommend using `plot_ecdf()` or `plot_ecdf_diff()`. The uncertainty about the coverage can however be useful for guiding decisions on whether more SBC steps should be performed (i.e. whether we can rule out that the coverage of the given backend differs too much for our purposes from the optimal value).

Usage

```
empirical_coverage(stats, width, prob = 0.95, interval_type = "central")
```

Arguments

`stats`	a data.frame of rank statistics (e.g. as returned in the `$stats` component of SBC_results), at minimum should have at least `variable`, `rank` and `max_rank` columns)
`width`	a vector of values between 0 and 1 representing widths of credible intervals for which we compute coverage.
`prob`	determines width of the uncertainty interval around the observed coverage
`inteval_type`	`"central"` to show coverage of central credible intervals or `"leftmost"` to show coverage of leftmost credible intervals (i.e. the observed CDF).

Details

Note that while coverage of central posterior intervals (with the default `type = "central"`) is often of the biggest practical interest, perfect calibration of central intervals still leaves space for substantial problems with the model (e.g. if the posterior 25% - 50% intervals contain 50% of the true values and the posterior 50% - 75% interval never contains the true value, the central 50% interval still has the ideal 50% coverage), so investigating central intervals should always be accompanied by checks with `plot_ecdf()` or `plot_ecdf_diff()` or by using `type = "leftmost"`, because if all leftmost credible intervals are well calibrated, then all intervals are well calibrated.

Value

A `data.frame` with columns `variable`, `width` (width of the interval as given in the `width` parameter), `width_represented` the closest width that can be represented by the ranks in the input (any discrepancy needs to be judged against this rather than `width`), `estimate` - observed coverage for the interval, `ci_low`, `ci_high` the uncertainty interval around `estimate` (width of the interval is given by the `prob` argument).

See Also

`plot_coverage()`

`extract_attribute_arguments_stats` *Extract arguments of a given attribute from the* `$stats` *element of the results*

Description

Extract arguments of a given attribute from the `$stats` element of the results

Usage

```
extract_attribute_arguments_stats(attribute_name, x)
```

gaffke_ci *Non-parametric test for the mean of a bounded variable.*

Description

Test a null hypothesis about the mean of i.i.d. samples. The test is based on Gaffke 2005, though a more detailed analysis and exposition can be found in Learned-Miller & Thomas 2020.

Usage

```
gaffke_ci(probs, B = 10000, alpha = 0.05)

gaffke_p(
  probs,
  mu = 0.5,
  alpha = 0.05,
  B = 10000,
  alternative = c("two.sided", "less", "greater")
)

gaffke_test(
  x,
  mu = 0.5,
  alpha = 0.05,
  lb = 0,
  ub = 1,
  B = 10000,
  alternative = c("two.sided", "less", "greater")
)
```

Arguments

B	number of bootstrap samples for the null distribution
alpha	the level of the test
mu	the mean under null hypothesis
alternative	the alternative for the test.
x	a vector of observed values
lb	the lower bound for x
ub	the upper bound for x

Details

The test is expected to be valid for any bounded distribution without further assumptions. The test has been proven valid only for special cases but no counterexample is known despite some efforts in the literature to find some. The test also provides way more power (and tighter confidence intervals)

than other non-parametric tests for bounded means - in fact its power converges quite quickly to that of the t-test, which is known to be optimal in the large data limit.

In SBC the test is useful for testing the data-averaged posterior for binary variables.

Value

`gaffke_test` returns an object of class `htest`, `gaffke_p` and `gaffke_ci` return just the p-value / CI as numeric for easier use in batch workflows.

References

Gaffke, N. (2005). "Three test statistics for a nonparametric one-sided hypothesis on the mean of a nonnegative variable." Mathematical Methods of Statistics, 14(4): 451–467.

Learned-Miller, E. and Thomas, P. S. (2020). "A New Confidence Interval for the Mean of a Bounded Random Variable." https://arxiv.org/abs/1905.06208

---

generate_datasets *Generate datasets.*

Description

Generate datasets.

Usage

```
generate_datasets(generator, n_sims, n_datasets = NULL)
```

Arguments

`generator`	a generator object - build e.g. via `SBC_generator_function` or `SBC_generator_brms`.
`n_sims`	the number of simulated datasets to use
`n_datasets`	DEPRECATED, use `n_sims` instead.

Value

object of class `SBC_datasets`

---

get_stats_for_submodel *Extract stats for a submodel from a Bayes factor results*

Description

Extract stats for a submodel from a Bayes factor results

Usage

```
get_stats_for_submodel(stats, submodel_id)
```

guess_rank_hist_bins *Guess the number of bins for* plot_rank_hist().

Description

Guess the number of bins for plot_rank_hist().

Usage

```
guess_rank_hist_bins(max_rank, N)
```

Arguments

max_rank	the maximum rank observed
N	the number of ranks observed

plot_contraction *Prior/posterior contraction plot.*

Description

The rationale for this plot and its interpretation is explained in Mike Betancourt's Towards A Principled Bayesian Workflow.

Usage

```
plot_contraction(
  x,
  prior_sd,
  variables = NULL,
  scale = "sd",
  alpha = 0.8,
  ...,
  parameters = NULL
)

## S3 method for class 'data.frame'
plot_contraction(
  x,
  prior_sd,
  variables = NULL,
  scale = "sd",
  alpha = 0.8,
  show_hidden = FALSE,
  parameters = NULL
)
```

Arguments

x	object containing results (a data.frame or SBC_results() object).
prior_sd	a named vector of prior standard deviations for your variables. Either pass in analytically obtained values or use calculate_prior_sd() to get an empirical estimate from an SBC_datasets object.
variables	variables to show in the plot or NULL to show all must correspond a field already computed in the results (most likely "mean" and "median").
scale	which scale of variability you want to see - either "sd" for standard deviation or "var" for variance.
alpha	the alpha for the points
parameters	DEPRECATED, use variables instead.
show_hidden	Show variables marked with hidden_var_attribute() (by default, those are not shown, available only when x is a data.frame)

Value

a ggplot2 plot object

---

plot_coverage *Plot the observed coverage and its uncertainty.*

Description

plot_coverage will plot the observed coverage, while plot_coverage_diff will show the difference between observed and expected coverage. Please refer to empirical_coverage() for details on computation and limitations of this plot as well as details on the arguments. See vignette("rank_visualizations") for more details.

Usage

```
plot_coverage(
  x,
  variables = NULL,
  prob = 0.95,
  interval_type = "central",
  ...,
  parameters = NULL,
  max_points = NULL
)

plot_coverage_diff(
  x,
  variables = NULL,
  prob = 0.95,
  interval_type = "central",
```

```
  ...,
  parameters = NULL,
  max_points = NULL
)
```

Arguments

x	object containing results (a data.frame or SBC_results() object).
variables	variables to show in the plot or NULL to show all
prob	the with of the uncertainty interval to be shown
parameters	DEPRECATED. Use variables instead.
max_points	maximum number of points where to evaluate the coverage. If set to NULL, coverage is evaluated across the whole range of ranks. Setting to some smaller number may reduce memory footprint and increase speed.

Value

a ggplot2 plot object

See Also

empirical_coverage

---

plot_ecdf *Plot the ECDF-based plots.*

Description

See vignette("rank_visualizations") for more details. See the methods for data_for_ecdf_plots() for available data formats.

Usage

```
plot_ecdf(
  x,
  variables = NULL,
  K = NULL,
  gamma = NULL,
  prob = 0.95,
  size = 1,
  alpha = 0.33,
  combine_variables = NULL,
  ecdf_alpha = NULL,
  show_hidden = FALSE,
  facet_args = list(),
  ...,
```

```
  parameters = NULL
)

plot_ecdf_diff(
  x,
  variables = NULL,
  K = NULL,
  gamma = NULL,
  prob = 0.95,
  size = 1,
  alpha = 0.33,
  combine_variables = NULL,
  ecdf_alpha = NULL,
  show_hidden = FALSE,
  facet_args = list(),
  ...,
  parameters = NULL
)
```

Arguments

x	object supporting the data_for_ecdf_plots() method.
variables	optional subset of variables to show in the plot
K	number of uniformly spaced evaluation points for the ECDF or ECDFs. Affects the granularity of the plot and can significantly speed up the computation of the simultaneous confidence bands. Default value is chosen heuristically. You can also use "max" to represent the number of ranks or "min" to choose a lower but still sensible value.
gamma	TODO
prob	the width of the plotted confidence interval for the ECDF.
size	size (linewidth) passed to ggplot2::geom_ribbon() for the confidence band
alpha	alpha level of the confidence band
combine_variables	optionally specify a named list where each entry is a character vectors which specifies a group of variables that will be displayed in the same panel. Panel title will be the name of the list element. A function that takes a character vector as an input and produces a list can also be specified (see combine-functions).
ecdf_alpha	the alpha level of the empirical CDF. Can be either a single number or a function taking the number of variables that were combined (when combine_variables is specified) and returns a number. By default, plots showing many ECDFs will have reduced alpha.
show_hidden	Show variables marked with hidden_var_attribute() (by default, those are not shown, available only when x is a data.frame)
facet_args	extra arguments for the call to ggplot2::facet_wrap() when rendering the plot.

...	additional arguments passed to data_for_ecdf_plots(). Most notably, if x is matrix, a max_rank parameter needs to be given.
parameters	DEPRECATED, use variables instead.

Details

arxiv::1903.08008 by A. Vehtari et al.

See Also

plot_coverage()

---

plot_rank_hist *Plot rank histogram of an SBC results.*

Description

The expected uniform distribution and an approximate confidence interval is also shown. The confidence interval cannot be taken too seriously as it is derived assuming the bins are independent (which they are not). The plot_ecdf() and plot_ecdf_diff() plots provide better confidence interval but are somewhat less interpretable. See vignette("rank_visualizations") for more details.

Usage

```
plot_rank_hist(
  x,
  variables = NULL,
  bins = NULL,
  prob = 0.95,
  ...,
  parameters = NULL
)

## S3 method for class 'data.frame'
plot_rank_hist(
  x,
  variables = NULL,
  bins = NULL,
  prob = 0.95,
  max_rank = x$max_rank,
  show_hidden = FALSE,
  parameters = NULL,
  facet_args = list()
)
```

Arguments

x	Object supporting the plotting method.
variables	Names of variables to show
bins	number of bins to be used in the histogram, if left unspecified, it is determined by guess_rank_hist_bins().
prob	The width of the approximate confidence interval shown.
show_hidden	Show variables marked with hidden_var_attribute() (by default, those are not shown)
facet_args	extra arguments for the call to ggplot2::facet_wrap() when rendering the plot.

Details

By default the support is for SBC_results objects and data frames in the same format as the $stats element of SBC_results.

---

plot_sim_estimated *Plot the simulated "true" values versus posterior estimates*

Description

Plot the simulated "true" values versus posterior estimates

Usage

```
plot_sim_estimated(
  x,
  variables = NULL,
  estimate = "mean",
  uncertainty = c("q5", "q95"),
  alpha = NULL,
  ...,
  parameters = NULL
)

## S3 method for class 'data.frame'
plot_sim_estimated(
  x,
  variables = NULL,
  estimate = "mean",
  uncertainty = c("q5", "q95"),
  alpha = NULL,
  show_hidden = FALSE,
  parameters = NULL
)
```

Arguments

x	object containing results (a data.frame or SBC_results() object).
variables	variables to show in the plot or NULL to show all
estimate	which estimate to use for the central tendency, must correspond a field already computed in the results (most likely "mean" and "median").
uncertainty	which estimates to use for uncertainty (a character vector of length 2) must correspond a field already computed in the results. Pass NULL to avoid showing uncertainty at all.
alpha	the alpha for the points and uncertainty intervals
parameters	DEPRECATED, use variables instead
show_hidden	Show variables marked with hidden_var_attribute() (by default, those are not shown, available only when x is a data.frame)

Value

a ggplot2 plot object

---

rbind.SBC_bridgesampling_diagnostics

Custom rbind implementation maintainig information about submodels

---

Description

Custom rbind implementation maintainig information about submodels

Usage

```
## S3 method for class 'SBC_bridgesampling_diagnostics'
rbind(...)
```

---

recompute_SBC_statistics

Recompute SBC statistics without refitting models.

Description

Useful for example to recompute SBC ranks with a different choice of thin_ranks or added derived quantities.

Usage

```
recompute_SBC_statistics(
  old_results,
  datasets,
  backend,
  thin_ranks = SBC_backend_default_thin_ranks(backend),
  ensure_num_ranks_divisor = 2,
  dquants = NULL,
  gen_quants = NULL
)
```

Arguments

datasets	an object of class SBC_datasets
backend	backend used to fit the results. Used to pull various defaults and other setting influencing the computation of statistics.
thin_ranks	how much thinning should be applied to posterior draws before computing ranks for SBC. Should be large enough to avoid any noticeable autocorrelation of the thinned draws See details below.
ensure_num_ranks_divisor	Potentially drop some posterior samples to ensure that this number divides the total number of SBC ranks (see Details).
dquants	Derived quantities to include in SBC. Use derived_quantities() to construct them.
gen_quants	Deprecated, use dquants instead

Value

An S3 object of class SBC_results with updated $stats and $default_diagnostics fields.

---

remove_attribute_from_stats

Remove an attribute from a data.frame *column with attributes.*

Description

Remove an attribute from a data.frame column with attributes.

Usage

```
remove_attribute_from_stats(attribute, x)
```

Details

By design, only one copy of an attribute is removed, if there are multiple copies

SBC_backend_bridgesampling *Backend using bridgesampling to do Bayes factor calculation (and Bayesian Model Averaging) over two candidate models.*

Description

Backend using bridgesampling to do Bayes factor calculation (and Bayesian Model Averaging) over two candidate models.

Usage

```
SBC_backend_bridgesampling(
  ...,
  model_var = "model",
  prior_probs = NULL,
  prior_prob1 = NULL,
  bridgesampling_args = list()
)
```

Arguments

...	passed to bridgesampling::bridge_sampler().

SBC_backend_brms *Build a backend based on the* brms *package.*

Description

Build a backend based on the brms package.

Usage

```
SBC_backend_brms(
  ...,
  template_data,
  out_stan_file = NULL,
  template_dataset = NULL
)
```

Arguments

...	arguments passed to brm.
template_data	a representative value for the data argument in brm that can be used to generate code.
out_stan_file	A filename for the generated Stan code. Useful for debugging and for avoiding unnecessary recompilations.
template_dataset	DEPRECATED. Use template_data

SBC_backend_brms_from_generator *Build a brms backend, reusing the compiled model from a previously created* SBC_generator_brms *object.*

Description

Build a brms backend, reusing the compiled model from a previously created SBC_generator_brms object.

Usage

```
SBC_backend_brms_from_generator(generator, ...)
```

SBC_backend_cached *A backend that wraps another backend with a caching scheme.*

Description

The cache is stored in files named based on dataset and underlying backend hashes, within the given directory, you can use cached_fit_filename() to manually access the cache.

Usage

```
SBC_backend_cached(cache_dir, backend)
```

Arguments

cache_dir	directory where the cache files (one file per fit) will be stored
backend	any other backend that does the actual computation

Details

The cached fit will try to re-emit all output, warnings and messages. The ordering within each category is preserved, but the relative ordering of e.g. outputs vs. messages (and all other combinations) may not be.

SBC_backend_cmdstan_sample *Backend based on sampling via* cmdstanr.

Description

Backend based on sampling via cmdstanr.

Usage

```
SBC_backend_cmdstan_sample(model, ..., init_factory = NULL)
```

Arguments

model	an object of class CmdStanModel (as created by cmdstanr::cmdstan_model)
...	other arguments passed to the $sample() method of the model. The data and parallel_chains arguments cannot be set this way as they need to be controlled by the SBC package.
init_factory	an optional function that takes in a dataset and returns a value that can be passed to the init argument of $sample(). This allows for data-dependent inits. The caching mechanism in compute_SBC() ignores the environment of the function, i.e. if the init factory takes values from its environment and those values change between runs, this will not by itself cause cached results to be recomputed.

SBC_backend_cmdstan_variational *Backend based on variational approximation via* cmdstanr.

Description

Backend based on variational approximation via cmdstanr.

Usage

```
SBC_backend_cmdstan_variational(model, ..., n_retries_init = 1)
```

Arguments

model	an object of class CmdStanModel (as created by cmdstanr::cmdstan_model)
...	other arguments passed to the $variational() method of the model. The data argument cannot be set this way as they need to be controlled by the SBC package.
n_retries_init	number of times to retry the variational fit if the algorithm has trouble initializing (e.g. too many dropped evaluations (see https://discourse.mc-stan.org/t/advi-too-many-dropped-evaluations-even-for-well-behaved-models/24338), or "cannot compute ELBO using the initial variational distribution")

SBC_backend_default_thin_ranks *S3 generic to get backend-specific default thinning for rank computation.*

Description

The default implementation plays it relatively safe and returns 10, unless `SBC_backend_iid_draws()` returns `TRUE` in which case it returns 1.

Usage

```
SBC_backend_default_thin_ranks(backend)

## Default S3 method:
SBC_backend_default_thin_ranks(backend)
```

SBC_backend_extractBF_comparison *A backend for Bayes factor workflow with BayesFactor::extractBF.*

Description

A backend for Bayes factor workflow with BayesFactor::extractBF.

Usage

```
SBC_backend_extractBF_comparison(backend_H0, backend_H1, model_var = "model")
```

SBC_backend_function *A backend that just calls a function.*

Description

If the function returns a `draws_matrix` object, no other work is necessary to make it work with SBC. Useful for quick tests.

Usage

```
SBC_backend_function(
  func,
  generated_arg = "generated",
  cores_arg = NULL,
  args = list(),
  iid_draws = FALSE,
  default_thin_ranks = 10
)
```

Arguments

func	the function that will be called in SBC_fit()
generated_arg	name of the argument of func that will receive the generated data. If NULL, data is not passed to the function.
cores_arg	name of the argument of func that will receive the number of cores to use. If NULL, information on cores is not passed.
args	a (named) list of additional arguments to the function
iid_draws	does the result of the backend have independent identically distribute draws (will be returned by SBC_backend_iid_draws() for this backend).
default_thin_ranks	suggested thinning if user does not specify any (will be returned by SBC_backend_default_thin_ranks(for this backend).

Examples

```
# Generate t-distributed variables as a ratio of standard normal (z)
# and transformed chi-squared (v) variables.
# What is the conditional distribution of z if t is observed?
# See https://math.stackexchange.com/a/5085538/423833 for derivation that it
# is generalized gamma distribution. Here we test this is correct.

library(ggplot2)
theme_set(theme_minimal())

N_sims <- 100
df <- 5
z <- rnorm(N_sims)
v <- rchisq(N_sims, df = df)
t <- z / sqrt(v/df)

# Bundle in a dataset with extra quantities
my_data <- SBC_datasets(
  variables = posterior::draws_matrix
  (z = z, v = v, lik = abs(z) * dchisq(df * z^2 / t^2, df = df)),
  generated = purrr::map(t, \(t) list(t = t, df = df)))

# Main workhorse function
my_post_func <- function(generated) {
  df <- generated$df
  t <- generated$t
  gg_d <- df + 1
  gg_p <- 2
  gg_a <- 1/sqrt(df/(2*t^2) + 0.5)

  # Transform to parametrization used by ggamma
  gg_b <- gg_p
  gg_k <- gg_d / gg_p

  abs_z <- ggamma::rggamma(1000, gg_a, gg_b, gg_k)
  v <- df * abs_z^2 / t^2
```

```
  lik = abs_z * dchisq(df * abs_z^2 / t^2, df = df)
  posterior::draws_matrix(z = abs_z * sign(generated$t),
                          v = v,
                          lik = lik)
}

backend <- SBC_backend_function(my_post_func, iid_draws = TRUE)
res <- compute_SBC(my_data, backend, keep_fits = FALSE)

plot_ecdf_diff(res)
plot_sim_estimated(res)
```

SBC_backend_hash_for_cache *Get hash used to identify cached results.*

Description

S3 generic that allows backends to override how a hash is computed. By default `rlang::hash()` is used.

Usage

```
SBC_backend_hash_for_cache(backend)
```

SBC_backend_iid_draws *S3 generic to let backends signal that they produced independent draws.*

Description

Most backends (e.g. those based on variatns of MCMC) don't produce independent draws and thus diagnostics like Rhat and ESS are important and draws may need thinning. Backends that already produce independent draws (e.g. ADVI/optimizing) can implement this method to return `TRUE` to signal this is the case. If this method returns `TRUE`, ESS and Rhat will always attain their best possible values and `SBC_backend_default_thin_ranks()` will return `1`. The default implementation returns `FALSE`.

Usage

```
SBC_backend_iid_draws(backend)

## Default S3 method:
SBC_backend_iid_draws(backend)
```

Arguments

backend	to check

SBC_backend_lmBF *A backend using* BayesFactor::lmBF()

Description

A backend using BayesFactor::lmBF()

Usage

```
SBC_backend_lmBF(..., iterations = 2000)
```

Arguments

...	all parameters are passed directly to BayesFactor::lmBF().

SBC_backend_mock *A mock backend.*

Description

Mock backend is useful for testing the package. It will ignore all data passed to it and always provide `result` as the draws generated by the backend.

Usage

```
SBC_backend_mock(
  result = posterior::draws_matrix(a = rnorm(100)),
  output = NULL,
  message = NULL,
  warning = NULL,
  error = NULL,
  bridgesampler = NULL
)
```

Arguments

result	a `draws_matrix` that will be returned regardless of the data

SBC_backend_postprocess_cached_fit

Allows the backend to do any pre-/post- processing on a fit stored to / loaded from cache.

Description

This is useful e.g. to provide a live `stanmodel` instance to an rstan fit, as the one loaded from cache won't work for some cases.

Usage

```
SBC_backend_postprocess_cached_fit(backend, generated, fit)

SBC_backend_preprocess_fit_for_cache(backend, generated, fit)
```

Details

The deafult implementation does nothing.

SBC_backend_rjags *Create a JAGS backend using* `rjags`

Description

Create a JAGS backend using `rjags`

Usage

```
SBC_backend_rjags(
  file,
  n.iter,
  n.burnin,
  variable.names,
  thin = 1,
  na.rm = TRUE,
  ...
)
```

Arguments

`file`	model file or connection to model code (passed to `rjags::jags.model()`)
`n.iter`	number of iterations for sampling (passed to [rjags::coda.samples())
`n.burnin`	number of iterations used for burnin
`variable.names`	names of variables to monitor (passed to `rjags::coda.samples()`)

thin	thinning (passed to rjags::coda.samples())
na.rm	whether to omit variables containing NA (passed to rjags::coda.samples())
...	additional optional arguments passed to rjags::jags.model() • most notably n.chains, n.adapt and inits.

SBC_backend_rstan_optimizing

SBC backend using the optimizing *method from* rstan.

Description

SBC backend using the optimizing method from rstan.

Usage

```
SBC_backend_rstan_optimizing(model, ..., n_retries_hessian = 1)
```

Arguments

model	a stanmodel object (created via rstan::stan_model)
...	other arguments passed to optimizing (number of iterations, ...). Argument data cannot be set this way as they need to be controlled by the package.
n_retries_hessian	the number of times the backend is allow to retry optimization (with different seeed) to produce a usable Hessian that can produce draws. In some cases, the Hessian may be numerically unstable and not be positive definite.

SBC_backend_rstan_sample

SBC backend using the sampling *method from* rstan.

Description

SBC backend using the sampling method from rstan.

Usage

```
SBC_backend_rstan_sample(model, ...)
```

Arguments

model	a stanmodel object (created via rstan::stan_model)
...	other arguments passed to sampling (number of iterations, ...). Arguments data and cores cannot be set this way as they need to be controlled by the package.

SBC_datasets *Create new* SBC_datasets *object.*

Description

In most cases, you may want to use `generate_datasets` to build the object, but for full control, you can also create datasets directly via this function.

Usage

```
SBC_datasets(variables, generated, var_attributes = NULL, parameters = NULL)
```

Arguments

`variables`	draws of "true" values of unobserved parameters or other derived variables. An object of class `draws_matrix` (from the `posterior` package)
`generated`	a list of objects that can be passed as data to the backend you plan to use. (e.g. list of values for Stan-based backends, a data frame for `SBC_backend_brms`)
`parameters`	DEPRECATED. Use variables instead.

SBC_datasets_for_bf *Combine two or more datasets to check Bayes factor computation.*

Description

Merge two or more datasets of the same length, each generated by a different model. The "true" model will be chosen randomly for each dataset.

Usage

```
SBC_datasets_for_bf(..., probs = NULL, model_var = "model", prob_H1 = NULL)
```

Details

The merged dataset will keep track of all parameters and later allow splitting results with `split_SBC_results_for_bf()`, to this, a number of submodel-specific variables will be stored, but marked as `hidden_var_attribute()` to not clutter default visualisations.

SBC_example_backend *Construct a backend to be used in the examples.*

Description

Note that this will involve compiling a Stan model and may take a while.

Usage

```
SBC_example_backend(
  example = c("normal_sd", "normal_bad"),
  interface = c("rstan", "cmdstanr", "rjags")
)
```

Arguments

example	name of the example model. `normal_sd` is a simple model fitting a normal distribution parametrized as mean and standard deviation. `normal_bad` is a model that *tries* to implement the `normal_sd` model, but assumes an incorrect parametrization of the normal distribution. For Stan-based backends, the model is written as if Stan parametrized normal distribution with precision (while Stan uses sd), for JAGS-based backends the model is written as if JAGS parametrized normal distribution with sd (while JAGS uses precision).
interface	name of the interface to be used to fit the model

SBC_example_generator *Construct a generator used in the examples.*

Description

Construct a generator used in the examples.

Usage

```
SBC_example_generator(example = c("normal"), N = 100)
```

Arguments

example	name of example
N	size of the dataset the generator should simulate

Value

an object that can be passed to `generate_datasets()`

SBC_example_results *Combine an example backend with an example generator to provide full results that can be used to test other functions in the package.*

Description

Except for `example = "visualizations"`, all examples will actually compile and fit Stan models and thus may take a while to complete.

Usage

```
SBC_example_results(
  example = c("normal_ok", "normal_bad", "visualizations"),
  interface = c("rstan", "cmdstanr", "rjags"),
  N = 100,
  n_sims = 50
)
```

Arguments

`example`	• name of the example. `normal_ok` is an example where the generator matches the model (using the `normal` generator and `normal_sd` backend), while `normal_bad` is an example with a mismatch between the generator and backend that manifests in SBC (`normal_bad` combines the `normal` generator with `normal_bad` backend). `visualizations` creates a purely artificial results that are meant to showcase the built-in plots (the `interface` parameter will be ignored).
`interface`	name of the interface to be used for the backend
`N`	number of datapoints to simulate from the generator for each simulation
`n_sims`	number of simulations to perform

SBC_fit *S3 generic using backend to fit a model to data.*

Description

Needs to be implemented by all backends. All implementations have to return an object for which you can safely call `SBC_fit_to_draws_matrix()` and get some draws. If that's not possible an error should be raised.

Usage

```
SBC_fit(backend, generated, cores)
```

`SBC_fit_to_BFBayesFactor` *A fits need to implement this S3 generic to be useful for the* `SBC_backend_extractBF_comparison()` *backend.*

Description

A fits need to implement this S3 generic to be useful for the `SBC_backend_extractBF_comparison()` backend.

Usage

```
SBC_fit_to_BFBayesFactor(fit)
```

`SBC_fit_to_bridge_sampler` *Convert a fit for a given backend into a bridge sampler object.*

Description

Users should typically not call this method and instead let `SBC_backend_bridgesampling()` to call it for them.

Usage

```
SBC_fit_to_bridge_sampler(backend, fit, generated, ...)
```

Arguments

`backend`	the underlying backend
`fit`	corresponding to the backend
`generated`	the generated data used to fit the underlying model
`...`	passed to `bridgesampling::bridge_sampler()`.

Value

an object of class `bridge` or `bridge_list`.

See Also

`bridgesampling::bridge_sampler()`

SBC_fit_to_diagnostics

S3 generic to get backend-specific diagnostics.

Description

The diagnostics object has to be a `data.frame` but may inherit additional classes - in particular it may be useful for the returning object to implement `diagnostic_types()`.

Usage

```
SBC_fit_to_diagnostics(fit, fit_output, fit_messages, fit_warnings)
```

Arguments

`fit`	The fit returned by `SBC_fit`
`fit_output`	a character string capturing what the backend wrote to stdout
`fit_messages`	a character vector of messages the backend raised
`fit_warnings`	a character vector of warnings the backend raised

Value

an single row `data.frame` that includes diagnostics or NULL, if no diagnostics available.

SBC_fit_to_draws_matrix

S3 generic converting a fitted model to a `draws_matrix` *object.*

Description

Needs to be implemented for all types of objects the backend can return from `SBC_fit()`. Default implementation just calls, `posterior::as_draws_matrix()`, so if the fit object already supports this, it will work out of the box.

Usage

```
SBC_fit_to_draws_matrix(fit)

## Default S3 method:
SBC_fit_to_draws_matrix(fit)
```

SBC_generator_brms *Create a brms generator.*

Description

Brms generator uses a brms model with `sample_prior = "only"` to generate new datasets.

Usage

```
SBC_generator_brms(
  formula,
  data,
  ...,
  generate_lp = TRUE,
  generate_lp_chunksize = 5000/nrow(data),
  out_stan_file = NULL
)
```

Arguments

`formula, data, ...`	arguments passed to `brms::brm`
`generate_lp`	whether to compute the overall log-likelihood of the model as an additional variable. This can be somewhat computationally expensive, but improves sensitivity of the SBC process.
`generate_lp_chunksize`	Will determine whether the chunk size used with `future.apply::future_mapply()`. Set quite high by default as the parallelism only benefits for large individual datasets/number of simulations.
`out_stan_file`	A filename for the generated Stan code. Useful for debugging and for avoiding unnecessary recompilations.

SBC_generator_custom *Wrap a function the creates a complete dataset.*

Description

This creates a very thin wrapper around the function and can store additional arguments, but does not do anything more..

Usage

```
SBC_generator_custom(f, ...)
```

Arguments

f	function accepting at least an n_sims argument and returning and SBC_datasets object
...	Additional arguments passed to f

Details

Running:

```
gen <- SBC_generator_custom(f, <<some other args>>)
datasets <- generate_datasets(gen, n_sims = my_n_sims)
```

is equivalent to just running

```
datasets <- f(<<some other args>>, n_sims = my_n_sims)
```

So whenever you control the code calling generate_datasets, it usually makes more sense to just create an SBC_datasets object directly and avoid using SBC_generator_custom and generate_datasets at all. SBC_generator_custom can however be useful, when a code you do not control calls generate_datasets for you and the built-in generators do not provide you with enough flexibility.

SBC_generator_function

Generate datasets via a function that creates a single dataset.

Description

The generator supports progress reporting with the progressr package. Run progressr::handlers(global = TRUE) to enable progress reporting globally, see the progressr docs for more options.

Usage

```
SBC_generator_function(
  f,
  future.chunk.size = getOption("SBC.generator_chunk_size", +Inf),
  future.globals = TRUE,
  ...
)
```

Arguments

f	function returning a list with elements variables (prior draws, a list or anything that can be converted to draws_rvars) and generated (observed dataset, ready to be passed to backend)

`future.chunk.size`
: if finite, datasets will be generated in parallel using the `future` package with this chunk size (see future.apply::future_lapply). Defaults to the `SBC.generator_chunk_size` option, which defaults to +Inf (no paralellism).

`...`
: Additional arguments passed to `f`

`SBC_posterior_cdf` *Use a fitted model to obtain a posterior CDF at a given point.*

Description

Backends that represent explicit posterior distributions (i.e. not just samples) can implement this S3 generic to evaluate the CDF of that distribution at the simulated values.

Usage

```
SBC_posterior_cdf(fit, variables)

## Default S3 method:
SBC_posterior_cdf(fit, variables)
```

Arguments

`fit`	an object returned by the `SBC_fit()` method.
`variables`	a named vector of values at which to evaluate the CDF

Details

A backend may choose to return explicit CDF only for some model parameters. return from `SBC_fit()`.

Value

either `NULL` (the default implementation) or a `data.frame` with a row for each variable. The columns must include: `variable` (variable name), and either `cdf` (if there are no ties) or the pair `cdf_low` and `cdf_high` indicate the possible tied values for the CDF.

SBC_print_example_model *Print the Stan code of a model used in the examples.*

Description

Print the Stan code of a model used in the examples.

Usage

```
SBC_print_example_model(
  example = c("normal_sd", "normal_bad"),
  interface = c("rstan", "cmdstanr", "rjags")
)
```

Arguments

example	name of the example model.

SBC_results *Create an* SBC_results *object*

Description

This will build and validate an SBC_results object from its constituents.

Usage

```
SBC_results(
  stats,
  fits,
  backend_diagnostics,
  default_diagnostics,
  outputs,
  messages,
  warnings,
  errors
)
```

Details

The SBC_results contains the following fields:

- $stats statistics for all variables and fits (one row per variable-fit combination)
- $fits the raw fits (unless keep_fits = FALSE) or NULL if the fit failed
- $errors error messages that caused fit failures

- `$outputs`, `$messages`, `$warnings` the outputs/messages/warnings written by fits
- `$default_diagnostics` a data frame of default convergence/correctness diagnostics (one row per fit)
- `$backend_diagnostics` a data frame of backend-specific diagnostics (one row per fit)

`SBC_statistics_from_single_fit` *Recompute SBC statistics given a single fit.*

Description

Potentially useful for doing some advanced stuff, but should not be used in regular workflow. Use `recompute_SBC_statistics()` to update an `[SBC_results]` objects with different `thin_ranks` or other settings.

Usage

```
SBC_statistics_from_single_fit(
  fit,
  variables,
  generated,
  thin_ranks,
  ensure_num_ranks_divisor,
  dquants,
  backend,
  var_attributes = NULL,
  gen_quants = NULL
)
```

Arguments

`thin_ranks`	how much thinning should be applied to posterior draws before computing ranks for SBC. Should be large enough to avoid any noticeable autocorrelation of the thinned draws See details below.
`ensure_num_ranks_divisor`	Potentially drop some posterior samples to ensure that this number divides the total number of SBC ranks (see Details).
`dquants`	Derived quantities to include in SBC. Use `derived_quantities()` to construct them.
`backend`	the model + sampling algorithm. The built-in backends can be constructed using `SBC_backend_cmdstan_sample()`, `SBC_backend_cmdstan_variational()`, `SBC_backend_rstan_sam` `SBC_backend_rstan_optimizing()` and `SBC_backend_brms()`. (more to come: issue 31, 38, 39). The backend is an S3 class supporting at least the `SBC_fit()`, `SBC_fit_to_draws_matrix()` methods.
`gen_quants`	Deprecated, use dquants instead

See Also

recompute_SBC_statistics()

`select.SBC_bridgesampling_diagnostics` *Custom select implementation maintaining information about submodels*

Description

Custom select implementation maintaining information about submodels

Usage

```
## S3 method for class 'SBC_bridgesampling_diagnostics'
select(diags, ...)
```

`split_SBC_results_for_bf` *Split stats from SBC of Bayes factor into stats for the individual submodels.*

Description

Split stats from SBC of Bayes factor into stats for the individual submodels.

Usage

```
split_SBC_results_for_bf(results_bf)
```

Value

list of two stats `data.frames`, each for simulations where the corresponding model was chosen.

`validate_derived_quantities` *Validate a definition of derived quantities evaluated in R.*

Description

Validate a definition of derived quantities evaluated in R.

Usage

```
validate_derived_quantities(x)
```

var_attributes *Predefined constants to use for variable attributes recognized by SBC.*

Description

Attributes give additional information useful for presenting SBC results concerning the variables.

Usage

```
var_attributes(...)

validate_var_attributes(var_attr)

possibly_constant_var_attribute()

binary_var_attribute()

hidden_var_attribute()

na_valid_var_attribute()

inf_valid_var_attribute()

submodel_var_attribute(sub_id)
```

Details

Should be passed via an instance of the `var_attributes` class to `SBC_datasets()` (e.g. by returning a `$var_attributes` element from a function passed to `SBC_generator_function()`).

`possibly_constant_var_attribute` attribute signals that having all posterior draws identical is possible and thus no warnings should be made for the resulting NAs in rhat and ESS checks.

`binary_var_attribute` marks the attribute as a binary variable (0 or 1) and thus eligible for some special handling and visualisations (e.g., `binary_probabilites_from_stats()`, `binary_calibration_from_bp()`).

`hidden_var_attribute` will hide the variable in default visualisations, unless the variable is explicitly mentioned.

`na_valid_var_attribute` will treat NAs as potentially equal to any other value in rank ordering. This gives the expected results when NA represents rare problems in computation that should be ignored (see `calculate_ranks_draws_matrix()`). Setting this attribute also changes the ESS/Rhat computation to ignore NAs.

`inf_valid_var_attribute` means infinity values may appear in the samples (this is useful e.g. to note that the parameter is actually not present for the given draw). Setting this attribute also changes the ESS/Rhat computation to ignore infinities. It is also assume that it is OK if *all* of the draws are infinity.

`submodel_var_attribute` signals that the parameter belongs to a submodel which can be extracted individually

In SBC results, the attributes of a variable are summarised in the `attributes` column of the `$stats` data.frame. Use `attribute_present_stats()` to check for presence of an attribute there.

Index