Package: ernest (via r-universe)

June 10, 2026

Title A Toolkit for Nested Sampling

Version 1.2.1

Description Bayesian evidence estimation and posterior inference with the nested sampling algorithm, along with S3 methods for simulating uncertainty and creating visualisations.

License GPL (>= 3)

URL https://github.com/ropensci/ernest, https://docs.ropensci.org/ernest/

BugReports https://github.com/ropensci/ernest/issues

Depends R (>= 4.1.0)

Imports cli, generics, ggplot2, lifecycle, matrixStats, posterior, rlang (>= 1.1.0), vctrs, withr

Suggests brms, distributional, extraDistr, ggdist, knitr, mvtnorm, patchwork, rmarkdown, testthat (>= 3.0.0), tidyselect, uniformly, vdiffr, xml2

LinkingTo cpp11, cpp11eigen, testthat

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first nested_sampling_impl, plot

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE, roclets = c (` `namespace", ` `rd", ` `srr::srr_stats_roclet"))

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-10 22:21:19 UTC

RemoteUrl https://github.com/ropensci/ernest

RemoteRef main

RemoteSha 1a6824166c4e23bc7303ab396ae3dca053261b53

Contents

`as_draws.ernest_run` *Transform nested sampling runs to* `draws` *objects*

Description

Access the posterior sample and weights from a nested sampling run as an object supported by the posterior package.

Usage

```
## S3 method for class 'ernest_run'
as_draws(x, units = c("original", "unit_cube"), ...)

## S3 method for class 'ernest_run'
as_draws_rvars(x, units = c("original", "unit_cube"), ...)

## S3 method for class 'ernest_run'
as_draws_matrix(x, units = c("original", "unit_cube"), ...)
```

Arguments

x	[ernest_run] Results from a nested sampling run.
units	[character(1)] The scale of the sampled points: • "original": Points are on the scale of the prior space. • "unit_cube": Points are on the (0, 1) unit hypercube scale.
...	These dots are for future extensions and must be empty.

Value

posterior::draws_matrix() or posterior::draws_rvars()
A object containing the posterior samples from the nested sampling run, with a hidden .weights column containing the importance weights for each sample.

Note

To produce a weighted posterior sample, use posterior::resample_draws() to reweigh an object from as_draws using its importance weights.

See Also

posterior::as_draws()

Examples

```
library(posterior)
data(example_run)

# View importance weights
dm <- as_draws(example_run)
str(dm)
weights(dm) |> head()

# Summarise points after resampling
dm |>
  resample_draws() |>
  summarize_draws()

# Extract the same coordinates in the unit space coordinates
dm_unit <- as_draws_rvars(example_run, units = "unit_cube")
str(dm_unit)
```

`calculate.ernest_run` *Estimate Evidence using a Nested Sampling Run*

Description

Computes evidence and related quantities from a nested sampling run, optionally by simulating the volumes of each nested likelihood shell.

Usage

```
## S3 method for class 'ernest_run'
calculate(x, ndraws = 1000L, ...)
```

Arguments

`x`	[ernest_run] Results from a nested sampling run.
`ndraws`	`[integer(1)]` The number of log-volume sequences to simulate. If equal to zero, log-volume simulation is skipped and error in the log-evidence estimates is approximated with analytical error estimates.
`...`	These dots are for future extensions and must be empty.

Value

An `ernest_estimate` object, which inherits from `tbl_df`, `tbl`, and `data.frame`.

The iterative estimates from the nested sampling run. Contains the following columns for each point:

- `log_lik`: `[[double()]]` The log-likelihood of the point.
- `log_volume`: `[[posterior::rvar()]]` The log-volume of the prior space associated with the point.
- `log_weight`: `[[posterior::rvar()]]` The estimated contribution of the point to the log-evidence estimate (i.e., the unnormalized posterior log-weight).
- `log_evidence`: `[[posterior::rvar()]]` The current log-evidence estimate.

If `ndraws > 0`, `log_volume`, `log_weight`, and `log_evidence` each contain `ndraws` simulated draws per iteration.

If `ndraws = 0`, `log_volume` and `log_weight` contain a single deterministic draw per iteration, and `log_evidence` contains draws from a normal approximation based on analytical variance estimates (see the package vignetttes for more information). The number of draws is controlled with getOption("posterior.rvar_ndraws"), with a default of 1000.

References

Higson, E., Handley, W., Hobson, M., & Lasenby, A. (2019). Nestcheck: Diagnostic Tests for Nested Sampling Calculations. Monthly Notices of the Royal Astronomical Society, 483(2), 2044–2056. doi:10.1093/mnras/sty3090

See Also

weights.ernest_run()

Examples

```
# Load an example run
data(example_run)

# View results and analytical evidence errors.
calculate(example_run, ndraws = 0)

# Simulate 100 log-volume shrinkage sequences across the run.
calculate(example_run, ndraws = 100)
```

---

compile.ernest_sampler

Compile the live set of points for nested sampling

Description

Prepares an object for nested sampling by validating and (re)generating its live set. This ensures the sampler is viable before new points are drawn during the nested sampling algorithm.

Usage

```
## S3 method for class 'ernest_sampler'
compile(object, ...)

## S3 method for class 'ernest_run'
compile(object, clear = FALSE, ...)
```

Arguments

object	[ernest_run] or [ernest_sampler] Results from a nested sampling run.
...	These dots are for future extensions and must be empty.
clear	[logical(1)] If TRUE, clears results from previous runs before compiling. If FALSE, retains previous results and validates the live set.

Details

`compile()` validates the live set bound to `object`, ensuring that:

- Each point in the set is within the unit hypercube.
- The likelihood function returns valid values (finite double or `-Inf`) for each point.
- The live set does not represent a perfect likelihood plateau (i.e., that all points share the same likelihood). A warning is issued if more than 25% of points share the same likelihood value.

If validation fails, the live set is removed from `object`, preventing further sampling until the issue is resolved.

Value

A copy of `[object]`.

The copy is guaranteed to have a valid live set, created according to the class of `object` and the value of `clear`:

- If `object` is an `ernest_sampler`, or if `clear = TRUE`, a new live set is created from scratch.
- If `object` is an `ernest_run`, the live set is regenerated from previous results.

See Also

`generate.ernest_run()`

Examples

```
prior <- create_uniform_prior(lower = c(-1, -1), upper = 1)
ll_fn <- function(x) -sum(x^2)
sampler <- ernest_sampler(ll_fn, prior, nlive = 100)

# Compile the sampler to add a live set
compile(sampler)
head(sampler$live_env$unit)

# Continue a previous run
# run <- data(example_run)
# sampler_2 <- compile(example_run)
# sampler_2

# Make a new sampler from a previous run
sampler_3 <- compile(example_run, clear = TRUE)
sampler_3
```

`create_likelihood` *Prepare a likelihood function for nested sampling*

Description

Creates a modified version of a log-likelihood function that always returns either a finite value or `-Inf` for each vector of parameters provided.

Usage

```
create_likelihood(
  scalar_fn,
  vectorized_fn,
  on_nonfinite = c("warn", "quiet", "abort")
)
```

Arguments

`scalar_fn, vectorized_fn`
: `[function]`
The log-likelihood function. Provide either `scalar_fn` or `vectorized_fn`:
- `scalar_fn`: Should accept a parameter as a numeric vector and return a single numeric value representing the log-likelihood, or `-Inf`.
- `vectorized_fn`: Should accept a matrix of parameter vectors (rows as samples, columns as elements of the parameter vector) and return a vector of log-likelihoods or `-Inf` values for each row.

`on_nonfinite`
: `[character]`
How the sampler should handle values returned by `fn` or `matrix_fn` that are not finite and not equal to `-Inf`. Must be one of:
- `"warn"`: Issue a warning and return `-Inf`.
- `"quiet"`: Silently return `-Inf`.
- `"abort"`: Stop execution and signal an error.

Details

Provide model likelihoods as a log-density function, which take a vector of free parameter values and return the corresponding log-likelihood value.

Likelihoods are typically the most computationally expensive function to evaluate in a nested sampling run. ernest allows you to implement your likelihood as a function over a single parameter vector (`scalar_fn`) or over a matrix of parameters (`vectorized_fn`).

ernest expects the log-likelihood function to return a finite double or `-Inf` for each parameter vector. The behaviour when encountering non-finite values other than `-Inf` (such as `NaN`, `Inf`, or `NA`) is controlled by `on_nonfinite`.

If your log-likelihood depends on additional data (e.g., an observation matrix or data frame), provide these using an (anonymous function)`rlang::as_function()` (see Examples).

Value

[ernest_likelihood], which inherits from function.

See Also

cubature for examples of vectorized and non-vectorized likelihood functions.

Examples

```
library(mvtnorm)

# Multivariate Normal Distribution
m <- 3
mean <- rep(0, m)
sigma <- diag(m)
sigma[2, 1] <- sigma[1, 2] <- 3 / 5
sigma[3, 1] <- sigma[1, 3] <- 1 / 3
sigma[3, 2] <- sigma[2, 3] <- 11 / 15
prec <- solve(sigma)
log_det <- -sum(log(diag(chol(sigma))))

# Provide a Scalar Log-Likelihood Function:
log_lik <- function(x) {
  log_det - 0.5 * m * log(2 * pi) - 0.5 * (t(x) %*% prec %*% x)
}
log_lik(c(0, 0, 0))

# `create_likelihood` allows scalar fns. to accept matrix inputs:
try(log_lik(matrix(rep(0, m * 2), nrow = 2)))
scalar_ll <- create_likelihood(scalar_fn = log_lik)
scalar_ll(matrix(rep(0, m * 2), nrow = 2))

# Provide a Vectorized Log-Likelihood Function:
v_log_lik <- function(x) {
  dmvnorm(x, mean = mean, sigma = sigma, log = TRUE)
}
v_log_lik(c(0, 0, 0))
v_log_lik(matrix(rep(0, m * 2), nrow = 2))

vector_ll <- create_likelihood(vectorized_fn = v_log_lik)
vector_ll

# Control Behaviour when Nonfinite Likelihood Values are Encountered
# Default: Warn and replace with `-Inf`
vector_ll(c(0, 0, NA))

# Signal an error
abort_ll <- create_likelihood(log_lik, on_nonfinite = "abort")
try(abort_ll(c(0, 0, NA)))

# Silently replace all non-finite values
quiet_ll <- create_likelihood(vectorized_fn = v_log_lik, on_nonfinite = "quiet")
```

```
quiet_ll(c(0, 0, NA))
```

`create_normal_prior` *Uniform distribution*

Description

Uniform distribution

Usage

```
create_normal_prior(
  names = NULL,
  mean = 0,
  sd = 1,
  lower = -Inf,
  upper = Inf,
 repair = c("unique", "universal", "check_unique", "unique_quiet", "universal_quiet")
)

create_uniform_prior(
  names = NULL,
  lower = 0,
  upper = 1,
 repair = c("unique", "universal", "check_unique", "unique_quiet", "universal_quiet")
)
```

Arguments

`names`	`[character()]` Optional names for each parameter. If `NULL`, default names and indices are generated.
`mean, sd`	`[double()]` Mean and standard deviation for each marginal normal distribution. `sd` must be strictly positive.
`lower, upper`	`[double()]` Lower and upper bounds for each parameter. If used with `create_normal_prior()`, these define the truncation limits.
`repair`	`[character(1)]` Name repair strategy for `names`. One of `"check_unique"`, `"unique"`, `"universal"`, `"unique_quiet"`, or `"universal_quiet"`. See `vctrs::vec_as_names()` for details.

Details

The provided transformations are vectorized: they accept a matrix of points in the unit hypercube and return a matrix of transformed values.

Value

An ernest_prior, additionally inheriting from the specialized class `uniform_prior` or `normal_prior`.

See Also

`create_prior()`

Other priors: `create_prior()`

Examples

```
# Specify a prior with independent marginals
normal <- create_normal_prior(
  names = c("beta0", "beta1", "beta2"),
  mean = 0,
  sd = 5
)
uniform <- create_uniform_prior(names = "sd", lower = 0, upper = 5)
composite <- normal + uniform
composite

# Propose a conditional (hierarchical) prior in vectorized form
fn <- function(x) {
  n <- nrow(x)
  out <- matrix(NA_real_, nrow = n, ncol = 3)
  # x[1] follows N(5, 1)
  out[, 1] <- stats::qnorm(x[, 1], mean = 5, sd = 1)
  # log10(x[2]) follows Uniform(-1, 1)
  out[, 2] <- 10^stats::qunif(x[, 2], min = -1, max = 1)
  # x[3] follows N(x[1], x[2])
  out[, 3] <- stats::qnorm(x[, 3], mean = out[, 1], sd = out[, 2])
  out
}

conditional_prior <- create_prior(
  vectorized_fn = fn,
  names = c("mean", "sd", "x"),
  lower = c(-Inf, 0, -Inf)
)

# Plot the marginals
sample <- conditional_prior$fn(matrix(runif(1000 * 3), nrow = 1000))
hist(sample[, 1], main = "mean")
hist(sample[, 2], main = "sd")
hist(sample[, 3], main = "x")
```

`create_prior` *Prepare a prior transformation for nested sampling*

Description

Creates a prior transformation object of class `ernest_prior`, which defines how to map points from the unit hypercube to the parameter space for use in a nested sampling run.

Usage

```
create_prior(
  point_fn,
  vectorized_fn,
  names,
  lower = -Inf,
  upper = Inf,
 repair = c("unique", "universal", "check_unique", "unique_quiet", "universal_quiet")
)

## S3 method for class 'ernest_prior'
x + y
```

Arguments

`point_fn`, `vectorized_fn`
: `[function]`
The prior transformation function. Provide either `point_fn` or `vectorized_fn`:
  - `point_fn`: Should accept a single parameter vector (numeric vector of length equal to the number of parameters) and return a vector in the original parameter space.
  - `vectorized_fn`: Should accept a matrix of points in the unit hypercube (rows as parameter vectors) and return a matrix of the same shape in the original parameter space.

`names`
: `[character()]`
Unique names for each variable in the prior distribution.

`lower`, `upper`
: `[double()]`
Expected lower and upper bounds for the parameter vectors after transformation.

`repair`
: `[character(1)]`
Name repair strategy for `names`. One of `"check_unique"`, `"unique"`, `"universal"`, `"unique_quiet"`, or `"universal_quiet"`. See `vctrs::vec_as_names()` for details.

`x`, `y`
: [ernest_prior]
Prior objects to combine.

Details

The prior transformation encodes points in the parameter space as independent and identically distributed points within a unit hypercube. Nested sampling implementations, including ernest, use this transformation to simplify likelihood-restricted prior sampling and avoid unnecessary rejection steps.

Provide your prior as a transformation function. For factorisable priors, this can simply transform each value in (0, 1) using the inverse CDF for each parameter. For more complex cases, you can specify hierarchical or conditionally dependent priors.

`create_prior` performs regularity checks on your prior function to catch basic errors that may affect nested sampling. The function must take in a vector (or matrix) of points (each between 0 and 1) and return a vector or matrix of the same shape containing only finite values.

If your prior depends on additional data, provide these using an anonymous function (see Examples).

Value

`[ernest_prior]`, an object describing the prior transformation, with `names`, `lower`, and `upper` recycled to the same length.

Note

See vctrs::vector_recycling_rules for information on how parameters are recycled to a common length.

See Also

Other priors: `create_normal_prior()`

Examples

```
# Specify a prior with independent marginals
normal <- create_normal_prior(
  names = c("beta0", "beta1", "beta2"),
  mean = 0,
  sd = 5
)
uniform <- create_uniform_prior(names = "sd", lower = 0, upper = 5)
composite <- normal + uniform
composite

# Propose a conditional (hierarchical) prior in vectorized form
fn <- function(x) {
  n <- nrow(x)
  out <- matrix(NA_real_, nrow = n, ncol = 3)
  # x[1] follows N(5, 1)
  out[, 1] <- stats::qnorm(x[, 1], mean = 5, sd = 1)
  # log10(x[2]) follows Uniform(-1, 1)
  out[, 2] <- 10^stats::qunif(x[, 2], min = -1, max = 1)
  # x[3] follows N(x[1], x[2])
  out[, 3] <- stats::qnorm(x[, 3], mean = out[, 1], sd = out[, 2])
  out
}

conditional_prior <- create_prior(
  vectorized_fn = fn,
  names = c("mean", "sd", "x"),
```

```
  lower = c(-Inf, 0, -Inf)
)

# Plot the marginals
sample <- conditional_prior$fn(matrix(runif(1000 * 3), nrow = 1000))
hist(sample[, 1], main = "mean")
hist(sample[, 2], main = "sd")
hist(sample[, 3], main = "x")
```

`ernest_sampler` *Prepare a new nested sampling run*

Description

Initializes an `ernest_sampler` object containing the components required to perform nested sampling. This object can then be used to build sequences of nested samples with `generate.ernest_run()`.

Usage

```
ernest_sampler(
  log_lik,
  prior,
  sampler = rwmh_cube(),
  nlive = 500,
  first_update = NULL,
  update_interval = NULL,
  seed = NA
)
```

Arguments

`log_lik`	`[function]` or [ernest_likelihood] A function which computes the log-likelihood of a given model. If a function, it is wrapped with `create_likelihood()`.
`prior`	[ernest_prior] Describes the prior space within which to generate samples.
`sampler`	[ernest_lrps] Specifies the likelihood-restricted prior sampling method used to replace points within the live set.
`nlive`	`[integer(1)]` The number of points to generate within the live set. Strictly positive.
`first_update`	`[integer(1)]` The number of likelihood calls to make with the default uniform LRPS method before swapping to the technique described by `sampler`. Optional; if left `NULL` this is set to `nlive * 2.5`.

update_interval
: [integer(1)]
The number of likelihood calls between updates to the sampler object. Optional; if left NULL this is set to nlive * 1.5.

seed
: [integer(1)]
Sets the random seed controlling the random number generator for nested sampling runs. Optional; if left NA the .Random.seed set within R is preserved and restored after a run.

Details

The ernest_sampler object is tested with compile.ernest_run() before it is returned. This helps to catch errors with the likelihood and prior specifications. If this compilation step fails, review your log_lik_fn and prior objects for their compliance.

Value

[ernest_sampler]
A named list, containing a specification for a nested sampling run. Contains the arguments passed to this function as well as an environment live_env, which is used to store the live set during sampling.

Verbosity

Messages from ernest can be silenced with the global options rlib_message_verbosity and rlib_warning_verbosity. These options take the values:

- "default": Verbose unless the .frequency argument is supplied.
- "verbose": Always verbose.
- "quiet": Always quiet.

When set to quiet, messages are not displayed and the condition is not signaled. See rlang::abort() for more information.

Examples

```
prior <- create_uniform_prior(lower = c(-1, -1), upper = 1)
ll_fn <- function(x) -sum(x^2)
sampler <- ernest_sampler(ll_fn, prior, nlive = 100)
sampler

# Use a unit-cube LRPS (not recommended in practice)
unit_sampler <- ernest_sampler(
  ll_fn,
  prior,
  nlive = 100,
  sampler = unif_cube()
)
unit_sampler
```

example_run *Example Nested Sampling Run with Ernest*

Description

Load a precomputed example nested sampling run generated using the ernest package. It demonstrates a typical output from a nested sampling run on a simple 3-dimensional Gaussian likelihood, with a uniform prior over each dimension. This dataset is intended for use in documentation, tutorials, and gaining experience with `ernest_run`'s S3 methods.

Usage

```
example_run
```

Format

An object of class `ernest_run` containing the results of a nested sampling run.

Details

The likelihood used to generate the points is $MVN(0, \Sigma)$, with each variance in Σ set to 1 and each covariance set to 0.95. The prior for each parameter is uniform on the interval `[-10, 10\]`.

This run uses the following non-default settings:

- `log_lik`: A 3D multivariate Gaussian with mean zero and covariance matrix `diag(0.95, 3)`.
- `prior`: Uniform over each dimension (x, y, z) in the range [-10, 10].
- `seed`: 42

View the `$spec` element of `example_run` to see the full R specification of the likelihood and prior.

[-10, 10]: R:-10,%2010%5C [-10, 10]: R:-10,%2010

Source

This example problem comes from the crash course for the dynesty Python-based nested sampling software.

`generate.ernest_sampler` *Run nested sampling to estimate Bayesian evidence*

Description

Executes the nested sampling algorithm, iteratively replacing the worst live point with a new sample from a likelihood-restricted prior until a stopping criterion is met.

Usage

```
## S3 method for class 'ernest_sampler'
generate(
  x,
  max_iterations = NULL,
  max_evaluations = NULL,
  min_logz = 0.05,
  show_progress = NULL,
  ...
)

## S3 method for class 'ernest_run'
generate(
  x,
  max_iterations = NULL,
  max_evaluations = NULL,
  min_logz = 0.05,
  show_progress = NULL,
  ...
)
```

Arguments

`x`	[ernest_sampler] or [ernest_run] A nested sampling specification.
`max_iterations`	`[integer(1)]` The maximum number of iterations to perform. Optional; if `NULL` this criterion is ignored.
`max_evaluations`	`[integer(1)]` The maximum number of times the run can evaluate the likelihood function. Optional; if `NULL` this criterion is ignored.
`min_logz`	`[double(1)]` The minimum log-ratio between the current estimated evidence and the remaining evidence. Must be non-negative; if set to zero, this criterion is ignored.

show_progress [logical(1)]
: If TRUE, displays a progress spinner and iteration counter during sampling. Optional; if NULL the global option rlib_message_verbosity is used to determine whether to show progress.

... Arguments passed on to compile.ernest_run

clear [logical(1)]
: If TRUE, clears results from previous runs before compiling. If FALSE, retains previous results and validates the live set.

Details

At least one of max_iterations, max_evaluations, or min_logz must specify a valid stopping criterion. Setting min_logz to zero while leaving max_iterations and max_evaluations at their defaults will result in an error.

If x is an ernest_run object, the stopping criteria are checked against the current state of the run. An error is thrown if the stopping criteria have already been satisfied by x.

The min_logz parameter controls the relative tolerance for the remaining evidence in the unexplored parameter space. Sampling stops when the estimated remaining evidence is sufficiently small compared to the accumulated evidence.

rcrd will have size niter + nlive: The first niter entries correspond to the points removed during the run, the last nlive entries correspond to the points within the live set at the end of the run.

Value

An [ernest_run] object with the nested sampling results.

This inherits from ernest_sampler and adds:

- niter: [integer(1)] Number of iterations performed.
- neval: [integer(1)] Number of times the likelihood function was evaluated.
- log_evidence: [double(1)] The log-evidence estimate.
- log_evidence_err: [double(1)] The standard error of the log-evidence estimate, derived using information.
- information: [double(1)] The estimated Kullback-Leibler divergence.
- log_weight: [double(nsample)] Each sample's contribution to the log-evidence estimate, computed from its log-likelihood and prior volume.
- rcrd: [ernest_rcrd] An object storing an internal record of each point generated during the run.

References

Skilling, J. (2006). Nested Sampling for General Bayesian Computation. Bayesian Analysis, 1(4), 833–859. doi:10.1214/06BA127

See Also

calculate.ernest_run() summary.ernest_run() weights.ernest_run()

Examples

```
prior <- create_uniform_prior(lower = c(-1, -1), upper = 1)
ll_fn <- function(x) -sum(x^2)
sampler <- ernest_sampler(ll_fn, prior, nlive = 100)
sampler

# Stop sampling after a set number of iterations or likelihood evaluations.
generate(sampler, max_iterations = 100)

# Use the default stopping criteria
## Not run:  generate(sampler)
```

`multi_ellipsoid` *Generate new points from multiple spanning ellipsoids*

Description

Partitions the prior space into a set of ellipsoids whose union bounds the live set. New points are created by randomly selecting an ellipsoid (weighted by their respective volumes), then using it to generate a random point as in unif_ellipsoid. Effective for multimodal posteriors where a single ellipsoid would be inefficient.

Usage

```
multi_ellipsoid(enlarge = 1.25)
```

Arguments

`enlarge`	`[double(1)]` Factor by which to inflate the bounding ellipsoid's volume before sampling (see Details). Must be at least 1.0.

Details

Nested likelihood contours for multimodal distributions are poorly represented by a single ellipsoid. This method fits multiple ellipsoids to better capture disconnected or elongated regions.

Ellipsoids are generated using the following procedure:

1. A single ellipsoid is fit to the live set, with volume V.
2. The live set is clustered into two groups using k-means clustering.
3. Ellipsoids are fit to each cluster.
4. The split ellipsoids are accepted if both ellipsoids are non-degenerate, and if the combined volume of the split ellipsoids is significantly smaller than the original ellipsoid (calculated using Bayes' Information Criterion).
5. Steps 2–4 are repeated recursively on each new ellipsoid until no further splits are accepted, updating V to the volume of the currently split ellipsoid.

Value

[multi_ellipsoid], a named list that inherits from [ernest_lrps].

References

- Feroz, F., Hobson, M. P., Bridges, M. (2009) MULTINEST: An Efficient and Robust Bayesian Inference Tool for Cosmology and Particle Physics. Monthly Notices of the Royal Astronomical Society. 398(4), 1601–1614. doi:10.1111/j.13652966.2009.14548.x
- Speagle, J. S. (2020). Dynesty: A Dynamic Nested Sampling Package for Estimating Bayesian Posteriors and Evidences. Monthly Notices of the Royal Astronomical Society, 493, 3132–3158. doi:10.1093/mnras/staa278

See Also

Other ernest_lrps: `rwmh_cube()`, `slice_rectangle()`, `unif_cube()`, `unif_ellipsoid()`

Examples

```
data(example_run)
lrps <- multi_ellipsoid(enlarge = 1.25)

ernest_sampler(example_run$log_lik_fn, example_run$prior, sampler = lrps)
```

---

plot.ernest_estimate *Plot diagnostics from nested sampling results*

Description

Visualizes key diagnostics from nested sampling outputs as functions of log-prior volume.

Usage

```
## S3 method for class 'ernest_estimate'
plot(x, which = c("evidence", "weight", "likelihood"), n = 512, ...)

## S3 method for class 'ernest_run'
plot(x, which = c("evidence", "weight", "likelihood"), n = 512, ...)

## S3 method for class 'ernest_estimate'
summary(
  object,
  which = c("evidence", "weight", "likelihood"),
  n = 512,
  width = NULL,
  ...
)
```

Arguments

x, object	[ernest_run] or [ernest_estimate] An object containing results from nested sampling.
which	[character()] One or more diagnostics to display. Must be any of "evidence", "weight", and "likelihood".
n	[integer(1)] Number of evaluation points along the log-volume axis used to summarize each curve.
...	These dots are for future extensions and must be empty.
width	[numeric()] Numeric vector of widths for uncertainty intervals. Defaults to the 50%, 80%, and 95% highest mean depth intervals.

Details

plot() is a visualization wrapper around summary.ernest_estimate(), followed by autoplot(). Use which to select diagnostics:

- which = "evidence": Estimated marginal likelihood as a function of log-prior volume.
- which = "weight": Posterior mass concentration across log-prior volume.
- which = "likelihood": Normalized likelihood across log-prior volume.

If x is an ernest_run, plotting first computes calculate(x, ndraws = 0). In this mode, log_volume and log_weight are deterministic, and evidence uncertainty comes from the analytical normal approximation generated from the original sampling run.

If x is an ernest_estimate generated with ndraws > 0, diagnostics are summarized over simulated log-volume trajectories. For these simulated estimates, uncertainty bands for evidence and weight are computed as interval summaries on interpolated curves.

To get the underlying data frames used for plotting, use summary.ernest_estimate(). This is useful when you want full control over plotting.

Value

For plot, the ggplot object, invisibly. It is also printed as a side effect. For summary, a list with possible elements evidence, weight, and likelihood. Each element is a data frame.

Note

Plotting multiple diagnostics with which requires **patchwork**. Plotting evidence or weight diagnostics requires **ggdist**.

See Also

calculate.ernest_run()

Other visualizations: visualize.ernest_run()

Examples

```
# Plot diagnostics from a run (analytical uncertainty for evidence).
data(example_run)
plot(example_run)

# Plot diagnostics from simulated log-volume trajectories.
set.seed(123)
est <- calculate(example_run, ndraws = 100)
plot(est)
```

`rwmh_cube` *Generate new points with a random walk*

Description

Create new samples for the live set by evolving a current point in the set through a Metropolis-Hastings random walk, rejecting steps that fail to meet the likelihood criterion.

Usage

```
rwmh_cube(steps = 25, target_acceptance = 0.5)
```

Arguments

`steps`	`[integer(1)]` Number of steps to take when generating a proposal point. Must be greater or equal to 2.
`target_acceptance`	`[double(1)]` Target acceptance rate for proposed points. Must be a number between `1 / steps` and 1.

Details

The random walk LRPS generates proposals by performing a fixed number of Metropolis-Hastings steps within the unit hypercube. Each step proposes a new location by adding a random perturbation to the current position, accepting or rejecting the step based on whether it satisfies the likelihood criterion. This process continues for the specified number of steps, with the final accepted position returned as the proposal.

Step-size Adaptation: The step size ϵ is adapted between sampling rounds using `update_lrps()`. The adaptation uses a Newton-like method to target the desired acceptance rate α_*. Given the current acceptance rate α_i and number of dimensions d, the step size is updated with:

$$\epsilon_i * \exp(\frac{\alpha_i - \alpha^*}{d \cdot \alpha_*})$$

Given the previously-accepted sample X_{i-1} and the number of dimensions d, proposed points are generated from:

$$X_{i-1} + S_d(0, \epsilon)$$

where $S(0, \epsilon)$ is a point drawn uniformly from the d-dimensional ball centered on the origin with radius ϵ.

Value

`[rwmh_cube]`, a named list that inherits from [ernest_lrps].

Control Parameters

- `steps`: Start with 25. Increase to generate points that more closely follow the posterior distribution; decrease for computational efficiency.
- `target_acceptance`: Start with 0.4-0.6. Lower values encourage more global exploration of the posterior, higher values encourage explorations close to the starting point.

References

- Skilling, J. (2006). Nested Sampling for General Bayesian Computation. Bayesian Analysis, 1(4), 833–859. doi:10.1214/06BA127
- Speagle, J. S. (2020). Dynesty: A Dynamic Nested Sampling Package for Estimating Bayesian Posteriors and Evidences. Monthly Notices of the Royal Astronomical Society, 493, 3132–3158. doi:10.1093/mnras/staa278

See Also

Other ernest_lrps: `multi_ellipsoid()`, `slice_rectangle()`, `unif_cube()`, `unif_ellipsoid()`

Examples

```
# Basic usage with default parameters
lrps <- rwmh_cube()

# A faster sampler for simple-to-traverse posterior surfaces
fast_lrps <- rwmh_cube(
  steps = 20,
  target_acceptance = 0.7
)
```

`slice_rectangle` *Generate new points with slice sampling*

Description

Create new samples for the live set by evolving a current point in the set through slice sampling within a bounding hyperrectangle, shrinking the rectangle when proposals are rejected.

Usage

```
slice_rectangle(enlarge = NA)
```

Arguments

`enlarge`	`[double(1)]` Factor by which to inflate the hyperrectangle's volume before sampling (see Details). Optional, and must be greater or equal to 1 if provided; if left `NA`, sampling is initially bounded by the unit hypercube at each iteration.

Details

The slice LRPS generates proposals by uniformly sampling within a bounding hyperrectangle that contains regions of the parameter space satisfying the likelihood criterion. Sampling begins by selecting a known live point θ that satisfies the criterion. Each iteration proposes a new point within this rectangle via uniform sampling and compares it against the criterion; if rejected, a new hyperrectangle is drawn such that the proposed point is on its boundary and θ is in its interior. This continues until either a valid proposal is found or the rectangle has shrunk to the point where no further clamping operations can be performed.

By default, the hyperrectangle spans the extreme values of the current live set in each dimension. This may risk excluding valid regions of the parameter space, particularly where the posterior is multimodal or highly non-Gaussian. To mitigate this, set `enlarge > 1`, which inflates the hyperrectagle's volume by the specified factor before sampling. Setting `enlarge` to `NA` disables this behaviour, instead slicing from the unit hypercube at each iteration.

Value

`[slice_rectangle]`, a named list that inherits from [ernest_lrps].

References

Neal, R. M. (2000). Slice Sampling (Version 1). arXiv. doi:10.48550/ARXIV.PHYSICS/0009028

See Also

Other ernest_lrps: `multi_ellipsoid()`, `rwmh_cube()`, `unif_cube()`, `unif_ellipsoid()`

Examples

```
# Basic usage with default parameters
lrps <- slice_rectangle()

# More patient sampler for complex posteriors
patient_lrps <- slice_rectangle(enlarge = 1.25)
```

`summary.ernest_run` *Summarize a nested sampling run*

Description

Returns a concise summary of an `ernest_run` object, including key statistics and a description of the posterior distribution.

Usage

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

Arguments

`object`	[ernest_run] Results from a nested sampling run.
`...`	These dots are for future extensions and must be empty.

Value

A named list, containing:

- `nlive`: `[integer(1)]` Number of points in the live set.
- `niter`: `[integer(1)]` Number of iterations performed.
- `neval`: `[integer(1)]` Number of times the likelihood function was evaluated.
- `log_evidence`: `[numeric(1)]` Log-evidence estimate.
- `log_evidence_err`: `[numeric(1)]` Standard error of log-evidence.
- `information`: `[numeric(1)]` Estimated Kullback-Leibler divergence between the prior and posterior.
- `reweighted_samples`: [posterior::draws_matrix] Posterior samples, resampled by normalized weights.
- `mle`: `[list]` Maximum likelihood estimate extracted during the run, stored in a list with the elements:
  - `log_lik`: `[double(1)]` The maximum log-likelihood value.
  - `original`, `unit_cube`: `[double(nvar)]` The parameter values at the MLE, expressed in the original parameter space and within the unit cube.

- posterior: [data.frame] with columns for the posterior mean, sd, median, and the 15th and 85th percentiles for each parameter.
- seed: The RNG seed used.

See Also

generate.ernest_run() as_draws.ernest_run()

Examples

```
data(example_run)
run_sm <- summary(example_run)
run_sm
run_sm$posterior
```

---

unif_cube *Generate new points from the unconstrained prior distribution*

Description

Use rejection sampling across the entire prior distribution to create new samples. This is highly inefficient as an LRPS, but may be useful for testing the behaviour of a nested sampling specification.

Usage

```
unif_cube()
```

Value

[unif_cube], a named list that inherits from [ernest_lrps].

References

Speagle, J. S. (2020). Dynesty: A Dynamic Nested Sampling Package for Estimating Bayesian Posteriors and Evidences. Monthly Notices of the Royal Astronomical Society, 493, 3132–3158. doi:10.1093/mnras/staa278

See Also

Other ernest_lrps: multi_ellipsoid(), rwmh_cube(), slice_rectangle(), unif_ellipsoid()

Examples

```
data(example_run)
lrps <- unif_cube()

ernest_sampler(example_run$log_lik_fn, example_run$prior, sampler = lrps)
```

`unif_ellipsoid` *Generate new points from the spanning ellipsoid*

Description

Uses the bounding ellipsoid of the live set to define the region of prior space that contains new points. Effective for unimodal and roughly-Gaussian posteriors.

Usage

```
unif_ellipsoid(enlarge = 1.25)
```

Arguments

`enlarge`	`[double(1)]` Factor by which to inflate the bounding ellipsoid's volume before sampling (see Details). Must be at least 1.0.

Details

Nested likelihood contours rarely form perfect ellipses, so sampling from the spanning ellipsoid without enlargement may exclude valid regions. This can bias proposals towards the ellipsoid centre and overestimate evidence. Setting `enlarge = 1` will produce a warning.

The covariance matrix of the points is used to estimate the ellipsoid's shape. In exceptional cases (e.g., perfect collinearity), this matrix may be singular. Should this occur, the covariance matrix is reconditioned by adjusting its eigenvalues. Should this also fail, the algorithm falls back to sampling from the circumscribed sphere bounding the unit hypercube.

Value

`[unif_ellipsoid]`, a named list that inherits from [ernest_lrps].

Ellipsoids

Ellipsoids are stored in the `cache` environment of the LRPS object. Ellipsoids are defined by their centre c and shape matrix A. The set of points x contained within the ellipsoid is given by

$$x \in \mathbf{R}^n | (x-c)A(x-c)' \leq 1$$

The volume of the ellipsoid is $V = \mathrm{Vol}(S_n)\sqrt{\det(A^{-1})}$, where $\mathrm{Vol}(S_n)$ is the volume of the unit hypersphere.

For sampling, we store the matrix $A^{-1/2}$, the inverse of the positive-semidefinite square root of A. The ellipsoid can equivalently be defined as the set of points

$$x = A^{-1/2}y + c,$$

where y are points from the unit hypersphere.

For more on ellipsoids and their operations, see Algorithms for Ellipsoids by S.B. Pope, Cornell University Report FDA 08-01 (2008).

References

Feroz, F., Hobson, M. P., Bridges, M. (2009) MULTINEST: An Efficient and Robust Bayesian Inference Tool for Cosmology and Particle Physics. Monthly Notices of the Royal Astronomical Society. 398(4), 1601–1614. doi:10.1111/j.13652966.2009.14548.x

Mukherjee, P., Parkinson, D., & Liddle, A. R. (2006). A Nested Sampling Algorithm for Cosmological Model Selection. The Astrophysical Journal, 638(2), L51. doi:10.1086/501068

See Also

Other ernest_lrps: `multi_ellipsoid()`, `rwmh_cube()`, `slice_rectangle()`, `unif_cube()`

Examples

```
data(example_run)
lrps <- unif_ellipsoid(enlarge = 1.25)

ernest_sampler(example_run$log_lik_fn, example_run$prior, sampler = lrps)
```

---

visualize.ernest_run *Visualize posterior distributions or traces from a nested sampling run*

Description

Produces visualizations of the posterior distributions or the evolution of variables along the log-prior volume from a nested sampling run.

Usage

```
## S3 method for class 'ernest_run'
visualize(
  x,
  ...,
  .which = c("density", "trace"),
  .units = c("original", "unit_cube")
)
```

Arguments

x	[ernest_run] The results of a nested sampling run.
...	<tidy-select> One or more variables to plot from the run. If omitted, all variables are plotted.
.which	[charcacter(1)] Character string specifying the type of plot to produce. Options are "density" for the posterior density of each parameter or "trace" for the trace of variables along log-volume.
.units	[character(1)] The scale of the sampled points: • "original": Points are on the scale of the prior space. • "unit_cube": Points are on the (0, 1) unit hypercube scale.

Details

The `visualize()` function is designed to quickly explore the results of a nested sampling run through two types of plots:

- **Density plots** show the marginal posterior for each selected variable, using `ggdist::stat_halfeye()` to visualize uncertainty and distribution shape.
- **Trace plots** display the evolution of variables as a function of log-volume, with points coloured by posterior weight. This can help diagnose sampling behavior and identify regions of interest in the prior volume.

Posterior weights are derived from the individual contributions of each sampled point in the prior space to a run's log-evidence estimate. A point's weight is a function of (a) the point's likelihood and (b) the estimated amount of volume within that point's likelihood contour. See ernest's vignettes for more information.

Value

A `ggplot2::ggplot()` object.

Note

This function requires **tidyselect**. If `which = "trace"` is selected, **ggdist** is also required.

See Also

`as_draws_rvars()`

Other visualizations: `plot.ernest_estimate()`

Examples

```
# Load example run
library(ggdist)
data(example_run)

# Plot posterior densities for all parameters
visualize(example_run, .which = "density")
```

`weights.ernest_run` *Extract the posterior sample weights from a nested sampling run*

Description

Return the normalised posterior importance weights for the dead points in a nested sampling run.

Usage

```
## S3 method for class 'ernest_run'
weights(object, log = FALSE, ...)
```

Arguments

`object`	[ernest_run] A nested sampling run.
`log`	`[logical(1)]` Whether to return the weights on the log scale.
`...`	These dots are for future extensions and must be empty.

Details

The log-weights in a nested sampling run are the individual contributions of each sample to the log-evidence estimate. The unnormalised weight of the ith sampled point is given as

$$w_i = \frac{L_{i-1} + L_i}{2} * (V_{i-1} - V_i)$$

where L_i is the likelihood value for the point and V_i is the prior volume at which the point was sampled.

The posterior importance weights are obtained by normalising the log-weights with the final log-evidence estimate. They can be used to reweight posterior samples from the run so they approximate the posterior distribution.

Value

`[double()]` A numeric vector of normalised importance weights. When `log = FALSE`, the values are exponentiated so they sum to one.

See Also

as_draws.ernest_run

Examples

```
data(example_run)
weights(example_run) |> head()
weights(example_run, log = TRUE) |> head()
```

Index