Package 'rmcmc'

February 4, 2025

Title Robust Markov Chain Monte Carlo Methods

Version 0.1.1

Description Functions for simulating Markov chains using the Barker proposal to compute Markov chain Monte Carlo (MCMC) estimates of expectations with respect to a target distribution on a real-valued vector space. The Barker proposal, described in Livingstone and Zanella (2022) <doi:10.1111/rssb.12482>, is a gradient-based MCMC algorithm inspired by the Barker accept-reject rule. It combines the robustness of simpler MCMC schemes, such as random-walk Metropolis, with the efficiency of gradient-based methods, such as the Metropolis adjusted Langevin algorithm. The key function provided by the package is sample_chain(), which allows sampling a Markov chain with a specified target distribution as its stationary distribution. The chain is sampled by generating proposals and accepting or rejecting them using a Metropolis-Hasting acceptance rule. During an initial warm-up stage, the parameters of the proposal distribution can be adapted, with adapters available to both: tune the scale of the proposals by coercing the average acceptance rate to a target value; tune the shape of the proposals to match covariance estimates under the target distribution. As well as the default Barker proposal, the package also provides implementations of alternative proposal distributions, such as (Gaussian) random walk and Langevin proposals. Optionally, if 'BridgeStan's R interface <https://roualdes.github.io/bridgestan/latest/languages/r.html>, available on GitHub <https://github.com/roualdes/bridgestan>, is installed, then 'BridgeStan' can be used to specify the target distribution to sample from.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests bridgestan (>= 2.5.0), knitr, posterior, progress, ramcmc, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/UCL/rmcmc, http://github-pages.ucl.ac.uk/rmcmc/

BugReports https://github.com/UCL/rmcmc/issues

Imports Matrix, rlang, withr

VignetteBuilder knitr

NeedsCompilation no

Author Matthew M. Graham [aut, cre] (<https://orcid.org/0000-0001-9104-7960>),
Samuel Livingstone [aut] (<https://orcid.org/0000-0002-7277-086X>),
University College London [cph],
Engineering and Physical Sciences Research Council [fnd]

Maintainer Matthew M. Graham <m.graham@ucl.ac.uk>

Repository CRAN

Date/Publication 2025-02-04 17:50:02 UTC

Contents

`barker_proposal` *Create a new Barker proposal object.*

Description

The Barker proposal is a gradient-based proposal inspired by the Barker accept-reject rule and proposed in Livingstone and Zanella (2022). It offers improved robustness compared to alternative gradient-based proposals such as Langevin proposals.

Usage

```
barker_proposal(
  scale = NULL,
  shape = NULL,
  sample_auxiliary = stats::rnorm,
  sample_uniform = stats::runif
)
```

Arguments

scale	Scale parameter of proposal distribution. A non-negative scalar value determining scale of steps proposed.
shape	Shape parameter of proposal distribution. Either a vector corresponding to a diagonal shape matrix with per-dimension scaling factors, or a matrix allowing arbitrary linear transformations.
sample_auxiliary	Function which generates a random vector from auxiliary variable distribution.
sample_uniform	Function which generates a random vector from standard uniform distribution given an integer size.

Details

For more details see the vignette: `vignette("barker-proposal", package = "rmcmc")`

Value

Proposal object. A list with entries

- `sample`: a function to generate sample from proposal distribution given current chain state,
- `log_density_ratio`: a function to compute log density ratio for proposal for a given pair of current and proposed chain states,
- `update`: a function to update parameters of proposal,
- `parameters`: a function to return list of current parameter values.
- `default_target_accept_prob`: a function returning the default target acceptance rate to use for any scale adaptation.
- `default_initial_scale`: a function which given a dimension gives a default value to use for the initial proposal scale parameter.

References

Livingstone, S., & Zanella, G. (2022). The Barker proposal: combining robustness and efficiency in gradient-based MCMC. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(2), 496-523.

Examples

```
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)
proposal <- barker_proposal(scale = 1.)
state <- chain_state(c(0., 0.))
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
log_density_ratio <- proposal$log_density_ratio(
  state, proposed_state, target_distribution
)
proposal$update(scale = 0.5)
```

`bimodal_barker_proposal` *Create a new Barker proposal object with bimodal noise distribution.*

Description

Convenience function for creating a Barker proposal with bimodal auxiliary noise variable distribution, corresponding to equally-weighted normal components with shared variance `sigma` and means `±sqrt(1 - sigma^2)`. This choice of noise distribution was suggested in Vogrinc et al. (2023) and found to give improved performance over the default choice of a standard normal auxiliary noise distribution in a range of targets.

Usage

```
bimodal_barker_proposal(
  sigma = 0.1,
  scale = NULL,
  shape = NULL,
  sample_uniform = stats::runif
)
```

Arguments

`sigma`	Standard deviation of equally-weighted normal components in bimodal auxiliary noise distribution, with corresponding means of `±sqrt(1 - sigma^2)`.
`scale`	Scale parameter of proposal distribution. A non-negative scalar value determining scale of steps proposed.
`shape`	Shape parameter of proposal distribution. Either a vector corresponding to a diagonal shape matrix with per-dimension scaling factors, or a matrix allowing arbitrary linear transformations.
`sample_uniform`	Function which generates a random vector from standard uniform distribution given an integer size.

Details

For more details see the vignette: `vignette("adjusting-noise-distribution", package = "rmcmc")`

Value

Proposal object. A list with entries

- `sample`: a function to generate sample from proposal distribution given current chain state,
- `log_density_ratio`: a function to compute log density ratio for proposal for a given pair of current and proposed chain states,
- `update`: a function to update parameters of proposal,
- `parameters`: a function to return list of current parameter values.
- `default_target_accept_prob`: a function returning the default target acceptance rate to use for any scale adaptation.
- `default_initial_scale`: a function which given a dimension gives a default value to use for the initial proposal scale parameter.

References

Vogrinc, J., Livingstone, S., & Zanella, G. (2023). Optimal design of the Barker proposal and other locally balanced Metropolis–Hastings algorithms. *Biometrika*, 110(3), 579-595.

See Also

`barker_proposal()`

Examples

```
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)
proposal <- bimodal_barker_proposal(scale = 1.)
state <- chain_state(c(0., 0.))
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
log_density_ratio <- proposal$log_density_ratio(
  state, proposed_state, target_distribution
)
proposal$update(scale = 0.5)
```

`chain_state` *Construct a new chain state.*

Description

The chain state object provides cached access to target distribution log density and its gradient.

Usage

```
chain_state(position, momentum = NULL)
```

Arguments

`position`	Position component of chain state.
`momentum`	Momentum component of chain state. Optional.

Value

New chain state object. A list with entries

- `position`: A zero-argument function to evaluate position vector.
- `momentum`: A zero-argument function to evaluate momentum vector.
- `dimension`: A zero-argument function evaluate dimension of position and momentum vectors.
- `update`: A function accepting arguments `position` and `momentum` for updating the value of one or both of these state components.
- `copy`: A function for creating a copy of the state object including any cached values.
- `log_density`: A function accepting argument `target_distribution` for evaluating the log density of the target distribution at the current state, with caching of the value to avoid recomputation on subsequent calls.
- `gradient_log_density`: A function accepting argument `target_distribution` for evaluating the gradient of the log density of the target distribution at the current state, with caching of the value to avoid recomputation on subsequent calls.

Examples

```
state <- chain_state(c(0.1, -0.5))
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)
state$gradient_log_density(target_distribution)
```

`covariance_shape_adapter` *Create object to adapt proposal with shape based on estimate of target distribution covariance matrix.*

Description

Corresponds to Algorithm 2 in Andrieu and Thoms (2009), which is itself a restatement of method proposed in Haario et al. (2001).

Usage

```
covariance_shape_adapter(kappa = 1)
```

Arguments

`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate. Value of 1 (default) corresponds to computing empirical covariance matrix.

Details

Requires `ramcmc` package to be installed for access to efficient rank-1 Cholesky update function `ramcmc::chol_update`.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing*, 18, 343-373.

Haario, H., Saksman, E., & Tamminen, J. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7(2): 223-242.

Examples

```
proposal <- barker_proposal()
adapter <- covariance_shape_adapter()
adapter$initialize(proposal, chain_state(c(0, 0)))
```

dual_averaging_scale_adapter *Create object to adapt proposal scale to coerce average acceptance rate using dual averaging scheme of Nesterov (2009) and Hoffman and Gelman (2014).*

Description

Create object to adapt proposal scale to coerce average acceptance rate using dual averaging scheme of Nesterov (2009) and Hoffman and Gelman (2014).

Usage

```
dual_averaging_scale_adapter(
  initial_scale = NULL,
  target_accept_prob = NULL,
  kappa = 0.75,
  gamma = 0.05,
  iteration_offset = 10,
  mu = NULL
)
```

Arguments

`initial_scale`	Initial value to use for scale parameter. If not set explicitly a proposal and dimension dependent default will be used.
`target_accept_prob`	Target value for average accept probability for chain. If not set a proposal dependent default will be used.
`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate. Defaults to value recommended in Hoffman and Gelman (2014).
`gamma`	Regularization coefficient for (log) scale in dual averaging algorithm. Controls amount of regularization of (log) scale towards `mu`. Should be set to a non-negative value. Defaults to value recommended in Hoffman and Gelman (2014).
`iteration_offset`	Offset to chain iteration used for the iteration based weighting of the adaptation statistic error estimate. Should be set to a non-negative value. A value greater than zero has the effect of stabilizing early iterations. Defaults to value recommended in Hoffman and Gelman (2014).
`mu`	Value to regularize (log) scale towards. If `NULL` (the default), `mu` will be set to `log(10 * initial_scale)`, as recommended in Hoffman and Gelman (2014).

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Nesterov, Y. (2009). Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1), 221-259.

Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1), 1593-1623.

Examples

```
proposal <- barker_proposal()
adapter <- dual_averaging_scale_adapter(
  initial_scale = 1., target_accept_prob = 0.4
)
adapter$initialize(proposal, chain_state(c(0, 0)))
```

---

example_gaussian_stan_model

Construct an example BridgeStan `StanModel` *object for a Gaussian model.*

Description

Requires BridgeStan package to be installed. Generative model is assumed to be of the form `y ~ normal(mu, sigma)` for unknown `mu ~ normal(0, 3)` and `sigma ~ half_normal(0, 3)`.

Usage

```
example_gaussian_stan_model(n_data = 50, seed = 1234L)
```

Arguments

`n_data`	Number of independent data points `y` to generate and condition model against from `normal(0, 1)`.
`seed`	Integer seed for Stan model.

Value

BridgeStan StanModel object.

Examples

```
model <- example_gaussian_stan_model(n_data = 5)
model$param_names()
```

`hamiltonian_proposal` *Create a new Hamiltonian proposal object.*

Description

The Hamiltonian proposal augments the target distribution with normally distributed auxiliary momenta variables and simulates the dynamics for a Hamiltonian function corresponding to the negative logarithm of the density of the resulting joint target distribution using a leapfrog integrator, with the proposed new state being the forward integrate state with momenta negated to ensure reversibility.

Usage

```
hamiltonian_proposal(
  n_step,
  scale = NULL,
  shape = NULL,
  sample_auxiliary = function(state) stats::rnorm(state$dimension()),
  sample_n_step = NULL
)
```

Arguments

`n_step`	Number of leapfrog steps to simulate Hamiltonian dynamics for in each proposed move, or parameter passed to function specified by `sample_n_step` argument if not `NULL`.
`scale`	Scale parameter of proposal distribution. A non-negative scalar value determining scale of steps proposed.
`shape`	Shape parameter of proposal distribution. Either a vector corresponding to a diagonal shape matrix with per-dimension scaling factors, or a matrix allowing arbitrary linear transformations.
`sample_auxiliary`	A function which samples new values for auxiliary variables (corresponding to a linear transform of momentum) given current chain state, leaving their standard normal target distribution invariant. Defaults to a function sampling independent standard normal random variates but can be used to implement alternative updates such as partial momentum refreshment. Function should accept a single argument which is passed the current chain state.

`sample_n_step` Optionally a function which randomly samples number of leapfrog steps to simulate in each proposed move from some integer-valued distribution, or `NULL` (the default) to use a fixed deterministic number of steps as specified by `n_step` argument. If a function it should accept a single argument which will be passed the value of `n_step` which can be used to specify parameter(s) of distribution to sample number of steps from.

Value

Proposal object. A list with entries

- `sample`: a function to generate sample from proposal distribution given current chain state,
- `log_density_ratio`: a function to compute log density ratio for proposal for a given pair of current and proposed chain states,
- `update`: a function to update parameters of proposal,
- `parameters`: a function to return list of current parameter values.
- `default_target_accept_prob`: a function returning the default target acceptance rate to use for any scale adaptation.
- `default_initial_scale`: a function which given a dimension gives a default value to use for the initial proposal scale parameter.

References

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid Monte Carlo. *Physics Letters B*, 195(2), 216-222.

Neal, R. M. (2011). MCMC Using Hamiltonian Dynamics. In *Handbook of Markov Chain Monte Carlo* (pp. 113-162). Chapman and Hall/CRC.

Examples

```
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)

# Proposal with fixed number of leapfrog steps
proposal <- hamiltonian_proposal(scale = 1., n_step = 5)
state <- chain_state(c(0., 0.))
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
log_density_ratio <- proposal$log_density_ratio(
  state, proposed_state, target_distribution
)
proposal$update(scale = 0.5)

# Proposal with number of steps randomly sampled uniformly from 5:10
sample_uniform_int <- function(lower, upper) {
  lower + sample.int(upper - lower + 1, 1) - 1
```

```
}
proposal <- hamiltonian_proposal(
  scale = 1.,
  n_step = c(5, 10),
  sample_n_step = function(n_step) sample_uniform_int(n_step[1], n_step[2])
)
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)

# Proposal with partial momentum refreshment
partial_momentum_update <- function(state, phi = pi / 4) {
  momentum <- state$momentum()
  if (is.null(momentum)) {
    stats::rnorm(state$dimension())
  } else {
    cos(phi) * momentum + sin(phi) * stats::rnorm(length(momentum))
  }
}
proposal <- hamiltonian_proposal(
  scale = 1.,
  n_step = 1,
  sample_auxiliary = partial_momentum_update
)
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
```

`langevin_proposal` *Create a new Langevin proposal object.*

Description

The Langevin proposal is a gradient-based proposal corresponding to a Euler-Maruyama time discretisation of a Langevin diffusion.

Usage

```
langevin_proposal(scale = NULL, shape = NULL, sample_auxiliary = stats::rnorm)
```

Arguments

`scale`	Scale parameter of proposal distribution. A non-negative scalar value determining scale of steps proposed.
`shape`	Shape parameter of proposal distribution. Either a vector corresponding to a diagonal shape matrix with per-dimension scaling factors, or a matrix allowing arbitrary linear transformations.
`sample_auxiliary`	Function which generates a random vector from auxiliary variable distribution.

Value

Proposal object. A list with entries

- `sample`: a function to generate sample from proposal distribution given current chain state,
- `log_density_ratio`: a function to compute log density ratio for proposal for a given pair of current and proposed chain states,
- `update`: a function to update parameters of proposal,
- `parameters`: a function to return list of current parameter values.
- `default_target_accept_prob`: a function returning the default target acceptance rate to use for any scale adaptation.
- `default_initial_scale`: a function which given a dimension gives a default value to use for the initial proposal scale parameter.

References

Besag, J. (1994). "Comments on "Representations of knowledge in complex systems" by U. Grenander and MI Miller". *Journal of the Royal Statistical Society, Series B*. 56: 591–592.

Roberts, G. O., & Tweedie, R. L. (1996). Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli* 2 (4), 341 - 363.

Examples

```
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)
proposal <- langevin_proposal(scale = 1.)
state <- chain_state(c(0., 0.))
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
log_density_ratio <- proposal$log_density_ratio(
  state, proposed_state, target_distribution
)
proposal$update(scale = 0.5)
```

---

random_walk_proposal *Create a new (Gaussian) random walk proposal object.*

Description

The Gaussian random walk proposal samples a new proposed state by perturbing the current state with zero-mean normally distributed noise.

Usage

```
random_walk_proposal(
  scale = NULL,
  shape = NULL,
  sample_auxiliary = stats::rnorm
)
```

Arguments

scale	Scale parameter of proposal distribution. A non-negative scalar value determining scale of steps proposed.
shape	Shape parameter of proposal distribution. Either a vector corresponding to a diagonal shape matrix with per-dimension scaling factors, or a matrix allowing arbitrary linear transformations.
sample_auxiliary	Function which generates a random vector from auxiliary variable distribution.

Value

Proposal object. A list with entries

- `sample`: a function to generate sample from proposal distribution given current chain state,
- `log_density_ratio`: a function to compute log density ratio for proposal for a given pair of current and proposed chain states,
- `update`: a function to update parameters of proposal,
- `parameters`: a function to return list of current parameter values.
- `default_target_accept_prob`: a function returning the default target acceptance rate to use for any scale adaptation.
- `default_initial_scale`: a function which given a dimension gives a default value to use for the initial proposal scale parameter.

Examples

```
target_distribution <- list(log_density = function(x) -sum(x^2) / 2)
proposal <- random_walk_proposal(scale = 1.)
state <- chain_state(c(0., 0.))
withr::with_seed(
  876287L, proposed_state <- proposal$sample(state, target_distribution)
)
log_density_ratio <- proposal$log_density_ratio(
  state, proposed_state, target_distribution
)
proposal$update(scale = 0.5)
```

`robust_shape_adapter` *Create object to adapt proposal shape (and scale) using robust adaptive Metropolis algorithm of Vihola (2012).*

Description

Requires `ramcmc` package to be installed.

Usage

```
robust_shape_adapter(
  initial_scale = NULL,
  target_accept_prob = NULL,
  kappa = 0.6
)
```

Arguments

`initial_scale`	Initial value to use for scale parameter. If not set explicitly a proposal and dimension dependent default will be used.
`target_accept_prob`	Target value for average accept probability for chain. If not set a proposal dependent default will be used.
`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Vihola, M. (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22, 997-1008. doi:10.1007/s1122201192695

Examples

```
proposal <- barker_proposal()
adapter <- robust_shape_adapter(initial_scale = 1., target_accept_prob = 0.4)
adapter$initialize(proposal, chain_state(c(0, 0)))
```

`sample_chain` *Sample a Markov chain*

Description

Sample a Markov chain using Metropolis-Hastings kernel with a user-specified target distribution and proposal (defaulting to Barker proposal), optionally adapting proposal parameters in a warm-up stage.

Usage

```
sample_chain(
  target_distribution,
  initial_state,
  n_warm_up_iteration,
  n_main_iteration,
  proposal = barker_proposal(),
  adapters = list(scale_adapter(), shape_adapter()),
  show_progress_bar = TRUE,
  trace_warm_up = FALSE
)
```

Arguments

`target_distribution`

Target stationary distribution for chain. One of:

- A one-sided formula specifying expression for log density of target distribution which will be passed to `target_distribution_from_log_density_formula()` to construct functions to evaluate log density and its gradient using `deriv()`.
- A `bridgestan::StanModel` instance (requires `bridgestan` to be installed) specifying target model and data. Will be passed to `target_distribution_from_stan_model()` using default values for optional arguments - to override call `target_distribution_from_stan_mo` directly and pass the returned list as the `target_distribution` argument here.
- A list with named entries `log_density` and `gradient_log_density` corresponding to respectively functions for evaluating the logarithm of the (potentially unnormalized) density of the target distribution and its gradient (only required for gradient-based proposals). As an alternative to `gradient_log_density` an entry `value_and_gradient_log_density` may instead be provided which is a function returning both the value and gradient of the logarithm of the (unnormalized) density of the target distribution as a list under the names `value` and `gradient` respectively. The list may also contain a named entry `trace_function`, correspond to a function which given current chain state outputs a named vector or list of variables to trace on each main (non-adaptive) chain iteration. If a `trace_function` entry is not specified, then the default behaviour is to trace the position

component of the chain state along with the log density of the target distribution.

`initial_state` Initial chain state. Either a vector specifying just the position component of the chain state or a list output by `chain_state` specifying the full chain state.

`n_warm_up_iteration` Number of warm-up (adaptive) chain iterations to run.

`n_main_iteration` Number of main (non-adaptive) chain iterations to run.

`proposal` Proposal distribution object. Defaults to Barker proposal, that is the output of `barker_proposal()`. Proposal objects are lists which must minimally define entries `sample`, a function to generate sample from proposal distribution given current chain state and `log_density_ratio`, a function to compute log density ratio for proposal for a given pair of current and proposed chain states. If adapters are being used to adaptively tune the proposal scale and shape parameters, which is the default behaviour of `sample_chain`, then additionally the list must also define entries: `update` a function for updating parameters of proposal, `parameters` a function for getting current proposal parameter values, `default_target_accept_prob` a function for getting proposal specific default target acceptance probability for scale adaptation and `default_initial_scale` a function for getting proposal and dimension dependent default initial value for scale parameter.

`adapters` List of adapters to tune proposal parameters during warm-up. Defaults to using list with instances of `scale_adapter()` and `shape_adapter()`, corresponding to respectively, adapting the scale to coerce the average acceptance rate to a target value using a dual-averaging algorithm, and adapting the shape to an estimate of the covariance of the target distribution.

`show_progress_bar` Whether to show progress bars during sampling. Requires `progress` package to be installed to have an effect.

`trace_warm_up` Whether to record chain traces and adaptation / transition statistics during (adaptive) warm-up iterations in addition to (non-adaptive) main chain iterations.

Value

A list with entries

- `final_state`: the final chain state,
- `traces`: a matrix with named columns contained traced variables for each main chain iteration, with variables along columns and iterations along rows.
- `statistics`: a matrix with named columns containing transition statistics for each main chain iteration, with statistics along columns and iterations along rows.
- `warm_up_traces`: a matrix with named columns contained traced variables for each warm-up chain iteration, with variables along columns and iterations along rows. Only present if `trace_warm_up = TRUE`.
- `warm_up_statistics`: a matrix with named columns containing adaptation and transition statistics for each warm-up chain iteration, with statistics along columns and iterations along rows. Only present if `trace_warm_up = TRUE`.

Examples

```
target_distribution <- list(
  log_density = function(x) -sum(x^2) / 2,
  gradient_log_density = function(x) -x
)
withr::with_seed(876287L, {
  results <- sample_chain(
    target_distribution,
    initial_state = stats::rnorm(2),
    n_warm_up_iteration = 1000,
    n_main_iteration = 1000
  )
})
```

`scale_adapter` *Create object to adapt proposal scale to coerce average acceptance rate.*

Description

Create object to adapt proposal scale to coerce average acceptance rate.

Usage

```
scale_adapter(
  algorithm = "dual_averaging",
  initial_scale = NULL,
  target_accept_prob = NULL,
  ...
)
```

Arguments

`algorithm`	String specifying algorithm to use. One of: • "stochastic_approximation" to use a Robbins-Monro (1951) based scheme, • "dual_averaging" to use dual-averaging scheme of Nesterov (2009).
`initial_scale`	Initial value to use for scale parameter. If not set explicitly a proposal and dimension dependent default will be used.
`target_accept_prob`	Target value for average accept probability for chain. If not set a proposal dependent default will be used.
`...`	Any additional algorithmic parameters to pass through to `dual_averaging_scale_adapter()` or `stochastic_approximation_scale_adapter()`.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Nesterov, Y. (2009). Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 120(1), 221-259.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 400-407.

See Also

`dual_averaging_scale_adapter()`, `stochastic_approximation_scale_adapter()`

Examples

```
proposal <- barker_proposal()
adapter <- scale_adapter(initial_scale = 1., target_accept_prob = 0.4)
adapter$initialize(proposal, chain_state(c(0, 0)))
```

`shape_adapter` *Create object to adapt proposal shape.*

Description

Create object to adapt proposal shape.

Usage

```
shape_adapter(type = "covariance", kappa = 1)
```

Arguments

`type`	Type of shape adapter to use. One of: • "variance": Diagonal shape matrix adaptation based on estimates of target distribution variances (see `variance_shape_adapter()`), • "covariance": Dense shape matrix adaptation based on estimates of target distribution covariance matrix (see `covariance_shape_adapter()`).
`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate. Value of 1 (default) corresponds to computing empirical (co)variances.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

See Also

`variance_shape_adapter()`, `covariance_shape_adapter()`

Examples

```
proposal <- barker_proposal()
adapter <- shape_adapter()
adapter$initialize(proposal, chain_state(c(0, 0)))
```

---

stochastic_approximation_scale_adapter

Create object to adapt proposal scale to coerce average acceptance rate using a Robbins and Monro (1951) scheme.

Description

When combined with `covariance_shape_adapter()` corresponds to Algorithm 4 in Andrieu and Thoms (2009).

Usage

```
stochastic_approximation_scale_adapter(
  initial_scale = NULL,
  target_accept_prob = NULL,
  kappa = 0.6
)
```

Arguments

`initial_scale`	Initial value to use for scale parameter. If not set explicitly a proposal and dimension dependent default will be used.
`target_accept_prob`	Target value for average accept probability for chain. If not set a proposal dependent default will be used.
`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing*, 18, 343-373.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 400-407.

Examples

```
proposal <- barker_proposal()
adapter <- stochastic_approximation_scale_adapter(
  initial_scale = 1., target_accept_prob = 0.4
)
adapter$initialize(proposal, chain_state(c(0, 0)))
```

`target_distribution_from_log_density_formula` *Construct target distribution from a formula specifying log density.*

Description

Construct target distribution from a formula specifying log density.

Usage

```
target_distribution_from_log_density_formula(log_density_formula)
```

Arguments

`log_density_formula`
: Formula for which right-hand side specifies expression for logarithm of (unnormalized) density of target distribution.

Value

A list with entries

- `log_density`: A function to evaluate log density function for target distribution given current position vector.
- `value_and_gradient_log_density`: A function to evaluate value and gradient of log density function for target distribution given current position vector, returning as a list with entries `value` and `gradient`.

Examples

```
target_distribution <- target_distribution_from_log_density_formula(
  ~ (-(x^2 + y^2) / 8 - (x^2 - y)^2 - (x - 1)^2 / 10)
)
target_distribution$value_and_gradient_log_density(c(0.1, -0.3))
```

---

target_distribution_from_stan_model

Construct target distribution from a BridgeStan `StanModel` *object.*

Description

Construct target distribution from a BridgeStan `StanModel` object.

Usage

```
target_distribution_from_stan_model(
  model,
  include_log_density = TRUE,
  include_generated_quantities = FALSE,
  include_transformed_parameters = FALSE
)
```

Arguments

`model`	Stan model object to use for target (posterior) distribution.
`include_log_density`	Whether to include an entry `log_density` corresponding to current log density for target distribution in values returned by trace function.
`include_generated_quantities`	Whether to included generated quantities in Stan model definition in values returned by trace function.
`include_transformed_parameters`	Whether to include transformed parameters in Stan model definition in values returned by trace function.

Value

A list with entries

- `log_density`: A function to evaluate log density function for target distribution given current position vector.
- `value_and_gradient_log_density`: A function to evaluate value and gradient of log density function for target distribution given current position vector, returning as a list with entries `value` and `gradient`.
- `trace_function`: A function which given a `chain_state()` object returns a named vector of values to trace during sampling. The constrained parameter values of model will always be included.

Examples

```
model <- example_gaussian_stan_model()
target_distribution <- target_distribution_from_stan_model(model)
withr::with_seed(
  876287L, state <- chain_state(stats::rnorm(model$param_unc_num()))
)
state$log_density(target_distribution)
target_distribution$trace_function(state)
```

---

variance_shape_adapter

Create object to adapt proposal with per dimension scales based on estimates of target distribution variances.

Description

Corresponds to variance variant of Algorithm 2 in Andrieu and Thoms (2009), which is itself a restatement of method proposed in Haario et al. (2001).

Usage

```
variance_shape_adapter(kappa = 1)
```

Arguments

`kappa`	Decay rate exponent in `[0.5, 1]` for adaptation learning rate. Value of 1 (default) corresponds to computing empirical variances.

Value

List of functions with entries

- `initialize`, a function for initializing adapter state and proposal parameters at beginning of chain,
- `update` a function for updating adapter state and proposal parameters on each chain iteration,
- `finalize` a function for performing any final updates to adapter state and proposal parameters on completion of chain sampling (may be `NULL` if unused).
- `state` a zero-argument function for accessing current values of adapter state variables.

References

Andrieu, C., & Thoms, J. (2008). A tutorial on adaptive MCMC. *Statistics and Computing*, 18, 343-373.

Haario, H., Saksman, E., & Tamminen, J. (2001). An adaptive Metropolis algorithm. *Bernoulli*, 7(2): 223-242.

Examples

```
proposal <- barker_proposal()
adapter <- variance_shape_adapter()
adapter$initialize(proposal, chain_state(c(0, 0)))
```

Index