Package: nutpieR (via r-universe)

June 6, 2026

Title R Bindings for the Nutpie NUTS Sampler

Version 1.7.5

Description Lightweight R interface to the nutpie No-U-Turn Sampler (NUTS) implemented in Rust (nuts-rs), using BridgeStan for model gradients.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports digest, jsonlite, posterior

Suggests bayesplot, posteriordb, reticulate, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/rextendr/version 0.4.2

SystemRequirements Cargo (Rust's package manager), rustc >= 1.85, GNU make

Depends R (>= 4.2)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs make libclang-dev

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

Date/Publication 2026-06-06 14:47:17 UTC

RemoteUrl https://github.com/andytimm/nutpieR

RemoteRef HEAD

RemoteSha d64060c6e8a70ac490302df7bc213bf87d380357

Contents

`nutpie_cache_dir` *Path to the nutpieR compile cache directory*

Description

Returns the directory under which `nutpie_compile_model()` stores its content-hashed compile artifacts (one subdirectory per unique source + flags + BridgeStan version). Useful for inspection, troubleshooting, or `unlink()`-ing a single entry.

Usage

```
nutpie_cache_dir()
```

Value

A character string with the path to the cache root.

Examples

```
nutpie_cache_dir()
```

`nutpie_clear_cache` *Clear the nutpieR compile cache*

Description

Removes the current resolved compile cache tree under `nutpie_cache_dir()`. Cached compiled models will be recompiled on next use.

Usage

```
nutpie_clear_cache()
```

Value

Invisibly `NULL`.

Warning

This deletes the underlying `_model.so` files. If you hold a `nutpie_model` object whose library hasn't been opened yet (no prior `nutpie_sample()` call on it), its `lib_path` will point at a deleted file and subsequent use will fail. Models that were already opened in the current session keep working — once loaded, the OS retains the mapped library independently of the file on disk.

Only the *active* cache root is cleared. If `R_USER_CACHE_DIR` was previously unset (or pointed somewhere else) and a different root was resolved earlier in the session, that older directory is left alone so models still backed by it remain valid.

Examples

```
nutpie_clear_cache()
```

---

nutpie_compile_model *Compile a Stan model*

Description

Compiles a Stan model to a shared library using BridgeStan. Downloads BridgeStan sources on first use (this is slow).

Usage

```
nutpie_compile_model(
  stan_file = NULL,
  code = NULL,
  stanc_args = character(),
  compile_args = character(),
  verbose = 1L,
  cache = TRUE
)
```

Arguments

`stan_file`	Path to a `.stan` file. Exactly one of `stan_file` or `code` must be provided.
`code`	A string containing Stan model code.
`stanc_args`	Character vector of extra arguments passed to the `stanc` compiler (e.g., `"--O1"` for optimization).
`compile_args`	Character vector of extra arguments passed to `make` during compilation.
`verbose`	Integer controlling compilation output. `0` = silent, `1` (default) = print status messages. Note: full make/stanc output (verbose=2) is not yet supported because bridgestan captures subprocess output internally rather than streaming it.
`cache`	Logical, default `TRUE`. When `TRUE`, reuse a previously compiled artifact when the source, BridgeStan version, and compile flags all match. When `FALSE`, compile to a fresh tempdir without touching the persistent cache.

Value

An object of class "nutpie_model" containing the path to the compiled shared library.

Caching

Compiled artifacts are stored in a content-hashed cache under nutpie_cache_dir() (one subdirectory per unique source + flags + BridgeStan version), regardless of whether the model was passed as stan_file = ... or code = "...". A subsequent call with identical inputs is a near-instant cache hit.

For stan_file = ..., the transitive #include set is hashed together with the main file, so editing an included file (or the main file itself) busts the cache and triggers a recompile.

The cache is bounded by nutpie_prune_cache(), which runs automatically at the end of every successful compile (cap: 16 entries, min age before eviction: 14 days).

Cache controls:

- cache = FALSE on a single call — compile to a fresh tempdir for this call only, without touching the persistent cache.
- Sys.setenv(NUTPIER_DISABLE_COMPILE_CACHE = "1") — same effect process-wide.
- nutpie_clear_cache() wipes the cache.

Note on storage location

Prior nutpieR versions wrote <basename>_model.so *next to* the source .stan file, matching cmdstanr's convention. nutpieR now uses a content-hashed cache directory instead. This change is required for correctness: when the same .stan path is reloaded after a recompile, the OS dynamic linker (dlopen) returns the previously loaded library rather than the new one, so edits silently had no effect (see GitHub issue #23). Distinct content → distinct path → fresh dlopen. Any stale <basename>_model.so and <basename>_model.cache_meta files left over from earlier versions can be deleted; nutpieR no longer reads or writes them.

Examples

```
## Not run:
# From a .stan file
model <- nutpie_compile_model(stan_file = "my_model.stan")

# From an inline code string
model <- nutpie_compile_model(code = "
  data { int<lower=0> N; array[N] int<lower=0,upper=1> y; }
  parameters { real<lower=0,upper=1> theta; }
  model { theta ~ beta(1, 1); y ~ bernoulli(theta); }
")

## End(Not run)
```

`nutpie_diagnostics` *Extract sampler diagnostics from nutpie draws*

Description

Diagnostics are extracted directly from the nuts-rs sample-stats schema, so the exact set of fields depends on the installed nuts-rs version and the sampling options used. Count fields (`depth`, `n_steps`, `chain`, `draw`, `index_in_trajectory`) are returned as R `integer` when every value fits in `i32`, else as `numeric`; floating-point fields (`logp`, `energy`, `step_size`, etc.) are always `numeric`. `NA`s use the matching sentinel (`NA_integer_` / `NA_real_`).

Usage

```
nutpie_diagnostics(draws)
```

Arguments

`draws`	A `posterior::draws_array` returned by `nutpie_sample()`.

Value

A `nutpie_diagnostics` object (a named list with a print method). Commonly available scalar fields: `diverging`, `tuning`, `maxdepth_reached` (logical); `depth`, `n_steps`, `chain`, `draw`, `index_in_trajectory` (integer when fits in `i32`, else numeric); `logp`, `energy`, `energy_error`, `step_size`, `step_size_bar`, `mean_tree_accept`, `mean_tree_accept_sym` (numeric). Wide fields (one row per draw): when `store_unconstrained = TRUE`, `unconstrained_draw` (`NA` rows where unrecorded); when `store_gradient = TRUE`, `gradient` (`NA` rows where unrecorded); when `store_mass_matrix = TRUE` **and** `save_warmup = TRUE`, `mass_matrix_inv` (and `mass_matrix_eigvals` / `mass_matrix_stds` when reported), with the most recently recorded value carried forward into draws between updates — the inverse mass matrix is piecewise-constant between adapter steps, not undefined. Requires `save_warmup = TRUE` because adaptation (the only time the matrix changes) occurs during warmup; without warmup draws in the trace there is no value to carry forward. These surface as `(n_draws * n_chains, ndim_unc)` numeric matrices when every recorded row has the same width; mixed-width columns fall back to a list of numeric vectors (one per draw, `NULL` when not recorded). With `store_divergences = TRUE`: `divergence_start`, `divergence_end`, `divergence_momentum`, `divergence_start_gradient` (lists, only present when at least one draw diverged).

Indexing conventions

`chain` is 1-indexed in `1:num_chains`. `draw` is 1-indexed in `1:num_draws` for post-warmup diagnostics and `1:num_warmup` for warmup diagnostics (returned via `nutpie_warmup_diagnostics()`). Matches `posterior::draws_array` conventions, so a `data.frame` of diagnostics joins cleanly against draws indexed by `(chain, iteration)`.

Examples

```
## Not run:
draws <- nutpie_sample(model, data = dat, num_draws = 1000, num_chains = 4)
diag <- nutpie_diagnostics(draws)
diag                              # printed summary
sum(diag$diverging)               # divergence count
max(diag$depth)                   # peak treedepth across all draws

## End(Not run)
```

`nutpie_nuts_params` *NUTS sampler parameters in bayesplot's long format*

Description

Reshapes the diagnostics from `nutpie_diagnostics()` into the four-column long-format `data.frame` that bayesplot's NUTS plotting helpers (e.g. `bayesplot::mcmc_pairs(np = ...)`, `bayesplot::mcmc_nuts_energy()`) expect. Names match Stan's CSV convention (`accept_stat__`, `divergent__`, `treedepth__`, `n_leapfrog__`, `stepsize__`, `energy__`). Other bayesplot NUTS helpers (e.g. `mcmc_nuts_divergence()`, `mcmc_nuts_acceptance()`) additionally need a per-draw `lp` data frame, which this helper does not produce.

Usage

```
nutpie_nuts_params(draws)
```

Arguments

`draws`	A `posterior::draws_array` returned by `nutpie_sample()`.

Value

A `data.frame` with columns:

`Chain` Integer chain index (1-indexed).

`Iteration` Integer post-warmup draw index (1-indexed within chain, in `1:num_draws`).

`Parameter` Character; one of `"accept_stat__"`, `"divergent__"`, `"treedepth__"`, `"n_leapfrog__"`, `"stepsize__"`, `"energy__"`.

`Value` Numeric value of the corresponding diagnostic.

The data frame has `num_draws * num_chains * 6` rows.

See Also

`bayesplot::mcmc_pairs()` for the most common consumer of this format. `nutpie_diagnostics()` for the raw diagnostics.

Examples

```
## Not run:
draws <- nutpie_sample(model, data = dat, num_chains = 4)
np <- nutpie_nuts_params(draws)
bayesplot::mcmc_pairs(draws, np = np, pars = c("mu", "tau"))

## End(Not run)
```

`nutpie_param_names` *Parameter names of a compiled Stan model*

Description

Returns the parameter names of a compiled Stan model, converted from BridgeStan's dot-indexed form (`"beta.1.2"`) to Stan's bracket convention (`"beta[1,2]"`).

Usage

```
nutpie_param_names(
  model,
  data = NULL,
  which = c("block", "unconstrained", "full"),
  unconstrained = NULL
)
```

Arguments

`model`	A `"nutpie_model"` object from `nutpie_compile_model()`.
`data`	Model data (same format as `nutpie_sample()`'s `data` argument). Required if the model has a `data` block.
`which`	One of `"block"`, `"unconstrained"`, `"full"`:
	`"block"` **(default)** Block-level parameter names — the names you pass as keys to `nutpie_sample()`'s `init` argument.
	`"unconstrained"` Unconstrained internal parameter names.
	`"full"` Block parameters plus transformed parameters and generated quantities (the full output draws space).
`unconstrained`	Deprecated. If non-`NULL`, overrides `which`: `TRUE` maps to `"unconstrained"`, `FALSE` maps to `"full"`. Emits a deprecation warning and will be removed in a future version.

Value

A character vector of parameter names.

Examples

```
## Not run:
model <- nutpie_compile_model(stan_file = "my_model.stan")
nutpie_param_names(model, data = dat)                       # block (default)
nutpie_param_names(model, data = dat, which = "unconstrained")
nutpie_param_names(model, data = dat, which = "full")       # incl. TP + GQ

## End(Not run)
```

`nutpie_prune_cache` *Prune the nutpieR compile cache*

Description

Evicts older entries from the nutpieR compile cache so it stays bounded. Mirrors Python nutpie's policy: cap the cache at `max_entries`, but only evict entries at least `min_age_days` old (oldest first). Called automatically at the end of every successful compile – you usually don't need to invoke it directly, but it's here for one-off manual cleanup or scripted maintenance.

Usage

```
nutpie_prune_cache(max_entries = 16L, min_age_days = 14L)
```

Arguments

`max_entries`	Maximum number of valid (fully compiled) cache entries to retain. Defaults to 16.
`min_age_days`	Minimum age (in days, by `ok` marker mtime) before an entry is eligible for eviction. Defaults to 14, so frequently re-used models aren't evicted just because the cache is hot.

Value

Invisibly, the number of entries removed.

Examples

```
nutpie_prune_cache()
nutpie_prune_cache(max_entries = 8, min_age_days = 7)
```

`nutpie_sample` *Sample from a Stan model using the NUTS sampler*

Description

Runs the nuts-rs NUTS sampler on a compiled Stan model.

Usage

```
nutpie_sample(
  model,
  data = NULL,
  num_draws = 1000L,
  num_warmup = 400L,
  num_chains = 4L,
  seed = NULL,
  max_treedepth = NULL,
  mindepth = NULL,
  target_accept = NULL,
  max_energy_error = NULL,
  extra_doublings = NULL,
  refresh = 100L,
  init = NULL,
  init_mean = NULL,
  save_warmup = FALSE,
  cores = NULL,
  pars = NULL,
  include = TRUE,
  store_divergences = FALSE,
  store_mass_matrix = FALSE,
  store_unconstrained = FALSE,
  store_gradient = FALSE,
  adaptation = c("diag", "low_rank", "low-rank"),
  low_rank_modified_mass_matrix = FALSE,
  mass_matrix_gamma = NULL,
  mass_matrix_eigval_cutoff = NULL
)
```

Arguments

`model`	A `"nutpie_model"` object from `nutpie_compile_model()`, or a path to a compiled shared library.
`data`	Model data. Can be: • A named list (will be converted to JSON via `jsonlite::toJSON()`) • A JSON string • A path to a `.json` file

- NULL for models with no data block

num_draws — Number of post-warmup draws per chain.

num_warmup — Number of warmup (tuning) draws per chain.

num_chains — Number of parallel chains.

seed — Random seed for reproducibility.

max_treedepth — Maximum tree depth for NUTS. The number of leapfrog steps per draw is at most 2^max_treedepth. Default NULL keeps the nuts-rs default (currently 10).

mindepth — Minimum tree depth for NUTS. The number of leapfrog steps per draw is at least 2^mindepth. Default NULL keeps the nuts-rs default (currently 0).

target_accept — Target acceptance probability for step size adaptation. Default NULL keeps the nuts-rs default (currently 0.8).

max_energy_error — Energy-error threshold above which a leapfrog step is treated as a divergence. Default NULL keeps the nuts-rs default.

extra_doublings — Number of additional tree doublings allowed after a turning point is reached. Default NULL keeps the nuts-rs default (currently 0).

refresh — How often to print progress updates, in draws per chain. Set to 0 to suppress progress output. Default is 100.

init — Initial values for each chain. Single entry point that dispatches on the input shape:

NULL **(default)** Each chain starts from a Uniform(-2, 2) draw on the unconstrained scale (nuts-rs default).

Scalar numeric x Each chain starts from a Uniform(-x, x) draw on the unconstrained scale. x = 0 starts every chain at the origin. Must be non-negative.

Named list, e.g. list(mu = 0, sigma = 1) Constrained values used to start each chain. If fully specified, every chain starts at the same point. If partial (some parameters missing), each chain gets its own random fill for the missing parameters (per-chain seeds derived from seed).

List of num_chains **named lists** One constrained start per chain. Each element may be partial.

Function function(chain_id) ... Called once per chain with chain_id in 1:num_chains; must return a (possibly partial) named list of constrained parameter values.

Character path(s) A JSON file path, or a character vector of num_chains JSON file paths (CmdStan-style). Parsed values are treated as constrained named lists.

No jitter is applied; starting points are used exactly (after any random fill for partial constrained inits). To work on the unconstrained scale, see nutpie_unconstrain() to convert constrained values first.

Chain assignment is unspecified. When supplying per-chain starts (list-of-lists or function(chain_id)), the N positions are guaranteed to be distributed one-per-chain, but the mapping from list index / chain_id to the output chain dimension is not currently guaranteed. Use this to provide N dispersed starts; do

not rely on "chain 1 starts at element 1" for downstream identifiability. (Threading the true chain id requires an upstream change in nuts-rs; tracked for a future release.)

`init_mean` Deprecated. Scalar or numeric vector on the unconstrained scale, with ±0.5 uniform jitter per chain. Use `init = function(chain_id) ...`, `init = <scalar>` for Uniform(-x, x), or `init = <named list>` instead. Will be removed in a future version.

`save_warmup` If `TRUE`, warmup draws and diagnostics are attached as attributes. Retrieve them with `nutpie_warmup_draws()`.

`cores` Number of CPU cores to use for parallel sampling. Defaults to `min(num_chains, parallel::detectCores())`.

`pars` An optional character vector of block-level parameter names (e.g. `"beta"`, `"sigma"`). When supplied, only these parameters (or all parameters *except* these, depending on `include`) are returned in the output draws. By default (`NULL`), all parameters are returned. Parameter names should be the Stan block names, not indexed names — e.g. `"beta"` will match `beta`, `beta[1]`, `beta[2]`, etc. When the kept set excludes the entire transformed-parameter and/or generated-quantities blocks, those slices are skipped at sample time (see `NEWS.md` for benchmarks).

`include` Logical (default `TRUE`). If `TRUE`, `pars` specifies the parameters to *keep* (whitelist). If `FALSE`, `pars` specifies parameters to *exclude* (blacklist). Ignored when `pars` is `NULL`.

`store_divergences`
If `TRUE`, store detailed information about each divergent transition (start/end positions, momentum, gradient). Surfaces as list columns on diagnostics, only present when at least one draw diverged. Default `FALSE`.

`store_mass_matrix`
If `TRUE`, store inverse mass matrix updates in diagnostics. Because adaptation occurs during warmup, these diagnostics require `save_warmup = TRUE`; otherwise warmup storage is skipped and no mass-matrix values are available to carry forward. Surfaces as a numeric matrix `(n_draws * n_chains, ndim_unc)` when uniform-width; falls back to a list of vectors when widths differ. Default `FALSE`.

`store_unconstrained`
If `TRUE`, store the unconstrained position at each draw on diagnostics (`unconstrained_draw`). Surfaces as a numeric matrix `(n_draws * n_chains, ndim_unc)` when uniform-width; falls back to a list of vectors when widths differ. For high-dimensional models this can rival the draws matrix in size. Default `FALSE`.

`store_gradient` If `TRUE`, store the log-density gradient at each draw on diagnostics (`gradient`). Same shape profile as `store_unconstrained`. Default `FALSE`.

`adaptation` Mass matrix adaptation strategy. One of:

`"diag"` **(default)** Diagonal mass matrix (the standard NUTS choice).

`"low_rank"` / `"low-rank"` Low-rank modified mass matrix adaptation. Captures posterior correlations and can improve efficiency on models with strongly correlated parameters.

Matches the Python nutpie adaptation= API. Additional strategies ("draw_diag", "flow") may be added in future releases.

low_rank_modified_mass_matrix
: Deprecated. If TRUE, equivalent to adaptation = "low_rank". Will be removed in a future release.

mass_matrix_gamma
: Regularisation parameter for low-rank mass matrix adaptation. Only used when adaptation = "low_rank"; ignored otherwise. Default NULL keeps the nuts-rs default (currently 1e-5).

mass_matrix_eigval_cutoff
: Eigenvalue cutoff for low-rank mass matrix. Eigenvalues outside (1/cutoff, cutoff) are ignored. Only used when adaptation = "low_rank"; ignored otherwise. Default NULL keeps the nuts-rs default (currently 2.0).

Value

A posterior::draws_array with dimensions (num_draws, num_chains, n_params). Sampler diagnostics are attached as an attribute and can be retrieved with nutpie_diagnostics(); see ?nutpie_diagnostics for the chain / draw indexing convention (1-indexed, phase-relative). The attributes "num_warmup" and "num_draws" record the sampling configuration (accessible via attr(draws, "num_warmup") etc.). The "sampler_config" attribute is a JSON string capturing the *effective* nuts-rs settings used (including any defaults that were left unspecified by the caller, and exposing num_warmup to match the function argument name); parse with jsonlite::fromJSON().

Examples

```
## Not run:
model <- nutpie_compile_model(code = "
  data { int<lower=0> N; array[N] int<lower=0,upper=1> y; }
  parameters { real<lower=0,upper=1> theta; }
  model { theta ~ beta(1, 1); y ~ bernoulli(theta); }
")

draws <- nutpie_sample(
  model,
  data = list(N = 10, y = c(0, 1, 0, 0, 0, 0, 0, 0, 0, 1)),
  num_draws = 1000, num_chains = 4, seed = 604
)

dim(draws)                      # (num_draws, num_chains, n_params)
posterior::variables(draws)
posterior::summarize_draws(draws)
nutpie_diagnostics(draws)

## End(Not run)
```

`nutpie_unconstrain` *Map a constrained parameter list to the unconstrained scale*

Description

Introspection / debugging helper. Takes a named list of parameter values in the constrained (user-facing) space and returns the corresponding unconstrained vector that BridgeStan uses internally. Useful when inspecting how Stan's unconstraining transform maps your values, or when sanity-checking a model's parameter order.

Usage

```
nutpie_unconstrain(model, params, data = NULL)
```

Arguments

`model`	A `"nutpie_model"` object.
`params`	A named list of parameter values. Names must match the parameter names in the Stan program. Values may be scalars, vectors, matrices, or arrays — matching the declared shape.
`data`	Model data (same format as `nutpie_sample()`'s `data`). Required if the model has a `data` block.

Details

For setting sampler starting points, pass constrained values directly to `nutpie_sample()`'s `init` argument (e.g. `init = list(mu = 0, sigma = 1)`) — this function is not part of the normal init workflow.

All parameters declared in the `parameters` block must be supplied.

Value

A named numeric vector whose names are the unconstrained parameter names (in BridgeStan's internal order).

Examples

```
## Not run:
model <- nutpie_compile_model(stan_file = "my_model.stan")
nutpie_unconstrain(model, params = list(mu = 0, sigma = 1), data = dat)

## End(Not run)
```

nutpie_warmup_diagnostics *Extract warmup diagnostics from nutpie output*

Description

Extract warmup diagnostics from nutpie output

Usage

```
nutpie_warmup_diagnostics(draws)
```

Arguments

draws	A posterior::draws_array returned by nutpie_sample() with save_warmup = TRUE.

Value

A named list of diagnostic vectors for the warmup phase. chain is 1-indexed and draw is 1-indexed in 1:num_warmup; see nutpie_diagnostics() for the full indexing convention.

Examples

```
## Not run:
draws <- nutpie_sample(model, data = dat, save_warmup = TRUE)
wd <- nutpie_warmup_diagnostics(draws)
max(wd$depth)                       # warmup treedepth peak

## End(Not run)
```

nutpie_warmup_draws *Extract warmup draws from nutpie output*

Description

Extract warmup draws from nutpie output

Usage

```
nutpie_warmup_draws(draws)
```

Arguments

draws	A posterior::draws_array returned by nutpie_sample() with save_warmup = TRUE.

Value

A `posterior::draws_array` containing the warmup draws, or `NULL` if warmup draws were not saved.

Examples

```
## Not run:
draws <- nutpie_sample(model, data = dat, save_warmup = TRUE)
warmup <- nutpie_warmup_draws(draws)
posterior::summarize_draws(warmup)

## End(Not run)
```

Index