Package: bridgestan (via r-universe)

January 18, 2025

Title BridgeStan, Accessing Stan Model Functions in R

Version 2.6.1

License BSD_3_clause

Description BridgeStan provides efficient in-memory access to the methods of a Stan model, including log densities, gradients, Hessians, and constraining and unconstraining transforms.

Encoding UTF-8

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE, r6 = TRUE)

Suggests testthat (>= 3.0.0), withr (>= 3.0.0)

Imports R6 (>= 2.4.0)

Config/testthat/edition 3

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

RemoteUrl https://github.com/roualdes/bridgestan

RemoteRef b161ca7a1d417213b084c4a8c2d53fa44f7e8284

RemoteSha b161ca7a1d417213b084c4a8c2d53fa44f7e8284

RemoteSubdir R

Contents

compile_model *Function* compile_model()

Description

Compiles a Stan model.

Usage

```
compile_model(stan_file, stanc_args = NULL, make_args = NULL)
```

Arguments

stan_file	A path to a Stan model file.
make_args	A vector of additional arguments to pass to Make. For example, `c('STAN_THREADS=True')` will enable threading for the compiled model. If the same flags are defined in `make/local`, the versions passed here will take precedent.
stanc_arg	A vector of arguments to pass to stanc3. For example, `c('--O1')` will enable compiler optimization level 1.

Details

Run BridgeStan's Makefile on a `.stan` file, creating the `.so` used by the StanModel class. This function checks that the path to BridgeStan is valid and will error if not. This can be set with `set_bridgestan_path`.

Value

Path to the compiled model.

See Also

`set_bridgestan_path()`

get_bridgestan_path *Get the path to BridgeStan.*

Description

By default this is set to the value of the environment variable `BRIDGESTAN`.

Usage

```
get_bridgestan_path(download = TRUE)
```

Details

If there is no path set and the argument `download` is TRUE, this function will download a copy of the BridgeStan source code for the currently installed version under a folder called `.bridgestan` in the user's home directory if one is not already present.

See Also

`set_bridgestan_path()`

`set_bridgestan_path` *Function* `set_bridgestan_path()`

Description

Set the path to BridgeStan.

Usage

```
set_bridgestan_path(path)
```

Details

This should point to the top-level folder of the repository.

`StanModel` *StanModel*

Description

R6 Class representing a compiled BridgeStan model.

This model exposes log density, gradient, and Hessian information as well as constraining and unconstraining transforms.

Methods

Public methods:

- `StanModel$new()`
- `StanModel$name()`
- `StanModel$model_info()`
- `StanModel$model_version()`
- `StanModel$param_names()`
- `StanModel$param_unc_names()`
- `StanModel$param_num()`
- `StanModel$param_unc_num()`

- StanModel$param_constrain()
- StanModel$new_rng()
- StanModel$param_unconstrain()
- StanModel$param_unconstrain_json()
- StanModel$log_density()
- StanModel$log_density_gradient()
- StanModel$log_density_hessian()
- StanModel$log_density_hessian_vector_product()

Method `new()`**:** Create a Stan Model instance.

Usage:

```
StanModel$new(
  lib,
  data,
  seed,
  stanc_args = NULL,
  make_args = NULL,
  warn = TRUE
)
```

Arguments:

`lib` A path to a compiled BridgeStan Shared Object file or a .stan file (will be compiled).

`data` Either a JSON string literal, a path to a data file in JSON format ending in ".json", or the empty string.

`seed` Seed for the RNG used in constructing the model.

`stanc_args` A list of arguments to pass to stanc3 if the model is not already compiled.

`make_args` A list of additional arguments to pass to Make if the model is not already compiled.

`warn` If false, the warning about re-loading the same shared object is suppressed.

Returns: A new StanModel.

Method `name()`**:** Get the name of this StanModel.

Usage:

```
StanModel$name()
```

Returns: A character vector of the name.

Method `model_info()`**:** Get compile information about this Stan model.

Usage:

```
StanModel$model_info()
```

Returns: A character vector of the Stan version and important flags.

Method `model_version()`**:** Get the version of BridgeStan used in the compiled model.

Usage:

```
StanModel$model_version()
```

Method `param_names()`**:** Return the indexed names of the (constrained) parameters. For containers, indexes are separated by periods (.).
For example, the scalar `a` has indexed name "a", the vector entry `a[1]` has indexed name "a.1" and the matrix entry `a[2, 3]` has indexed name "a.2.3". Parameter order of the output is column major and more generally last-index major for containers.

Usage:

```
StanModel$param_names(include_tp = FALSE, include_gq = FALSE)
```

Arguments:

`include_tp` Whether to include variables from transformed parameters.

`include_gq` Whether to include variables from generated quantities.

Returns: A list of character vectors of the names.

Method `param_unc_names()`**:** Return the indexed names of the unconstrained parameters. For containers, indexes are separated by periods (.).
For example, the scalar `a` has indexed name "a", the vector entry `a[1]` has indexed name "a.1" and the matrix entry `a[2, 3]` has indexed name "a.2.3". Parameter order of the output is column major and more generally last-index major for containers.

Usage:

```
StanModel$param_unc_names()
```

Returns: A list of character vectors of the names.

Method `param_num()`**:** Return the number of (constrained) parameters in the model.

Usage:

```
StanModel$param_num(include_tp = FALSE, include_gq = FALSE)
```

Arguments:

`include_tp` Whether to include variables from transformed parameters.

`include_gq` Whether to include variables from generated quantities.

Returns: The number of parameters in the model.

Method `param_unc_num()`**:** Return the number of unconstrained parameters in the model.
This function is mainly different from `param_num` when variables are declared with constraints. For example, `simplex[5]` has a constrained size of 5, but an unconstrained size of 4.

Usage:

```
StanModel$param_unc_num()
```

Returns: The number of parameters in the model.

Method `param_constrain()`**:** Returns a vector of constrained parameters given the unconstrained parameters. See also `StanModel$param_unconstrain()`, the inverse of this function.

Usage:

```
StanModel$param_constrain(
  theta_unc,
  include_tp = FALSE,
  include_gq = FALSE,
  rng
)
```

Arguments:

`theta_unc` The vector of unconstrained parameters.

`include_tp` Whether to also output the transformed parameters of the model.

`include_gq` Whether to also output the generated quantities of the model.

`rng` The random number generator to use if `include_gq` is `TRUE`. See `StanModel$new_rng()`.

Returns: The constrained parameters of the model.

Method `new_rng()`**:** Create a new persistent PRNG object for use in `param_constrain()`.

Usage:

```
StanModel$new_rng(seed)
```

Arguments:

`seed` The seed for the PRNG.

Returns: A `StanRNG` object.

Method `param_unconstrain()`**:** Returns a vector of unconstrained parameters give the constrained parameters.

It is assumed that these will be in the same order as internally represented by the model (e.g., in the same order as `StanModel$param_names()`). If structured input is needed, use `StanModel$param_unconstrain_json()`. See also `StanModel$param_constrain()`, the inverse of this function.

Usage:

```
StanModel$param_unconstrain(theta)
```

Arguments:

`theta` The vector of constrained parameters.

Returns: The unconstrained parameters of the model.

Method `param_unconstrain_json()`**:** This accepts a JSON string of constrained parameters and returns the unconstrained parameters.

The JSON is expected to be in the JSON Format for CmdStan.

Usage:

```
StanModel$param_unconstrain_json(json)
```

Arguments:

`json` Character vector containing a string representation of JSON data.

Returns: The unconstrained parameters of the model.

Method `log_density()`**:** Return the log density of the specified unconstrained parameters.

Usage:

```
StanModel$log_density(theta_unc, propto = TRUE, jacobian = TRUE)
```

Arguments:

`theta_unc` The vector of unconstrained parameters.

`propto` If `TRUE`, drop terms which do not depend on the parameters.

`jacobian` If `TRUE`, include change of variables terms for constrained parameters.

Returns: The log density.

Method `log_density_gradient()`**:** Return the log density and gradient of the specified unconstrained parameters.

Usage:

```
StanModel$log_density_gradient(theta_unc, propto = TRUE, jacobian = TRUE)
```

Arguments:

`theta_unc` The vector of unconstrained parameters.

`propto` If `TRUE`, drop terms which do not depend on the parameters.

`jacobian` If `TRUE`, include change of variables terms for constrained parameters.

Returns: List containing entries `val` (the log density) and `gradient` (the gradient).

Method `log_density_hessian()`**:** Return the log density, gradient, and Hessian of the specified unconstrained parameters.

Usage:

```
StanModel$log_density_hessian(theta_unc, propto = TRUE, jacobian = TRUE)
```

Arguments:

`theta_unc` The vector of unconstrained parameters.

`propto` If `TRUE`, drop terms which do not depend on the parameters.

`jacobian` If `TRUE`, include change of variables terms for constrained parameters.

Returns: List containing entries `val` (the log density), `gradient` (the gradient), and `hessian` (the Hessian).

Method `log_density_hessian_vector_product()`**:** Return the log density and the product of the Hessian with the specified vector.

Usage:

```
StanModel$log_density_hessian_vector_product(
  theta_unc,
  v,
  propto = TRUE,
  jacobian = TRUE
)
```

Arguments:

`theta_unc` The vector of unconstrained parameters.

`v` The vector to multiply the Hessian by.

`propto` If `TRUE`, drop terms which do not depend on the parameters.

`jacobian` If `TRUE`, include change of variables terms for constrained parameters.

Returns: List containing entries `val` (the log density) and `Hvp` (the hessian-vector product).

Index