Package: cmdstanr (via r-universe)

August 5, 2024

Title R Interface to 'CmdStan'

Version 0.8.1

Date 2024-06-06

Description A lightweight interface to 'Stan' <https://mc-stan.org>. The 'CmdStanR' interface is an alternative to 'RStan' that calls the command line interface for compilation and running algorithms instead of interfacing with C++ via 'Rcpp'. This has many benefits including always being compatible with the latest version of Stan, fewer installation errors, fewer unexpected crashes in RStudio, and a more permissive license.

License BSD_3_clause + file LICENSE

URL https://mc-stan.org/cmdstanr/, https://discourse.mc-stan.org

BugReports https://github.com/stan-dev/cmdstanr/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE, r6 = FALSE)

SystemRequirements CmdStan (https://mc-stan.org/users/interfaces/cmdstan)

Depends R (>= 3.5.0)

Imports checkmate, data.table, jsonlite (>= 1.2.0), posterior (>= 1.4.1), processx (>= 3.5.0), R6 (>= 2.4.0), withr (>= 2.5.0), rlang (>= 0.4.7)

Suggests bayesplot, ggplot2, knitr (>= 1.37), loo (>= 2.0.0), rmarkdown, testthat (>= 2.1.0), Rcpp

VignetteBuilder knitr

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

RemoteUrl https://github.com/stan-dev/cmdstanr

RemoteRef v0.8.1

RemoteSha 02259ef7aa2a8b1c8de2fa3fc42a9feafd789288

Contents

`cmdstanr-package` *CmdStanR: the R interface to CmdStan*

Description

CmdStanR: the R interface to CmdStan.

Details

CmdStanR (**cmdstanr** package) is an interface to Stan (mc-stan.org) for R users. It provides the necessary objects and functions to compile a Stan program and run Stan's algorithms from R via CmdStan, the shell interface to Stan (mc-stan.org/users/interfaces/cmdstan).

Different ways of interfacing with Stan's C++:

The RStan interface (**rstan** package) is an in-memory interface to Stan and relies on R packages like **Rcpp** and **inline** to call C++ code from R. On the other hand, the CmdStanR interface does not directly call any C++ code from R, instead relying on the CmdStan interface behind the scenes for compilation, running algorithms, and writing results to output files.

Advantages of RStan:

- Allows other developers to distribute R packages with *pre-compiled* Stan programs (like **rstanarm**) on CRAN. (Note: As of 2023, this can mostly be achieved with CmdStanR as well. See Developing using CmdStanR.)
- Avoids use of R6 classes, which may result in more familiar syntax for many R users.
- CRAN binaries available for Mac and Windows.

Advantages of CmdStanR:

- Compatible with latest versions of Stan. Keeping up with Stan releases is complicated for RStan, often requiring non-trivial changes to the **rstan** package and new CRAN releases of both **rstan** and **StanHeaders**. With CmdStanR the latest improvements in Stan will be available from R immediately after updating CmdStan using `cmdstanr::install_cmdstan()`.
- Running Stan via external processes results in fewer unexpected crashes, especially in RStudio.
- Less memory overhead.
- More permissive license. RStan uses the GPL-3 license while the license for CmdStanR is BSD-3, which is a bit more permissive and is the same license used for CmdStan and the Stan C++ source code.

Getting started

CmdStanR requires a working version of CmdStan. If you already have CmdStan installed see `cmdstan_model()` to get started, otherwise see `install_cmdstan()` to install CmdStan. The vignette *Getting started with CmdStanR* demonstrates the basic functionality of the package.

For a list of global options see cmdstanr_global_options.

Author(s)

Maintainer: Andrew Johnson <andrew.johnson@arjohnsonau.com> (ORCID)

Authors:

- Jonah Gabry <jsg2201@columbia.edu>
- Rok Češnovar <rok.cesnovar@fri.uni-lj.si>
- Steve Bronder

Other contributors:

- Ben Bales [contributor]
- Mitzi Morris [contributor]
- Mikhail Popov [contributor]
- Mike Lawrence [contributor]
- William Michael Landau <will.landau@gmail.com> (ORCID) [contributor]
- Jacob Socolar [contributor]
- Martin Modrák [contributor]
- Ven Popov [contributor]

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Useful links:

- https://mc-stan.org/cmdstanr/
- https://discourse.mc-stan.org
- Report bugs at https://github.com/stan-dev/cmdstanr/issues

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)
```

```
# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()

# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
```

```
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

`as_draws.CmdStanMCMC` *Create a* `draws` *object from a CmdStanR fitted model object*

Description

Create a `draws` object supported by the **posterior** package. These methods are just wrappers around CmdStanR's `$draws()` method provided for convenience.

Usage

```
## S3 method for class 'CmdStanMCMC'
as_draws(x, ...)

## S3 method for class 'CmdStanMLE'
as_draws(x, ...)

## S3 method for class 'CmdStanLaplace'
as_draws(x, ...)

## S3 method for class 'CmdStanVB'
as_draws(x, ...)

## S3 method for class 'CmdStanGQ'
as_draws(x, ...)
```

```
## S3 method for class 'CmdStanPathfinder'
as_draws(x, ...)
```

Arguments

x	A CmdStanR fitted model object.
...	Optional arguments passed to the $draws() method (e.g., variables, inc_warmup, etc.).

Details

To subset iterations, chains, or draws, use the posterior::subset_draws() method after creating the draws object.

Examples

```
## Not run:
fit <- cmdstanr_example()
as_draws(fit)

# posterior's as_draws_*() methods will also work
posterior::as_draws_rvars(fit)
posterior::as_draws_list(fit)

## End(Not run)
```

---

as_mcmc.list *Convert* CmdStanMCMC *to* mcmc.list

Description

This function converts a CmdStanMCMC object to an mcmc.list object compatible with the **coda** package. This is primarily intended for users of Stan coming from BUGS/JAGS who are used to **coda** for plotting and diagnostics. In general we recommend the more recent MCMC diagnostics in **posterior** and the **ggplot2**-based plotting functions in **bayesplot**, but for users who prefer **coda** this function provides compatibility.

Usage

```
as_mcmc.list(x)
```

Arguments

x	A CmdStanMCMC object.

Value

An `mcmc.list` object compatible with the **coda** package.

Examples

```
## Not run:
fit <- cmdstanr_example()
x <- as_mcmc.list(fit)

## End(Not run)
```

`CmdStanDiagnose` *CmdStanDiagnose objects*

Description

A `CmdStanDiagnose` object is the object returned by the `$diagnose()` method of a `CmdStanModel` object.

Methods

`CmdStanDiagnose` objects have the following associated methods:

Method	Description
`$gradients()`	Return gradients from diagnostic mode.
`$lp()`	Return the total log probability density (`target`).
`$init()`	Return user-specified initial values.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: `CmdStanGQ`, `CmdStanLaplace`, `CmdStanMCMC`, `CmdStanMLE`, `CmdStanPathfinder`, `CmdStanVB`

Examples

```
## Not run:
test <- cmdstanr_example("logistic", method = "diagnose")

# retrieve the gradients
test$gradients()

## End(Not run)
```

`CmdStanGQ` *CmdStanGQ objects*

Description

A `CmdStanGQ` object is the fitted model object returned by the `$generate_quantities()` method of a `CmdStanModel` object.

Methods

`CmdStanGQ` objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of generated quantities object:

Method	Description
`$draws()`	Return the generated quantities as a `draws_array`.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$code()`	Return Stan code as a character vector.

Summarize inferences:

Method	Description
`$summary()`	Run `posterior::summarise_draws()`.

Save fitted model object and temporary files:

Method	Description
`$save_object()`	Save fitted model object to a file.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.

Report run times, console output, return codes:

Method	Description
$time()	Report the total run time.
$output()	Return the stdout and stderr of all chains or pretty print the output for a single chain.
$return_codes()	Return the return codes from the CmdStan runs.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: CmdStanDiagnose, CmdStanLaplace, CmdStanMCMC, CmdStanMLE, CmdStanPathfinder, CmdStanVB

Examples

```
## Not run:
# first fit a model using MCMC
mcmc_program <- write_stan_file(
  "data {
    int<lower=0> N;
    array[N] int<lower=0,upper=1> y;
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  model {
    y ~ bernoulli(theta);
  }"
)
mod_mcmc <- cmdstan_model(mcmc_program)

data <- list(N = 10, y = c(1,1,0,0,0,1,0,1,0,0))
fit_mcmc <- mod_mcmc$sample(data = data, seed = 123, refresh = 0)

# stan program for standalone generated quantities
# (could keep model block, but not necessary so removing it)
gq_program <- write_stan_file(
  "data {
    int<lower=0> N;
    array[N] int<lower=0,upper=1> y;
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  generated quantities {
    array[N] int y_rep = bernoulli_rng(rep_vector(theta, N));
  }"
)
```

```
mod_gq <- cmdstan_model(gq_program)
fit_gq <- mod_gq$generate_quantities(fit_mcmc, data = data, seed = 123)
str(fit_gq$draws())

library(posterior)
as_draws_df(fit_gq$draws())

## End(Not run)
```

`CmdStanLaplace` *CmdStanLaplace objects*

Description

A `CmdStanLaplace` object is the fitted model object returned by the `$laplace()` method of a `CmdStanModel` object.

Methods

`CmdStanLaplace` objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of fitted model object:

Method	Description
`$draws()`	Return approximate posterior draws as a `draws_matrix`.
`$mode()`	Return the mode as a `CmdStanMLE` object.
`$lp()`	Return the total log probability density (`target`) computed in the model block of the Stan program.
`$lp_approx()`	Return the log density of the approximation to the posterior.
`$init()`	Return user-specified initial values.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$code()`	Return Stan code as a character vector.

Summarize inferences:

Method	Description
`$summary()`	Run `posterior::summarise_draws()`.

Save fitted model object and temporary files:

Method	Description
`$save_object()`	Save fitted model object to a file.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.

`$save_latent_dynamics_files()`	Save diagnostic CSV files to a specified location.

Report run times, console output, return codes:

Method	Description
`$time()`	Report the run time of the Laplace sampling step.
`$output()`	Pretty print the output that was printed to the console.
`$return_codes()`	Return the return codes from the CmdStan runs.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: `CmdStanDiagnose`, `CmdStanGQ`, `CmdStanMCMC`, `CmdStanMLE`, `CmdStanPathfinder`, `CmdStanVB`

`CmdStanMCMC` *CmdStanMCMC objects*

Description

A `CmdStanMCMC` object is the fitted model object returned by the `$sample()` method of a `CmdStanModel` object. Like `CmdStanModel` objects, `CmdStanMCMC` objects are R6 objects.

Methods

`CmdStanMCMC` objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of fitted model object:

Method	Description
`$draws()`	Return posterior draws using formats from the **posterior** package.
`$sampler_diagnostics()`	Return sampler diagnostics as a `draws_array`.
`$lp()`	Return the total log probability density (`target`).
`$inv_metric()`	Return the inverse metric for each chain.
`$init()`	Return user-specified initial values.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$num_chains()`	Return the number of MCMC chains.
`$code()`	Return Stan code as a character vector.

Summarize inferences and diagnostics:

Method	Description
`$print()`	Run `posterior::summarise_draws()`.
`$summary()`	Run `posterior::summarise_draws()`.
`$diagnostic_summary()`	Get summaries of sampler diagnostics and warning messages.
`$cmdstan_summary()`	Run and print CmdStan's `bin/stansummary`.
`$cmdstan_diagnose()`	Run and print CmdStan's `bin/diagnose`.
`$loo()`	Run `loo::loo.array()` for approximate LOO-CV

Save fitted model object and temporary files:

Method	Description
`$save_object()`	Save fitted model object to a file.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.
`$save_latent_dynamics_files()`	Save diagnostic CSV files to a specified location.

Report run times, console output, return codes:

Method	Description
`$output()`	Return the stdout and stderr of all chains or pretty print the output for a single chain.
`$time()`	Report total and chain-specific run times.
`$return_codes()`	Return the return codes from the CmdStan runs.

Expose Stan functions and additional methods to R:

Method	Description
`$expose_functions()`	Expose Stan functions for use in R.
`$init_model_methods()`	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
`$log_prob()`	Calculate log-prob.
`$grad_log_prob()`	Calculate log-prob and gradient.
`$hessian()`	Calculate log-prob, gradient, and hessian.
`$constrain_variables()`	Transform a set of unconstrained parameter values to the constrained scale.
`$unconstrain_variables()`	Transform a set of parameter values to the unconstrained scale.
`$unconstrain_draws()`	Transform all parameter draws to the unconstrained scale.
`$variable_skeleton()`	Helper function to re-structure a vector of constrained parameter values.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: `CmdStanDiagnose`, `CmdStanGQ`, `CmdStanLaplace`, `CmdStanMLE`, `CmdStanPathfinder`, `CmdStanVB`

CmdStanMLE *CmdStanMLE objects*

Description

A CmdStanMLE object is the fitted model object returned by the $optimize() method of a CmdStanModel object.

Methods

CmdStanMLE objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of fitted model object:

Method	Description
draws()	Return the point estimate as a 1-row draws_matrix.
$mle()	Return the point estimate as a numeric vector.
$lp()	Return the total log probability density (target).
$init()	Return user-specified initial values.
$metadata()	Return a list of metadata gathered from the CmdStan CSV files.
$code()	Return Stan code as a character vector.

Summarize inferences:

Method	Description
$summary()	Run posterior::summarise_draws().

Save fitted model object and temporary files:

Method	Description
$save_object()	Save fitted model object to a file.
$save_output_files()	Save output CSV files to a specified location.
$save_data_file()	Save JSON data file to a specified location.

Report run times, console output, return codes:

Method	Description
$time()	Report the total run time.
$output()	Pretty print the output that was printed to the console.
$return_codes()	Return the return codes from the CmdStan runs.

Expose Stan functions and additional methods to R:

Method	Description
`$expose_functions()`	Expose Stan functions for use in R.
`$init_model_methods()`	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
`$log_prob()`	Calculate log-prob.
`$grad_log_prob()`	Calculate log-prob and gradient.
`$hessian()`	Calculate log-prob, gradient, and hessian.
`$constrain_variables()`	Transform a set of unconstrained parameter values to the constrained scale.
`$unconstrain_variables()`	Transform a set of parameter values to the unconstrained scale.
`$unconstrain_draws()`	Transform all parameter draws to the unconstrained scale.
`$variable_skeleton()`	Helper function to re-structure a vector of constrained parameter values.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: `CmdStanDiagnose`, `CmdStanGQ`, `CmdStanLaplace`, `CmdStanMCMC`, `CmdStanPathfinder`, `CmdStanVB`

`CmdStanModel` *CmdStanModel objects*

Description

A `CmdStanModel` object is an R6 object created by the `cmdstan_model()` function. The object stores the path to a Stan program and compiled executable (once created), and provides methods for fitting the model using Stan's algorithms.

Methods

`CmdStanModel` objects have the following associated methods, many of which have their own (linked) documentation pages:

Stan code:

Method	Description
`$stan_file()`	Return the file path to the Stan program.
`$code()`	Return Stan program as a character vector.
`$print()`	Print readable version of Stan program.
`$check_syntax()`	Check Stan syntax without having to compile.
`$format()`	Format and canonicalize the Stan model code.

Compilation:

Method	Description
`$compile()`	Compile Stan program.
`$exe_file()`	Return the file path to the compiled executable.
`$hpp_file()`	Return the file path to the `.hpp` file containing the generated C++ code.
`$save_hpp_file()`	Save the `.hpp` file containing the generated C++ code.
`$expose_functions()`	Expose Stan functions for use in R.

Diagnostics:

Method	Description
`$diagnose()`	Run CmdStan's `"diagnose"` method to test gradients, return `CmdStanDiagnose` object.

Model fitting:

Method	Description
`$sample()`	Run CmdStan's `"sample"` method, return `CmdStanMCMC` object.
`$sample_mpi()`	Run CmdStan's `"sample"` method with MPI, return `CmdStanMCMC` object.
`$optimize()`	Run CmdStan's `"optimize"` method, return `CmdStanMLE` object.
`$variational()`	Run CmdStan's `"variational"` method, return `CmdStanVB` object.
`$pathfinder()`	Run CmdStan's `"pathfinder"` method, return `CmdStanPathfinder` object.
`$generate_quantities()`	Run CmdStan's `"generate quantities"` method, return `CmdStanGQ` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
```

```
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()

# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
```

```
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
```

```
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

`CmdStanPathfinder` *CmdStanPathfinder objects*

Description

A `CmdStanPathfinder` object is the fitted model object returned by the `$pathfinder()` method of a `CmdStanModel` object.

Methods

`CmdStanPathfinder` objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of fitted model object:

Method	Description
`$draws()`	Return approximate posterior draws as a `draws_matrix`.
`$lp()`	Return the total log probability density (`target`) computed in the model block of the Stan program.
`$lp_approx()`	Return the log density of the approximation to the posterior.
`$init()`	Return user-specified initial values.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$code()`	Return Stan code as a character vector.

Summarize inferences:

Method	Description
`$summary()`	Run `posterior::summarise_draws()`.
`$cmdstan_summary()`	Run and print CmdStan's `bin/stansummary`.

Save fitted model object and temporary files:

Method	Description
`$save_object()`	Save fitted model object to a file.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.
`$save_latent_dynamics_files()`	Save diagnostic CSV files to a specified location.

Report run times, console output, return codes:

Method	Description
`$time()`	Report the total run time.
`$output()`	Pretty print the output that was printed to the console.
`$return_codes()`	Return the return codes from the CmdStan runs.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: `CmdStanDiagnose`, `CmdStanGQ`, `CmdStanLaplace`, `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`

`cmdstanr_example` *Fit models for use in examples*

Description

Fit models for use in examples

Usage

```
cmdstanr_example(
  example = c("logistic", "schools", "schools_ncp"),
 method = c("sample", "optimize", "laplace", "variational", "pathfinder", "diagnose"),
  ...,
  quiet = TRUE,
  force_recompile = getOption("cmdstanr_force_recompile", default = FALSE)
)

print_example_program(example = c("logistic", "schools", "schools_ncp"))
```

Arguments

`example` (string) The name of the example. The currently available examples are

- `"logistic"`: logistic regression with intercept and 3 predictors.
- `"schools"`: the so-called "eight schools" model, a hierarchical meta-analysis. Fitting this model will result in warnings about divergences.
- `"schools_ncp"`: non-centered parameterization of the "eight schools" model that fixes the problem with divergences.

To print the Stan code for a given `example` use `print_example_program(example)`.

`method` (string) Which fitting method should be used? The default is the `"sample"` method (MCMC).

`...` Arguments passed to the chosen `method`. See the help pages for the individual methods for details.

`quiet` (logical) If `TRUE` (the default) then fitting the model is wrapped in `utils::capture.output()`.

`force_recompile`
Passed to the $compile() method.

Value

The fitted model object returned by the selected `method`.

Examples

```
## Not run:
print_example_program("logistic")
fit_logistic_mcmc <- cmdstanr_example("logistic", chains = 2)
fit_logistic_mcmc$summary()

fit_logistic_optim <- cmdstanr_example("logistic", method = "optimize")
fit_logistic_optim$summary()

fit_logistic_vb <- cmdstanr_example("logistic", method = "variational")
fit_logistic_vb$summary()

print_example_program("schools")
fit_schools_mcmc <- cmdstanr_example("schools")
fit_schools_mcmc$summary()

print_example_program("schools_ncp")
fit_schools_ncp_mcmc <- cmdstanr_example("schools_ncp")
fit_schools_ncp_mcmc$summary()

# optimization fails for hierarchical model
cmdstanr_example("schools", "optimize", quiet = FALSE)

## End(Not run)
```

---

cmdstanr_global_options

CmdStanR global options

Description

These options can be set via `options()` for an entire R session.

Details

- `cmdstanr_draws_format`: Which format provided by the **posterior** package should be used when returning the posterior or approximate posterior draws? The default depends on the model fitting method. See draws for more details.
- `cmdstanr_force_recompile`: Should the default be to recompile models even if there were no Stan code changes since last compiled? See compile for more details. The default is `FALSE`.
- `cmdstanr_max_rows`: The maximum number of rows of output to print when using the `$print()` method. The default is 10.
- `cmdstanr_no_ver_check`: Should the check for a more recent version of CmdStan be disabled? The default is `FALSE`.
- `cmdstanr_output_dir`: The directory where CmdStan should write its output CSV files when fitting models. The default is a temporary directory. Files in a temporary directory are removed as part of R garbage collection, while files in an explicitly defined directory are not automatically deleted.
- `cmdstanr_verbose`: Should more information be printed when compiling or running models, including showing how CmdStan was called internally? The default is `FALSE`.
- `cmdstanr_warn_inits`: Should a warning be thrown if initial values are only provided for a subset of parameters? The default is `TRUE`.
- `cmdstanr_write_stan_file_dir`: The directory where `write_stan_file()` should write Stan files. The default is a temporary directory. Files in a temporary directory are removed as part of R garbage collection, while files in an explicitly defined directory are not automatically deleted.
- `mc.cores`: The number of cores to use for various parallelization tasks (e.g. running MCMC chains, installing CmdStan). The default depends on the use case and is documented with the methods that make use of `mc.cores`.

`CmdStanVB` *CmdStanVB objects*

Description

A `CmdStanVB` object is the fitted model object returned by the `$variational()` method of a `CmdStanModel` object.

Methods

`CmdStanVB` objects have the following associated methods, all of which have their own (linked) documentation pages.

Extract contents of fitted model object:

Method	Description
`$draws()`	Return approximate posterior draws as a `draws_matrix`.
`$lp()`	Return the total log probability density (`target`) computed in the model block of the Stan program.

`$lp_approx()`	Return the log density of the variational approximation to the posterior.
`$init()`	Return user-specified initial values.
`$metadata()`	Return a list of metadata gathered from the CmdStan CSV files.
`$code()`	Return Stan code as a character vector.

Summarize inferences:

Method	Description
`$summary()`	Run `posterior::summarise_draws()`.
`$cmdstan_summary()`	Run and print CmdStan's `bin/stansummary`.

Save fitted model object and temporary files:

Method	Description
`$save_object()`	Save fitted model object to a file.
`$save_output_files()`	Save output CSV files to a specified location.
`$save_data_file()`	Save JSON data file to a specified location.
`$save_latent_dynamics_files()`	Save diagnostic CSV files to a specified location.

Report run times, console output, return codes:

Method	Description
`$time()`	Report the total run time.
`$output()`	Pretty print the output that was printed to the console.
`$return_codes()`	Return the return codes from the CmdStan runs.

Expose Stan functions and additional methods to R:

Method	Description
`$expose_functions()`	Expose Stan functions for use in R.
`$init_model_methods()`	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
`$log_prob()`	Calculate log-prob.
`$grad_log_prob()`	Calculate log-prob and gradient.
`$hessian()`	Calculate log-prob, gradient, and hessian.
`$constrain_variables()`	Transform a set of unconstrained parameter values to the constrained scale.
`$unconstrain_variables()`	Transform a set of parameter values to the unconstrained scale.
`$unconstrain_draws()`	Transform all parameter draws to the unconstrained scale.
`$variable_skeleton()`	Helper function to re-structure a vector of constrained parameter values.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation

- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other fitted model objects: CmdStanDiagnose, CmdStanGQ, CmdStanLaplace, CmdStanMCMC, CmdStanMLE, CmdStanPathfinder

---

`cmdstan_coercion` *Coercion methods for CmdStan objects*

Description

These are generic functions intended to primarily be used by developers of packages that interface with on CmdStanR. Developers can define methods on top of these generics to coerce objects into CmdStanR's fitted model objects.

Usage

```
as.CmdStanMCMC(object, ...)

as.CmdStanMLE(object, ...)

as.CmdStanLaplace(object, ...)

as.CmdStanVB(object, ...)

as.CmdStanPathfinder(object, ...)

as.CmdStanGQ(object, ...)

as.CmdStanDiagnose(object, ...)
```

Arguments

`object`	to be coerced
`...`	additional arguments

`cmdstan_model` *Create a new CmdStanModel object*

Description

Create a new CmdStanModel object from a file containing a Stan program or from an existing Stan executable. The CmdStanModel object stores the path to a Stan program and compiled executable (once created), and provides methods for fitting the model using Stan's algorithms.

See the `compile` and `...` arguments for control over whether and how compilation happens.

Usage

```
cmdstan_model(stan_file = NULL, exe_file = NULL, compile = TRUE, ...)
```

Arguments

stan_file	(string) The path to a .stan file containing a Stan program. The helper function write_stan_file() is provided for cases when it is more convenient to specify the Stan program as a string. If stan_file is not specified then exe_file must be specified.
exe_file	(string) The path to an existing Stan model executable. Can be provided instead of or in addition to stan_file (if stan_file is omitted some CmdStanModel methods like $code() and $print() will not work). This argument can only be used with CmdStan 2.27+.
compile	(logical) Do compilation? The default is TRUE. If FALSE compilation can be done later via the $compile() method.
...	Optionally, additional arguments to pass to the $compile() method if compile=TRUE. These options include specifying the directory for saving the executable, turning on pedantic mode, specifying include paths, configuring C++ options, and more. See $compile() for details.

Value

A CmdStanModel object.

See Also

install_cmdstan(), $compile(), $check_syntax()

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)
```

```
# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()

# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()
```

```
# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
```

```
fit_optim_w_init_list$init()

## End(Not run)
```

`draws_to_csv` *Write posterior draws objects to CSV files suitable for running standalone generated quantities with CmdStan.*

Description

Write posterior draws objects to CSV files suitable for running standalone generated quantities with CmdStan.

Usage

```
draws_to_csv(
  draws,
  sampler_diagnostics = NULL,
  dir = tempdir(),
  basename = "fittedParams"
)
```

Arguments

`draws`	A `posterior::draws_*` object.
`sampler_diagnostics`	Either `NULL` or a `posterior::draws_*` object of sampler diagnostics.
`dir`	(string) An optional path to the directory where the CSV files will be written. If not set, temporary directory is used.
`basename`	(string) If `dir` is specified, 'basename" is used for naming the output CSV files. If not specified, the file names are randomly generated.

Details

`draws_to_csv()` generates a CSV suitable for running standalone generated quantities with CmdStan. The CSV file contains a single comment `#num_samples`, which equals the number of iterations in the supplied draws object.

The comment is followed by the column names. The first column is the `lp__` value, followed by sampler diagnostics and finnaly other variables of the draws object. #' If the draws object does not contain the `lp__` or sampler diagnostics variables, columns with zeros are created in order to conform with the requirements of the standalone generated quantities method of CmdStan.

The column names line is finally followed by the values of the draws in the same order as the column names.

Value

Paths to CSV files (one per chain).

Examples

```
## Not run:
draws <- posterior::example_draws()

draws_csv_files <- draws_to_csv(draws)
print(draws_csv_files)

# draws_csv_files <- draws_to_csv(draws,
#                                 sampler_diagnostic = sampler_diagnostics,
#                                 dir = "~/my_folder",
#                                 basename = "my-samples")

## End(Not run)
```

`eng_cmdstan` *CmdStan knitr engine for Stan*

Description

This provides a knitr engine for Stan, suitable for usage when attempting to render Stan chunks and compile the model code within to an executable with CmdStan. Use `register_knitr_engine()` to make this the default engine for `stan` chunks. See the vignette R Markdown CmdStan Engine for an example.

Usage

```
eng_cmdstan(options)
```

Arguments

`options`	(named list) Chunk options, as provided by `knitr` during chunk execution.

Examples

```
## Not run:
knitr::knit_engines$set(stan = cmdstanr::eng_cmdstan)

## End(Not run)
```

`fit-method-cmdstan_summary`

Run CmdStan's `stansummary` *and* `diagnose` *utilities*

Description

Run CmdStan's `stansummary` and `diagnose` utilities. These are documented in the CmdStan Guide:

- https://mc-stan.org/docs/cmdstan-guide/stansummary.html
- https://mc-stan.org/docs/cmdstan-guide/diagnose.html

Although these methods can be used for models fit using the `$variational()` method, much of the output is currently only relevant for models fit using the `$sample()` method.

See the $summary() for computing similar summaries in R rather than calling CmdStan's utilites.

Usage

```
cmdstan_summary(flags = NULL)

cmdstan_diagnose()
```

Arguments

`flags`	An optional character vector of flags (e.g. `flags = c("--sig_figs=1")`).

See Also

`CmdStanMCMC`, fit-method-summary

Examples

```
## Not run:
fit <- cmdstanr_example("logistic")
fit$cmdstan_diagnose()
fit$cmdstan_summary()

## End(Not run)
```

fit-method-code *Return Stan code*

Description

Return Stan code

Usage

```
code()
```

Value

A character vector with one element per line of code.

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
fit <- cmdstanr_example()
fit$code() # character vector
cat(fit$code(), sep = "\n") # pretty print

## End(Not run)
```

fit-method-constrain_variables *Transform a set of unconstrained parameter values to the constrained scale*

Description

The `$constrain_variables()` method transforms input parameters to the constrained scale.

Usage

```
constrain_variables(
  unconstrained_variables,
  transformed_parameters = TRUE,
  generated_quantities = TRUE
)
```

Arguments

`unconstrained_variables`
: (numeric) A vector of unconstrained parameters to constrain.

`transformed_parameters`
: (logical) Whether to return transformed parameters implied by newly-constrained parameters (defaults to TRUE).

`generated_quantities`
: (logical) Whether to return generated quantities implied by newly-constrained parameters (defaults to TRUE).

See Also

`log_prob()`, `grad_log_prob()`, `constrain_variables()`, `unconstrain_variables()`, `unconstrain_draws()`, `variable_skeleton()`, `hessian()`

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
fit_mcmc$constrain_variables(unconstrained_variables = c(0.5, 1.2, 1.1, 2.2))

## End(Not run)
```

`fit-method-diagnostic_summary` *Sampler diagnostic summaries and warnings*

Description

Warnings and summaries of sampler diagnostics. To instead get the underlying values of the sampler diagnostics for each iteration and chain use the `$sampler_diagnostics()` method.

Currently parameter-specific diagnostics like R-hat and effective sample size are *not* handled by this method. Those diagnostics are provided via the `$summary()` method (using `posterior::summarize_draws()`).

Usage

```
diagnostic_summary(
  diagnostics = c("divergences", "treedepth", "ebfmi"),
  quiet = FALSE
)
```

Arguments

diagnostics	(character vector) One or more diagnostics to check. The currently supported diagnostics are "divergences", "treedepth", and "ebfmi". The default is to check all of them.
quiet	(logical) Should warning messages about the diagnostics be suppressed? The default is FALSE, in which case warning messages are printed in addition to returning the values of the diagnostics.

Value

A list with as many named elements as diagnostics selected. The possible elements and their values are:

- "num_divergent": A vector of the number of divergences per chain.
- "num_max_treedepth": A vector of the number of times max_treedepth was hit per chain.
- "ebfmi": A vector of E-BFMI values per chain.

See Also

CmdStanMCMC and the $sampler_diagnostics() method

Examples

```
## Not run:
fit <- cmdstanr_example("schools")
fit$diagnostic_summary()
fit$diagnostic_summary(quiet = TRUE)

## End(Not run)
```

---

fit-method-draws *Extract posterior draws*

Description

Extract posterior draws after MCMC or approximate posterior draws after variational approximation using formats provided by the **posterior** package.

The variables include the parameters, transformed parameters, and generated quantities from the Stan program as well as lp__, the total log probability (target) accumulated in the model block.

Usage

```
draws(
  variables = NULL,
  inc_warmup = FALSE,
  format = getOption("cmdstanr_draws_format")
)
```

Arguments

`variables` (character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If `NULL` (the default) then all variables are included.
- If an empty string (`variables=""`) then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - `variables = "theta"` selects all elements of `theta`;
  - `variables = c("theta[1]", "theta[3]")` selects only the 1st and 3rd elements.

`inc_warmup` (logical) Should warmup draws be included? Defaults to `FALSE`. Ignored except when used with CmdStanMCMC objects.

`format` (string) The format of the returned draws or point estimates. Must be a valid format from the **posterior** package. The defaults are the following.

- For sampling and generated quantities the default is `"draws_array"`. This format keeps the chains separate. To combine the chains use any of the other formats (e.g. `"draws_matrix"`).
- For point estimates from optimization and approximate draws from variational inference the default is `"draws_matrix"`.

To use a different format it can be specified as the full name of the format from the **posterior** package (e.g. `format = "draws_df"`) or omitting the `"draws_"` prefix (e.g. `format = "df"`).

Changing the default format: To change the default format for an entire R session use `options(cmdstanr_draws_format = format)`, where `format` is the name (in quotes) of a valid format from the posterior package. For example `options(cmdstanr_draws_format = "draws_df")` will change the default to a data frame.

Note about efficiency: For models with a large number of parameters (20k+) we recommend using the `"draws_list"` format, which is the most efficient and RAM friendly when combining draws from multiple chains. If speed or memory is not a constraint we recommend selecting the format that most suits the coding style of the post processing phase.

Value

Depends on the value of `format`. The defaults are:

- For MCMC, a 3-D `draws_array` object (iteration x chain x variable).
- For standalone generated quantities, a 3-D `draws_array` object (iteration x chain x variable).
- For variational inference, a 2-D `draws_matrix` object (draw x variable) because there are no chains. An additional variable `lp_approx__` is also included, which is the log density of the variational approximation to the posterior evaluated at each of the draws.
- For optimization, a 1-row `draws_matrix` with one column per variable. These are *not* actually draws, just point estimates stored in the `draws_matrix` format. See `$mle()` to extract them as a numeric vector.

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
# logistic regression with intercept alpha and coefficients beta
fit <- cmdstanr_example("logistic", method = "sample")

# returned as 3-D array (see ?posterior::draws_array)
draws <- fit$draws()
dim(draws)
str(draws)

# can easily convert to other formats (data frame, matrix, list)
# using the posterior package
head(posterior::as_draws_matrix(draws))

# or can specify 'format' argument to avoid manual conversion
# matrix format combines all chains
draws <- fit$draws(format = "matrix")
head(draws)

# can select specific parameters
fit$draws("alpha")
fit$draws("beta")  # selects entire vector beta
fit$draws(c("alpha", "beta[2]"))

# can be passed directly to bayesplot plotting functions
bayesplot::color_scheme_set("brightblue")
bayesplot::mcmc_dens(fit$draws(c("alpha", "beta")))
bayesplot::mcmc_scatter(fit$draws(c("beta[1]", "beta[2]")), alpha = 0.3)


# example using variational inference
fit <- cmdstanr_example("logistic", method = "variational")
head(fit$draws("beta")) # a matrix by default
head(fit$draws("beta", format = "df"))

## End(Not run)
```

`fit-method-gradients` *Extract gradients after diagnostic mode*

Description

Return the data frame containing the gradients for all parameters.

Usage

```
gradients()
```

Value

A list of lists. See **Examples**.

See Also

CmdStanDiagnose

Examples

```
## Not run:
test <- cmdstanr_example("logistic", method = "diagnose")

# retrieve the gradients
test$gradients()

## End(Not run)
```

---

fit-method-grad_log_prob

Calculate the log-probability and the gradient w.r.t. each input for a given vector of unconstrained parameters

---

Description

The `$grad_log_prob()` method provides access to the Stan model's `log_prob` function and its derivative.

Usage

```
grad_log_prob(
  unconstrained_variables,
  jacobian = TRUE,
  jacobian_adjustment = NULL
)
```

Arguments

`unconstrained_variables`	(numeric) A vector of unconstrained parameters.
`jacobian`	(logical) Whether to include the log-density adjustments from un/constraining variables.
`jacobian_adjustment`	Deprecated. Please use `jacobian` instead.

See Also

log_prob(), grad_log_prob(), constrain_variables(), unconstrain_variables(), unconstrain_draws(), variable_skeleton(), hessian()

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
fit_mcmc$grad_log_prob(unconstrained_variables = c(0.5, 1.2, 1.1, 2.2))

## End(Not run)
```

---

fit-method-hessian *Calculate the log-probability , the gradient w.r.t. each input, and the hessian for a given vector of unconstrained parameters*

Description

The $hessian() method provides access to the Stan model's log_prob, its derivative, and its hessian.

Usage

```
hessian(unconstrained_variables, jacobian = TRUE, jacobian_adjustment = NULL)
```

Arguments

unconstrained_variables	(numeric) A vector of unconstrained parameters.
jacobian	(logical) Whether to include the log-density adjustments from un/constraining variables.
jacobian_adjustment	Deprecated. Please use jacobian instead.

See Also

log_prob(), grad_log_prob(), constrain_variables(), unconstrain_variables(), unconstrain_draws(), variable_skeleton(), hessian()

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
# fit_mcmc$init_model_methods(hessian = TRUE)
# fit_mcmc$hessian(unconstrained_variables = c(0.5, 1.2, 1.1, 2.2))

## End(Not run)
```

`fit-method-init` *Extract user-specified initial values*

Description

Return user-specified initial values. If the user provided initial values files or R objects (list of lists or function) via the `init` argument when fitting the model then these are returned (always in the list of lists format). Currently it is not possible to extract initial values generated automatically by CmdStan, although CmdStan may support this in the future.

Usage

```
init()
```

Value

A list of lists. See **Examples**.

See Also

`CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`

Examples

```
## Not run:
init_fun <- function() list(alpha = rnorm(1), beta = rnorm(3))
fit <- cmdstanr_example("logistic", init = init_fun, chains = 2)
str(fit$init())

# partial inits (only specifying for a subset of parameters)
init_list <- list(
  list(mu = 10, tau = 2),
  list(mu = -10, tau = 1)
)
fit <- cmdstanr_example("schools_ncp", init = init_list, chains = 2, adapt_delta = 0.9)

# only user-specified inits returned
str(fit$init())

## End(Not run)
```

fit-method-init_model_methods *Compile additional methods for accessing the model log-probability function and parameter constraining and unconstraining.*

Description

The `$init_model_methods()` method compiles and initializes the `log_prob`, `grad_log_prob`, `constrain_variables`, `unconstrain_variables` and `unconstrain_draws` functions. These are then available as methods of the fitted model object. This requires the additional `Rcpp` package, which are not required for fitting models using CmdStanR.

Note: there may be many compiler warnings emitted during compilation but these can be ignored so long as they are warnings and not errors.

Usage

```
init_model_methods(seed = 1, verbose = FALSE, hessian = FALSE)
```

Arguments

`seed`	(integer) The random seed to use when initializing the model.
`verbose`	(logical) Whether to show verbose logging during compilation.
`hessian`	(logical) Whether to expose the (experimental) hessian method.

See Also

`log_prob()`, `grad_log_prob()`, `constrain_variables()`, `unconstrain_variables()`, `unconstrain_draws()`, `variable_skeleton()`, `hessian()`

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)

## End(Not run)
```

fit-method-inv_metric *Extract inverse metric (mass matrix) after MCMC*

Description

Extract the inverse metric (mass matrix) for each MCMC chain.

Usage

```
inv_metric(matrix = TRUE)
```

Arguments

matrix	(logical) If a diagonal metric was used, setting matrix = FALSE returns a list containing just the diagonals of the matrices instead of the full matrices. Setting matrix = FALSE has no effect for dense metrics.

Value

A list of length equal to the number of MCMC chains. See the matrix argument for details.

See Also

CmdStanMCMC

Examples

```
## Not run:
fit <- cmdstanr_example("logistic")
fit$inv_metric()
fit$inv_metric(matrix=FALSE)

fit <- cmdstanr_example("logistic", metric = "dense_e")
fit$inv_metric()

## End(Not run)
```

---

fit-method-log_prob *Calculate the log-probability given a provided vector of unconstrained parameters.*

Description

The $log_prob() method provides access to the Stan model's log_prob function.

Usage

```
log_prob(unconstrained_variables, jacobian = TRUE, jacobian_adjustment = NULL)
```

Arguments

unconstrained_variables	(numeric) A vector of unconstrained parameters.
jacobian	(logical) Whether to include the log-density adjustments from un/constraining variables.
jacobian_adjustment	Deprecated. Please use jacobian instead.

See Also

log_prob(), grad_log_prob(), constrain_variables(), unconstrain_variables(), unconstrain_draws(), variable_skeleton(), hessian()

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
fit_mcmc$log_prob(unconstrained_variables = c(0.5, 1.2, 1.1, 2.2))

## End(Not run)
```

---

fit-method-loo *Leave-one-out cross-validation (LOO-CV)*

Description

The $loo() method computes approximate LOO-CV using the **loo** package. In order to use this method you must compute and save the pointwise log-likelihood in your Stan program. See loo::loo.array() and the **loo** package vignettes for details.

Usage

```
loo(variables = "log_lik", r_eff = TRUE, moment_match = FALSE, ...)
```

Arguments

variables	(character vector) The name(s) of the variable(s) in the Stan program containing the pointwise log-likelihood. The default is to look for "log_lik". This argument is passed to the $draws() method.
r_eff	(multiple options) How to handle the r_eff argument for loo(): • TRUE (the default) will automatically call loo::relative_eff.array() to compute the r_eff argument to pass to loo::loo.array(). • FALSE or NULL will avoid computing r_eff (which can sometimes be slow), but the reported ESS and MCSE estimates can be over-optimistic if the posterior draws are not (near) independent. • If r_eff is anything else, that object will be passed as the r_eff argument to loo::loo.array().
moment_match	(logical) Whether to use a moment-matching correction for problematic observations. The default is FALSE. Using moment_match=TRUE will result in compiling the additional methods described in fit-method-init_model_methods. This allows CmdStanR to automatically supply the functions for the log_lik_i, unconstrain_pars, log_prob_upars, and log_lik_i_upars arguments to loo::loo_moment_match()
...	Other arguments (e.g., cores, save_psis, etc.) passed to loo::loo.array() or loo::loo_moment_match.default() (if moment_match = TRUE is set).

Value

The object returned by `loo::loo.array()` or `loo::loo_moment_match.default()`.

See Also

The **loo** package website with documentation and vignettes.

Examples

```
## Not run:
# the "logistic" example model has "log_lik" in generated quantities
fit <- cmdstanr_example("logistic")
loo_result <- fit$loo(cores = 2)
print(loo_result)

## End(Not run)
```

`fit-method-lp` *Extract log probability (target)*

Description

The `$lp()` method extracts `lp__`, the total log probability (`target`) accumulated in the model block of the Stan program. For variational inference the log density of the variational approximation to the posterior is available via the `$lp_approx()` method. For Laplace approximation the unnormalized density of the approximation to the posterior is available via the `$lp_approx()` method.

See the Log Probability Increment vs. Sampling Statement section of the Stan Reference Manual for details on when normalizing constants are dropped from log probability calculations.

Usage

```
lp()

lp_approx()

lp_approx()
```

Value

A numeric vector with length equal to the number of (post-warmup) draws or length equal to 1 for optimization.

Details

`lp__` is the unnormalized log density on Stan's unconstrained space. This will in general be different than the unnormalized model log density evaluated at a posterior draw (which is on the constrained space). `lp__` is intended to diagnose sampling efficiency and evaluate approximations.

For variational inference `lp_approx__` is the log density of the variational approximation to `lp__` (also on the unconstrained space). It is exposed in the variational method for performing the checks described in Yao et al. (2018) and implemented in the **loo** package.

For Laplace approximation `lp_approx__` is the unnormalized density of the Laplace approximation. It can be used to perform the same checks as in the case of the variational method described in Yao et al. (2018).

References

Yao, Y., Vehtari, A., Simpson, D., and Gelman, A. (2018). Yes, but did it work?: Evaluating variational inference. *Proceedings of the 35th International Conference on Machine Learning*, PMLR 80:5581–5590.

See Also

`CmdStanMCMC`, `CmdStanMLE`, `CmdStanLaplace`, `CmdStanVB`

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic")
head(fit_mcmc$lp())

fit_mle <- cmdstanr_example("logistic", method = "optimize")
fit_mle$lp()

fit_vb <- cmdstanr_example("logistic", method = "variational")
plot(fit_vb$lp(), fit_vb$lp_approx())

## End(Not run)
```

`fit-method-metadata` *Extract metadata from CmdStan CSV files*

Description

The `$metadata()` method returns a list of information gathered from the CSV output files, including the CmdStan configuration used when fitting the model. See **Examples** and `read_cmdstan_csv()`.

Usage

```
metadata()
```

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample")
str(fit_mcmc$metadata())

fit_mle <- cmdstanr_example("logistic", method = "optimize")
str(fit_mle$metadata())

fit_vb <- cmdstanr_example("logistic", method = "variational")
str(fit_vb$metadata())

## End(Not run)
```

---

fit-method-mle *Extract (penalized) maximum likelihood estimate after optimization*

Description

The `$mle()` method is only available for CmdStanMLE objects. It returns the penalized maximum likelihood estimate (posterior mode) as a numeric vector with one element per variable. The returned vector does *not* include `lp__`, the total log probability (`target`) accumulated in the model block of the Stan program, which is available via the `$lp()` method and also included in the `$draws()` method.

Usage

```
mle(variables = NULL)
```

Arguments

`variables`	(character vector) The variables (parameters, transformed parameters, and generated quantities) to include. If NULL (the default) then all variables are included.

Value

A numeric vector. See **Examples**.

See Also

CmdStanMLE

Examples

```
## Not run:
fit <- cmdstanr_example("logistic", method = "optimize")
fit$mle("alpha")
fit$mle("beta")
fit$mle("beta[2]")

## End(Not run)
```

`fit-method-num_chains` *Extract number of chains after MCMC*

Description

The `$num_chains()` method returns the number of MCMC chains.

Usage

```
num_chains()
```

Value

An integer.

See Also

CmdStanMCMC

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example(chains = 2)
fit_mcmc$num_chains()

## End(Not run)
```

fit-method-output *Access console output*

Description

For MCMC, the `$output()` method returns the stdout and stderr of all chains as a list of character vectors if `id=NULL`. If the `id` argument is specified it instead pretty prints the console output for a single chain.

For optimization and variational inference `$output()` just pretty prints the console output.

Usage

```
output(id = NULL)
```

Arguments

`id`	(integer) The chain id. Ignored if the model was not fit using MCMC.

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample")
fit_mcmc$output(1)
out <- fit_mcmc$output()
str(out)

fit_mle <- cmdstanr_example("logistic", method = "optimize")
fit_mle$output()

fit_vb <- cmdstanr_example("logistic", method = "variational")
fit_vb$output()

## End(Not run)
```

`fit-method-profiles` *Return profiling data*

Description

The `$profiles()` method returns a list of data frames with profiling data if any profiling data was written to the profile CSV files. See `save_profile_files()` to control where the files are saved.

Support for profiling Stan programs is available with CmdStan >= 2.26 and requires adding profiling statements to the Stan program.

Usage

```
profiles()
```

Value

A list of data frames with profiling data if the profiling CSV files were created.

See Also

`CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanGQ`

Examples

```
## Not run:
# first fit a model using MCMC
mcmc_program <- write_stan_file(
  'data {
    int<lower=0> N;
    array[N] int<lower=0,upper=1> y;
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  model {
    profile("likelihood") {
      y ~ bernoulli(theta);
    }
  }
  generated quantities {
    array[N] int y_rep;
    profile("gq") {
      y_rep = bernoulli_rng(rep_vector(theta, N));
    }
  }
'
)
mod_mcmc <- cmdstan_model(mcmc_program)
```

```
data <- list(N = 10, y = c(1,1,0,0,0,1,0,1,0,0))
fit <- mod_mcmc$sample(data = data, seed = 123, refresh = 0)

fit$profiles()

## End(Not run)
```

`fit-method-return_codes`

Extract return codes from CmdStan

Description

The `$return_codes()` method returns a vector of return codes from the CmdStan run(s). A return code of 0 indicates a successful run.

Usage

```
return_codes()
```

Value

An integer vector of return codes with length equal to the number of CmdStan runs (number of chains for MCMC and one otherwise).

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
# example with return codes all zero
fit_mcmc <- cmdstanr_example("schools", method = "sample")
fit_mcmc$return_codes() # should be all zero

# example of non-zero return code (optimization fails for hierarchical model)
fit_opt <- cmdstanr_example("schools", method = "optimize")
fit_opt$return_codes() # should be non-zero

## End(Not run)
```

`fit-method-sampler_diagnostics` *Extract sampler diagnostics after MCMC*

Description

Extract the values of sampler diagnostics for each iteration and chain of MCMC. To instead get summaries of these diagnostics and associated warning messages use the `$diagnostic_summary()` method.

Usage

```
sampler_diagnostics(
  inc_warmup = FALSE,
  format = getOption("cmdstanr_draws_format", "draws_array")
)
```

Arguments

`inc_warmup`	(logical) Should warmup draws be included? Defaults to `FALSE`.
`format`	(string) The draws format to return. See draws for details.

Value

Depends on `format`, but the default is a 3-D `draws_array` object (iteration x chain x variable). The variables for Stan's default MCMC algorithm are `"accept_stat__"`, `"stepsize__"`, `"treedepth__"`, `"n_leapfrog__"`, `"divergent__"`, `"energy__"`.

See Also

`CmdStanMCMC`

Examples

```
## Not run:
fit <- cmdstanr_example("logistic")
sampler_diagnostics <- fit$sampler_diagnostics()
str(sampler_diagnostics)

library(posterior)
as_draws_df(sampler_diagnostics)

# or specify format to get a data frame instead of calling as_draws_df
fit$sampler_diagnostics(format = "df")

## End(Not run)
```

`fit-method-save_object` *Save fitted model object to a file*

Description

This method is a wrapper around `base::saveRDS()` that ensures that all posterior draws and diagnostics are saved when saving a fitted model object. Because the contents of the CmdStan output CSV files are only read into R lazily (i.e., as needed), the `$save_object()` method is the safest way to guarantee that everything has been read in before saving.

Usage

```
save_object(file, ...)
```

Arguments

`file`	(string) Path where the file should be saved.
`...`	Other arguments to pass to `base::saveRDS()` besides `object` and `file`.

See Also

`CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanGQ`

Examples

```
## Not run:
fit <- cmdstanr_example("logistic")

temp_rds_file <- tempfile(fileext = ".RDS")
fit$save_object(file = temp_rds_file)
rm(fit)

fit <- readRDS(temp_rds_file)
fit$summary()

## End(Not run)
```

fit-method-save_output_files

Save output and data files

Description

All fitted model objects have methods for saving (moving to a specified location) the files created by CmdStanR to hold CmdStan output csv files and input data files. These methods move the files from their current location (possibly the temporary directory) to a user-specified location. **The paths stored in the fitted model object will also be updated to point to the new file locations.**

The versions without the save_ prefix (e.g., $output_files()) return the current file paths without moving any files.

Usage

```
save_output_files(dir = ".", basename = NULL, timestamp = TRUE, random = TRUE)

save_latent_dynamics_files(
  dir = ".",
  basename = NULL,
  timestamp = TRUE,
  random = TRUE
)

save_profile_files(dir = ".", basename = NULL, timestamp = TRUE, random = TRUE)

save_data_file(dir = ".", basename = NULL, timestamp = TRUE, random = TRUE)

save_config_files(dir = ".", basename = NULL, timestamp = TRUE, random = TRUE)

save_metric_files(dir = ".", basename = NULL, timestamp = TRUE, random = TRUE)

output_files(include_failed = FALSE)

profile_files(include_failed = FALSE)

latent_dynamics_files(include_failed = FALSE)

data_file()

config_files(include_failed = FALSE)

metric_files(include_failed = FALSE)
```

Arguments

dir	(string) Path to directory where the files should be saved.

basename	(string) Base filename to use. See **Details**.
timestamp	(logical) Should a timestamp be added to the file name(s)? Defaults to TRUE. See **Details**.
random	(logical) Should random alphanumeric characters be added to the end of the file name(s)? Defaults to TRUE. See **Details**.
include_failed	(logical) Should CmdStan runs that failed also be included? The default is FALSE.

Value

The $save_* methods print a message with the new file paths and (invisibly) return a character vector of the new paths (or NA for any that couldn't be copied). They also have the side effect of setting the internal paths in the fitted model object to the new paths.

The methods *without* the save_ prefix return character vectors of file paths without moving any files.

Details

For $save_output_files() the files moved to dir will have names of the form basename-timestamp-id-random, where

- basename is the user's provided basename argument;
- timestamp is of the form format(Sys.time(), "%Y%m%d%H%M");
- id is the MCMC chain id (or 1 for non MCMC);
- random contains six random alphanumeric characters;

For $save_latent_dynamics_files() everything is the same as for $save_output_files() except "-diagnostic-" is included in the new file name after basename.

For $save_profile_files() everything is the same as for $save_output_files() except "-profile-" is included in the new file name after basename.

For $save_metric_files() everything is the same as for $save_output_files() except "-metric-" is included in the new file name after basename.

For $save_config_files() everything is the same as for $save_output_files() except "-config-" is included in the new file name after basename.

For $save_data_file() no id is included in the file name because even with multiple MCMC chains the data file is the same.

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
fit <- cmdstanr_example()
fit$output_files()
fit$data_file()
```

```
# just using tempdir for the example
my_dir <- tempdir()
fit$save_output_files(dir = my_dir, basename = "banana")
fit$save_output_files(dir = my_dir, basename = "tomato", timestamp = FALSE)
fit$save_output_files(dir = my_dir, basename = "lettuce", timestamp = FALSE, random = FALSE)

## End(Not run)
```

`fit-method-summary` *Compute a summary table of estimates and diagnostics*

Description

The `$summary()` method runs `summarise_draws()` from the **posterior** package and returns the output. For MCMC, only post-warmup draws are included in the summary.

There is also a `$print()` method that prints the same summary stats but removes the extra formatting used for printing tibbles and returns the fitted model object itself. The `$print()` method may also be faster than `$summary()` because it is designed to only compute the summary statistics for the variables that will actually fit in the printed output whereas `$summary()` will compute them for all of the specified variables in order to be able to return them to the user. See **Examples**.

Usage

```
summary(variables = NULL, ...)
```

Arguments

`variables`	(character vector) The variables to include.
`...`	Optional arguments to pass to `posterior::summarise_draws()`.

Value

The `$summary()` method returns the tibble data frame created by `posterior::summarise_draws()`.

The `$print()` method returns the fitted model object itself (invisibly), which is the standard behavior for print methods in R.

See Also

`CmdStanMCMC`, `CmdStanMLE`, `CmdStanLaplace`, `CmdStanVB`, `CmdStanGQ`

Examples

```
## Not run:
fit <- cmdstanr_example("logistic")
fit$summary()
fit$print()
fit$print(max_rows = 2) # same as print(fit, max_rows = 2)

# include only certain variables
fit$summary("beta")
fit$print(c("alpha", "beta[2]"))

# include all variables but only certain summaries
fit$summary(NULL, c("mean", "sd"))

# can use functions created from formulas
# for example, calculate Pr(beta > 0)
fit$summary("beta", prob_gt_0 = ~ mean(. > 0))

# can combine user-specified functions with
# the default summary functions
fit$summary(variables = c("alpha", "beta"),
  posterior::default_summary_measures()[1:4],
  quantiles = ~ quantile2(., probs = c(0.025, 0.975)),
  posterior::default_convergence_measures()
  )

# the functions need to calculate the appropriate
# value for a matrix input
fit$summary(variables = "alpha", dim)

# the usual [stats::var()] is therefore not directly suitable as it
# will produce a covariance matrix unless the data is converted to a vector
fit$print(c("alpha", "beta"), var2 = ~var(as.vector(.x)))


## End(Not run)
```

`fit-method-time` *Report timing of CmdStan runs*

Description

Report the run time in seconds. For MCMC additional information is provided about the run times of individual chains and the warmup and sampling phases. For Laplace approximation the time only include the time for drawing the approximate sample and does not include the time taken to run the `$optimize()` method.

Usage

```
time()
```

Value

A list with elements

- `total`: (scalar) The total run time. For MCMC this may be different than the sum of the chain run times if parallelization was used.
- `chains`: (data frame) For MCMC only, timing info for the individual chains. The data frame has columns `"chain_id"`, `"warmup"`, `"sampling"`, and `"total"`.

See Also

CmdStanMCMC, CmdStanMLE, CmdStanVB, CmdStanGQ

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample")
fit_mcmc$time()

fit_vb <- cmdstanr_example("logistic", method = "variational")
fit_vb$time()

fit_mle <- cmdstanr_example("logistic", method = "optimize", jacobian = TRUE)
fit_mle$time()

# use fit_mle to draw samples from laplace approximation
fit_laplace <- cmdstanr_example("logistic", method = "laplace", mode = fit_mle)
fit_laplace$time() # just time for drawing sample not for running optimize
fit_laplace$time()$total + fit_mle$time()$total # total time

## End(Not run)
```

`fit-method-unconstrain_draws` *Transform all parameter draws to the unconstrained scale*

Description

The `$unconstrain_draws()` method transforms all parameter draws to the unconstrained scale. The method returns a list for each chain, containing the parameter values from each iteration on the unconstrained scale. If called with no arguments, then the draws within the fit object are unconstrained. Alternatively, either an existing draws object or a character vector of paths to CSV files can be passed.

Usage

```
unconstrain_draws(
  files = NULL,
  draws = NULL,
  format = getOption("cmdstanr_draws_format", "draws_array"),
  inc_warmup = FALSE
)
```

Arguments

files	(character vector) The paths to the CmdStan CSV files. These can be files generated by running CmdStanR or running CmdStan directly.
draws	A `posterior::draws_*` object.
format	(string) The format of the returned draws. Must be a valid format from the **posterior** package.
inc_warmup	(logical) Should warmup draws be included? Defaults to `FALSE`.

See Also

`log_prob()`, `grad_log_prob()`, `constrain_variables()`, `unconstrain_variables()`, `unconstrain_draws()`, `variable_skeleton()`, `hessian()`

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)

# Unconstrain all internal draws
unconstrained_internal_draws <- fit_mcmc$unconstrain_draws()

# Unconstrain external CmdStan CSV files
unconstrained_csv <- fit_mcmc$unconstrain_draws(files = fit_mcmc$output_files())

# Unconstrain existing draws object
unconstrained_draws <- fit_mcmc$unconstrain_draws(draws = fit_mcmc$draws())

## End(Not run)
```

---

fit-method-unconstrain_variables

Transform a set of parameter values to the unconstrained scale

---

Description

The `$unconstrain_variables()` method transforms input parameters to the unconstrained scale.

Usage

```
unconstrain_variables(variables)
```

Arguments

variables	(list) A list of parameter values to transform, in the same format as provided to the `init` argument of the `$sample()` method.

See Also

log_prob(), grad_log_prob(), constrain_variables(), unconstrain_variables(), unconstrain_draws(), variable_skeleton(), hessian()

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
fit_mcmc$unconstrain_variables(list(alpha = 0.5, beta = c(0.7, 1.1, 0.2)))

## End(Not run)
```

---

fit-method-variable_skeleton

Return the variable skeleton for `relist`

Description

The `$variable_skeleton()` method returns the variable skeleton needed by `utils::relist()` to re-structure a vector of constrained parameter values to a named list.

Usage

```
variable_skeleton(transformed_parameters = TRUE, generated_quantities = TRUE)
```

Arguments

transformed_parameters	(logical) Whether to include transformed parameters in the skeleton (defaults to TRUE).
generated_quantities	(logical) Whether to include generated quantities in the skeleton (defaults to TRUE).

See Also

log_prob(), grad_log_prob(), constrain_variables(), unconstrain_variables(), unconstrain_draws(), variable_skeleton(), hessian()

Examples

```
## Not run:
fit_mcmc <- cmdstanr_example("logistic", method = "sample", force_recompile = TRUE)
fit_mcmc$variable_skeleton()

## End(Not run)
```

`install_cmdstan` *Install CmdStan or clean and rebuild an existing installation*

Description

The `install_cmdstan()` function attempts to download and install the latest release of CmdStan. Installing a previous release or a new release candidate is also possible by specifying the `version` or `release_url` argument. See the first few sections of the CmdStan installation guide for details on the C++ toolchain required for installing CmdStan.

The `rebuild_cmdstan()` function cleans and rebuilds the CmdStan installation. Use this function in case of any issues when compiling models.

The `cmdstan_make_local()` function is used to read/write makefile flags and variables from/to the `make/local` file of a CmdStan installation. Writing to the `make/local` file can be used to permanently add makefile flags/variables to an installation. For example adding specific compiler switches, changing the C++ compiler, etc. A change to the `make/local` file should typically be followed by calling `rebuild_cmdstan()`.

The `check_cmdstan_toolchain()` function attempts to check for the required C++ toolchain. It is called internally by `install_cmdstan()` but can also be called directly by the user.

Usage

```
install_cmdstan(
  dir = NULL,
  cores = getOption("mc.cores", 2),
  quiet = FALSE,
  overwrite = FALSE,
  timeout = 1200,
  version = NULL,
  release_url = NULL,
  release_file = NULL,
  cpp_options = list(),
  check_toolchain = TRUE,
  wsl = FALSE
)

rebuild_cmdstan(
  dir = cmdstan_path(),
  cores = getOption("mc.cores", 2),
```

```
  quiet = FALSE,
  timeout = 600
)

cmdstan_make_local(dir = cmdstan_path(), cpp_options = NULL, append = TRUE)

check_cmdstan_toolchain(fix = FALSE, quiet = FALSE)
```

Arguments

dir	(string) The path to the directory in which to install CmdStan. The default is to install it in a directory called `.cmdstan` within the user's home directory (i.e, `file.path(Sys.getenv("HOME"), ".cmdstan")`).
cores	(integer) The number of CPU cores to use to parallelize building CmdStan and speed up installation. If `cores` is not specified then the default is to look for the option `"mc.cores"`, which can be set for an entire R session by `options(mc.cores=value)`. If the `"mc.cores"` option has not been set then the default is `2`.
quiet	(logical) For `install_cmdstan()`, should the verbose output from the system processes be suppressed when building the CmdStan binaries? The default is `FALSE`. For `check_cmdstan_toolchain()`, should the function suppress printing informational messages? The default is `FALSE`. If `TRUE` only errors will be printed.
overwrite	(logical) Should CmdStan still be downloaded and installed even if an installation of the same version is found in `dir`? The default is `FALSE`, in which case an informative error is thrown instead of overwriting the user's installation.
timeout	(positive real) Timeout (in seconds) for the build stage of the installation.
version	(string) The CmdStan release version to install. The default is `NULL`, which downloads the latest stable release from https://github.com/stan-dev/cmdstan/releases.
release_url	(string) The URL for the specific CmdStan release or release candidate to install. See https://github.com/stan-dev/cmdstan/releases. The URL should point to the tarball (`.tar.gz` file) itself, e.g., `release_url="https://github.com/stan-dev/cmdstar` If both `version` and `release_url` are specified then `version` will be used.
release_file	(string) A file path to a CmdStan release tar.gz file downloaded from the releases page: https://github.com/stan-dev/cmdstan/releases. For example: `release_file="./cmdstan-2.33.1.tar.gz"`. If `release_file` is specified then both `release_url` and `version` will be ignored.
cpp_options	(list) Any makefile flags/variables to be written to the `make/local` file. For example, `list("CXX" = "clang++")` will force the use of clang for compilation.
check_toolchain	(logical) Should `install_cmdstan()` attempt to check that the required toolchain is installed and properly configured. The default is `TRUE`.
wsl	(logical) Should CmdStan be installed and run through the Windows Subsystem for Linux (WSL). The default is `FALSE`.
append	(logical) For `cmdstan_make_local()`, should the listed makefile flags be appended to the end of the existing `make/local` file? The default is `TRUE`. If `FALSE` the file is overwritten.

`fix` For `check_cmdstan_toolchain()`, should CmdStanR attempt to fix any detected toolchain problems? Currently this option is only available on Windows. The default is `FALSE`, in which case problems are only reported along with suggested fixes.

Value

For `cmdstan_make_local()`, if `cpp_options=NULL` then the existing contents of `make/local` are returned without writing anything, otherwise the updated contents are returned.

Examples

```
## Not run:
check_cmdstan_toolchain()

# install_cmdstan(cores = 4)

cpp_options <- list(
  "CXX" = "clang++",
  "CXXFLAGS+= -march=native",
  PRECOMPILED_HEADERS = TRUE
)
# cmdstan_make_local(cpp_options = cpp_options)
# rebuild_cmdstan()

## End(Not run)
```

---

model-method-check_syntax

Check syntax of a Stan program

Description

The `$check_syntax()` method of a `CmdStanModel` object checks the Stan program for syntax errors and returns `TRUE` (invisibly) if parsing succeeds. If invalid syntax in found an error is thrown.

Usage

```
check_syntax(
  pedantic = FALSE,
  include_paths = NULL,
  stanc_options = list(),
  quiet = FALSE
)
```

Arguments

pedantic	(logical) Should pedantic mode be turned on? The default is FALSE. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual.
include_paths	(character vector) Paths to directories where Stan should look for files specified in #include directives in the Stan program.
stanc_options	(list) Any other Stan-to-C++ transpiler options to be used when compiling the model. See the documentation for the $compile() method for details.
quiet	(logical) Should informational messages be suppressed? The default is FALSE, which will print a message if the Stan program is valid or the compiler error message if there are syntax errors. If TRUE, only the error message will be printed.

Value

The $check_syntax() method returns TRUE (invisibly) if the model is valid.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: model-method-compile, model-method-diagnose, model-method-expose_functions, model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variables, model-method-variational

Examples

```
## Not run:
file <- write_stan_file("
data {
  int N;
  array[N] int y;
}
parameters {
  // should have <lower=0> but omitting to demonstrate pedantic mode
  real lambda;
}
model {
  y ~ poisson(lambda);
}
")
mod <- cmdstan_model(file, compile = FALSE)

# the program is syntactically correct, however...
```

```
mod$check_syntax()

# pedantic mode will warn that lambda should be constrained to be positive
# and that lambda has no prior distribution
mod$check_syntax(pedantic = TRUE)

## End(Not run)
```

`model-method-compile` *Compile a Stan program*

Description

The `$compile()` method of a `CmdStanModel` object checks the syntax of the Stan program, translates the program to C++, and creates a compiled executable. To just check the syntax of a Stan program without compiling it use the `$check_syntax()` method instead.

In most cases the user does not need to explicitly call the `$compile()` method as compilation will occur when calling `cmdstan_model()`. However it is possible to set `compile=FALSE` in the call to `cmdstan_model()` and subsequently call the `$compile()` method directly.

After compilation, the paths to the executable and the `.hpp` file containing the generated C++ code are available via the `$exe_file()` and `$hpp_file()` methods. The default is to create the executable in the same directory as the Stan program and to write the generated C++ code in a temporary directory. To save the C++ code to a non-temporary location use `$save_hpp_file(dir)`.

Usage

```
compile(
  quiet = TRUE,
  dir = NULL,
  pedantic = FALSE,
  include_paths = NULL,
  user_header = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = getOption("cmdstanr_force_recompile", default = FALSE),
  compile_model_methods = FALSE,
  compile_standalone = FALSE,
  dry_run = FALSE,
  compile_hessian_method = FALSE,
  threads = FALSE
)
```

Arguments

`quiet`	(logical) Should the verbose output from CmdStan during compilation be suppressed? The default is `TRUE`, but if you encounter an error we recommend trying again with `quiet=FALSE` to see more of the output.

`dir` (string) The path to the directory in which to store the CmdStan executable (or `.hpp` file if using `$save_hpp_file()`). The default is the same location as the Stan program.

`pedantic` (logical) Should pedantic mode be turned on? The default is `FALSE`. Pedantic mode attempts to warn you about potential issues in your Stan program beyond syntax errors. For details see the *Pedantic mode* chapter in the Stan Reference Manual. **Note:** to do a pedantic check for a model without compiling it or for a model that is already compiled the `$check_syntax()` method can be used instead.

`include_paths` (character vector) Paths to directories where Stan should look for files specified in `#include` directives in the Stan program.

`user_header` (string) The path to a C++ file (with a .hpp extension) to compile with the Stan model.

`cpp_options` (list) Any makefile options to be used when compiling the model (`STAN_THREADS`, `STAN_MPI`, `STAN_OPENCL`, etc.). Anything you would otherwise write in the `make/local` file. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

`stanc_options` (list) Any Stan-to-C++ transpiler options to be used when compiling the model. See the **Examples** section below as well as the `stanc` chapter of the CmdStan Guide for more details on available options: https://mc-stan.org/docs/cmdstan-guide/stanc.html.

`force_recompile` (logical) Should the model be recompiled even if was not modified since last compiled. The default is `FALSE`. Can also be set via a global `cmdstanr_force_recompile` option.

`compile_model_methods` (logical) Compile additional model methods (`log_prob()`, `grad_log_prob()`, `constrain_variables()`, `unconstrain_variables()`).

`compile_standalone` (logical) Should functions in the Stan model be compiled for use in R? If `TRUE` the functions will be available via the `functions` field in the compiled model object. This can also be done after compilation using the `$expose_functions()` method.

`dry_run` (logical) If `TRUE`, the code will do all checks before compilation, but skip the actual C++ compilation. Used to speedup tests.

`compile_hessian_method` (logical) Should the (experimental) `hessian()` method be be compiled with the model methods?

`threads` Deprecated and will be removed in a future release. Please turn on threading via `cpp_options = list(stan_threads = TRUE)` instead.

Value

The `$compile()` method is called for its side effect of creating the executable and adding its path to the `CmdStanModel` object, but it also returns the `CmdStanModel` object invisibly.

After compilation, the `$exe_file()`, `$hpp_file()`, and `$save_hpp_file()` methods can be used and return file paths.

See Also

The $check_syntax() method to check Stan syntax or enable pedantic model without compiling.

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: model-method-check_syntax, model-method-diagnose, model-method-expose_functi
model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize,
model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variables,
model-method-variational

Examples

```
## Not run:
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")

# by default compilation happens when cmdstan_model() is called.
# to delay compilation until calling the $compile() method set compile=FALSE
mod <- cmdstan_model(file, compile = FALSE)
mod$compile()
mod$exe_file()

# turn on threading support (for using functions that support within-chain parallelization)
mod$compile(force_recompile = TRUE, cpp_options = list(stan_threads = TRUE))
mod$exe_file()

# turn on pedantic mode (new in Stan v2.24)
file_pedantic <- write_stan_file("
parameters {
  real sigma;  // pedantic mode will warn about missing <lower=0>
}
model {
  sigma ~ exponential(1);
}
")
mod <- cmdstan_model(file_pedantic, pedantic = TRUE)


## End(Not run)
```

---

model-method-diagnose *Run Stan's diagnose method*

Description

The `$diagnose()` method of a `CmdStanModel` object runs Stan's basic diagnostic feature that will calculate the gradients of the initial state and compare them with gradients calculated by finite differences. Discrepancies between the two indicate that there is a problem with the model or initial states or else there is a bug in Stan.

Usage

```
diagnose(
  data = NULL,
  seed = NULL,
  init = NULL,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  epsilon = NULL,
  error = NULL
)
```

Arguments

`data` (multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- `NULL` or an empty list if the Stan program has no data block.

`seed` (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

`init` (multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.;
- The number `0`. This initializes *all* parameters to `0`;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan.

- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`output_dir` (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at `NULL` (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If `NULL` (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename` (string) A string to use as a prefix for the names of the output CSV files of

	CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.
epsilon	(positive real) The finite difference step size. Default value is 1e-6.
error	(positive real) The error threshold. Default value is 1e-6.

Value

A CmdStanDiagnose object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: model-method-check_syntax, model-method-compile, model-method-expose_functio
model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize,
model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variables,
model-method-variational

Examples

```
## Not run:
test <- cmdstanr_example("logistic", method = "diagnose")

# retrieve the gradients
test$gradients()

## End(Not run)
```

---

model-method-expose_functions

Expose Stan functions to R

Description

The $expose_functions() method of a CmdStanModel object will compile the functions in the Stan program's functions block and expose them for use in R. This can also be specified via the compile_standalone argument to the $compile() method.

This method is also available for fitted model objects (CmdStanMCMC, CmdStanVB, etc.). See **Examples**.

Note: there may be many compiler warnings emitted during compilation but these can be ignored so long as they are warnings and not errors.

Usage

```
expose_functions(global = FALSE, verbose = FALSE)
```

Arguments

`global`	(logical) Should the functions be added to the Global Environment? The default is `FALSE`, in which case the functions are available via the `functions` field of the R6 object.
`verbose`	(logical) Should detailed information about generated code be printed to the console? Defaults to `FALSE`.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-optimize`, `model-method-pathfinder`, `model-method-sample`, `model-method-sample_mpi`, `model-method-variables`, `model-method-variational`

Examples

```
## Not run:
stan_file <- write_stan_file(
 "
 functions {
   real a_plus_b(real a, real b) {
     return a + b;
   }
 }
 parameters {
   real x;
 }
 model {
   x ~ std_normal();
 }
 "
)
mod <- cmdstan_model(stan_file)
mod$expose_functions()
mod$functions$a_plus_b(1, 2)

fit <- mod$sample(refresh = 0)
fit$expose_functions() # already compiled because of above but this would compile them otherwise
fit$functions$a_plus_b(1, 2)
```

```
## End(Not run)
```

`model-method-format` *Run stanc's auto-formatter on the model code.*

Description

The `$format()` method of a `CmdStanModel` object runs stanc's auto-formatter on the model code. Either saves the formatted model directly back to the file or prints it for inspection.

Usage

```
format(
  overwrite_file = FALSE,
  canonicalize = FALSE,
  backup = TRUE,
  max_line_length = NULL,
  quiet = FALSE
)
```

Arguments

`overwrite_file`	(logical) Should the formatted code be written back to the input model file. The default is `FALSE`.
`canonicalize`	(list or logical) Defines whether or not the compiler should 'canonicalize' the Stan model, removing things like deprecated syntax. Default is `FALSE`. If `TRUE`, all canonicalizations are run. You can also supply a list of strings which represent options. In that case the options are passed to stanc (new in Stan 2.29). See the User's guide section for available canonicalization options.
`backup`	(logical) If `TRUE`, create stanfile.bak backups before writing to the file. Disable this option if you're sure you have other copies of the file or are using a version control system like Git. Defaults to `TRUE`. The value is ignored if `overwrite_file = FALSE`.
`max_line_length`	(integer) The maximum length of a line when formatting. The default is `NULL`, which defers to the default line length of stanc.
`quiet`	(logical) Should informational messages be suppressed? The default is `FALSE`.

Value

The `$format()` method returns `TRUE` (invisibly) if the model is valid.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: model-method-check_syntax, model-method-compile, model-method-diagnose, model-method-expose_functions, model-method-generate-quantities, model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variables, model-method-variational

Examples

```
## Not run:

# Example of fixing old syntax
# real x[2] --> array[2] real x;
file <- write_stan_file("
parameters {
  real x[2];
}
model {
  x ~ std_normal();
}
")

# set compile=FALSE then call format to fix old syntax
mod <- cmdstan_model(file, compile = FALSE)
mod$format(canonicalize = list("deprecations"))

# overwrite the original file instead of just printing it
mod$format(canonicalize = list("deprecations"), overwrite_file = TRUE)
mod$compile()


# Example of removing unnecessary whitespace
file <- write_stan_file("
data {
  int N;
  array[N] int y;
}
parameters {
  real                    lambda;
}
model {
  target +=
 poisson_lpmf(y | lambda);
}
")
mod <- cmdstan_model(file, compile = FALSE)
```

```
mod$format(canonicalize = TRUE)

## End(Not run)
```

`model-method-generate-quantities` *Run Stan's standalone generated quantities method*

Description

The `$generate_quantities()` method of a `CmdStanModel` object runs Stan's standalone generated quantities to obtain generated quantities based on previously fitted parameters.

Usage

```
generate_quantities(
  fitted_params,
  data = NULL,
  seed = NULL,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  parallel_chains = getOption("mc.cores", 1),
  threads_per_chain = NULL,
  opencl_ids = NULL
)
```

Arguments

`fitted_params` (multiple options) The parameter draws to use. One of the following:

- A CmdStanMCMC or CmdStanVB fitted model object.
- A posterior::draws_array (for MCMC) or posterior::draws_matrix (for VB) object returned by CmdStanR's `$draws()` method.
- A character vector of paths to CmdStan CSV output files.

NOTE: if you plan on making many calls to `$generate_quantities()` then the most efficient option is to pass the paths of the CmdStan CSV output files (this avoids CmdStanR having to rewrite the draws contained in the fitted model object to CSV each time). If you no longer have the CSV files you can use `draws_to_csv()` once to write them and then pass the resulting file paths to `$generate_quantities()` as many times as needed.

`data` (multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- `NULL` or an empty list if the Stan program has no data block.

`seed` (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

`output_dir` (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at `NULL` (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If `NULL` (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename`

(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

`sig_figs` (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`parallel_chains`

(positive integer) The *maximum* number of MCMC chains to run in parallel. If `parallel_chains` is not specified then the default is to look for the option `"mc.cores"`, which can be set for an entire R session by `options(mc.cores=value)`. If the `"mc.cores"` option has not been set then the default is 1.

`threads_per_chain`

(positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when

using the Stan functions `reduce_sum()` or `map_rect()`). This is in contrast with `parallel_chains`, which specifies the number of chains to run in parallel. The actual number of CPU cores used is `parallel_chains*threads_per_chain`. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

`opencl_ids` (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

Value

A `CmdStanGQ` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-laplace`, `model-method-optimize`, `model-method-pathfinder`, `model-method-sample`, `model-method-sample_mpi`, `model-method-variables`, `model-method-variational`

Examples

```
## Not run:
# first fit a model using MCMC
mcmc_program <- write_stan_file(
  "data {
    int<lower=0> N;
    array[N] int<lower=0,upper=1> y;
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  model {
    y ~ bernoulli(theta);
  }"
)
mod_mcmc <- cmdstan_model(mcmc_program)

data <- list(N = 10, y = c(1,1,0,0,0,1,0,1,0,0))
fit_mcmc <- mod_mcmc$sample(data = data, seed = 123, refresh = 0)

# stan program for standalone generated quantities
# (could keep model block, but not necessary so removing it)
gq_program <- write_stan_file(
  "data {
```

```
    int<lower=0> N;
    array[N] int<lower=0,upper=1> y;
  }
  parameters {
    real<lower=0,upper=1> theta;
  }
  generated quantities {
    array[N] int y_rep = bernoulli_rng(rep_vector(theta, N));
  }"
)

mod_gq <- cmdstan_model(gq_program)
fit_gq <- mod_gq$generate_quantities(fit_mcmc, data = data, seed = 123)
str(fit_gq$draws())

library(posterior)
as_draws_df(fit_gq$draws())

## End(Not run)
```

`model-method-laplace` *Run Stan's Laplace algorithm*

Description

The `$laplace()` method of a `CmdStanModel` object produces a sample from a normal approximation centered at the mode of a distribution in the unconstrained space. If the mode is a maximum a posteriori (MAP) estimate, the samples provide an estimate of the mean and standard deviation of the posterior distribution. If the mode is a maximum likelihood estimate (MLE), the sample provides an estimate of the standard error of the likelihood. Whether the mode is the MAP or MLE depends on the value of the `jacobian` argument when running optimization. See the CmdStan User's Guide for more details.

Any argument left as `NULL` will default to the default value used by the installed version of CmdStan.

Usage

```
laplace(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  threads = NULL,
  opencl_ids = NULL,
```

```
  mode = NULL,
  opt_args = NULL,
  jacobian = TRUE,
  draws = NULL,
  show_messages = TRUE,
  show_exceptions = TRUE,
  save_cmdstan_config = NULL
)
```

Arguments

`data`	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following: • A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan. • A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats. • `NULL` or an empty list if the Stan program has no data block.
`seed`	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
`refresh`	(non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.
`init`	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following: • A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.; • The number `0`. This initializes *all* parameters to `0`; • A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan. • A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**. • A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no

arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`save_latent_dynamics`

Ignored for this method.

`output_dir` (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at `NULL` (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If `NULL` (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename`

(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

`sig_figs` (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant

figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`threads` (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections (e.g., when using the Stan functions `reduce_sum()` or `map_rect()`).

`opencl_ids` (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

`mode` (multiple options) The mode to center the approximation at. One of the following:

- A `CmdStanMLE` object from a previous run of `$optimize()`.
- The path to a CmdStan CSV file from running optimization.
- `NULL`, in which case $optimize() will be run with `jacobian=jacobian` (see the `jacobian` argument below).

In all cases the total time reported by `$time()` will be the time of the Laplace sampling step only and does not include the time taken to run the `$optimize()` method.

`opt_args` (named list) A named list of optional arguments to pass to $optimize() if `mode=NULL`.

`jacobian` (logical) Whether or not to enable the Jacobian adjustment for constrained parameters. The default is `TRUE`. See the Laplace Sampling section of the CmdStan User's Guide for more details. If `mode` is not `NULL` then the value of `jacobian` must match the value used when optimization was originally run. If `mode` is `NULL` then the value of `jacobian` specified here is used when running optimization.

`draws` (positive integer) The number of draws to take.

`show_messages` (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions` (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`save_cmdstan_config` (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

Value

A `CmdStanLaplace` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-optimize`, `model-method-pathfinder`, `model-method-sample`, `model-method-sample_mpi`, `model-method-variables`, `model-method-variational`

Examples

```
## Not run:
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()

stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))
fit_mode <- mod$optimize(data = stan_data, jacobian = TRUE)
fit_laplace <- mod$laplace(data = stan_data, mode = fit_mode)
fit_laplace$summary()

# if mode isn't specified optimize is run internally first
fit_laplace <- mod$laplace(data = stan_data)
fit_laplace$summary()

# plot approximate posterior
bayesplot::mcmc_hist(fit_laplace$draws("theta"))

## End(Not run)
```

`model-method-optimize` *Run Stan's optimization algorithms*

Description

The `$optimize()` method of a `CmdStanModel` object runs Stan's optimizer to obtain a (penalized) maximum likelihood estimate or a maximum a posteriori estimate (if `jacobian=TRUE`). See the CmdStan User's Guide for more details.

Any argument left as `NULL` will default to the default value used by the installed version of CmdStan. See the CmdStan User's Guide for more details on the default arguments. The default values can also be obtained by checking the metadata of an example model, e.g., `cmdstanr_example(method="optimize")$metadata`

Usage

```
optimize(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  threads = NULL,
  opencl_ids = NULL,
  algorithm = NULL,
  jacobian = FALSE,
  init_alpha = NULL,
  iter = NULL,
  tol_obj = NULL,
  tol_rel_obj = NULL,
  tol_grad = NULL,
  tol_rel_grad = NULL,
  tol_param = NULL,
  history_size = NULL,
  show_messages = TRUE,
  show_exceptions = TRUE,
  save_cmdstan_config = NULL
)
```

Arguments

`data` (multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan.
- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- `NULL` or an empty list if the Stan program has no data block.

`seed` (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

`refresh` (non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.

`init` (multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.;
- The number `0`. This initializes *all* parameters to `0`;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`save_latent_dynamics`

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content

may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the $save_latent_dynamics_files() method.

output_dir (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using options(cmdstanr_output_dir). The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

sig_figs (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for sig_figs is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

threads (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections (e.g., when using the Stan functions reduce_sum() or map_rect()).

opencl_ids (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with cpp_options = list(stan_opencl = TRUE) for this argument to have an effect.

algorithm (string) The optimization algorithm. One of "lbfgs", "bfgs", or "newton". The control parameters below are only available for "lbfgs" and "bfgs. For their default values and more details see the CmdStan User's Guide. The default values can also be obtained by running cmdstanr_example(method="optimize")$metadata().

jacobian (logical) Whether or not to use the Jacobian adjustment for constrained variables. For historical reasons, the default is FALSE, meaning optimization yields the (regularized) maximum likelihood estimate. Setting it to TRUE yields the maximum a posteriori estimate. See the Maximum Likelihood Estimation section of the CmdStan User's Guide for more details. For use later with $laplace() the jacobian argument should typically be set to TRUE.

init_alpha (positive real) The initial step size parameter.

iter (positive integer) The maximum number of iterations.

tol_obj (positive real) Convergence tolerance on changes in objective function value.

tol_rel_obj (positive real) Convergence tolerance on relative changes in objective function value.

`tol_grad` (positive real) Convergence tolerance on the norm of the gradient.

`tol_rel_grad` (positive real) Convergence tolerance on the relative norm of the gradient.

`tol_param` (positive real) Convergence tolerance on changes in parameter value.

`history_size` (positive integer) The size of the history used when approximating the Hessian. Only available for L-BFGS.

`show_messages` (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions` (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`save_cmdstan_config` (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

Value

A `CmdStanMLE` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-pathfinder`, `model-method-sample`, `model-method-sample_mpi`, `model-method-variables`, `model-method-variational`

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
```

```
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()
```

```
# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
```

```
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

---

model-method-pathfinder

Run Stan's Pathfinder Variational Inference Algorithm

Description

The `$pathfinder()` method of a `CmdStanModel` object runs Stan's Pathfinder algorithms. Pathfinder is a variational method for approximately sampling from differentiable log densities. Starting from a random initialization, Pathfinder locates normal approximations to the target density along a quasi-Newton optimization path in the unconstrained space, with local covariance estimated using the negative inverse Hessian estimates produced by the LBFGS optimizer. Pathfinder selects the normal approximation with the lowest estimated Kullback-Leibler (KL) divergence to the true posterior. Finally Pathfinder draws from that normal approximation and returns the draws transformed to the constrained scale. See the CmdStan User's Guide for more details.

Any argument left as `NULL` will default to the default value used by the installed version of CmdStan

Usage

```
pathfinder(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  opencl_ids = NULL,
  num_threads = NULL,
  init_alpha = NULL,
  tol_obj = NULL,
  tol_rel_obj = NULL,
  tol_grad = NULL,
  tol_rel_grad = NULL,
  tol_param = NULL,
  history_size = NULL,
```

```
  single_path_draws = NULL,
  draws = NULL,
  num_paths = 4,
  max_lbfgs_iters = NULL,
  num_elbo_draws = NULL,
  save_single_paths = NULL,
  psis_resample = NULL,
  calculate_lp = NULL,
  show_messages = TRUE,
  show_exceptions = TRUE,
  save_cmdstan_config = NULL
)
```

Arguments

`data`	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following: • A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan. • A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats. • `NULL` or an empty list if the Stan program has no data block.
`seed`	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
`refresh`	(non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.
`init`	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following: • A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.; • The number `0`. This initializes *all* parameters to `0`; • A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan. • A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have

named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.

- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`save_latent_dynamics`

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is `FALSE`, which is appropriate for almost every use case. To save the temporary files created when `save_latent_dynamics=TRUE` see the `$save_latent_dynamics_files()` method.

`output_dir`

(string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at `NULL` (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If `NULL` (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary

files are removed when the fitted model object is garbage collected (manually or automatically).

- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename`
: (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

`sig_figs`
: (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`opencl_ids`
: (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

`num_threads`
: (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections (e.g., for multi-path pathfinder as well as `reduce_sum`).

`init_alpha`
: (positive real) The initial step size parameter.

`tol_obj`
: (positive real) Convergence tolerance on changes in objective function value.

`tol_rel_obj`
: (positive real) Convergence tolerance on relative changes in objective function value.

`tol_grad`
: (positive real) Convergence tolerance on the norm of the gradient.

`tol_rel_grad`
: (positive real) Convergence tolerance on the relative norm of the gradient.

`tol_param`
: (positive real) Convergence tolerance on changes in parameter value.

`history_size`
: (positive integer) The size of the history used when approximating the Hessian.

`single_path_draws`
: (positive integer) Number of draws a single pathfinder should return. The number of draws PSIS sampling samples from will be equal to `single_path_draws * num_paths`.

`draws`
: (positive integer) Number of draws to return after performing pareto smooted importance sampling (PSIS). This should be smaller than `single_path_draws * num_paths` (future versions of CmdStan will throw a warning).

`num_paths`
: (positive integer) Number of single pathfinders to run.

`max_lbfgs_iters`
: (positive integer) The maximum number of iterations for LBFGS.

`num_elbo_draws`
: (positive integer) Number of draws to make when calculating the ELBO of the approximation at each iteration of LBFGS.

`save_single_paths`
: (logical) Whether to save the results of single pathfinder runs in multi-pathfinder.

`psis_resample`
: (logical) Whether to perform pareto smoothed importance sampling. If `TRUE`, the number of draws returned will be equal to `draws`. If `FALSE`, the number of draws returned will be equal to `single_path_draws * num_paths`.

`calculate_lp` (logical) Whether to calculate the log probability of the draws. If `TRUE`, the log probability will be calculated and given in the output. If `FALSE`, the log probability will only be returned for draws used to determine the ELBO in the pathfinder steps. All other draws will have a log probability of `NA`. A value of `FALSE` will also turn off pareto smoothed importance sampling as the lp calculation is needed for PSIS.

`show_messages` (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions` (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`save_cmdstan_config` (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

Value

A `CmdStanPathfinder` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-optimize`, `model-method-sample`, `model-method-sample_mpi`, `model-method-variables`, `model-method-variational`

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
```

```
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()
```

```
# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
```

```
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

---

model-method-sample *Run Stan's MCMC algorithms*

Description

The `$sample()` method of a `CmdStanModel` object runs Stan's main Markov chain Monte Carlo algorithm.

Any argument left as `NULL` will default to the default value used by the installed version of CmdStan. See the CmdStan User's Guide for more details.

After model fitting any diagnostics specified via the `diagnostics` argument will be checked and warnings will be printed if warranted.

Usage

```
sample(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  chains = 4,
  parallel_chains = getOption("mc.cores", 1),
  chain_ids = seq_len(chains),
  threads_per_chain = NULL,
  opencl_ids = NULL,
  iter_warmup = NULL,
  iter_sampling = NULL,
  save_warmup = FALSE,
  thin = NULL,
  max_treedepth = NULL,
  adapt_engaged = TRUE,
  adapt_delta = NULL,
  step_size = NULL,
```

```
  metric = NULL,
  metric_file = NULL,
  inv_metric = NULL,
  init_buffer = NULL,
  term_buffer = NULL,
  window = NULL,
  fixed_param = FALSE,
  show_messages = TRUE,
  show_exceptions = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi"),
  save_metric = NULL,
  save_cmdstan_config = NULL,
  cores = NULL,
  num_cores = NULL,
  num_chains = NULL,
  num_warmup = NULL,
  num_samples = NULL,
  validate_csv = NULL,
  save_extra_diagnostics = NULL,
  max_depth = NULL,
  stepsize = NULL
)
```

Arguments

`data`	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following: • A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan. • A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats. • `NULL` or an empty list if the Stan program has no data block.
`seed`	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
`refresh`	(non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.
`init`	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.;
- The number `0`. This initializes *all* parameters to `0`;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`save_latent_dynamics`
: (logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is `FALSE`, which is appropriate for almost every use case. To save the temporary files created when `save_latent_dynamics=TRUE` see the `$save_latent_dynamics_files()` method.

`output_dir`
: (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be

a single file. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename`

(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

`sig_figs` (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`chains` (positive integer) The number of Markov chains to run. The default is 4.

`parallel_chains`

(positive integer) The *maximum* number of MCMC chains to run in parallel. If `parallel_chains` is not specified then the default is to look for the option `"mc.cores"`, which can be set for an entire R session by `options(mc.cores=value)`. If the `"mc.cores"` option has not been set then the default is 1.

`chain_ids` (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

`threads_per_chain`

(positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections *within* an MCMC chain (e.g., when using the Stan functions `reduce_sum()` or `map_rect()`). This is in contrast with `parallel_chains`, which specifies the number of chains to run in parallel. The actual number of CPU cores used is `parallel_chains*threads_per_chain`. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.

`opencl_ids` (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

`iter_warmup` (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_warmup`.

`iter_sampling` (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_samples`.

`save_warmup` (logical) Should warmup iterations be saved? The default is FALSE.

`thin` (positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.

`max_treedepth` (positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.

`adapt_engaged` (logical) Do warmup adaptation? The default is `TRUE`. If a precomputed inverse metric is specified via the `inv_metric` argument (or `metric_file`) then, if `adapt_engaged=TRUE`, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set `adapt_engaged=FALSE`.

`adapt_delta` (real in `(0,1)`) The adaptation target acceptance statistic.

`step_size` (positive real) The *initial* step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.

`metric` (string) One of `"diag_e"`, `"dense_e"`, or `"unit_e"`, specifying the geometry of the base manifold. See the *Euclidean Metric* section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the `inv_metric` argument below.

`metric_file` (character vector) The paths to JSON or Rdump files (one per chain) compatible with CmdStan that contain precomputed inverse metrics. The `metric_file` argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named `inv_metric`, referring to the *inverse* metric. We recommend instead using CmdStanR's `inv_metric` argument (see below) to specify an inverse metric directly using a vector or matrix from your R session.

`inv_metric` (vector, matrix) A vector (if `metric='diag_e'`) or a matrix (if `metric='dense_e'`) for initializing the inverse metric. This can be used as an alternative to the `metric_file` argument. A vector is interpreted as a diagonal metric. The inverse metric is usually set to an estimate of the posterior covariance. See the `adapt_engaged` argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.

`init_buffer` (nonnegative integer) Width of initial fast timestep adaptation interval during warmup.

`term_buffer` (nonnegative integer) Width of final fast timestep adaptation interval during warmup.

`window` (nonnegative integer) Initial width of slow timestep/metric adaptation interval.

`fixed_param` (logical) When `TRUE`, call CmdStan with argument `"algorithm=fixed_param"`. The default is `FALSE`. The fixed parameter sampler generates a new sample without changing the current state of the Markov chain; only generated quantities may change. This can be useful when, for example, trying to generate pseudo-data using the generated quantities block. If the parameters block is empty then using `fixed_param=TRUE` is mandatory. When `fixed_param=TRUE` the `chains` and `parallel_chains` arguments will be set to `1`.

`show_messages` (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions`
: (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`diagnostics`
: (character vector) The diagnostics to automatically check and warn about after sampling. Setting this to an empty string `""` or `NULL` can be used to prevent CmdStanR from automatically reading in the sampler diagnostics from CSV if you wish to manually read in the results and validate them yourself, for example using `read_cmdstan_csv()`. The currently available diagnostics are `"divergences"`, `"treedepth"`, and `"ebfmi"` (the default is to check all of them).

  These diagnostics are also available after fitting. The `$sampler_diagnostics()` method provides access the diagnostic values for each iteration and the `$diagnostic_summary()` method provides summaries of the diagnostics and can regenerate the warning messages.

  Diagnostics like R-hat and effective sample size are *not* currently available via the `diagnostics` argument but can be checked after fitting using the `$summary()` method.

`save_metric`
: (logical) When `TRUE`, call CmdStan with argument `"adaptation save_metric=1"` to save the adapted metric in separate JSON file with elements "stepsize", "metric_type" and "inv_metric". The default is `TRUE`. This option is only available in CmdStan 2.34.0 and later.

`save_cmdstan_config`
: (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

`cores`, `num_cores`, `num_chains`, `num_warmup`, `num_samples`, `save_extra_diagnostics`, `max_depth`, `stepsize`, `validate_csv`
: Deprecated and will be removed in a future release.

Value

A `CmdStanMCMC` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: `model-method-check_syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-generate-quantities`,

model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample_mpi, model-method-variables, model-method-variational

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)

# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))
```

```
# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()

# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()
```

```
# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

`model-method-sample_mpi`

Run Stan's MCMC algorithms with MPI

Description

The `$sample_mpi()` method of a `CmdStanModel` object is identical to the `$sample()` method but with support for MPI (message passing interface). The target audience for MPI are those with large computer clusters. For other users, the `$sample()` method provides both parallelization of chains and threading support for within-chain parallelization.

In order to use MPI with Stan, an MPI implementation must be installed. For Unix systems the most commonly used implementations are MPICH and OpenMPI. The implementations provide an MPI C++ compiler wrapper (for example mpicxx), which is required to compile the model.

An example of compiling with MPI:

```
mpi_options = list(STAN_MPI=TRUE, CXX="mpicxx", TBB_CXX_TYPE="gcc")
mod = cmdstan_model("model.stan", cpp_options = mpi_options)
```

The C++ options that must be supplied to the compile call are:

- `STAN_MPI`: Enables the use of MPI with Stan if `TRUE`.
- `CXX`: The name of the MPI C++ compiler wrapper. Typically `"mpicxx"`.
- `TBB_CXX_TYPE`: The C++ compiler the MPI wrapper wraps. Typically `"gcc"` on Linux and `"clang"` on macOS.

In the call to the `$sample_mpi()` method it is also possible to provide the name of the MPI launcher (`mpi_cmd`, defaulting to `"mpiexec"`) and any other MPI launch arguments (`mpi_args`). In most cases, it is enough to only define the number of processes. To use `n_procs` processes specify `mpi_args = list("n" = n_procs)`.

Usage

```
sample_mpi(
  data = NULL,
  mpi_cmd = "mpiexec",
  mpi_args = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  chains = 1,
  chain_ids = seq_len(chains),
  iter_warmup = NULL,
  iter_sampling = NULL,
  save_warmup = FALSE,
  thin = NULL,
  max_treedepth = NULL,
  adapt_engaged = TRUE,
  adapt_delta = NULL,
  step_size = NULL,
  metric = NULL,
  metric_file = NULL,
  inv_metric = NULL,
  init_buffer = NULL,
  term_buffer = NULL,
  window = NULL,
  fixed_param = FALSE,
  sig_figs = NULL,
  show_messages = TRUE,
  show_exceptions = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi"),
  save_cmdstan_config = NULL,
  validate_csv = TRUE
)
```

Arguments

`data` (multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:

- A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()`

for details on the conversions performed on R objects before they are passed to Stan.

- A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats.
- `NULL` or an empty list if the Stan program has no data block.

`mpi_cmd` (string) The MPI launcher used for launching MPI processes. The default launcher is `"mpiexec"`.

`mpi_args` (list) A list of arguments to use when launching MPI processes. For example, `mpi_args = list("n" = 4)` launches the executable as `mpiexec -n 4 model_executable`, followed by CmdStan arguments for the model executable.

`seed` (positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.

`refresh` (non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.

`init` (multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:

- A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.;
- The number `0`. This initializes *all* parameters to `0`;
- A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan.
- A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**.
- A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When

a CmdStanPathfinder fit object is used as the init, if . psis_resample was set to FALSE and calculate_lp was set to TRUE (default), then resampling without replacement with Pareto smoothed weights will be used. If psis_resample was set to TRUE or calculate_lp was set to FALSE then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for CmdStanVB, and CmdStanLaplace fit objects.

- A type inheriting from posterior::draws. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

save_latent_dynamics

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's diagnostic_file argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is FALSE, which is appropriate for almost every use case. To save the temporary files created when save_latent_dynamics=TRUE see the $save_latent_dynamics_files() method.

output_dir (string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at NULL (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using options(cmdstanr_output_dir). The behavior of output_dir is as follows:

- If NULL (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the $save_* methods of the fitted model object (e.g., $save_output_files()). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in output_dir with names corresponding to the defaults used by $save_output_files().

output_basename

(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If NULL (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

chains (positive integer) The number of Markov chains to run. The default is 4.

chain_ids (integer vector) A vector of chain IDs. Must contain as many unique positive integers as the number of chains. If not set, the default chain IDs are used (integers starting from 1).

iter_warmup (positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as num_warmup.

`iter_sampling` (positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as `num_samples`.

`save_warmup` (logical) Should warmup iterations be saved? The default is `FALSE`.

`thin` (positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.

`max_treedepth` (positive integer) The maximum allowed tree depth for the NUTS engine. See the *Tree Depth* section of the CmdStan User's Guide for more details.

`adapt_engaged` (logical) Do warmup adaptation? The default is `TRUE`. If a precomputed inverse metric is specified via the `inv_metric` argument (or `metric_file`) then, if `adapt_engaged=TRUE`, Stan will use the provided inverse metric just as an initial guess during adaptation. To turn off adaptation when using a precomputed inverse metric set `adapt_engaged=FALSE`.

`adapt_delta` (real in `(0,1)`) The adaptation target acceptance statistic.

`step_size` (positive real) The *initial* step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.

`metric` (string) One of `"diag_e"`, `"dense_e"`, or `"unit_e"`, specifying the geometry of the base manifold. See the *Euclidean Metric* section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the `inv_metric` argument below.

`metric_file` (character vector) The paths to JSON or Rdump files (one per chain) compatible with CmdStan that contain precomputed inverse metrics. The `metric_file` argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named `inv_metric`, referring to the *inverse* metric. We recommend instead using CmdStanR's `inv_metric` argument (see below) to specify an inverse metric directly using a vector or matrix from your R session.

`inv_metric` (vector, matrix) A vector (if `metric='diag_e'`) or a matrix (if `metric='dense_e'`) for initializing the inverse metric. This can be used as an alternative to the `metric_file` argument. A vector is interpreted as a diagonal metric. The inverse metric is usually set to an estimate of the posterior covariance. See the `adapt_engaged` argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.

`init_buffer` (nonnegative integer) Width of initial fast timestep adaptation interval during warmup.

`term_buffer` (nonnegative integer) Width of final fast timestep adaptation interval during warmup.

`window` (nonnegative integer) Initial width of slow timestep/metric adaptation interval.

`fixed_param` (logical) When `TRUE`, call CmdStan with argument `"algorithm=fixed_param"`. The default is `FALSE`. The fixed parameter sampler generates a new sample without changing the current state of the Markov chain; only generated quantities may change. This can be useful when, for example, trying to generate pseudo-data using the generated quantities block. If the parameters block is empty then using `fixed_param=TRUE` is mandatory. When `fixed_param=TRUE` the `chains` and `parallel_chains` arguments will be set to `1`.

`sig_figs` (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`show_messages` (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions` (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`diagnostics` (character vector) The diagnostics to automatically check and warn about after sampling. Setting this to an empty string `""` or `NULL` can be used to prevent CmdStanR from automatically reading in the sampler diagnostics from CSV if you wish to manually read in the results and validate them yourself, for example using `read_cmdstan_csv()`. The currently available diagnostics are `"divergences"`, `"treedepth"`, and `"ebfmi"` (the default is to check all of them).

These diagnostics are also available after fitting. The `$sampler_diagnostics()` method provides access the diagnostic values for each iteration and the `$diagnostic_summary()` method provides summaries of the diagnostics and can regenerate the warning messages.

Diagnostics like R-hat and effective sample size are *not* currently available via the `diagnostics` argument but can be checked after fitting using the `$summary()` method.

`save_cmdstan_config` (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

`validate_csv` Deprecated. Use `diagnostics` instead.

Value

A `CmdStanMCMC` object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

The Stan Math Library's documentation (mc-stan.org/math) for more details on MPI support in Stan.

Other CmdStanModel methods: model-method-check_syntax, model-method-compile, model-method-diagnose, model-method-expose_functions, model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample, model-method-variables, model-method-variational

Examples

```
## Not run:
# mpi_options <- list(STAN_MPI=TRUE, CXX="mpicxx", TBB_CXX_TYPE="gcc")
# mod <- cmdstan_model("model.stan", cpp_options = mpi_options)
# fit <- mod$sample_mpi(..., mpi_args = list("n" = 4))

## End(Not run)
```

`model-method-variables`

Input and output variables of a Stan program

Description

The `$variables()` method of a `CmdStanModel` object returns a list, each element representing a Stan model block: `data`, `parameters`, `transformed_parameters` and `generated_quantities`.

Each element contains a list of variables, with each variables represented as a list with infromation on its scalar type (`real` or `int`) and number of dimensions.
`transformed data` is not included, as variables in that block are not part of the model's input or output.

Usage

```
variables()
```

Value

The `$variables()` returns a list with information on input and output variables for each of the Stan model blocks.

See Also

Other CmdStanModel methods: model-method-check_syntax, model-method-compile, model-method-diagnose, model-method-expose_functions, model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variational

Examples

```
## Not run:
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")

# create a `CmdStanModel` object, compiling the model is not required
mod <- cmdstan_model(file, compile = FALSE)

mod$variables()


## End(Not run)
```

`model-method-variational`

Run Stan's variational approximation algorithms

Description

The `$variational()` method of a `CmdStanModel` object runs Stan's Automatic Differentiation Variational Inference (ADVI) algorithms. The approximation is a Gaussian in the unconstrained variable space. Stan implements two ADVI algorithms: the `algorithm="meanfield"` option uses a fully factorized Gaussian for the approximation; the `algorithm="fullrank"` option uses a Gaussian with a full-rank covariance matrix for the approximation. See the CmdStan User's Guide for more details.

Any argument left as `NULL` will default to the default value used by the installed version of CmdStan.

Usage

```
variational(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  save_latent_dynamics = FALSE,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  threads = NULL,
  opencl_ids = NULL,
  algorithm = NULL,
  iter = NULL,
  grad_samples = NULL,
  elbo_samples = NULL,
  eta = NULL,
  adapt_engaged = NULL,
  adapt_iter = NULL,
```

```
  tol_rel_obj = NULL,
  eval_elbo = NULL,
  output_samples = NULL,
  draws = NULL,
  show_messages = TRUE,
  show_exceptions = TRUE,
  save_cmdstan_config = NULL
)
```

Arguments

`data`	(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following: • A named list of R objects with the names corresponding to variables declared in the data block of the Stan program. Internally this list is then written to JSON for CmdStan using `write_stan_json()`. See `write_stan_json()` for details on the conversions performed on R objects before they are passed to Stan. • A path to a data file compatible with CmdStan (JSON or R dump). See the appendices in the CmdStan guide for details on using these formats. • `NULL` or an empty list if the Stan program has no data block.
`seed`	(positive integer(s)) A seed for the (P)RNG to pass to CmdStan. In the case of multi-chain sampling the single `seed` will automatically be augmented by the the run (chain) ID so that each chain uses a different seed. The exception is the transformed data block, which defaults to using same seed for all chains so that the same data is generated for all chains if RNG functions are used. The only time `seed` should be specified as a vector (one element per chain) is if RNG functions are used in transformed data and the goal is to generate *different* data for each chain.
`refresh`	(non-negative integer) The number of iterations between printed screen updates. If `refresh = 0`, only error messages will be printed.
`init`	(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following: • A real number `x>0`. This initializes *all* parameters randomly between `[-x,x]` on the *unconstrained* parameter space.; • The number `0`. This initializes *all* parameters to `0`; • A character vector of paths (one per chain) to JSON or Rdump files containing initial values for all or some parameters. See `write_stan_json()` to write R objects to JSON files compatible with CmdStan. • A list of lists containing initial values for all or some parameters. For MCMC the list should contain a sublist for each chain. For other model fitting methods there should be just one sublist. The sublists should have named elements corresponding to the parameters for which you are specifying initial values. See **Examples**. • A function that returns a single list with names corresponding to the parameters for which you are specifying initial values. The function can take no

arguments or a single argument `chain_id`. For MCMC, if the function has argument `chain_id` it will be supplied with the chain id (from 1 to number of chains) when called to generate the initial values. See **Examples**.

- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanVB`, `CmdStanPathfinder`, or `CmdStanLaplace` fit object. If the fit object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model. For the `sample` and `pathfinder` method, if the fit object has fewer draws than the requested number of chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. When a `CmdStanPathfinder` fit object is used as the init, if . `psis_resample` was set to `FALSE` and `calculate_lp` was set to `TRUE` (default), then resampling without replacement with Pareto smoothed weights will be used. If `psis_resample` was set to `TRUE` or `calculate_lp` was set to `FALSE` then sampling without replacement with uniform weights will be used to select the draws. PSIS resampling is used to select the draws for `CmdStanVB`, and `CmdStanLaplace` fit objects.
- A type inheriting from `posterior::draws`. If the draws object has less samples than the number of requested chains/paths then the inits will be drawn using sampling with replacement. Otherwise sampling without replacement will be used. If the draws object's parameters are only a subset of the model parameters then the other parameters will be drawn by Stan's default initialization. The fit object must have at least some parameters that are the same name and dimensions as the current Stan model.

`save_latent_dynamics`

(logical) Should auxiliary diagnostic information about the latent dynamics be written to temporary diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument and the content written to CSV is controlled by the user's CmdStan installation and not CmdStanR (for some algorithms no content may be written). The default is `FALSE`, which is appropriate for almost every use case. To save the temporary files created when `save_latent_dynamics=TRUE` see the `$save_latent_dynamics_files()` method.

`output_dir`

(string) A path to a directory where CmdStan should write its output CSV files. For MCMC there will be one file per chain; for other methods there will be a single file. For interactive use this can typically be left at `NULL` (temporary directory) since CmdStanR makes the CmdStan output (posterior draws and diagnostics) available in R via methods of the fitted model objects. This can be set for an entire R session using `options(cmdstanr_output_dir)`. The behavior of `output_dir` is as follows:

- If `NULL` (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the `$save_*` methods of the fitted model object (e.g., `$save_output_files()`). These temporary files are removed when the fitted model object is garbage collected (manually or automatically).
- If a path, then the files are created in `output_dir` with names corresponding to the defaults used by `$save_output_files()`.

`output_basename`
: (string) A string to use as a prefix for the names of the output CSV files of CmdStan. If `NULL` (the default), the basename of the output CSV files will be comprised from the model name, timestamp, and 5 random characters.

`sig_figs`
: (positive integer) The number of significant figures used when storing the output values. By default, CmdStan represent the output values with 6 significant figures. The upper limit for `sig_figs` is 18. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.

`threads`
: (positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections (e.g., when using the Stan functions `reduce_sum()` or `map_rect()`).

`opencl_ids`
: (integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with `cpp_options = list(stan_opencl = TRUE)` for this argument to have an effect.

`algorithm`
: (string) The algorithm. Either `"meanfield"` or `"fullrank"`.

`iter`
: (positive integer) The *maximum* number of iterations.

`grad_samples`
: (positive integer) The number of samples for Monte Carlo estimate of gradients.

`elbo_samples`
: (positive integer) The number of samples for Monte Carlo estimate of ELBO (objective function).

`eta`
: (positive real) The step size weighting parameter for adaptive step size sequence.

`adapt_engaged`
: (logical) Do warmup adaptation?

`adapt_iter`
: (positive integer) The *maximum* number of adaptation iterations.

`tol_rel_obj`
: (positive real) Convergence tolerance on the relative norm of the objective.

`eval_elbo`
: (positive integer) Evaluate ELBO every Nth iteration.

`output_samples`
: (positive integer) Use `draws` argument instead. `output_samples` will be deprecated in the future.

`draws`
: (positive integer) Number of approximate posterior samples to draw and save.

`show_messages`
: (logical) When `TRUE` (the default), prints all output during the execution process, such as iteration numbers and elapsed times. If the output is silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`show_exceptions`
: (logical) When `TRUE` (the default), prints all informational messages, for example rejection of the current proposal. Disable if you wish to silence these messages, but this is not usually recommended unless you are very confident that the model is correct up to numerical error. If the messages are silenced then the `$output()` method of the resulting fit object can be used to display the silenced messages.

`save_cmdstan_config`
: (logical) When `TRUE` (the default), call CmdStan with argument `"output save_config=1"` to save a json file which contains the argument tree and extra information (equivalent to the output CSV file header). This option is only available in CmdStan 2.34.0 and later.

Value

A CmdStanVB object.

See Also

The CmdStanR website (mc-stan.org/cmdstanr) for online documentation and tutorials.

The Stan and CmdStan documentation:

- Stan documentation: mc-stan.org/users/documentation
- CmdStan User's Guide: mc-stan.org/docs/cmdstan-guide

Other CmdStanModel methods: model-method-check_syntax, model-method-compile, model-method-diagnose, model-method-expose_functions, model-method-format, model-method-generate-quantities, model-method-laplace, model-method-optimize, model-method-pathfinder, model-method-sample, model-method-sample_mpi, model-method-variables

Examples

```
## Not run:
library(cmdstanr)
library(posterior)
library(bayesplot)
color_scheme_set("brightblue")

# Set path to CmdStan
# (Note: if you installed CmdStan via install_cmdstan() with default settings
# then setting the path is unnecessary but the default below should still work.
# Otherwise use the `path` argument to specify the location of your
# CmdStan installation.)
set_cmdstan_path(path = NULL)

# Create a CmdStanModel object from a Stan program,
# here using the example model that comes with CmdStan
file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.stan")
mod <- cmdstan_model(file)
mod$print()
# Print with line numbers. This can be set globally using the
# `cmdstanr_print_line_numbers` option.
mod$print(line_numbers = TRUE)

# Data as a named list (like RStan)
stan_data <- list(N = 10, y = c(0,1,0,0,0,0,0,0,0,1))

# Run MCMC using the 'sample' method
fit_mcmc <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  parallel_chains = 2
)
```

```
# Use 'posterior' package for summaries
fit_mcmc$summary()

# Check sampling diagnostics
fit_mcmc$diagnostic_summary()

# Get posterior draws
draws <- fit_mcmc$draws()
print(draws)

# Convert to data frame using posterior::as_draws_df
as_draws_df(draws)

# Plot posterior using bayesplot (ggplot2)
mcmc_hist(fit_mcmc$draws("theta"))

# For models fit using MCMC, if you like working with RStan's stanfit objects
# then you can create one with rstan::read_stan_csv()
# stanfit <- rstan::read_stan_csv(fit_mcmc$output_files())


# Run 'optimize' method to get a point estimate (default is Stan's LBFGS algorithm)
# and also demonstrate specifying data as a path to a file instead of a list
my_data_file <- file.path(cmdstan_path(), "examples/bernoulli/bernoulli.data.json")
fit_optim <- mod$optimize(data = my_data_file, seed = 123)
fit_optim$summary()

# Run 'optimize' again with 'jacobian=TRUE' and then draw from Laplace approximation
# to the posterior
fit_optim <- mod$optimize(data = my_data_file, jacobian = TRUE)
fit_laplace <- mod$laplace(data = my_data_file, mode = fit_optim, draws = 2000)
fit_laplace$summary()

# Run 'variational' method to use ADVI to approximate posterior
fit_vb <- mod$variational(data = stan_data, seed = 123)
fit_vb$summary()
mcmc_hist(fit_vb$draws("theta"))

# Run 'pathfinder' method, a new alternative to the variational method
fit_pf <- mod$pathfinder(data = stan_data, seed = 123)
fit_pf$summary()
mcmc_hist(fit_pf$draws("theta"))

# Run 'pathfinder' again with more paths, fewer draws per path,
# better covariance approximation, and fewer LBFGSs iterations
fit_pf <- mod$pathfinder(data = stan_data, num_paths=10, single_path_draws=40,
                         history_size=50, max_lbfgs_iters=100)

# Specifying initial values as a function
fit_mcmc_w_init_fun <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
```

```
  refresh = 0,
  init = function() list(theta = runif(1))
)
fit_mcmc_w_init_fun_2 <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = function(chain_id) {
    # silly but demonstrates optional use of chain_id
    list(theta = 1 / (chain_id + 1))
  }
)
fit_mcmc_w_init_fun_2$init()

# Specifying initial values as a list of lists
fit_mcmc_w_init_list <- mod$sample(
  data = stan_data,
  seed = 123,
  chains = 2,
  refresh = 0,
  init = list(
    list(theta = 0.75), # chain 1
    list(theta = 0.25)  # chain 2
  )
)
fit_optim_w_init_list <- mod$optimize(
  data = stan_data,
  seed = 123,
  init = list(
    list(theta = 0.75)
  )
)
fit_optim_w_init_list$init()

## End(Not run)
```

`read_cmdstan_csv` *Read CmdStan CSV files into R*

Description

`read_cmdstan_csv()` is used internally by CmdStanR to read CmdStan's output CSV files into R. It can also be used by CmdStan users as a more flexible and efficient alternative to `rstan::read_stan_csv()`. See the **Value** section for details on the structure of the returned list.

It is also possible to create CmdStanR's fitted model objects directly from CmdStan CSV files using the `as_cmdstan_fit()` function.

Usage

```
read_cmdstan_csv(
  files,
  variables = NULL,
  sampler_diagnostics = NULL,
  format = getOption("cmdstanr_draws_format", NULL)
)

as_cmdstan_fit(
  files,
  check_diagnostics = TRUE,
  format = getOption("cmdstanr_draws_format")
)
```

Arguments

`files` (character vector) The paths to the CmdStan CSV files. These can be files generated by running CmdStanR or running CmdStan directly.

`variables` (character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in.

- If `NULL` (the default) then all variables are included.
- If an empty string (`variables=""`) then none are included.
- For non-scalar variables all elements or specific elements can be selected:
  - `variables = "theta"` selects all elements of `theta`;
  - `variables = c("theta[1]", "theta[3]")` selects only the 1st and 3rd elements.

`sampler_diagnostics` (character vector) Works the same way as `variables` but for sampler diagnostic variables (e.g., `"treedepth__"`, `"accept_stat__"`, etc.). Ignored if the model was not fit using MCMC.

`format` (string) The format for storing the draws or point estimates. The default depends on the method used to fit the model. See draws for details, in particular the note about speed and memory for models with many parameters.

`check_diagnostics` (logical) For models fit using MCMC, should diagnostic checks be performed after reading in the files? The default is `TRUE` but set to `FALSE` to avoid checking for problems with divergences and treedepth.

Value

`as_cmdstan_fit()` returns a CmdStanMCMC, CmdStanMLE, CmdStanLaplace or CmdStanVB object. Some methods typically defined for those objects will not work (e.g. `save_data_file()`) but the important methods like `$summary()`, `$draws()`, `$sampler_diagnostics()` and others will work fine.

`read_cmdstan_csv()` returns a named list with the following components:

- metadata: A list of the meta information from the run that produced the CSV file(s). See **Examples** below.

The other components in the returned list depend on the method that produced the CSV file(s).

For sampling the returned list also includes the following components:

- time: Run time information for the individual chains. The returned object is the same as for the $time() method except the total run time can't be inferred from the CSV files (the chains may have been run in parallel) and is therefore NA.
- inv_metric: A list (one element per chain) of inverse mass matrices or their diagonals, depending on the type of metric used.
- step_size: A list (one element per chain) of the step sizes used.
- warmup_draws: If save_warmup was TRUE when fitting the model then a draws_array (or different format if format is specified) of warmup draws.
- post_warmup_draws: A draws_array (or different format if format is specified) of post-warmup draws.
- warmup_sampler_diagnostics: If save_warmup was TRUE when fitting the model then a draws_array (or different format if format is specified) of warmup draws of the sampler diagnostic variables.
- post_warmup_sampler_diagnostics: A draws_array (or different format if format is specified) of post-warmup draws of the sampler diagnostic variables.

For optimization the returned list also includes the following components:

- point_estimates: Point estimates for the model parameters.

For laplace and variational inference the returned list also includes the following components:

- draws: A draws_matrix (or different format if format is specified) of draws from the approximate posterior distribution.

For standalone generated quantities the returned list also includes the following components:

- generated_quantities: A draws_array of the generated quantities.

Examples

```
## Not run:
# Generate some CSV files to use for demonstration
fit1 <- cmdstanr_example("logistic", method = "sample", save_warmup = TRUE)
csv_files <- fit1$output_files()
print(csv_files)

# Creating fitting model objects

# Create a CmdStanMCMC object from the CSV files
fit2 <- as_cmdstan_fit(csv_files)
fit2$print("beta")

# Using read_cmdstan_csv
```

```
#
# Read in everything
x <- read_cmdstan_csv(csv_files)
str(x)

# Don't read in any of the sampler diagnostic variables
x <- read_cmdstan_csv(csv_files, sampler_diagnostics = "")

# Don't read in any of the parameters or generated quantities
x <- read_cmdstan_csv(csv_files, variables = "")

# Read in only specific parameters and sampler diagnostics
x <- read_cmdstan_csv(
  csv_files,
  variables = c("alpha", "beta[2]"),
  sampler_diagnostics = c("n_leapfrog__", "accept_stat__")
)

# For non-scalar parameters all elements can be selected or only some elements,
# e.g. all of the vector "beta" but only one element of the vector "log_lik"
x <- read_cmdstan_csv(
  csv_files,
  variables = c("beta", "log_lik[3]")
)

## End(Not run)
```

`register_knitr_engine` *Register CmdStanR's knitr engine for Stan*

Description

Registers CmdStanR's knitr engine `eng_cmdstan()` for processing Stan chunks. Refer to the vignette R Markdown CmdStan Engine for a demonstration.

Usage

```
register_knitr_engine(override = TRUE)
```

Arguments

`override`	(logical) Override knitr's built-in, RStan-based engine for Stan? The default is `TRUE`. See **Details**.

Details

If `override = TRUE` (default), this registers CmdStanR's knitr engine as the engine for `stan` chunks, replacing knitr's built-in, RStan-based engine. If `override = FALSE`, this registers a `cmdstan` engine

so that both engines may be used in the same R Markdown document. If the template supports syntax highlighting for the Stan language, the `cmdstan` chunks will have `stan` syntax highlighting applied to them.

See the vignette R Markdown CmdStan Engine for an example.

Note: When running chunks interactively in RStudio (e.g. when using R Notebooks), it has been observed that the built-in, RStan-based engine is used for `stan` chunks even when CmdStanR's engine has been registered in the session. When the R Markdown document is knit/rendered, the correct engine is used. As a workaround, when running chunks interactively, it is recommended to use the `override = FALSE` option and change `stan` chunks to be `cmdstan` chunks.

If you would like to keep `stan` chunks as `stan` chunks, it is possible to specify `engine = "cmdstan"` in the chunk options after registering the `cmdstan` engine with `override = FALSE`.

References

- Register a custom language engine for knitr
- knitr's built-in Stan language engine

`set_cmdstan_path` *Get or set the file path to the CmdStan installation*

Description

Use the `set_cmdstan_path()` function to tell CmdStanR where the CmdStan installation in located. Once the path has been set, `cmdstan_path()` will return the full path to the CmdStan installation and `cmdstan_version()` will return the CmdStan version number. See **Details** for how to avoid manually setting the path in each R session.

Usage

```
set_cmdstan_path(path = NULL)

cmdstan_path()

cmdstan_version(error_on_NA = TRUE)
```

Arguments

`path`	(string) The full file path to the CmdStan installation. If `NULL` (the default) then the path is set to the default path used by `install_cmdstan()` if it exists.
`error_on_NA`	(logical) Should an error be thrown if CmdStan is not found. The default is `TRUE`. If `FALSE`, `cmdstan_version()` returns `NULL`.

Details

Before the package can be used it needs to know where the CmdStan installation is located. When the package is loaded it tries to help automate this to avoid having to manually set the path every session:

- If the environment variable `"CMDSTAN"` exists at load time then its value will be automatically set as the default path to CmdStan for the R session. If the environment variable `"CMDSTAN"` is set, but a valid CmdStan is not found in the supplied path, the path is treated as a top folder that contains CmdStan installations. In that case, the CmdStan installation with the largest version number will be set as the path to CmdStan for the R session.
- If no environment variable is found when loaded but any directory in the form `".cmdstan/cmdstan-[version]"` (e.g., `".cmdstan/cmdstan-2.23.0"`), exists in the user's home directory (`Sys.getenv("HOME")`, *not* the current working directory) then the path to the cmdstan with the largest version number will be set as the path to CmdStan for the R session. This is the same as the default directory that `install_cmdstan()` would use to install the latest version of CmdStan.

It is always possible to change the path after loading the package using `set_cmdstan_path(path)`.

Value

A string. Either the file path to the CmdStan installation or the CmdStan version number.

CmdStan version string if available. If CmdStan is not found and `error_on_NA` is `FALSE`, `cmdstan_version()` returns `NULL`.

`write_stan_file` *Write Stan code to a file*

Description

Convenience function for writing Stan code to a (possibly temporary) file with a `.stan` extension. By default, the file name is chosen deterministically based on a hash of the Stan code, and the file is not overwritten if it already has correct contents. This means that calling this function multiple times with the same Stan code will reuse the compiled model. This also however means that the function is potentially not thread-safe. Using `hash_salt = Sys.getpid()` should ensure thread-safety in the rare cases when it is needed.

Usage

```
write_stan_file(
  code,
  dir = getOption("cmdstanr_write_stan_file_dir", tempdir()),
  basename = NULL,
  force_overwrite = FALSE,
  hash_salt = ""
)
```

Arguments

code	(character vector) The Stan code to write to the file. This can be a character vector of length one (a string) containing the entire Stan program or a character vector with each element containing one line of the Stan program.
dir	(string) An optional path to the directory where the file will be written. If omitted, a global option `cmdstanr_write_stan_file_dir` is used. If the global options is not set, temporary directory is used.
basename	(string) If `dir` is specified, optionally the basename to use for the file created. If not specified a file name is generated from hashing the code.
force_overwrite	(logical) If set to `TRUE` the file will always be overwritten and thus the resulting model will always be recompiled.
hash_salt	(string) Text to add to the model code prior to hashing to determine the file name if `basename` is not set.

Value

The path to the file.

Examples

```
# stan program as a single string
stan_program <- "
data {
  int<lower=0> N;
  array[N] int<lower=0,upper=1> y;
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  y ~ bernoulli(theta);
}
"

f <- write_stan_file(stan_program)
print(f)

lines <- readLines(f)
print(lines)
cat(lines, sep = "\n")

# stan program as character vector of lines
f2 <- write_stan_file(lines)
identical(readLines(f), readLines(f2))
```

write_stan_json *Write data to a JSON file readable by CmdStan*

Description

Write data to a JSON file readable by CmdStan

Usage

```
write_stan_json(data, file, always_decimal = FALSE)
```

Arguments

data	(list) A named list of R objects.
file	(string) The path to where the data file should be written.
always_decimal	(logical) Force generate non-integers with decimal points to better distinguish between integers and floating point values. If TRUE all R objects in data intended for integers must be of integer type.

Details

write_stan_json() performs several conversions before writing the JSON file:

- logical -> integer (TRUE -> 1, FALSE -> 0)
- data.frame -> matrix (via data.matrix())
- list -> array
- table -> vector, matrix, or array (depending on dimensions of table)

The list to array conversion is intended to make it easier to prepare the data for certain Stan declarations involving arrays:

- vector[J] v[K] (or equivalently array[K] vector[J] v as of Stan 2.27) can be constructed in R as a list with K elements where each element a vector of length J
- matrix[I,J] v[K] (or equivalently array[K] matrix[I,J] m as of Stan 2.27) can be constructed in R as a list with K elements where each element an IxJ matrix

These can also be passed in from R as arrays instead of lists but the list option is provided for convenience. Unfortunately for arrays with more than one dimension, e.g., vector[J] v[K,L] (or equivalently array[K,L] vector[J] v as of Stan 2.27) it is not possible to use an R list and an array must be used instead. For this example the array in R should have dimensions KxLxJ.

Examples

```
x <- matrix(rnorm(10), 5, 2)
y <- rpois(nrow(x), lambda = 10)
z <- c(TRUE, FALSE)
data <- list(N = nrow(x), K = ncol(x), x = x, y = y, z = z)

# write data to json file
file <- tempfile(fileext = ".json")
write_stan_json(data, file)

# check the contents of the file
cat(readLines(file), sep = "\n")


# demonstrating list to array conversion
# suppose x is declared as `vector[3] x[2]` (or equivalently `array[2] vector[3] x`)
# we can use a list of length 2 where each element is a vector of length 3
data <- list(x = list(1:3, 4:6))
file <- tempfile(fileext = ".json")
write_stan_json(data, file)
cat(readLines(file), sep = "\n")
```

Index