

Package: cmdstanr (via r-universe)

October 13, 2026

Title R Interface to 'CmdStan'

Version 0.9.0.9002

Date 2025-03-30

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

Roxygen list(markdown = TRUE, r6 = FALSE)

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

Depends R (>= 4.0.0)

Imports checkmate, data.table (>= 1.18.0), jsonlite (>= 1.8.7),
methods, posterior (>= 1.5.0), processx (>= 3.5.0), R6 (>= 2.4.0), vctrs, withr (>= 2.5.0), rlang (>= 0.4.7)

Suggests bayesplot, fs, ggplot2, knitr (>= 1.37), loo (>= 2.0.0), qs2,
rmarkdown, testthat (>= 3.3.0), Rcpp

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Repository <https://coalesce-lab.r-universe.dev>

Date/Publication 2026-10-13 19:44:40 UTC

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

RemoteRef HEAD

RemoteSha ee43ebd66754d9af30c5879e315ccd38b5345fb2

Contents

cmdstanr-package	3
as_draws.CmdStanMCMC	7
as_mcmc.list	9
cmdstan_coercion	10
cmdstan_model	11
CmdStanDiagnose	14
CmdStanGQ	15
CmdStanLaplace	17
CmdStanMCMC	19
CmdStanMLE	21
CmdStanModel	23
CmdStanPathfinder	26
cmdstanr_example	28
cmdstanr_global_options	30
CmdStanVB	31
draws_to_csv	33
eng_cmdstan	34
fit-method-cmdstan_summary	35
fit-method-code	36
fit-method-constrain_variables	36
fit-method-diagnostic_summary	37
fit-method-draws	38
fit-method-fitted_params_files	41
fit-method-grad_log_prob	41
fit-method-gradients	42
fit-method-hessian	43
fit-method-init	44
fit-method-init_model_methods	44
fit-method-inv_metric	45
fit-method-log_prob	47
fit-method-loo	48
fit-method-lp	49
fit-method-materialize	51
fit-method-metadata	51
fit-method-mle	52
fit-method-mode	53
fit-method-num_chains	53
fit-method-output	54
fit-method-profiles	55
fit-method-return_codes	56
fit-method-sampler_diagnostics	57
fit-method-save_object	58

fit-method-save_output_files	59
fit-method-summary	61
fit-method-time	63
fit-method-unconstrain_draws	64
fit-method-unconstrain_variables	65
fit-method-variable_skeleton	66
install_cmdstan	67
model-method-check_syntax	70
model-method-cmdstan_defaults	71
model-method-compile	72
model-method-diagnose	75
model-method-expose_functions	79
model-method-format	80
model-method-generate-quantities	82
model-method-laplace	86
model-method-model-info	90
model-method-optimize	92
model-method-pathfinder	98
model-method-sample	106
model-method-sample_mpi	114
model-method-variables	121
model-method-variational	122
print_stan_file	128
read_cmdstan_csv	130
register_knitr_engine	133
set_cmdstan_path	134
write_stan_file	135
write_stan_json	137

Index	140
--------------	------------

cmdstanr-package	<i>CmdStanR: the R interface to CmdStan</i>
------------------	---

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 command line interface to Stan (mc-stan.org/users/interfaces/cmdstan).

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

The RStan interface (**rstan** package) provides its core functionality through an in-memory interface to Stan and relies on R packages such as **Rcpp** to call C++ code from R. CmdStanR's core model compilation and inference workflow instead runs CmdStan in external processes and reads

the resulting output files. Only optional CmdStanR features, such as `$expose_functions()` and the additional model methods, use **Rcpp** to call compiled C++ code directly from R.

Advantages of RStan:

- CRAN provides binary versions of RStan for Windows and macOS. RStan-based packages can also include precompiled Stan models in their binary packages, which allows users to run the models without a local C++ toolchain. CmdStanR-based packages can use `instantiate` to compile models once during package installation. Because CRAN's build machines do not provide CmdStan, packages using this workflow currently need to be installed from source with CmdStan and a C++ toolchain available.
- Avoids use of R6 classes, which may result in more familiar syntax for many R users.

Advantages of CmdStanR:

- CmdStan is installed separately from CmdStanR, so users can often update to a new Stan release by updating CmdStan without waiting for a new CmdStanR release.
- Running CmdStan in external processes isolates inference from the R process, reducing the risk that a failure during inference terminates the R session.
- Potentially lower memory use in the R session. CmdStan writes results to CSV files, and CmdStanR loads draws into R only when requested. This can avoid retaining all output in memory during fitting.
- More permissive license. RStan uses the GPL (≥ 3) license while the license for CmdStanR is BSD 3-clause, 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 ≥ 2.35 . 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: Jonah Gabry <jgabry@gmail.com>

Authors:

- Jonah Gabry <jgabry@gmail.com>
- Rok Češnovar <rok.cesnovar@fri.uni-lj.si>
- Andrew Johnson <andrew.johnson@arjohnsonau.com> ([ORCID](#))
- 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]
- Visruth Srimath Kandali [contributor]
- Aki Vehtari [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"))

# 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 the Pathfinder variational inference 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 <code>\$draws()</code> 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.

Value

A `posterior::draws_*` object. The default format depends on the fitted model class and can be changed using arguments passed through `...`

See Also

`$draws()` and `posterior::as_draws()`

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.

See Also

`$draws()`, `CmdStanMCMC`, and `posterior::as_draws()`

Examples

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

## End(Not run)
```

cmdstan_coercion	<i>Coercion methods for CmdStan objects</i>
------------------	---

Description

These are generic functions intended to primarily be used by developers of packages that interface with 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	The object to be coerced.
...	Additional arguments to pass to methods.

Value

An object of the CmdStan fitted-model class corresponding to the generic, as returned by the dispatched method.

See Also

[as_cmdstan_fit\(\)](#), [CmdStanMCMC](#), [CmdStanMLE](#), [CmdStanLaplace](#), [CmdStanVB](#), [CmdStanPathfinder](#), [CmdStanGQ](#), and [CmdStanDiagnose](#)

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

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

# 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 the Pathfinder variational inference 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)

```

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
<code>\$gradients()</code>	Return gradients from diagnostic mode.
<code>\$lp()</code>	Return the target log density (<code>lp__</code>) evaluated by Stan.
<code>\$init()</code>	Return user-specified initial values.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	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
<code>\$draws()</code>	Return the generated quantities as a <code>draws_array</code> .
<code>\$fitted_params_files()</code>	Return paths to the fitted-parameter CSV files.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$profiles()</code>	Return profiling data.
<code>\$num_chains()</code>	Return the number of chains used for generated quantities.
<code>\$code()</code>	Return Stan code as a character vector.

Summarize inferences:

Method	Description
<code>\$print()</code>	Print a summary of the generated quantities.
<code>\$summary()</code>	Run <code>posterior::summarise_draws()</code> .

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all generated quantities into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.

<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.
<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	Transform all parameter draws to the unconstrained scale.
<code>\$variable_skeleton()</code>	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](#), [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.

Objects created from CSV files using `as_cmdstan_fit()` have a reduced set of available methods. See **Reconstructed fitted model objects** in the `as_cmdstan_fit()` documentation for details.

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
<code>\$draws()</code>	Return approximate posterior draws as a <code>draws_matrix</code> .
<code>\$mode()</code>	Return the mode as a <code>CmdStanMLE</code> object.
<code>\$lp()</code>	Return the target log density (<code>lp__</code>) evaluated by Stan.
<code>\$lp_approx()</code>	Return the log density of the approximation to the posterior.
<code>\$init()</code>	Return user-specified initial values.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$profiles()</code>	Return profiling data.
<code>\$code()</code>	Return Stan code as a character vector.

Summarize inferences:

Method	Description
<code>\$print()</code>	Print a summary of the approximate posterior draws.
<code>\$summary()</code>	Run <code>posterior::summarise_draws()</code> .

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all draws and diagnostics into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.
<code>\$config_files()</code>	Return paths to CmdStan configuration JSON files.
<code>\$save_config_files()</code>	Save CmdStan configuration JSON files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.

<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	Transform all parameter draws to the unconstrained scale.
<code>\$variable_skeleton()</code>	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](#), [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.

Objects created from CSV files using `as_cmdstan_fit()` have a reduced set of available methods.

See **Reconstructed fitted model objects** in the `as_cmdstan_fit()` documentation for details.

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
<code>\$draws()</code>	Return posterior draws using formats from the posterior package.
<code>\$sampler_diagnostics()</code>	Return sampler diagnostics as a <code>draws_array</code> .
<code>\$lp()</code>	Return the target log density (lp__) evaluated by Stan.
<code>\$inv_metric()</code>	Return the inverse metric for each chain.
<code>\$init()</code>	Return user-specified initial values.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$profiles()</code>	Return profiling data.
<code>\$num_chains()</code>	Return the number of MCMC chains.
<code>\$code()</code>	Return Stan code as a character vector.

Summarize inferences and diagnostics:

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

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all draws and diagnostics into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$latent_dynamics_files()</code>	Return paths to diagnostic CSV files.
<code>\$save_latent_dynamics_files()</code>	Save diagnostic CSV files to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.
<code>\$config_files()</code>	Return paths to CmdStan configuration JSON files.
<code>\$save_config_files()</code>	Save CmdStan configuration JSON files to a specified location.
<code>\$metric_files()</code>	Return paths to metric JSON files.
<code>\$save_metric_files()</code>	Save metric JSON files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.
<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	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 `$optimize()` method of a `CmdStanModel` object. The name "MLE" is retained for historical reasons. With `jacobian = FALSE`, the point estimate is a mode of the target in the constrained parameter space. With `jacobian = TRUE`, it is a mode of the corresponding density in the unconstrained parameter space. The `jacobian` argument does not control whether prior terms are included; all contributions to the Stan program's target are included under either setting. See `$optimize()` and the CmdStan User's Guide for more details.

Objects created from CSV files using `as_cmdstan_fit()` have a reduced set of available methods. See **Reconstructed fitted model objects** in the `as_cmdstan_fit()` documentation for details.

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
<code>\$draws()</code>	Return the point estimate as a 1-row <code>draws_matrix</code> .
<code>\$mle()</code>	Return the point estimate as a numeric vector.
<code>\$lp()</code>	Return the target log density (<code>lp__</code>) evaluated by Stan.
<code>\$init()</code>	Return user-specified initial values.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$profiles()</code>	Return profiling data.
<code>\$code()</code>	Return Stan code as a character vector.

Summarize inferences:

Method	Description
<code>\$print()</code>	Print a summary of the point estimate.
<code>\$summary()</code>	Run <code>posterior::summarise_draws()</code> .

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all draws and diagnostics into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.
<code>\$config_files()</code>	Return paths to CmdStan configuration JSON files.
<code>\$save_config_files()</code>	Save CmdStan configuration JSON files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.
<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	Transform all parameter draws to the unconstrained scale.
<code>\$variable_skeleton()</code>	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.
\$has_stan_file()	Check whether the model was created with a Stan file.
\$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.

Model information:

Method	Description
\$model_name()	Return the model name.
\$include_paths()	Return the Stan include paths.
\$cmdstan_version()	Return the CmdStan version associated with the model.
\$cpp_options()	Return the C++ options associated with the model.

Compilation:

Method	Description
\$compile()	Compile Stan program.
\$exe_file()	Return or set 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.
\$cmdstan_defaults()	Get CmdStan default argument values for a method.

Diagnostics:

Method	Description
<code>\$diagnose()</code>	Run CmdStan's "diagnose" method to test gradients, return <code>CmdStanDiagnose</code> object.

Model fitting:

Method	Description
<code>\$sample()</code>	Run CmdStan's "sample" method, return <code>CmdStanMCMC</code> object.
<code>\$sample_mpi()</code>	Run CmdStan's "sample" method with <code>MPI</code> , return <code>CmdStanMCMC</code> object.
<code>\$optimize()</code>	Run CmdStan's "optimize" method, return <code>CmdStanMLE</code> object.
<code>\$laplace()</code>	Run CmdStan's "laplace" method, return <code>CmdStanLaplace</code> object.
<code>\$variational()</code>	Run CmdStan's "variational" method, return <code>CmdStanVB</code> object.
<code>\$pathfinder()</code>	Run CmdStan's "pathfinder" method, return <code>CmdStanPathfinder</code> object.
<code>\$generate_quantities()</code>	Run CmdStan's "generate quantities" method, return <code>CmdStanGQ</code> 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"))

# 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 the Pathfinder variational inference 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.

Objects created from CSV files using `as_cmdstan_fit()` have a reduced set of available methods. See **Reconstructed fitted model objects** in the `as_cmdstan_fit()` documentation for details.

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
<code>\$draws()</code>	Return approximate posterior draws as a <code>draws_matrix</code> .
<code>\$lp()</code>	Return the target log density (<code>lp__</code>) evaluated by Stan.
<code>\$lp_approx()</code>	Return the log density of the approximation to the posterior.
<code>\$init()</code>	Return user-specified initial values.
<code>\$metadata()</code>	Return a list of metadata gathered from the CmdStan CSV files.
<code>\$profiles()</code>	Return profiling data.
<code>\$code()</code>	Return Stan code as a character vector.

Summarize inferences:

Method	Description
<code>\$print()</code>	Print a summary of the approximate posterior draws.
<code>\$summary()</code>	Run <code>posterior::summarise_draws()</code> .
<code>\$cmdstan_summary()</code>	Run and print CmdStan's <code>bin/stansummary</code> .
<code>\$cmdstan_diagnose()</code>	Run and print CmdStan's <code>bin/diagnose</code> .

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all draws and diagnostics into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.
<code>\$config_files()</code>	Return paths to CmdStan configuration JSON files.
<code>\$save_config_files()</code>	Save CmdStan configuration JSON files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.
<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	Transform all parameter draws to the unconstrained scale.
<code>\$variable_skeleton()</code>	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](#), [CmdStanVB](#)

cmdstanr_example	<i>Fit models for use in examples</i>
------------------	---------------------------------------

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 <code>print_example_program(example)</code>.
method	(string) The fitting method to use. One of "sample", "optimize", "laplace", "variational", "pathfinder", or "diagnose". The default is "sample" (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 <code>utils::capture.output()</code> .
force_recompile	Passed to the <code>\$compile()</code> method.

Value

`cmdstanr_example()` returns the fitted model object from the selected method. `print_example_program()` invisibly returns NULL after printing the Stan code.

See Also

`cmdstan_model()` for fitting your own Stan programs and `print_stan_file()` for displaying Stan source files

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_print_line_numbers`: Should line numbers be included when printing a Stan program? The default is FALSE.
- `cmdstanr_no_ver_check`: Should the check for a more recent version of CmdStan be disabled? The default is FALSE. Alternatively, set the `cmdstanr_no_ver_check` environment variable to "true" (case-insensitive). Configure the option or environment variable before attaching the package.
- `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_spinner`: Should a spinner be shown while CmdStan compiles a model, checks syntax, or is installed or rebuilt? The default is TRUE. The spinner is only ever shown in interactive sessions, so setting this to FALSE is only necessary to suppress it in an interactive session.
- `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`.
- `cmdstanr_save_metric`: Should the adapted metric be saved to a separate JSON file when running MCMC? The default is `FALSE`.
- `cmdstanr_save_config`: Should a JSON file be saved containing the argument tree and extra information when running CmdStan? The default is `FALSE`.

See Also

[cmdstan_model\(\)](#), [install_cmdstan\(\)](#), and [write_stan_file\(\)](#)

CmdStanVB

CmdStanVB objects

Description

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

Objects created from CSV files using [as_cmdstan_fit\(\)](#) have a reduced set of available methods. See **Reconstructed fitted model objects** in the [as_cmdstan_fit\(\)](#) documentation for details.

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 target log density (<code>lp__</code>) evaluated by Stan.
\$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.
\$profiles()	Return profiling data.
\$code()	Return Stan code as a character vector.

Summarize inferences:

Method	Description
\$print()	Print a summary of the approximate posterior draws.
\$summary()	Run <code>posterior::summarise_draws()</code> .
\$cmdstan_summary()	Run and print CmdStan's <code>bin/stansummary</code> .
\$cmdstan_diagnose()	Run and print CmdStan's <code>bin/diagnose</code> .

Save fitted model object and temporary files:

Method	Description
<code>\$materialize()</code>	Read all draws and diagnostics into memory.
<code>\$save_object()</code>	Save fitted model object to a file.
<code>\$output_files()</code>	Return paths to output CSV files.
<code>\$save_output_files()</code>	Save output CSV files to a specified location.
<code>\$data_file()</code>	Return the path to the JSON data file.
<code>\$save_data_file()</code>	Save JSON data file to a specified location.
<code>\$latent_dynamics_files()</code>	Return paths to diagnostic CSV files.
<code>\$save_latent_dynamics_files()</code>	Save diagnostic CSV files to a specified location.
<code>\$profile_files()</code>	Return paths to profiling CSV files.
<code>\$save_profile_files()</code>	Save profiling CSV files to a specified location.
<code>\$config_files()</code>	Return paths to CmdStan configuration JSON files.
<code>\$save_config_files()</code>	Save CmdStan configuration JSON files to a specified location.

Report run times, console output, return codes:

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

Expose Stan functions and additional methods to R:

Method	Description
<code>\$expose_functions()</code>	Expose Stan functions for use in R.
<code>\$init_model_methods()</code>	Expose methods for log-probability, gradients, parameter constraining and unconstraining.
<code>\$log_prob()</code>	Calculate log-prob.
<code>\$grad_log_prob()</code>	Calculate log-prob and gradient.
<code>\$hessian()</code>	Calculate log-prob, gradient, and Hessian.
<code>\$constrain_variables()</code>	Transform a set of unconstrained parameter values to the constrained scale.
<code>\$unconstrain_variables()</code>	Transform a set of parameter values to the unconstrained scale.
<code>\$unconstrain_draws()</code>	Transform all parameter draws to the unconstrained scale.
<code>\$variable_skeleton()</code>	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](#)

draws_to_csv	<i>Write posterior draws objects to CSV files suitable for running standalone generated quantities with CmdStan.</i>
--------------	--

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 <code>posterior::draws_*</code> object.
sampler_diagnostics	Either <code>NULL</code> or a <code>posterior::draws_*</code> 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) The base name for the output CSV files. The default is "fittedParams". A timestamp, chain ID, and six-character random hexadecimal suffix are appended to the base name.

Details

`draws_to_csv()` generates a CSV suitable for running standalone generated quantities with CmdStan. The CSV file contains a single comment `# num_samples = <n>`, where `<n>` is 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 finally 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).

See Also

[\\$generate_quantities\(\)](#) for using the generated CSV files

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_diagnostics = 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 supplied by knitr. The output.var element is required and must be a single character string naming the CmdStanModel object created by the chunk.

Value

A character vector containing the formatted chunk output produced by [knitr::engine_output\(\)](#).

See Also

[register_knitr_engine\(\)](#)

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:

- [stansummary](#)
- [diagnose](#)

These methods can be used for models fit using [\\$sample\(\)](#), [\\$variational\(\)](#), or [\\$pathfinder\(\)](#). Much of the output is currently only relevant for models fit using [\\$sample\(\)](#). These methods are not available for optimization, Laplace approximation, or standalone generated quantities.

See the [\\$summary\(\)](#) for computing similar summaries in R rather than calling CmdStan's utilities.

Usage

```
cmdstan_summary(flags = NULL)  
  
cmdstan_diagnose()
```

Arguments

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

Value

Both methods invisibly return a list containing the command's exit status, standard output, and standard error.

See Also

[fit-method-summary](#)

Examples

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

## End(Not run)
```

fit-method-code	<i>Return Stan code</i>
-----------------	-------------------------

Description

Return Stan code

Usage

```
code()
```

Value

A character vector with one element per line of code.

Examples

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

## End(Not run)
```

fit-method-constrain_variables	<i>Transform a set of unconstrained parameter values to the constrained scale</i>
--------------------------------	---

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).

Value

A named list of constrained parameter values and, if requested, transformed parameters and generated quantities.

See Also

[log_prob\(\)](#), [grad_log_prob\(\)](#), [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::summarise_draws()`).

Usage

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

Arguments

<code>diagnostics</code>	(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.
<code>quiet</code>	(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)
```

Description

Extract draws or point estimates from fitted model objects using formats provided by the **posterior** package. Depending on the fitting method, these are posterior draws from MCMC, approximate posterior draws from variational inference, Laplace approximation, or Pathfinder, standalone generated quantities, or a point estimate from optimization.

The variables include the parameters, transformed parameters, and generated quantities from the Stan program as well as `lp__`, the target log density evaluated by Stan, up to an additive constant. See `$lp()` for details.

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. <ul style="list-style-type: none"> • If <code>NULL</code> (the default) then all variables are included. • If an empty string (<code>variables=""</code>) then none are included. • For non-scalar variables all elements or specific elements can be selected: <ul style="list-style-type: none"> – <code>variables = "theta"</code> selects all elements of <code>theta</code>; – <code>variables = c("theta[1]", "theta[3]")</code> selects only the 1st and 3rd elements. |
| inc_warmup | (logical) Should warmup draws be included? Defaults to <code>FALSE</code> . Ignored except when used with <code>CmdStanMCMC</code> 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. <ul style="list-style-type: none"> • For sampling and generated quantities the default is <code>"draws_array"</code>. This format keeps the chains separate. To combine the chains use any of the other formats (e.g. <code>"draws_matrix"</code>). • For point estimates from optimization and approximate draws from variational inference, Laplace approximation, and Pathfinder the default is <code>"draws_matrix"</code>. |

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** and **Laplace approximation**, a 2-D `draws_matrix` object (draw x variable). An additional variable `lp_approx__` containing the log density of the corresponding approximation is also included.
- For **Pathfinder**, a 2-D `draws_matrix` object (draw x variable). Additional variables `lp_approx__` and `path__` are also included, with `path__` identifying the path associated with each draw.
- 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.

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-fitted_params_files

Extract the fitted-parameter CSV files used for generated quantities

Description

The `$fitted_params_files()` method returns the paths to the CmdStan CSV files used as the `fitted_params` input to standalone generated quantities.

Usage

```
fitted_params_files()
```

Value

A character vector of file paths.

See Also

[CmdStanGQ, \\$generate_quantities\(\)](#)

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)
```

Arguments

`unconstrained_variables`

(numeric) A vector of unconstrained parameters.

`jacobian`

(logical) Whether to include the log-density adjustments from constraining or unconstraining variables.

Value

A numeric vector containing the gradient, with the log probability stored in the "log_prob" attribute.

See Also

[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-gradients *Extract gradients after diagnostic mode*

Description

Return the data frame containing the gradients for all parameters.

Usage

```
gradients()
```

Value

A data frame containing the gradients for all parameters.

See Also

[CmdStanDiagnose](#)

Examples

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

# retrieve the gradients
test$gradients()

## End(Not run)
```

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

Description

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

Usage

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

Arguments

`unconstrained_variables`
(numeric) A vector of unconstrained parameters.

`jacobian`
(logical) Whether to include the log-density adjustments from constraining or unconstraining variables.

Value

A named list with elements `log_prob`, `grad_log_prob`, and `hessian`.

See Also

[log_prob\(\)](#), [grad_log_prob\(\)](#), [constrain_variables\(\)](#), [unconstrain_variables\(\)](#), [unconstrain_draws\(\)](#), [variable_skeleton\(\)](#)

Examples

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

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`, `hessian`, `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.

If a model or fit object was saved with `base::saveRDS()` and later reloaded, any previously compiled model-method bindings will be rebuilt in the current R session when this method is called.

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)
```

Arguments

<code>seed</code>	(integer) The random seed to use when initializing the model.
<code>verbose</code>	(logical) Whether to show verbose logging during compilation.

Value

NULL, invisibly.

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()

## End(Not run)
```

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

Description

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

The inverse metric is defined over the unconstrained parameter space, so its entries do not necessarily correspond one-to-one with the constrained parameters declared in the parameters block (some transformations from constrained to unconstrained change dimensions). See **Examples** for a way to add names to the entries of the inverse metric when the model has parameters that change dimensions when unconstrained.

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:
stan_file <- write_stan_file("
  parameters {
    simplex[3] theta;
  }
  model {
    theta ~ dirichlet(rep_vector(1, 3));
  }
")
mod <- cmdstan_model(stan_file)

# use higher output precision so the simplex remains valid after CSV rounding
fit <- mod$sample(chains = 1, sig_figs = 10)

# even though theta has 3 elements, the inverse metric is 2x2 because the
# simplex constraint reduces the dimension. we set `matrix = FALSE` in this case
# because we estimated a diagonal matrix (we didn't set `metric = "dense_e"`
# when fitting the model), so we can just look at the diagonal elements
inv_metric <- fit$inv_metric(matrix = FALSE)

# the list has 1 element since we only ran 1 chain for simplicity
print(inv_metric)

# get names of unconstrained parameters and add them to the inverse metric
# (unconstrain_draws() requires compiling additional methods)
inv_metric_names <- posterior::variables(fit$unconstrain_draws())
inv_metric <- lapply(inv_metric, stats::setNames, nm = inv_metric_names)

# the names will be theta[1] and theta[2], but these are not the same as
# the first two elements of the constrained theta in the Stan program
inv_metric

## End(Not run)
```

fit-method-log_prob	<i>Calculate the log-probability given a provided vector of unconstrained parameters.</i>
---------------------	---

Description

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

Usage

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

Arguments

unconstrained_variables	(numeric) A vector of unconstrained parameters.
jacobian	(logical) Whether to include the log-density adjustments from constraining or unconstraining variables.

Value

A numeric scalar containing the log probability.

See Also

[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	<i>Leave-one-out cross-validation (LOO-CV)</i>
----------------	--

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 = FALSE, moment_match = FALSE, ...)
```

Arguments

variables	(string) The name of the variable 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 <code>r_eff</code> argument for <code>loo()</code> . <code>r_eff</code> measures the amount of autocorrelation in MCMC draws, and is used to compute more accurate ESS and MCSE estimates for pointwise and total ELPDs. • TRUE will call loo::relative_eff.array() to compute the <code>r_eff</code> argument to pass to loo::loo.array(). • FALSE (the default) or NULL will avoid computing <code>r_eff</code>, which can be very slow. The reported ESS and MCSE estimates may be over-optimistic if the posterior draws are far from independent. • If <code>r_eff</code> is anything else, that object will be passed as the <code>r_eff</code> argument to loo::loo.array().
moment_match	(logical) Whether to use a moment-matching correction for problematic observations. The default is FALSE. Using <code>moment_match=TRUE</code> will result in compiling the additional methods described in fit-method-init_model_methods . This allows CmdStanR to automatically supply the functions for the <code>log_lik_i</code> , <code>unconstrain_pars</code> , <code>log_prob_upars</code> , and <code>log_lik_i_upars</code> arguments to loo::loo_moment_match() .
...	Other arguments (e.g., <code>cores</code> , <code>save_psis</code> , etc.) passed to loo::loo.array() or loo::loo_moment_match.default() (if <code>moment_match = TRUE</code> is set).

Value

The object returned by [loo::loo.array\(\)](#) or [loo::loo_moment_match.default\(\)](#).

References

- Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, 27(5), 1413-1432. doi:10.1007/s11222-016-9696-4.

- Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2024). Pareto smoothed importance sampling. *Journal of Machine Learning Research*, 25(72), 1-58.
- Paananen, T., Piironen, J., Buerkner, P.-C., and Vehtari, A. (2021). Implicitly adaptive importance sampling. *Statistics and Computing*, 31, 16. doi:10.1007/s11222-020-09982-2 (for `moment_match = TRUE`).

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 `$lp()` method extracts `lp_`, the target log density evaluated by Stan, up to an additive constant. For variational inference, Laplace approximation, and Pathfinder, the log density of the corresponding approximating distribution is available via `$lp_approx()`.

See the [Increment log density and Distribution Statements](#) sections of the Stan Reference Manual for details on when normalizing constants are dropped from log probability calculations.

Usage

`lp()`

`lp_approx()`

Value

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

Details

The target includes all contributions to the log probability, which can come from the transformed parameters and model blocks, including certain user-defined functions. The exact target represented by `lp__` depends on the inference method:

- For MCMC sampling, variational inference, Pathfinder, and diagnostic mode, `lp__` is the log density on Stan's **unconstrained space** and includes the Jacobian adjustments for constrained parameters.
- For optimization and Laplace approximation, whether the Jacobian adjustments are included depends on the `jacobian` argument.

For MCMC, `lp__` can be used to diagnose sampling efficiency; for approximation methods, it can be used to evaluate the approximation.

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 CmdStan's `log_q__`: the unnormalized log density of the normal approximation on the unconstrained space. It can be used to perform the same checks as in the case of the variational method described in Yao et al. (2018).

For Pathfinder `lp_approx__` is the log density of the approximating distribution on the unconstrained space.

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.

Examples

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

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

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

fit_pathfinder <- cmdstanr_example("logistic", method = "pathfinder")
plot(fit_pathfinder$lp(), fit_pathfinder$lp_approx())

## End(Not run)
```

`fit-method-materialize`*Materialize model object*

Description

This method collects all posterior draws and diagnostics of a fitted model object into R, since the contents of the CmdStan output CSV files are only read into R lazily (i.e., as needed).

Usage

```
materialize()
```

Value

The fitted model object, invisibly.

See Also

[save_object](#)

Examples

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

## 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()
```

Value

A named list of metadata.

Examples

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

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

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

## End(Not run)
```

fit-method-mle

Extract point estimate after optimization

Description

The `$mle()` method is only available for [CmdStanMLE](#) objects. The method name is retained for historical reasons. It returns the point estimate as a numeric vector with one element per variable. The returned vector does *not* include `lp__`, the target log density evaluated by Stan, up to an additive constant. `lp__` is available via the `$lp()` method and also included in the `$draws()` method.

With `jacobian = FALSE`, the point estimate is a mode of the target in the constrained parameter space. With `jacobian = TRUE`, it is a mode of the corresponding density in the unconstrained parameter space. See `$optimize()` and the CmdStan User's Guide for more details.

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-mode

*Extract the mode used for a Laplace approximation***Description**

The `$mode()` method returns the mode used to center the Laplace approximation. This method is only available for [CmdStanLaplace](#) objects returned by `$laplace()`, not objects reconstructed using `as_cmdstan_fit()`.

Usage

```
mode()
```

Value

A [CmdStanMLE](#) object.

See Also

[CmdStanLaplace](#), `$laplace()`

fit-method-num_chains

*Extract the number of chains***Description**

The `$num_chains()` method returns the number of chains in a [CmdStanMCMC](#) object or the number of chains used for standalone generated quantities in a [CmdStanGQ](#) object.

Usage

```
num_chains()
```

Value

An integer.

See Also

[CmdStanMCMC](#), [CmdStanGQ](#)

Examples

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

## End(Not run)
```

fit-method-output	<i>Access console output</i>
-------------------	------------------------------

Description

For MCMC and standalone generated quantities, 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, Laplace approximation, variational inference, and Pathfinder, `$output()` just pretty prints the console output.

Usage

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

Arguments

`id` (integer) The chain id. Ignored except for MCMC and standalone generated quantities.

Value

For MCMC and standalone generated quantities with `id=NULL`, a list of character vectors containing the console output for each chain. In all other cases, NULL invisibly.

Examples

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

fit_optim <- cmdstanr_example("logistic", method = "optimize")
fit_optim$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.

Profiling requires adding profiling statements to the Stan program. See **Examples** for a demonstration.

Usage

```
profiles()
```

Value

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

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).

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

By default objects are saved using `base::saveRDS()`. If you have a big object to save, we recommend setting `format = "qs2"`, which is faster and more memory efficient. Internally this will use `qs2::qs_save()` to save the object and you can read it back into R later with `qs2::qs_read()`.

Usage

```
save_object(file, format = c("rds", "qs2"), ...)
```

Arguments

<code>file</code>	(string) Path where the file should be saved.
<code>format</code>	(string) Serialization format for the object. The default is "rds". The "qs2" format uses <code>qs2::qs_save()</code> and requires the qs2 package.
<code>...</code>	Other arguments to pass to <code>base::saveRDS()</code> (for <code>format = "rds"</code>) or <code>qs2::qs_save()</code> (for <code>format = "qs2"</code>).

Value

The fitted model object, invisibly.

See Also

[materialize](#)

Examples

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

# using default format = "rds"
temp_rds_file <- tempfile(fileext = ".rds")
fit$save_object(file = temp_rds_file)
rm(fit)

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

```
# using format = "qs2"
temp_qs2_file <- tempfile(fileext = ".qs2")
fit$save_object(file = temp_qs2_file, format = "qs2")
rm(fit)

fit <- qs2::qs_read(temp_qs2_file)
fit$summary()

## End(Not run)
```

fit-method-save_output_files

Save output and data files

Description

Fitted model objects returned directly by a [CmdStanModel](#) method have methods for saving (moving to a specified location) files created by CmdStanR, including 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.

Objects created by [as_cmdstan_fit\(\)](#) support `$output_files()` but not the other methods documented on this page because the original CmdStan run is unavailable. See **Reconstructed fitted model objects** in the [as_cmdstan_fit\(\)](#) documentation for details.

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

<code>dir</code>	(string) Path to directory where the files should be saved.
<code>basename</code>	(string) Base filename to use. If NULL (the default), the model name is used. See Details .
<code>timestamp</code>	(logical) Should a timestamp be added to the file name(s)? Defaults to TRUE. See Details .
<code>random</code>	(logical) Should a six-character random hexadecimal suffix be added to the file name(s)? Defaults to TRUE. See Details .
<code>include_failed</code>	(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. If any file cannot be copied then the method errors and no original files are removed. The methods 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.csv`, where

- `basename` is the user's provided `basename` argument or, if NULL, the model name;
- `timestamp` is of the form `format(Sys.time(), "%Y%m%d%H%M")`;
- `id` is the MCMC chain id, zero-padded to at least two digits (or 01 for non-MCMC);
- `random` contains six random hexadecimal characters.

`$save_latent_dynamics_files()` uses the pattern `basename-diagnostic-timestamp-id-random.csv`. The `$latent_dynamics_files()` and `$save_latent_dynamics_files()` methods apply only to [CmdStanMCMC](#) and [CmdStanVB](#) objects created with `save_latent_dynamics = TRUE`.

`$save_profile_files()` uses the pattern `basename-profile-timestamp-id-random.csv`.

`$save_metric_files()` uses the pattern `basename-metric-timestamp-id-random.json`. Make sure to set `save_metric = TRUE` when fitting the model.

`$save_config_files()` uses the pattern `basename-config-timestamp-id-random.json`. Make sure to set `save_cmdstan_config = TRUE` when fitting the model.

`$save_data_file()` uses the pattern `basename-timestamp-random.ext`, where `.ext` is the original data file extension. No `id` is included because even with multiple MCMC chains the data file is the same.

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	<i>Compute a summary table of estimates and diagnostics</i>
--------------------	---

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. The `$print()` method accepts the same variables and `...` arguments as `$summary()`. It also has a `digits` argument for the number of digits to display after the decimal point (default 2) and a `max_rows` argument for the maximum number of rows to print (default `getOption("cmdstanr_max_rows", 10)`). See **Examples**.

Usage

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

Arguments

<code>variables</code>	(character vector) The variables to include.
<code>...</code>	Optional arguments to pass to <code>posterior::summarise_draws()</code> .

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.

References

- Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., and Buerkner, P.-C. (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC (with discussion). *Bayesian Analysis*, 16(2), 667-718. doi:10.1214/20-BA1221.
- Vehtari, A. (2021). Comparison of MCMC effective sample size estimators. https://avehtari.github.io/rhat_ess/ess_comp (for ESS diagnostics such as `ess_bulk` and `ess_tail`).

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 and standalone generated quantities additional information is provided about the run times of individual chains or processes. For MCMC, timing information is also provided for the warmup and sampling phases. For Laplace approximation the reported time includes only 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 and standalone generated quantities this may differ from the sum of the individual run times if parallelization was used.
- `chains`: (data frame) For MCMC and standalone generated quantities, timing information for the individual chains. For MCMC the data frame has columns `"chain_id"`, `"warmup"`, `"sampling"`, and `"total"`. For standalone generated quantities, each row corresponds to one fitted-parameter CSV file and one CmdStan process, and the data frame has columns `"chain_id"` and `"total"`. Variational or optimization input therefore produces one row. With CmdStan versions before 2.39, standalone generated quantities process times are reported as zero.

Examples

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

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

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

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

<code>files</code>	(character vector) The paths to the CmdStan CSV files. These can be files generated by running CmdStanR or running CmdStan directly.
<code>draws</code>	A <code>posterior::draws_*</code> object.
<code>format</code>	(string) The format of the returned draws. Must be a valid format from the posterior package. Defaults to <code>getOption("cmdstanr_draws_format", "draws_array")</code> .
<code>inc_warmup</code>	(logical) Should warmup draws be included when using draws from the fit object or CSV files? Defaults to FALSE. If <code>draws</code> is supplied, <code>inc_warmup</code> is ignored with a message.

Value

A `posterior::draws_*` object in the format specified by `format`.

See Also

[log_prob\(\)](#), [grad_log_prob\(\)](#), [constrain_variables\(\)](#), [unconstrain_variables\(\)](#), [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.

Value

A numeric vector of unconstrained parameter values.

See Also

[log_prob\(\)](#), [grad_log_prob\(\)](#), [constrain_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).

Value

A named list suitable for use as the skeleton argument to `utils::relist()`.

See Also

`log_prob()`, `grad_log_prob()`, `constrain_variables()`, `unconstrain_variables()`, `unconstrain_draws()`, `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. If the CmdStan installation currently in use has a non-empty `make/local` file, the flags in it can be copied to the new installation before it is built, so that no rebuild is needed afterwards. In an interactive session `install_cmdstan()` shows the file and asks. See the `copy_make_local` argument to decide without being asked.

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.

CmdStan versions older than 2.35.0 are no longer supported. If you need to work with an older CmdStan version we recommend installing an older CmdStanR release from GitHub.

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,
  copy_make_local = NULL
)

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

<code>dir</code>	(string) The path to the directory in which to install CmdStan. The default is to install it in a directory called <code>.cmdstan</code> within the user's home directory. On Windows the home directory is determined from <code>USERPROFILE</code> , falling back to <code>HOMEDRIVE</code> and <code>HOMEPATH</code> . On other platforms it is determined from <code>HOME</code> .
<code>cores</code>	(integer) The number of CPU cores to use to parallelize building CmdStan and speed up installation. If <code>cores</code> is not specified then the default is to look for the option <code>"mc.cores"</code> , which can be set for an entire R session by <code>options(mc.cores=value)</code> . If the <code>"mc.cores"</code> option has not been set then the default is 2.
<code>quiet</code>	(logical) For <code>install_cmdstan()</code> , should the verbose output from the system processes be suppressed when building the CmdStan binaries? The default is <code>FALSE</code> . For <code>check_cmdstan_toolchain()</code> , should the function suppress printing informational messages? The default is <code>FALSE</code> . If <code>TRUE</code> only errors will be printed.
<code>overwrite</code>	(logical) Should CmdStan still be downloaded and installed even if an installation of the same version is found in <code>dir</code> ? The default is <code>FALSE</code> , in which case an informative error is thrown instead of overwriting the user's installation.
<code>timeout</code>	(positive real) Timeout (in seconds) for the build stage of the installation. The default is 1200 seconds for <code>install_cmdstan()</code> and 600 seconds for <code>rebuild_cmdstan()</code> .
<code>version</code>	(string) The CmdStan release version to install. The default is <code>NULL</code> , which downloads the latest stable release from https://github.com/stan-dev/cmdstan/releases .
<code>release_url</code>	(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., <code>release_url="https://github.com/stan-dev/cmdstan/releases/download/v2.35.0/cmdstan-2.35.0.tar.gz"</code> . If both <code>version</code> and <code>release_url</code> are specified then <code>version</code> will be used.
<code>release_file</code>	(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: <code>release_file="./cmdstan-2.35.0.tar.gz"</code> . If <code>release_file</code> is specified then both <code>release_url</code> and <code>version</code> will be ignored.
<code>cpp_options</code>	(list) Any makefile flags/variables to be written to the <code>make/local</code> file. For example, <code>list("CXX" = "clang++")</code> will force the use of clang for compilation.
<code>check_toolchain</code>	(logical) Should <code>install_cmdstan()</code> attempt to check that the required toolchain is installed and properly configured? The default is <code>TRUE</code> .
<code>ws1</code>	(logical) Should CmdStan be installed and run through the Windows Subsystem for Linux (WSL). The default is <code>FALSE</code> .

copy_make_local	(logical) Should the make/local file of the CmdStan installation currently in use be copied to the new installation? The copy happens before CmdStan is built, so the flags are already in effect for that build. The default is NULL, which shows the previous make/local and asks in an interactive session, and copies nothing otherwise. The question comes before the download, so that the rest of the installation runs unattended. Use TRUE or FALSE to decide without being asked. Flags given in <code>cpp_options</code> are written after the copied ones and therefore take precedence.
append	(logical) For <code>cmdstan_make_local()</code> , 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. When appending, a flag that is already in make/local is not written again.
fix	Deprecated and will be removed in a future release. This argument is ignored and retained only for compatibility.

Value

If a build fails or times out, `install_cmdstan()` issues a warning and invisibly returns the process result.

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.

See Also

[set_cmdstan_path\(\)](#), [cmdstan_default_install_path\(\)](#), and [cmdstan_default_path\(\)](#)

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 is 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 <i>Pedantic mode</i> chapter in the Stan User's Guide.
include_paths	(character vector) Paths to directories where Stan should look for files specified in <code>#include</code> 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 <code>\$compile()</code> 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-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [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-cmdstan_defaults

Get CmdStan default argument values

Description

The `$cmdstan_defaults()` method of a [CmdStanModel](#) object queries the compiled model binary for the default argument values used by a given inference method. The returned list uses CmdStanR-style argument names (e.g., `iter_sampling` instead of CmdStan's `num_samples`).

The model must be compiled before calling this method.

Usage

```
cmdstan_defaults(
  method = c("sample", "optimize", "variational", "pathfinder", "laplace")
)
```

Arguments

method	(string) The inference method for which to retrieve default argument values. One of "sample", "optimize", "variational", "pathfinder", or "laplace". The default is "sample".
--------	---

Value

A named list of default argument values for the specified method, with CmdStanR-style argument names.

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-model-info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-sample](#), [model-method-sample_mpi](#), [model-method-variables](#), [model-method-variational](#)

Examples

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

mod$cmdstan_defaults("sample")
mod$cmdstan_defaults("optimize")

## 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 path to the executable is available via `$exe_file()`. If compilation generated C++ code instead of reusing an up-to-date executable, its path is also available via `$hpp_file()`. Use `force_recompile=TRUE` to force generation of the C++ code. By default, the executable is created in the same directory as the Stan program and the generated C++ code is written to 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
)

```

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. 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 <i>Pedantic mode section</i> in the Stan User's Guide. Note: to do a pedantic check for a model without compiling it or for a model that is already compiled the <code>\$check_syntax()</code> 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. Relative paths are resolved against the working directory when the model object is created (or when <code>\$compile()</code> is called) and stored as absolute paths, so subsequent changes to the working directory do not affect them. If <code>\$compile()</code> is called again without <code>include_paths</code> , the most recently supplied paths are reused, and changing them forces recompilation. Edits to the included files themselves do not; see <code>force_recompile</code> .
user_header	(string) The path to a C++ file (with a .hpp extension) to compile with the Stan model. If <code>\$compile()</code> is called again without <code>user_header</code> , the most recently supplied header is reused, and changing it forces recompilation. Pass <code>user_header = NULL</code> to compile without one. A header can also be supplied via <code>cpp_options</code> as <code>USER_HEADER</code> or <code>user_header</code> ; the <code>user_header</code> argument takes precedence over both. See <code>force_recompile</code> for the case of a header supplied for a program whose executable is already up to date.
cpp_options	(list) Any makefile options to be used when compiling the model (<code>stan_threads</code> , <code>stan_mpi</code> , <code>stan_opensl</code> , etc.). Anything you would otherwise write in the <code>make/local</code> file. For an example of using threading see the Stan case study <i>Reduce Sum: A Minimal Example</i> . Note: For historical reasons, CmdStan treats some options as enabled whenever their Make variable is non-empty. In particular, setting <code>stan_threads</code> to FALSE passes <code>STAN_THREADS=FALSE</code> to Make,

	which still enables threading! To leave threading disabled, either omit <code>stan_threads</code> entirely, which leaves any setting in <code>make/local</code> in place, or set it to <code>NULL</code> , which passes an empty <code>STAN_THREADS=</code> and so overrides <code>make/local</code> too.
<code>stanc_options</code>	(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 User's Guide for more details on available options.
<code>force_recompile</code>	(logical) Should the model be recompiled even if it has not been modified since it was last compiled? The default is <code>FALSE</code> . Can also be set via a global <code>cmdstanr_force_recompile</code> option. Only the Stan program itself and the user header (if any) are checked for modification. Files pulled in by <code>#include</code> directives are not, at any depth, so editing an included file does not on its own trigger recompilation. Use <code>force_recompile = TRUE</code> after changing one. Similarly, when a model object is created for a Stan program whose executable already exists and is up to date, CmdStanR cannot tell which <code>user_header</code> or <code>include_paths</code> that executable was built with, so supplying different ones does not force a rebuild.
<code>compile_model_methods</code>	(logical) Compile additional model methods (<code>log_prob()</code> , <code>grad_log_prob()</code> , <code>hessian()</code> , <code>constrain_variables()</code> , <code>unconstrain_variables()</code> , <code>unconstrain_draws()</code> , and <code>variable_skeleton()</code>). Note: the compiled model-method bindings are not preserved in a usable form when saving a model object. If you plan to save and reload the model object before model fitting, we recommend instead waiting to compile the model methods until after fitting via <code>fit\$init_model_methods()</code> .
<code>compile_standalone</code>	(logical) Should functions in the Stan model be compiled for use in R? If <code>TRUE</code> the functions will be available via the <code>functions</code> field in the compiled model object. This can also be done after compilation using the <code>\$expose_functions()</code> method.
<code>dry_run</code>	(logical) If <code>TRUE</code> , the code will do all checks before compilation, but skip the actual C++ compilation. Used to speedup tests.

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.

The `$exe_file()` method returns the executable path. If compilation generated C++ code, the `$hpp_file()` and `$save_hpp_file()` methods can also be used. See their linked documentation for return values.

See Also

The `$check_syntax()` method to check Stan syntax or enable pedantic mode 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-cmdstan_defaults](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate_quantiles](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-sample](#), [model-method-sample_mpi](#), [model-method-variables](#), [model-method-variational](#)

Examples

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

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

# turn on threading support for using functions that support within-chain
# parallelization or running multiple pathfinder paths in parallel
# (here we compile a copy of the model in a temporary directory so that the
# executable compiled without threading above is not overwritten)
stan_file_threads <- file.path(tempdir(), "bernoulli.stan")
file.copy(stan_file, stan_file_threads)
mod_threads <- cmdstan_model(stan_file_threads, compile = FALSE)
mod_threads$compile(cpp_options = list(stan_threads = TRUE))
mod_threads$cxx_options()

# turn on pedantic mode
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, compile = FALSE)
mod$compile(pedantic = TRUE)
# same as mod <- cmdstan_model(file_pedantic, pedantic = TRUE)

## End(Not run)
```

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.

Unlike other CmdStan methods, `$diagnose()` does not expose `show_messages` or `show_exceptions` arguments. CmdStan's standard output is not printed during execution, while standard error is always displayed. The captured console output can be inspected with the returned object's `$output()` method.

Usage

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

Arguments

- | | |
|------|---|
| data | <p>(multiple options) The data to use for the variables specified in the data block of the Stan program. One of the following:</p> <ul style="list-style-type: none"> • 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 <code>write_stan_json()</code>. See <code>write_stan_json()</code> 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 | <p>(non-negative 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 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 <i>different</i> data for each chain.</p> |
| init | <p>(multiple options) The initialization method to use for the variables declared in the parameters block of the Stan program. One of the following:</p> <ul style="list-style-type: none"> • A real number $x > 0$. This initializes <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. |

- The number 0. This initializes *all* parameters to 0 on the *unconstrained* parameter space.
- A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. 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, and for Pathfinder it should contain a sublist for each path. 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 and Pathfinder, the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For [CmdStanVB](#), [CmdStanLaplace](#), and [CmdStanPathfinder](#) fit objects the draws must additionally be *distinct*. A [CmdStanMLE](#) fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a [CmdStanPathfinder](#) fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 <code>options(cmdstanr_output_dir)</code>. The behavior of <code>output_dir</code> 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 <code>\$save_*</code> methods of the fitted model object (e.g., <code>\$save_output_files()</code>). 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 <code>output_dir</code> with names corresponding to the defaults used by <code>\$save_output_files()</code>.
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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.
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 [\\$gradients\(\)](#) method for accessing the gradients and the [\\$output\(\)](#) method for displaying the captured console output.

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-cmdstan_defaults](#), [model-method-compile](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate_quantities](#), [model-method-laplace](#), [model-method-model_info](#), [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 all fitted model objects. 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

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

Value

NULL, invisibly.

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-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [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	<i>Run stanc's auto-formatter on the model code.</i>
---------------------	--

Description

The `$format()` method of a [CmdStanModel](#) object runs stanc's auto-formatter on the model code. It 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

<code>overwrite_file</code>	(logical) Should the formatted code be written back to the input model file? The default is FALSE.
<code>canonicalize</code>	(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 <code>stanc</code> . See the User's guide section for available canonicalization options.
<code>backup</code>	(logical) If TRUE, create a backup before writing to the file. The backup filename is the Stan filename followed by <code>.bak-YYYYMMDDHHMMSS</code> , where the final digits encode the timestamp. 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 <code>overwrite_file = FALSE</code> .
<code>max_line_length</code>	(integer) The maximum length of a line when formatting. The default is NULL, which defers to the default line length of <code>stanc</code> .
<code>quiet</code>	(logical) Should informational messages be suppressed? The default is FALSE.

Value

The `$format()` method returns TRUE (invisibly) if formatting succeeds.

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-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-generate_quantities](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-sample](#), [model-method-sample_mpi](#), [model-method-variables](#), [model-method-variational](#)

Examples

```
## Not run:

# 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);
}
")

# 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()

## 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.

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. These values are also available via the `$cmdstan_defaults` method.

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,
  show_messages = TRUE,
  show_exceptions = TRUE
)

```

Arguments

- `fitted_params` (multiple options) The parameter draws to use. One of the following:
- A `CmdStanMCMC`, `CmdStanMLE`, `CmdStanLaplace`, `CmdStanVB`, or `CmdStanPathfinder` fitted model object.
 - A `posterior::draws_array` or `posterior::draws_matrix` object returned by `CmdStanR`'s `$draws()` method.
 - A character vector of paths to `CmdStan` CSV output files.
- For a `CmdStanMLE` object, optimization supplies one point estimate, so generated quantities that use RNG functions produce only one simulation. For `CmdStanLaplace`, `CmdStanVB`, and `CmdStanPathfinder` objects, generated quantities are evaluated once per approximate draw.
- NOTE: `CmdStan` CSV paths are used directly. A `CmdStanMCMC` object also reuses its original output files when they are available. If any of those files are unavailable, `CmdStanR` writes the in-memory draws to temporary CSV files. Other fitted model objects and posterior draws objects are converted to temporary CSV files on each call. For repeated calls that require this conversion, we recommend using `draws_to_csv()` once and passing the resulting paths to `$generate_quantities()`.
- `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` (non-negative 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 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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.

`sig_figs`

(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If NULL (the default), the default from the installed CmdStan version is used. Use `$cmdstan_defaults()` to check that default. 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_opengl = TRUE)` for this argument to have an effect.

`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.

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-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-laplace](#), [model-method-model-info](#), [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_gg$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. When the mode was found with the Jacobian adjustment, the draws provide an estimate of the mean and standard deviation of the posterior distribution. See the `jacobian` argument below for how this setting relates to the value used when running optimization, and 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. These values are also available via the `$cmdstan_defaults` method.

Usage

```
laplace(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  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 = getOption("cmdstanr_save_config", FALSE)
)
```

Arguments

<code>data</code>	(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** (non-negative 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 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 on the *unconstrained* parameter space.
 - A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. 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, and for Pathfinder it should contain a sublist for each path. 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 and Pathfinder, the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the

inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For `CmdStanVB`, `CmdStanLaplace`, and `CmdStanPathfinder` fit objects the draws must additionally be *distinct*. A `CmdStanMLE` fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a `CmdStanPathfinder` fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.

`sig_figs` (positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If NULL (the default), the default from the installed CmdStan version is used. Use `$cmdstan_defaults()` to check that default. 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 <code>reduce_sum()</code> or <code>map_rect()</code>).
openc1_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 <code>cpp_options = list(stan_openc1 = TRUE)</code> 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 <code>\$optimize()</code>. • The path to a CmdStan CSV file from running optimization. • NULL, in which case <code>\$optimize()</code> will be run with <code>jacobian=jacobian</code> (see the <code>jacobian</code> argument below). In all cases the total time reported by <code>\$time()</code> will be the time of the Laplace sampling step only and does not include the time taken to run the <code>\$optimize()</code> method.
opt_args	(named list) A named list of optional arguments to pass to <code>\$optimize()</code> if <code>mode=NULL</code> .
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 <code>mode</code> is not NULL then the value of <code>jacobian</code> must match the value used when optimization was originally run so the mode and the Laplace approximation use the same target density. If <code>mode</code> is NULL then the value of <code>jacobian</code> 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 <code>\$output()</code> 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 <code>\$output()</code> method of the resulting fit object can be used to display the silenced messages.
save_cmdstan_config	(logical) When TRUE, 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). The default is FALSE but can be set to TRUE for an entire R session by <code>options(cmdstanr_save_config = TRUE)</code> .

Value

A [CmdStanLaplace](#) object.

References

- Stan Development Team. Stan Reference Manual (Algorithms section, Laplace approximation): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

Other CmdStanModel methods: [model-method-check_syntax](#), [model-method-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-model-info](#), [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-model-info

Access information from a CmdStanModel object

Description

These methods access information stored in a [CmdStanModel](#) object, print its Stan program, and manage paths to its executable and generated C++ file.

```

stan_file()
has_stan_file()
code()
print(line_numbers = getOption("cmdstanr_print_line_numbers", FALSE))
model_name()
exe_file(path = NULL)
include_paths()
cmdstan_version()
cpp_options()
hpp_file()
save_hpp_file(dir = NULL)

```

Arguments

line_numbers	(logical) Should line numbers be printed? The default is <code>getOption("cmdstanr_print_line_numbers", FALSE)</code> .
path	(string) The path to a model executable. If <code>NULL</code> (the default), <code>\$exe_file()</code> returns the current path. Otherwise, the stored path is updated before being returned.
dir	(string) The directory in which to save the <code>.hpp</code> file. The default is the directory containing the Stan program.

Value

- `$stan_file()` returns a path as a string, or `character(0)` if the model was created without a Stan file.
- `$has_stan_file()` returns `TRUE` if the model was created with a Stan file and `FALSE` otherwise.
- `$code()` returns a character vector with one element per line of Stan code, or `NULL` if the model was created without a Stan file.
- `$print()` returns the `CmdStanModel` object invisibly.
- `$model_name()` returns the model name as a string.
- `$exe_file()` returns a path as a string, or `character(0)` if no executable path is set.
- `$include_paths()` returns a character vector of absolute paths or `NULL`.
- `$cmdstan_version()` returns a CmdStan version as a string.
- `$cpp_options()` returns a named list of C++ options.
- `$hpp_file()` returns the path to the `.hpp` file as a string when C++ code was generated while compiling this model object. It errors if no `.hpp` path is available, such as when an up-to-date executable was reused.
- `$save_hpp_file()` requires an available `.hpp` file. It moves the file to `dir`, updates the stored path, and returns the new path invisibly.

See Also

`$compile()` and `cmdstan_model()`

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-cmdstan_defaults](#), [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](#)

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

Description

The `$optimize()` method of a [CmdStanModel](#) object runs Stan's optimizer. Without the Jacobian adjustment (the default), optimization finds a mode of the target in the original constrained parameter space (if the mode exists). With the adjustment, it finds a mode of the corresponding density in the unconstrained parameter space.

The `jacobian` argument does not determine whether prior terms are included. Every contribution to the Stan program's target, including prior terms, is included under either setting. The Jacobian adjustment is particularly useful when making a distributional approximation in the unconstrained space (see [Laplace sampling](#)). If the model has only unconstrained parameters, including the Jacobian has no effect. 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. These values are also available via the `$cmdstan_defaults` method.

Usage

```
optimize(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  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 = getOption("cmdstanr_save_config", FALSE)
)

```

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	(non-negative 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 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 <i>different</i> 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 <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. • The number 0. This initializes <i>all</i> parameters to 0 on the <i>unconstrained</i> parameter space. • A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. 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, and for Pathfinder it should contain a sublist for each path. 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 and Pathfinder, the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For `CmdStanVB`, `CmdStanLaplace`, and `CmdStanPathfinder` fit objects the draws must additionally be *distinct*. A `CmdStanMLE` fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a `CmdStanPathfinder` fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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()`.

<code>output_basename</code>	(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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.
<code>sig_figs</code>	(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If NULL (the default), the default from the installed CmdStan version is used. Use <code>\$cmdstan_defaults()</code> to check that default. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
<code>threads</code>	(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 <code>reduce_sum()</code> or <code>map_rect()</code>).
<code>opencl_ids</code>	(integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with <code>cpp_options = list(stan_opencl = TRUE)</code> for this argument to have an effect.
<code>algorithm</code>	(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 <code>cmdstanr_example(method="optimize")\$metadata()</code> .
<code>jacobian</code>	(logical) Whether or not to use the Jacobian adjustment for constrained variables. For historical reasons, the default is FALSE. FALSE finds a mode of the target in the constrained parameter space and TRUE finds a mode in the unconstrained space. This argument does not control whether prior terms are included. See the Description section and the CmdStan User's Guide for more details. For use later with <code>\$laplace()</code> , the <code>jacobian</code> argument should typically be set to TRUE.
<code>init_alpha</code>	(positive real) The initial step size parameter.
<code>iter</code>	(positive integer) The maximum number of iterations.
<code>tol_obj</code>	(positive real) Convergence tolerance on changes in objective function value.
<code>tol_rel_obj</code>	(positive real) Convergence tolerance on relative changes in objective function value.
<code>tol_grad</code>	(positive real) Convergence tolerance on the norm of the gradient.
<code>tol_rel_grad</code>	(positive real) Convergence tolerance on the relative norm of the gradient.
<code>tol_param</code>	(positive real) Convergence tolerance on changes in parameter value.
<code>history_size</code>	(positive integer) The size of the history used when approximating the Hessian. Only available for L-BFGS.
<code>show_messages</code>	(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 <code>\$output()</code> 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, 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). The default is FALSE but can be set to TRUE for an entire R session by `options(cmdstanr_save_config = TRUE)`.

Value

A `CmdStanMLE` object.

References

- Stan Development Team. Stan Reference Manual (Algorithms section, optimization): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

Other `CmdStanModel` methods: [model-method-check_syntax](#), [model-method-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-pathfind](#), [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"))

# 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 the Pathfinder variational inference 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. See the [CmdStan User's Guide](#) for more details on the default arguments. These values are also available via the `$cmdstan_defaults` method.

Usage

```
pathfinder(
  data = NULL,
  seed = NULL,
  refresh = NULL,
  init = NULL,
  output_dir = getOption("cmdstanr_output_dir"),
  output_basename = NULL,
  sig_figs = NULL,
  threads = 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 = getOption("cmdstanr_save_config", FALSE)
  )

```

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 <code>write_stan_json()</code>. See <code>write_stan_json()</code> 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	(non-negative 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 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 <i>different</i> 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 <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. • The number 0. This initializes <i>all</i> parameters to 0 on the <i>unconstrained</i> parameter space. • A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. See <code>write_stan_json()</code> 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, and for Pathfinder it should contain a sublist for each path. 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 <code>chain_id</code>. For MCMC and Pathfinder, the function is called once for each chain or path. If the function has the <code>chain_id</code> argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For `CmdStanVB`, `CmdStanLaplace`, and `CmdStanPathfinder` fit objects the draws must additionally be *distinct*. A `CmdStanMLE` fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a `CmdStanPathfinder` fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.

<code>sig_figs</code>	(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If NULL (the default), the default from the installed CmdStan version is used. Use <code>\$cmdstan_defaults()</code> to check that default. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
<code>threads</code>	(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 <code>reduce_sum</code>).
<code>opencl_ids</code>	(integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with <code>cpp_options = list(stan_opencl = TRUE)</code> for this argument to have an effect.
<code>num_threads</code>	Deprecated and will be removed in a future release. Use <code>threads</code> instead.
<code>init_alpha</code>	(positive real) The initial step size parameter.
<code>tol_obj</code>	(positive real) Convergence tolerance on changes in objective function value.
<code>tol_rel_obj</code>	(positive real) Convergence tolerance on relative changes in objective function value.
<code>tol_grad</code>	(positive real) Convergence tolerance on the norm of the gradient.
<code>tol_rel_grad</code>	(positive real) Convergence tolerance on the relative norm of the gradient.
<code>tol_param</code>	(positive real) Convergence tolerance on changes in parameter value.
<code>history_size</code>	(positive integer) The size of the history used when approximating the Hessian.
<code>single_path_draws</code>	(positive integer) Number of draws a single pathfinder should return. The number of draws PSIS sampling samples from will be equal to <code>single_path_draws * num_paths</code> .
<code>draws</code>	(positive integer) Number of draws to return after performing Pareto smoothed importance sampling (PSIS). This should be smaller than <code>single_path_draws * num_paths</code> .
<code>num_paths</code>	(positive integer) Number of single pathfinders to run. The default is 4. The paths are run sequentially unless the model was compiled with <code>cpp_options = list(stan_threads = TRUE)</code> and <code>threads</code> is set, so running multiple paths in parallel requires both.
<code>max_lbfgs_iters</code>	(positive integer) The maximum number of iterations for LBFGS.
<code>num_elbo_draws</code>	(positive integer) Number of draws to make when calculating the ELBO of the approximation at each iteration of LBFGS.
<code>save_single_paths</code>	(logical) Whether to save the output from each single-Pathfinder run. For a multi-path run, CmdStan writes one Stan CSV file containing draws and one JSON file containing the L-BFGS and ELBO iterations for each path. For a single-path run, the main output CSV contains the draws and CmdStan writes an additional JSON file. The auxiliary files are written to <code>output_dir</code> , or to a temporary directory if <code>output_dir = NULL</code> . They are not included in the paths returned by the fitted object's <code>\$output_files()</code> method. See the CmdStan User's Guide for details.

- `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, 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). The default is FALSE but can be set to TRUE for an entire R session by `options(cmdstanr_save_config = TRUE)`.

Value

A `CmdStanPathfinder` object.

References

- Zhang, L., Carpenter, B., Gelman, A., and Vehtari, A. (2022). Pathfinder: parallel quasi-Newton variational inference. *Journal of Machine Learning Research*, 23(306), 1-49.
- Stan Development Team. Stan Reference Manual (Algorithms section, Pathfinder): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

Other `CmdStanModel` methods: `model-method-check_syntax`, `model-method-cmdstan_defaults`, `model-method-compile`, `model-method-diagnose`, `model-method-expose_functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-model-info`, `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"))

# 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 the Pathfinder variational inference 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.

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

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. These values are also available via the `$cmdstan_defaults` method.

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 = getOption("cmdstanr_save_metric", FALSE),
save_cmdstan_config = getOption("cmdstanr_save_config", FALSE)
)

```

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	(non-negative 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 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 <i>different</i> 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 <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. • The number 0. This initializes <i>all</i> parameters to 0 on the <i>unconstrained</i> parameter space.

- A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. 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, and for Pathfinder it should contain a sublist for each path. 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 and Pathfinder, the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For [CmdStanVB](#), [CmdStanLaplace](#), and [CmdStanPathfinder](#) fit objects the draws must additionally be *distinct*. A [CmdStanMLE](#) fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a [CmdStanPathfinder](#) fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 sampler or varia-

	tional algorithm be written to diagnostic CSV files? This argument replaces CmdStan's <code>diagnostic_file</code> argument. The content is controlled by the user's CmdStan installation. The default is <code>FALSE</code>, which is appropriate for almost every use case. To save temporary diagnostic files permanently, use <code>\$save_latent_dynamics_files()</code>.
<code>output_dir</code>	(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 <code>NULL</code> (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 <code>options(cmdstanr_output_dir)</code>. The behavior of <code>output_dir</code> is as follows: • If <code>NULL</code> (the default), then the CSV files are written to a temporary directory and only saved permanently if the user calls one of the <code>\$save_*</code> methods of the fitted model object (e.g., <code>\$save_output_files()</code>). 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 <code>output_dir</code> with names corresponding to the defaults used by <code>\$save_output_files()</code>.
<code>output_basename</code>	(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If <code>NULL</code> (the default), the basename of the output CSV files is composed of the model name, timestamp, and a six-character random hexadecimal suffix.
<code>sig_figs</code>	(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If <code>NULL</code> (the default), the default from the installed CmdStan version is used. Use <code>\$cmdstan_defaults()</code> to check that default. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
<code>chains</code>	(positive integer) The number of Markov chains to run. The default is 4.
<code>parallel_chains</code>	(positive integer) The <i>maximum</i> number of MCMC chains to run in parallel. If <code>parallel_chains</code> is not specified then the default is to look for the option <code>"mc.cores"</code>, which can be set for an entire R session by <code>options(mc.cores=value)</code>. If the <code>"mc.cores"</code> option has not been set then the default is 1.
<code>chain_ids</code>	(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).
<code>threads_per_chain</code>	(positive integer) If the model was compiled with threading support, the number of threads to use in parallelized sections <i>within</i> an MCMC chain (e.g., when using the Stan functions <code>reduce_sum()</code> or <code>map_rect()</code>). This is in contrast with <code>parallel_chains</code>, which specifies the number of chains to run in parallel. The actual number of CPU cores used is <code>parallel_chains*threads_per_chain</code>. For an example of using threading see the Stan case study Reduce Sum: A Minimal Example.
<code>openc1_ids</code>	(integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with <code>cpp_options = list(stan_openc1 = TRUE)</code> for this argument to have an effect.

<code>iter_warmup</code>	(positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as <code>num_warmup</code> .
<code>iter_sampling</code>	(positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as <code>num_samples</code> .
<code>save_warmup</code>	(logical) Should warmup iterations be saved? The default is FALSE.
<code>thin</code>	(positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.
<code>max_treedepth</code>	(positive integer) The maximum allowed tree depth for the NUTS engine. See the <i>Tree Depth</i> section of the CmdStan User's Guide for more details.
<code>adapt_engaged</code>	(logical) Do warmup adaptation? The default is TRUE. If a precomputed inverse metric is specified via the <code>inv_metric</code> argument (or <code>metric_file</code>) then, if <code>adapt_engaged=TRUE</code> , 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 <code>adapt_engaged=FALSE</code> .
<code>adapt_delta</code>	(real in $(0, 1)$) The adaptation target acceptance statistic.
<code>step_size</code>	(positive real) The <i>initial</i> step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.
<code>metric</code>	(string) One of "diag_e", "dense_e", or "unit_e", specifying the geometry of the base manifold. See the <i>Euclidean Metric</i> section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the <code>inv_metric</code> argument below.
<code>metric_file</code>	(character vector) One path, used for all chains, or one path per chain to JSON or Rdump files compatible with CmdStan that contain precomputed inverse metrics. The <code>metric_file</code> argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named <code>inv_metric</code> , referring to the <i>inverse</i> metric. An alternative to <code>metric_file</code> is the <code>inv_metric</code> argument (see below), which allows specifying an inverse metric directly using a vector or matrix from your R session.
<code>inv_metric</code>	(vector, matrix, or list) A vector (if <code>metric = "diag_e"</code>) or a matrix (if <code>metric = "dense_e"</code>) for initializing the inverse metric. A single vector or matrix is used for all chains. To initialize the chains with different inverse metrics, provide a list containing one vector or matrix per chain. A length-one list is also used for all chains. This can be used as an alternative to the <code>metric_file</code> argument. The inverse metric is usually set to an estimate of the posterior covariance. See the <code>adapt_engaged</code> argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.
<code>init_buffer</code>	(nonnegative integer) Width of initial fast timestep adaptation interval during warmup.
<code>term_buffer</code>	(nonnegative integer) Width of final fast timestep adaptation interval during warmup.
<code>window</code>	(nonnegative integer) Initial width of slow timestep/metric adaptation interval.
<code>fixed_param</code>	(logical) When TRUE, call CmdStan with argument <code>"algorithm=fixed_param"</code> . 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. For CmdStan versions before 2.36, `fixed_param = TRUE` is mandatory if the parameters block is empty.

<code>show_messages</code>	(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 <code>\$output()</code> method of the resulting fit object can be used to display the silenced messages.
<code>show_exceptions</code>	(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 <code>\$output()</code> method of the resulting fit object can be used to display the silenced messages.
<code>diagnostics</code>	(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 <code>read_cmdstan_csv()</code>. 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 <code>\$sampler_diagnostics()</code> method provides access to the diagnostic values for each iteration and the <code>\$diagnostic_summary()</code> method provides summaries of the diagnostics and can regenerate the warning messages. Diagnostics like R-hat and effective sample size are <i>not</i> currently available via the diagnostics argument but can be checked after fitting using the <code>\$summary()</code> method.
<code>save_metric</code>	(logical) When TRUE, call CmdStan with argument "adaptation save_metric=1" to save the adapted metric in a separate JSON file with elements "stepsize", "metric_type" and "inv_metric". The default is FALSE but can be set to TRUE for an entire R session by <code>options(cmdstanr_save_metric = TRUE)</code> .
<code>save_cmdstan_config</code>	(logical) When TRUE, 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). The default is FALSE but can be set to TRUE for an entire R session by <code>options(cmdstanr_save_config = TRUE)</code> .

Value

A `CmdStanMCMC` object.

References

- Hoffman, M. D., and Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(47), 1593-1623.

- Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. arXiv:1701.02434. Appendix A describes Stan's dynamic HMC/NUTS implementation.
- Stan Development Team. Stan Reference Manual (Algorithms section): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

Other CmdStanModel methods: [model-method-check_syntax](#), [model-method-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [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"))

# 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 the Pathfinder variational inference 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 = getOption("cmdstanr_save_config", FALSE)
)
```

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 <code>write_stan_json()</code>. See <code>write_stan_json()</code> 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, <code>mpi_args = list("n" = 4)</code> launches the executable as <code>mpiexec -n 4 model_executable</code>, followed by CmdStan arguments for the model executable.
seed	(non-negative 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 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 <i>different</i> data for each chain.
refresh	(non-negative integer) The number of iterations between printed screen updates. If <code>refresh = 0</code>, 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 <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. • The number 0. This initializes <i>all</i> parameters to 0 on the <i>unconstrained</i> parameter space. • A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. See <code>write_stan_json()</code> 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, and for Pathfinder it should contain a sublist for each path. 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 <code>chain_id</code>. For MCMC and Pathfinder,

the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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` methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For `CmdStanVB`, `CmdStanLaplace`, and `CmdStanPathfinder` fit objects the draws must additionally be *distinct*. A `CmdStanMLE` fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a `CmdStanPathfinder` fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 sampler or variational algorithm be written to diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument. The content is controlled by the user's CmdStan installation. The default is `FALSE`, which is appropriate for almost every use case. To save temporary diagnostic files permanently, use `$save_latent_dynamics_files()`.

`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()`.

<code>output_basename</code>	(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 is composed of the model name, timestamp, and a six-character random hexadecimal suffix.
<code>chains</code>	(positive integer) The number of Markov chains to run. The default is 4.
<code>chain_ids</code>	(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).
<code>iter_warmup</code>	(positive integer) The number of warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as <code>num_warmup</code> .
<code>iter_sampling</code>	(positive integer) The number of post-warmup iterations to run per chain. Note: in the CmdStan User's Guide this is referred to as <code>num_samples</code> .
<code>save_warmup</code>	(logical) Should warmup iterations be saved? The default is FALSE.
<code>thin</code>	(positive integer) The period between saved samples. This should typically be left at its default (no thinning) unless memory is a problem.
<code>max_treedepth</code>	(positive integer) The maximum allowed tree depth for the NUTS engine. See the <i>Tree Depth</i> section of the CmdStan User's Guide for more details.
<code>adapt_engaged</code>	(logical) Do warmup adaptation? The default is TRUE. If a precomputed inverse metric is specified via the <code>inv_metric</code> argument (or <code>metric_file</code>) then, if <code>adapt_engaged=TRUE</code> , 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 <code>adapt_engaged=FALSE</code> .
<code>adapt_delta</code>	(real in $(0, 1)$) The adaptation target acceptance statistic.
<code>step_size</code>	(positive real) The <i>initial</i> step size for the discrete approximation to continuous Hamiltonian dynamics. This is further tuned during warmup.
<code>metric</code>	(string) One of "diag_e", "dense_e", or "unit_e", specifying the geometry of the base manifold. See the <i>Euclidean Metric</i> section of the CmdStan User's Guide for more details. To specify a precomputed (inverse) metric, see the <code>inv_metric</code> argument below.
<code>metric_file</code>	(character vector) One path, used for all chains, or one path per chain to JSON or Rdump files compatible with CmdStan that contain precomputed inverse metrics. The <code>metric_file</code> argument is inherited from CmdStan but is confusing in that the entry in JSON or Rdump file(s) must be named <code>inv_metric</code> , referring to the <i>inverse</i> metric. An alternative to <code>metric_file</code> is the <code>inv_metric</code> argument (see below), which allows specifying an inverse metric directly using a vector or matrix from your R session.
<code>inv_metric</code>	(vector, matrix, or list) A vector (if <code>metric = "diag_e"</code>) or a matrix (if <code>metric = "dense_e"</code>) for initializing the inverse metric. A single vector or matrix is used for all chains. To initialize the chains with different inverse metrics, provide a

	list containing one vector or matrix per chain. A length-one list is also used for all chains. This can be used as an alternative to the <code>metric_file</code> argument. The inverse metric is usually set to an estimate of the posterior covariance. See the <code>adapt_engaged</code> argument above for details about (and control over) how specifying a precomputed inverse metric interacts with adaptation.
<code>init_buffer</code>	(nonnegative integer) Width of initial fast timestep adaptation interval during warmup.
<code>term_buffer</code>	(nonnegative integer) Width of final fast timestep adaptation interval during warmup.
<code>window</code>	(nonnegative integer) Initial width of slow timestep/metric adaptation interval.
<code>fixed_param</code>	(logical) When TRUE, call <code>CmdStan</code> with argument <code>"algorithm=fixed_param"</code> . 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. For <code>CmdStan</code> versions before 2.36, <code>fixed_param = TRUE</code> is mandatory if the parameters block is empty.
<code>sig_figs</code>	(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If NULL (the default), the default from the installed <code>CmdStan</code> version is used. Use <code>\$cmdstan_defaults()</code> to check that default. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
<code>show_messages</code>	(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 <code>\$output()</code> method of the resulting fit object can be used to display the silenced messages.
<code>show_exceptions</code>	(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 <code>\$output()</code> method of the resulting fit object can be used to display the silenced messages.
<code>diagnostics</code>	(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 <code>CmdStanR</code> 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 <code>read_cmdstan_csv()</code>. 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 <code>\$sampler_diagnostics()</code> method provides access to the diagnostic values for each iteration and the <code>\$diagnostic_summary()</code> method provides summaries of the diagnostics and can regenerate the warning messages. Diagnostics like R-hat and effective sample size are <i>not</i> currently available via the <code>diagnostics</code> argument but can be checked after fitting using the <code>\$summary()</code> method.

save_cmdstan_config

(logical) When TRUE, 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). The default is FALSE but can be set to TRUE for an entire R session by options(cmdstanr_save_config = TRUE).

Value

A `CmdStanMCMC` object.

References

- Hoffman, M. D., and Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(47), 1593-1623.
- Betancourt, M. (2017). A conceptual introduction to Hamiltonian Monte Carlo. arXiv:1701.02434. Appendix A describes Stan's dynamic HMC/NUTS implementation.
- Stan Development Team. Stan Reference Manual (Algorithms section): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

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-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate_quantities](#), [model-method-laplace](#), [model-method-model_info](#), [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 variable represented as a list with information on its scalar type (real or int) and number of dimensions.

The number of dimensions reported is the number of indexing dimensions in the declared Stan variable, equivalently the number of indices needed to access one scalar element. This means a scalar has 0 dimensions, a vector or one-dimensional array has 1, and a matrix or two-dimensional array has 2. Array dimensions are added to any vector or matrix dimensions, so `array[J] matrix[N, K]` has 3 dimensions. See **Examples**.

transformed data is not included, as variables in that block are not part of the model's input or output.

Usage

```
variables()
```

Value

The method returns a list with information on input and output variables for each of the Stan model blocks.

See Also

[write_stan_json\(\)](#) for writing data for CmdStan.

Other CmdStanModel methods: [model-method-check_syntax](#), [model-method-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate_quantities](#), [model-method-laplace](#), [model-method-model_info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-sample](#), [model-method-sample_mpi](#), [model-method-variational](#)

Examples

```
## Not run:
stan_file <- write_stan_file("
data {
  int N;
  array[2, 3] int y;
}
parameters {
  real alpha;
  vector[N] beta;
  array[2] matrix[3, 4] theta;
```

```

}
")

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

vars <- mod$variables()
str(vars)

## 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. See the [CmdStan User's Guide](#) for more details on the default arguments. These values are also available via the `$cmdstan_defaults` method.

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,
    draws = NULL,
    show_messages = TRUE,
    show_exceptions = TRUE,
    save_cmdstan_config = getOption("cmdstanr_save_config", FALSE)
  )

```

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 <code>write_stan_json()</code>. See <code>write_stan_json()</code> 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	(non-negative 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 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 <i>different</i> 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 <i>all</i> parameters randomly between $[-x, x]$ on the <i>unconstrained</i> parameter space. • The number 0. This initializes <i>all</i> parameters to 0 on the <i>unconstrained</i> parameter space. • A character vector of paths to JSON or Rdump files containing initial values for all or some parameters. For MCMC and Pathfinder, if only a single file is provided it will be reused for all chains and paths. See <code>write_stan_json()</code> 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, and for Pathfinder it should contain a sublist for each path. 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 and Pathfinder, the function is called once for each chain or path. If the function has the `chain_id` argument, it receives the chain or path number, starting at 1. 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 methods, which need one initialization per chain or path, the inits are drawn from the fit object without replacement, so it must contain at least as many draws as the number of chains/paths. For `CmdStanVB`, `CmdStanLaplace`, and `CmdStanPathfinder` fit objects the draws must additionally be *distinct*. A `CmdStanMLE` fit object is the exception: its single draw (the mode) is used to initialize every chain or path. When a `CmdStanPathfinder` fit object is used as the init, if CmdStan actually performed PSIS resampling (which requires `num_paths > 1`, `psis_resample = TRUE`, and `calculate_lp = TRUE`), CmdStanR selects from the returned draws using uniform weights to avoid applying importance weights again. If CmdStan did not PSIS-resample the output and `calculate_lp = TRUE`, CmdStanR selects draws using Pareto-smoothed importance weights. This includes single-path fits with `psis_resample = TRUE`, because CmdStan does not PSIS-resample single-path output. If `calculate_lp = FALSE`, uniform weights are used because importance weights cannot be calculated. 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 fewer draws than the number of requested chains/paths, the draws are reused in their existing order until each chain/path has an initialization. If there are more draws than requested chains/paths, draws are selected uniformly without replacement. 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 draws 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 sampler or variational algorithm be written to diagnostic CSV files? This argument replaces CmdStan's `diagnostic_file` argument. The content is controlled by the user's CmdStan installation. The default is `FALSE`, which is appropriate for almost every use case. To save temporary diagnostic files permanently, use `$save_latent_dynamics_files()`.

`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()`.

<code>output_basename</code>	(string) A string to use as a prefix for the names of the output CSV files of CmdStan. If <code>NULL</code> (the default), the basename of the output CSV files is composed of the model name, timestamp, and a six-character random hexadecimal suffix.
<code>sig_figs</code>	(positive integer) The number of significant figures (up to a maximum of 18) to use when storing the output values. If <code>NULL</code> (the default), the default from the installed CmdStan version is used. Use <code>\$cmdstan_defaults()</code> to check that default. Increasing this value will result in larger output CSV files and thus an increased usage of disk space.
<code>threads</code>	(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 <code>reduce_sum()</code> or <code>map_rect()</code>).
<code>opencl_ids</code>	(integer vector of length 2) The platform and device IDs of the OpenCL device to use for fitting. The model must be compiled with <code>cpp_options = list(stan_opencl = TRUE)</code> for this argument to have an effect.
<code>algorithm</code>	(string) The algorithm. Either "meanfield" or "fullrank".
<code>iter</code>	(positive integer) The <i>maximum</i> number of iterations.
<code>grad_samples</code>	(positive integer) The number of samples for Monte Carlo estimate of gradients.
<code>elbo_samples</code>	(positive integer) The number of samples for Monte Carlo estimate of ELBO (objective function).
<code>eta</code>	(positive real) The step size weighting parameter for adaptive step size sequence.
<code>adapt_engaged</code>	(logical) Do warmup adaptation?
<code>adapt_iter</code>	(positive integer) The <i>maximum</i> number of adaptation iterations.
<code>tol_rel_obj</code>	(positive real) Convergence tolerance on the relative norm of the objective.
<code>eval_elbo</code>	(positive integer) Evaluate ELBO every Nth iteration.
<code>draws</code>	(positive integer) Number of approximate posterior samples to draw and save.
<code>show_messages</code>	(logical) When <code>TRUE</code> (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.
<code>show_exceptions</code>	(logical) When <code>TRUE</code> (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, 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). The default is FALSE but can be set to TRUE for an entire R session by options(cmdstanr_save_config = TRUE).

Value

A `CmdStanVB` object.

References

- Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2017). Automatic differentiation variational inference. *Journal of Machine Learning Research*, 18(14), 1-45.
- Stan Development Team. Stan Reference Manual (Algorithms section, variational inference): <https://mc-stan.org/docs/reference-manual/>
- Stan Development Team. Stan documentation: <https://mc-stan.org/users/documentation/>
- Stan Development Team. CmdStan User's Guide: <https://mc-stan.org/docs/cmdstan-guide/>

See Also

Other CmdStanModel methods: [model-method-check_syntax](#), [model-method-cmdstan_defaults](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose_functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [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"))

# 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 the Pathfinder variational inference 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)

```

print_stan_file

Print a Stan file with syntax highlighting in Quarto and R Markdown

Description

Prints the contents of a Stan file, optionally with syntax highlighting when used in a Quarto or R Markdown document. When called inside a [knitr](#) code chunk with the chunk option `output: asis`

(or results: asis in R Markdown), the output is a fenced Stan code block that Quarto renders with syntax highlighting. When called interactively or without output: asis, the code is printed as plain text via `writelnLines()`.

Usage

```
print_stan_file(file, fold = FALSE, summary = "Stan model code")
```

Arguments

file	(string) Path to a .stan file.
fold	(logical) Whether to wrap the output in an HTML <details> block so that the code is collapsed (folded) by default. Only has an effect when rendering with output: asis and when outputting HTML. Defaults to FALSE.
summary	(string) The summary text shown in the fold toggle when fold = TRUE. Defaults to "Stan model code".

Value

The file path (invisibly).

Quarto usage

Use in a Quarto code chunk with output: asis to get syntax highlighting:

```
```{r}
#| output: asis
print_stan_file("path/to/model.stan")
```
```

To make the code block collapsible:

```
```{r}
#| output: asis
print_stan_file("path/to/model.stan", fold = TRUE)
```
```

See Also

`write_stan_file()` and `cmdstan_model()`

Examples

```
stan_file <- write_stan_file("
parameters {
  real y;
}
model {
  y ~ std_normal();
```

```

}
")

# Prints plain code at the console
print_stan_file(stan_file)

```

read_cmdstan_csv

Read CmdStan CSV files into R

Description

read_cmdstan_csv() reads CmdStan's output CSV files into R and returns the contents as a list (see the **Value** section for details). 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,
  variables = NULL,
  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. Compressed files ending in .csv.gz or .csv.bz2 can also be read. They are decompressed with gzip or bzip2 (on Windows the copies that come with Rtools) into a temporary file, so the temporary directory needs room for the uncompressed size of all the files. Some Linux distributions keep /tmp in memory, where that room runs out sooner than on disk. Set the TMPDIR environment variable before starting R to use a different directory. |
| variables | (character vector) Optionally, the names of the variables (parameters, transformed parameters, and generated quantities) to read in. <ul style="list-style-type: none"> • 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 fitted model object ([CmdStanMCMC](#), [CmdStanVB](#), etc.). A fitted model object created this way has some limitations compared to fitted model objects created directly by a model fitting method. See the **Reconstructed fitted model objects** section below for details.

`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 [MCMC](#) 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 the [laplace](#), [pathfinder](#) and [variational](#) methods, 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:

- time: Run time information for the individual processes, with one row in the chains data frame per CSV file. The returned object is the same as for the [\\$time\(\)](#) method except the total run time can't be inferred directly from the CSV files (they may have been generated in parallel) and is therefore NA. For CmdStan versions before 2.39 the individual process times are reported as zero.
- generated_quantities: A [draws_array](#) of the generated quantities.

Reconstructed fitted model objects

`as_cmdstan_fit()` reconstructs a fitted model object from CmdStan CSV files. The CSV files do not contain all of the information available to the model fitting methods, the Stan source, the console output, or the paths to most input and auxiliary files, so the reconstructed object has a reduced set of methods.

Only the following methods are available for every reconstructed object: `$draws()`, `$lp()`, `$materialize()`, `$metadata()`, `$output_files()`, `$print()`, `$save_object()`, and `$summary()`. Additional methods are available according to the inference method:

- For MCMC, `$diagnostic_summary()`, `$inv_metric()`, `$loo()`, `$num_chains()`, `$sampler_diagnostics()`, and `$time()` are available. However, the total time reported by `$time()` will be NA because the CSV files do not record whether the chains ran in parallel.
- For optimization, `$mle()` is available.
- For variational inference, Laplace approximation, and Pathfinder, `$lp_approx()` is available.

All other fitted-model methods are unavailable because they require information not contained in the CSV files. Calling an unavailable method produces an informative error.

See Also

[draws_to_csv\(\)](#) for creating compatible CSV files and [\\$output_files\(\)](#) for locating files created by CmdStanR

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 with as_cmdstan_fit()

# Create a CmdStanMCMC object from the CSV files
```

```

fit2 <- as_cmdstan_fit(csv_files)
fit2$print("beta")
str(fit2$draws())

# 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`.

Value

A named list containing the registered engine, invisibly.

References

- [Register a custom language engine for knitr](#)
- [knitr's built-in Stan language engine](#)

See Also

[eng_cmdstan\(\)](#)

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 is 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 using the "CMDSTAN" environment variable when available, otherwise the default path used by <code>install_cmdstan()</code> if it exists. |
| error_on_NA | (logical) Should an error be thrown if CmdStan is not found. The default is TRUE. If FALSE, <code>cmdstan_version()</code> 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" points directly to a valid CmdStan installation at load time, that path is used for the R session. If it instead points to an existing parent directory containing versioned CmdStan installations, the installation with the largest version number is used.
- If no environment variable is found when loaded but any directory in the form ".cmdstan/cmdstan-[version]" (e.g., ".cmdstan/cmdstan-2.35.0"), exists in the user's home directory (*not* the current working directory), then the path to the CmdStan installation with the largest version number is used for the R session. On Windows the home directory is determined from USERPROFILE, falling back to HOMEDRIVE and HOMEPATH. On other platforms it is determined from HOME. This is the same default directory that `install_cmdstan()` uses.

It is always possible to change the path after loading the package using `set_cmdstan_path(path)`.

Value

`set_cmdstan_path()` invisibly returns the supplied or automatically resolved path. If path = NULL and no installation is found, it invisibly returns NULL.

`cmdstan_path()` returns the current CmdStan path as a string or it errors if no path has been set.

`cmdstan_version()` returns the CmdStan version as a string. If CmdStan is not found, it errors when `error_on_NA = TRUE` and returns NULL when `error_on_NA = FALSE`.

See Also

`install_cmdstan()`, `cmdstan_default_install_path()`, and `cmdstan_default_path()`

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 <code>cmdstanr_write_stan_file_dir</code> is used. If the global option is not set, temporary directory is used. |
| basename | (string) If <code>dir</code> 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 <code>basename</code> is not set. |

Value

The path to the file.

See Also

[print_stan_file\(\)](#), [cmdstan_model\(\)](#), and [cmdstanr_global_options](#)

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)
- factor -> integer (the index of each value's level)
- data.frame -> matrix (via [data.matrix\(\)](#)); every column must be numeric, integer, logical, or factor

- list -> array
- table -> vector, matrix, or array (depending on dimensions of table)

Factor conversion:

Factors are written as their level indices, i.e., the position of each value in `levels(x)` rather than the value itself. The default levels are the sorted unique values, e.g., `factor(c(10, 9, 8))` has levels 8, 9, 10 and is written as `[3, 2, 1]`. An unused level shifts the indices of the levels after it. The fitting methods of a model compiled from a Stan file will error if a factor is supplied for a variable that is not declared as `int`, but if `write_stan_json()` is called directly by the user it has no declarations to check and so it always does the conversion.

List to array conversion:

The list to array conversion is intended to make it easier to prepare the data for certain Stan declarations involving arrays:

- `array[K] vector[J] v` can be constructed in R as a list with K elements where each element is a vector of length J
- `array[K] matrix[I,J] m` can be constructed in R as a list with K elements where each element is an IxJ matrix
- `array[K,I,J] int n` can be constructed in R as a list with K elements where each element is an IxJ matrix of integers

These can also be passed in from R as arrays instead of lists but the list option is provided for convenience. A list always contributes exactly one leading dimension, so `array[K,L] vector[J] v` can be supplied either as a list of K matrices each with dimensions LxJ or as a single R array with dimensions KxLxJ. Nested lists are not supported: every element of the list must be a vector, matrix, or array.

Scalar vs. length-1 vector:

Because R does not distinguish between a scalar and a vector of length 1, a length-1 vector like `c(42)` is written to JSON as a scalar (42) rather than an array (`[42]`). If a Stan variable is declared as a vector or array that may have length 1, wrap the value in `array()` to force array output. Because `array()` uses the length of its input as the default dimension, this works regardless of length:

- `write_stan_json(list(x = array(42)), file)` writes "x": `[42]`
- `write_stan_json(list(x = array(c(42, 43))), file)` writes "x": `[42, 43]`

This is only necessary when calling `write_stan_json()` directly. When passing a data list to the fitting methods of a model compiled from a Stan file (e.g., `$sample()`), CmdStanR uses the model's variable declarations to make this correction automatically.

Value

NULL, invisibly.

See Also

`$variables()` for inspecting the input and output variables of a Stan program.

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 `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

* **CmdStanModel** methods

- model-method-check_syntax, [70](#)
- model-method-cmdstan_defaults, [71](#)
- model-method-compile, [72](#)
- model-method-diagnose, [75](#)
- model-method-expose_functions, [79](#)
- model-method-format, [80](#)
- model-method-generate-quantities, [82](#)
- model-method-laplace, [86](#)
- model-method-model-info, [90](#)
- model-method-optimize, [92](#)
- model-method-pathfinder, [99](#)
- model-method-sample, [106](#)
- model-method-sample_mpi, [114](#)
- model-method-variables, [121](#)
- model-method-variational, [122](#)

* **fitted model objects**

- CmdStanDiagnose, [14](#)
- CmdStanGQ, [15](#)
- CmdStanLaplace, [17](#)
- CmdStanMCMC, [19](#)
- CmdStanMLE, [21](#)
- CmdStanPathfinder, [26](#)
- CmdStanVB, [31](#)
- \$check_syntax(), [11](#), [23](#), [72–74](#)
- \$cmdstan_defaults, [82](#), [86](#), [92](#), [99](#), [106](#), [122](#)
- \$cmdstan_defaults(), [23](#), [84](#), [88](#), [95](#), [102](#), [109](#), [119](#), [125](#)
- \$cmdstan_diagnose(), [20](#), [27](#), [31](#)
- \$cmdstan_summary(), [20](#), [27](#), [31](#)
- \$cmdstan_version(), [23](#)
- \$code(), [15](#), [18](#), [19](#), [21](#), [23](#), [27](#), [31](#)
- \$compile(), [11](#), [23](#), [29](#), [70](#), [79](#), [91](#)
- \$config_files(), [18](#), [20](#), [22](#), [27](#), [32](#)
- \$constrain_variables(), [16](#), [19](#), [20](#), [22](#), [28](#), [32](#)
- \$cpp_options(), [23](#)
- \$data_file(), [14](#), [16](#), [18](#), [20](#), [22](#), [27](#), [32](#)

- \$diagnose(), [14](#), [24](#)
- \$diagnostic_summary(), [20](#), [57](#), [111](#), [119](#)
- \$draws(), [7–9](#), [15](#), [18](#), [19](#), [21](#), [27](#), [31](#), [48](#), [52](#), [83](#)
- \$exe_file(), [23](#), [72](#), [74](#)
- \$expose_functions(), [16](#), [18](#), [20](#), [22](#), [23](#), [28](#), [32](#), [74](#)
- \$fitted_params_files(), [15](#)
- \$format(), [23](#)
- \$generate_quantities(), [15](#), [24](#), [34](#), [41](#)
- \$grad_log_prob(), [16](#), [19](#), [20](#), [22](#), [28](#), [32](#)
- \$gradients(), [14](#), [78](#)
- \$has_stan_file(), [23](#)
- \$hessian(), [16](#), [19](#), [20](#), [22](#), [28](#), [32](#)
- \$hpp_file(), [23](#), [72](#), [74](#)
- \$include_paths(), [23](#)
- \$init(), [14](#), [18](#), [19](#), [21](#), [27](#), [31](#)
- \$init_model_methods(), [16](#), [18](#), [20](#), [22](#), [28](#), [32](#)
- \$inv_metric(), [19](#)
- \$laplace(), [17](#), [24](#), [53](#), [95](#)
- \$latent_dynamics_files(), [20](#), [32](#)
- \$log_prob(), [16](#), [18](#), [20](#), [22](#), [28](#), [32](#)
- \$loo(), [20](#)
- \$lp(), [14](#), [18](#), [19](#), [21](#), [27](#), [31](#), [39](#), [52](#)
- \$lp_approx(), [18](#), [27](#), [31](#)
- \$materialize(), [15](#), [18](#), [20](#), [22](#), [27](#), [32](#), [58](#)
- \$metadata(), [14](#), [15](#), [18](#), [19](#), [21](#), [27](#), [31](#)
- \$metric_files(), [20](#)
- \$mle(), [21](#), [40](#)
- \$mode(), [18](#)
- \$model_name(), [23](#)
- \$num_chains(), [15](#), [19](#)
- \$optimize(), [21](#), [24](#), [52](#), [89](#)
- \$output(), [16](#), [18](#), [20](#), [22](#), [27](#), [32](#), [76](#), [78](#), [84](#), [89](#), [95](#), [96](#), [103](#), [111](#), [119](#), [125](#)
- \$output_files(), [14](#), [15](#), [18](#), [20](#), [22](#), [27](#), [32](#), [132](#)
- \$pathfinder(), [24](#), [26](#), [35](#)

`$print()`, 15, 18, 20, 22, 23, 27, 30, 31
`$profile_files()`, 16, 18, 20, 22, 27, 32
`$profiles()`, 15, 18, 19, 21, 27, 31
`$return_codes()`, 16, 18, 20, 22, 27, 32
`$sample()`, 19, 24, 35, 114
`$sample_mpi()`, 24
`$sampler_diagnostics()`, 19, 37, 38, 111, 119
`$save_config_files()`, 18, 20, 22, 27, 32
`$save_data_file()`, 14, 16, 18, 20, 22, 27, 32
`$save_hpp_file()`, 23, 74
`$save_hpp_file(dir)`, 72
`$save_latent_dynamics_files()`, 20, 32, 109, 117, 124
`$save_metric_files()`, 20
`$save_object()`, 15, 18, 20, 22, 27, 32
`$save_output_files()`, 14, 16, 18, 20, 22, 27, 32, 78, 84, 88, 94, 101, 109, 118, 125
`$save_profile_files()`, 16, 18, 20, 22, 27, 32
`$stan_file()`, 23
`$summary()`, 15, 18, 20, 22, 27, 31, 35, 37, 111, 119
`$time()`, 16, 18, 20, 22, 27, 32, 89, 131, 132
`$unconstrain_draws()`, 16, 19, 20, 22, 28, 32
`$unconstrain_variables()`, 16, 19, 20, 22, 28, 32
`$variable_skeleton()`, 16, 19, 21, 22, 28, 32
`$variables()`, 138
`$variational()`, 24, 31, 35

`array()`, 138
`as.CmdStanDiagnose(cmdstan_coercion)`, 10
`as.CmdStanGQ(cmdstan_coercion)`, 10
`as.CmdStanLaplace(cmdstan_coercion)`, 10
`as.CmdStanMCMC(cmdstan_coercion)`, 10
`as.CmdStanMLE(cmdstan_coercion)`, 10
`as.CmdStanPathfinder(cmdstan_coercion)`, 10
`as.CmdStanVB(cmdstan_coercion)`, 10
`as_cmdstan_fit(read_cmdstan_csv)`, 130
`as_cmdstan_fit()`, 10, 17, 19, 21, 27, 31, 53, 59
`as_draws(as_draws.CmdStanMCMC)`, 7
`as_draws.CmdStanMCMC`, 7
`as_mcmc.list`, 9

`base::saveRDS()`, 45, 58

`check_cmdstan_toolchain(install_cmdstan)`, 67
`check_syntax(model-method-check_syntax)`, 70
`cmdstan_coercion`, 10
`cmdstan_default_install_path()`, 69, 135
`cmdstan_default_path()`, 69, 135
`cmdstan_defaults(model-method-cmdstan_defaults)`, 71
`cmdstan_diagnose(fit-method-cmdstan_summary)`, 35
`cmdstan_make_local(install_cmdstan)`, 67
`cmdstan_model`, 11
`cmdstan_model()`, 4, 23, 29, 31, 72, 91, 129, 136
`cmdstan_path(set_cmdstan_path)`, 134
`cmdstan_summary(fit-method-cmdstan_summary)`, 35
`cmdstan_version(set_cmdstan_path)`, 134
`CmdStanDiagnose`, 10, 14, 16, 19, 21, 23, 24, 28, 32, 42, 78
`CmdStanGQ`, 10, 14, 15, 19, 21, 23, 24, 28, 32, 41, 53, 54, 84
`CmdStanLaplace`, 10, 14, 16, 17, 21, 23, 24, 28, 32, 53, 77, 83, 87–89, 94, 101, 108, 117, 124
`CmdStanMCMC`, 9, 10, 14, 16, 19, 19, 23, 24, 28, 32, 38, 39, 46, 53, 54, 57, 60, 77, 83, 87, 94, 101, 108, 111, 117, 120, 124, 131
`CmdStanMLE`, 10, 14, 16, 18, 19, 21, 21, 24, 28, 32, 52, 53, 77, 83, 87–89, 94, 96, 101, 108, 117, 124
`CmdStanModel`, 11, 14, 15, 17, 19, 21, 23, 26, 31, 59, 70–72, 74, 76, 79, 80, 82, 86, 90–92, 99, 106, 114, 121, 122
`CmdStanPathfinder`, 10, 14, 16, 19, 21, 23, 24, 26, 32, 77, 83, 87, 88, 94, 101, 103, 108, 117, 124
`CmdStanR(cmdstanr-package)`, 3
`cmdstanr(cmdstanr-package)`, 3
`cmdstanr-package`, 3
`cmdstanr_example`, 28
`cmdstanr_global_options`, 4, 30, 136

- CmdStanVB, [10](#), [14](#), [16](#), [19](#), [21](#), [23](#), [24](#), [28](#), [31](#),
[60](#), [77](#), [83](#), [87](#), [88](#), [94](#), [101](#), [108](#), [117](#),
[124](#), [126](#), [131](#)
- code (fit-method-code), [36](#)
- compile, [30](#), [115](#)
- compile (model-method-compile), [72](#)
- compiled, [84](#), [89](#), [95](#), [102](#), [109](#), [125](#)
- config_files
(fit-method-save_output_files),
[59](#)
- constrain_variables
(fit-method-constrain_variables),
[36](#)
- constrain_variables(), [42](#), [43](#), [45](#), [47](#),
[64–66](#)
- data.matrix(), [137](#)
- data_file
(fit-method-save_output_files),
[59](#)
- diagnose (model-method-diagnose), [75](#)
- diagnostic_summary
(fit-method-diagnostic_summary),
[37](#)
- draws, [30](#), [57](#), [131](#)
- draws (fit-method-draws), [38](#)
- draws_array, [15](#), [19](#), [40](#), [57](#), [131](#), [132](#)
- draws_matrix, [18](#), [21](#), [27](#), [31](#), [40](#), [132](#)
- draws_to_csv, [33](#)
- draws_to_csv(), [83](#), [132](#)
- eng_cmdstan, [34](#)
- eng_cmdstan(), [133](#), [134](#)
- environment variable, [135](#)
- expose_functions
(model-method-expose_functions),
[79](#)
- fit-method-cmdstan_diagnose
(fit-method-cmdstan_summary),
[35](#)
- fit-method-cmdstan_summary, [35](#)
- fit-method-code, [36](#)
- fit-method-constrain_variables, [36](#)
- fit-method-data_file
(fit-method-save_output_files),
[59](#)
- fit-method-diagnostic_summary, [37](#)
- fit-method-draws, [38](#)
- fit-method-expose_functions
(model-method-expose_functions),
[79](#)
- fit-method-fitted_params_files, [41](#)
- fit-method-grad_log_prob, [41](#)
- fit-method-gradients, [42](#)
- fit-method-hessian, [43](#)
- fit-method-init, [44](#)
- fit-method-init_model_methods, [44](#), [48](#)
- fit-method-inv_metric, [45](#)
- fit-method-latent_dynamics_files
(fit-method-save_output_files),
[59](#)
- fit-method-log_prob, [47](#)
- fit-method-loo, [48](#)
- fit-method-lp, [49](#)
- fit-method-materialize, [51](#)
- fit-method-metadata, [51](#)
- fit-method-mle, [52](#)
- fit-method-mode, [53](#)
- fit-method-num_chains, [53](#)
- fit-method-output, [54](#)
- fit-method-output_files
(fit-method-save_output_files),
[59](#)
- fit-method-print (fit-method-summary),
[61](#)
- fit-method-profile_files
(fit-method-save_output_files),
[59](#)
- fit-method-profiles, [55](#)
- fit-method-return_codes, [56](#)
- fit-method-sampler_diagnostics, [57](#)
- fit-method-save_config_files
(fit-method-save_output_files),
[59](#)
- fit-method-save_data_file
(fit-method-save_output_files),
[59](#)
- fit-method-save_latent_dynamics_files
(fit-method-save_output_files),
[59](#)
- fit-method-save_metric_files
(fit-method-save_output_files),
[59](#)
- fit-method-save_object, [58](#)
- fit-method-save_output_files, [59](#)
- fit-method-save_profile_files

- (fit-method-save_output_files), 59
- fit-method-summary, 35, 61
- fit-method-time, 63
- fit-method-unconstrain_draws, 64
- fit-method-unconstrain_variables, 65
- fit-method-variable_skeleton, 66
- fit\$init_model_methods(), 74
- fitted_params_files
 - (fit-method-fitted_params_files), 41
- format(model-method-format), 80
- garbage collected, 78, 84, 88, 94, 101, 109, 118, 125
- generate_quantities
 - (model-method-generate_quantities), 82
- generated quantities, 40
- grad_log_prob
 - (fit-method-grad_log_prob), 41
- grad_log_prob(), 37, 43, 45, 47, 64–66
- gradients(fit-method-gradients), 42
- hash, 136
- hashing, 136
- hessian(fit-method-hessian), 43
- hessian(), 37, 42, 45, 47, 64–66
- init(fit-method-init), 44
- init_model_methods
 - (fit-method-init_model_methods), 44
- install_cmdstan, 67
- install_cmdstan(), 4, 11, 31, 135
- inv_metric(fit-method-inv_metric), 45
- knitr, 128
- knitr::engine_output(), 34
- laplace, 132
- laplace(model-method-laplace), 86
- Laplace approximation, 40
- Laplace sampling, 92
- latent_dynamics_files
 - (fit-method-save_output_files), 59
- log_prob(fit-method-log_prob), 47
- log_prob(), 37, 42, 43, 45, 64–66
- loo(fit-method-loo), 48
- loo::loo.array(), 20, 48
- loo::loo_moment_match(), 48
- loo::loo_moment_match.default(), 48
- loo::relative_eff.array(), 48
- lp(fit-method-lp), 49
- lp_approx(fit-method-lp), 49
- materialize, 58
- materialize(fit-method-materialize), 51
- MCMC, 40, 131
- metadata(fit-method-metadata), 51
- metric_files
 - (fit-method-save_output_files), 59
- mle(fit-method-mle), 52
- mode(fit-method-mode), 53
- model-method-check_syntax, 70
- model-method-cmdstan_defaults, 71
- model-method-compile, 72
- model-method-diagnose, 75
- model-method-expose_functions, 79
- model-method-format, 80
- model-method-generate_quantities, 82
- model-method-laplace, 86
- model-method-model-info, 90
- model-method-optimize, 92
- model-method-pathfinder, 98
- model-method-sample, 106
- model-method-sample_mpi, 114
- model-method-variables, 121
- model-method-variational, 122
- moment-matching, 48
- num_chains(fit-method-num_chains), 53
- optimization, 40, 131
- optimize(model-method-optimize), 92
- options, 4
- options(), 30
- output(fit-method-output), 54
- output_files
 - (fit-method-save_output_files), 59
- Pathfinder, 40
- pathfinder, 132
- pathfinder(model-method-pathfinder), 99
- posterior::as_draws(), 8, 9

- posterior::draws_array, [83](#)
- posterior::draws_matrix, [83](#)
- posterior::subset_draws(), [8](#)
- posterior::summarise_draws(), [15](#), [18](#), [20](#), [22](#), [27](#), [31](#), [37](#), [61](#), [62](#)
- print.CmdStanGQ (fit-method-summary), [61](#)
- print.CmdStanLaplace (fit-method-summary), [61](#)
- print.CmdStanMCMC (fit-method-summary), [61](#)
- print.CmdStanMLE (fit-method-summary), [61](#)
- print.CmdStanPathfinder (fit-method-summary), [61](#)
- print.CmdStanVB (fit-method-summary), [61](#)
- print_example_program (cmdstanr_example), [28](#)
- print_stan_file, [128](#)
- print_stan_file(), [29](#), [136](#)
- profile_files (fit-method-save_output_files), [59](#)
- profiles (fit-method-profiles), [55](#)
- R6, [19](#), [23](#)
- read_cmdstan_csv, [130](#)
- read_cmdstan_csv(), [51](#), [111](#), [119](#)
- rebuild_cmdstan (install_cmdstan), [67](#)
- register_knitr_engine, [133](#)
- register_knitr_engine(), [34](#)
- return_codes (fit-method-return_codes), [56](#)
- sample (model-method-sample), [106](#)
- sample_mpi (model-method-sample_mpi), [114](#)
- sampler_diagnostics (fit-method-sampler_diagnostics), [57](#)
- save_config_files (fit-method-save_output_files), [59](#)
- save_data_file (fit-method-save_output_files), [59](#)
- save_latent_dynamics_files (fit-method-save_output_files), [59](#)
- save_metric_files (fit-method-save_output_files), [59](#)
- save_object, [51](#)
- save_object (fit-method-save_object), [58](#)
- save_output_files (fit-method-save_output_files), [59](#)
- save_profile_files (fit-method-save_output_files), [59](#)
- save_profile_files(), [55](#)
- set_cmdstan_path, [134](#)
- set_cmdstan_path(), [69](#)
- standalone generated quantities, [132](#)
- summarise_draws(), [61](#)
- summary (fit-method-summary), [61](#)
- temporary, [136](#)
- temporary directory, [33](#), [136](#)
- time (fit-method-time), [63](#)
- unconstrain_draws (fit-method-unconstrain_draws), [64](#)
- unconstrain_draws(), [37](#), [42](#), [43](#), [45](#), [47](#), [65](#), [66](#)
- unconstrain_variables (fit-method-unconstrain_variables), [65](#)
- unconstrain_variables(), [37](#), [42](#), [43](#), [45](#), [47](#), [64](#), [66](#)
- utils::capture.output(), [29](#)
- utils::relist(), [66](#)
- variable_skeleton (fit-method-variable_skeleton), [66](#)
- variable_skeleton(), [37](#), [42](#), [43](#), [45](#), [47](#), [64](#), [65](#)
- variables (model-method-variables), [121](#)
- variational, [132](#)
- variational (model-method-variational), [122](#)
- variational inference, [40](#)
- write_stan_file, [135](#)
- write_stan_file(), [11](#), [30](#), [31](#), [129](#)
- write_stan_json, [137](#)

`write_stan_json()`, [76](#), [77](#), [83](#), [87](#), [93](#), [100](#),
[107](#), [108](#), [116](#), [121](#), [123](#)
`writelnLines()`, [129](#)