

Package: stanr (via r-universe)

October 7, 2026

Title A Modern R Interface for Stan

Version 0.1.5

Description A modern, simpler R interface for Stan.

License BSD_3_clause + file LICENSE

URL <https://github.com/andrjohns/stanr>

BugReports <https://github.com/andrjohns/stanr/issues>

Depends R (>= 4.0)

Imports R6, QuickJSR, posterior, withr

Suggests testthat (>= 3.0.0), loo, V8

Encoding UTF-8

Roxygen list(markdown = TRUE)

NeedsCompilation yes

SystemRequirements GNU make, C++20

Config/testthat/edition 3

Config/testthat/parallel true

Config/roxygen2/version 8.0.0

Config/build/compilation-database true

Config/pak/sysreqs make

Repository <https://andrjohns.r-universe.dev>

Date/Publication 2026-09-07 12:24:09 UTC

RemoteUrl <https://github.com/andrjohns/stanr>

RemoteRef HEAD

RemoteSha 18faffb65577a776d96874392e7dca6d00f600bb

Contents

stanr-package	3
fit-method-diagnose	3
fit-method-draws	4
fit-method-expose-stan-functions	5
fit-method-fit-info	5
fit-method-gq	6
fit-method-laplace-mode	7
fit-method-loo	7
fit-method-lp	8
fit-method-mcmc	9
fit-method-mle	10
fit-method-model-methods	11
fit-method-print	12
fit-method-save	13
fit-method-summary	13
model-method-check-syntax	14
model-method-compile	14
model-method-diagnose	15
model-method-expose-stan-functions	16
model-method-format	18
model-method-generate-quantities	19
model-method-laplace	20
model-method-model-info	22
model-method-optimize	23
model-method-pathfinder	25
model-method-sample	27
model-method-variables	30
model-method-variational	31
stan_model	33
stanc	35
StanDiagnose	37
StanFit	37
StanGQ	38
StanLaplace	39
StanMCMC	39
StanMLE	40
StanModel	40
StanPathfinder	41
StanVB	42

Index

43

stanr-package	<i>The stanr package.</i>
---------------	---------------------------

Description

A modern, simpler R interface for Stan. Bundles all required Stan, Eigen, and TBB headers internally.

Author(s)

Maintainer: Andrew R. Johnson <andrew.johnson@arjohnsonau.com> ([ORCID](#))

Authors:

- Andrew R. Johnson <andrew.johnson@arjohnsonau.com> ([ORCID](#))

Other contributors:

- Stan Development Team (Stan sources and headers) [copyright holder]
- Walnutpie Developers (Walnutpie sources and headers) [copyright holder]
- Lawrence Livermore National Security (SUNDIALS sources and headers) [copyright holder]
- Intel Corporation (TBB sources and headers) [copyright holder]
- Posit Software, PBC (cpp11 sources and headers) [copyright holder]

See Also

Useful links:

- <https://github.com/andrjohns/stanr>
- Report bugs at <https://github.com/andrjohns/stanr/issues>

fit-method-diagnose	<i>Gradient diagnostic results</i>
---------------------	------------------------------------

Description

Access gradient checking results from a [StanDiagnose](#) object.

```
gradients()  
lp()  
num_failed()
```

Value

- `$gradients()` returns a data frame with gradient check results for each parameter, including columns `param_idx`, `value`, `model`, `finite_diff`, and `error`.
- `$lp()` returns the log probability evaluated at the initial parameter values.
- `$num_failed()` returns the number of parameters that failed the gradient check.

See Also[\\$draws\(\)](#)

Other StanFit methods: [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-draws

*Extract posterior draws***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.

Arguments

variables	(character vector) Which variables to extract. If NULL (the default), all variables are returned.
inc_warmup	(logical) Should warmup draws be included? Defaults to FALSE. Errors unless the fit is a \$sample() run with save_warmup = TRUE.
format	(string) The format of the returned draws. Must be a valid format from the posterior package. Defaults depend on the fitting method: "draws_array" for MCMC and generated quantities, "draws_matrix" for all other methods.

Value

An object from the **posterior** package in the requested format.

See Also[\\$summary\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-expose-stan-functions

Expose Stan functions from a fitted model

Description

The `$expose_stan_functions()` method of a [StanFit](#) object delegates to the bound [StanModel](#)'s `$expose_stan_functions()` – see that topic for the full description of `global/verbose`, `_rng` reproducibility, caching, and the serialization caveat. `$expose_functions()` is an alias. `$functions` is a read-only active binding returning the model's `$functions` environment – shared with the model and with every other fit from the same model, since exposure is memoized on the model, not per fit.

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

Arguments

<code>global</code>	(logical) Should the exposed functions also be assigned into the global environment?
<code>verbose</code>	(logical) Should compiler progress messages be printed?

Value

`$expose_stan_functions()/expose_functions()` return the model's `$functions` environment, invisibly. Both methods, and reading `$functions`, error if this fit does not retain a model binding (e.g. after restoring a fit that was saved without its model).

See Also

[\\$expose_stan_functions\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-fit-info *Access fit information*

Description

These methods access information stored in a [StanFit](#) object.

```
return_codes()
metadata()
output()
time()
init()
code()
```

Value

- `$return_codes()` returns an integer vector with the Stan service's return code replicated once per chain (the in-process services report a single code for the whole run). A return code of 0 indicates success.
- `$metadata()` returns a named list of fit metadata including the seed, data, arguments, and model name. For adaptive MCMC fits, `step_size_adaptation` is a numeric vector of one adapted step size per chain; it is absent when sampling ran without adaptation.
- `$output()` returns a character vector of Stan output messages from the run. Messages from all chains or paths are interleaved in one vector; per-chain attribution is not available.
- `$time()` returns a list with timing information.
- `$init()` returns the user-specified initial values, or NULL if none were provided.
- `$code()` returns the Stan program as a string, or `character()` if the model is not available.

See Also

[\\$draws\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-gq

Generated quantities chain count

Description

Return the number of chains in a [StanGQ](#) object's draws.

Value

An integer giving the number of chains, or 0 if no draws are available.

See Also

[\\$draws\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-laplace-mode

Extract Laplace mode

Description

Return the mode (point estimate) used as the center of the Laplace approximation. This is either a [StanMLE](#) object from a prior call to [\\$optimize\(\)](#), or a numeric vector provided directly.

Value

The mode object or numeric vector.

See Also

[\\$draws\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-loo

Leave-one-out cross-validation (LOO-CV)

Description

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

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 loo() . <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. <ul style="list-style-type: none"> • TRUE will call loo::relative_eff() 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 `moment_match = TRUE` uses the fit's model methods (see [fit-method-model-methods](#)) to automatically supply the functions for the `log_lik_i`, `unconstrain_pars`, `log_prob_upars`, and `log_lik_i_upars` arguments to `loo::loo_moment_match()`.

... Other arguments (e.g., `cores`, `save_psis`, etc.) passed to `loo::loo.array()` or `loo::loo_moment_match.default()` (if `moment_match = TRUE` is set).

Value

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

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](#).

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

`fit-method-lp`

Extract log density draws

Description

Extract the log density (`lp__`) or log density approximation (`lp_approx__`) as a numeric vector. `$lp_approx()` is only available on [StanLaplace](#), [StanVB](#), and [StanPathfinder](#) objects.

`lp()`
`lp_approx()`

Value

A numeric vector, with one element per (post-warmup) draw, or a single value for optimization.

See Also`$draws()`

Other StanFit methods: `fit-method-diagnose`, `fit-method-draws`, `fit-method-expose-stan-functions`, `fit-method-fit-info`, `fit-method-gq`, `fit-method-laplace-mode`, `fit-method-loo`, `fit-method-mcmc`, `fit-method-mle`, `fit-method-model-methods`, `fit-method-print`, `fit-method-save`, `fit-method-summary`

fit-method-mcmc	<i>MCMC-specific methods</i>
-----------------	------------------------------

Description

Methods specific to `StanMCMC` objects for accessing sampler diagnostics, chain information, and model evaluation.

```
sampler_diagnostics(inc_warmup = FALSE, format = "draws_array")
num_chains()
diagnostic_summary(diagnostics = c("divergences", "treedepth", "ebfmi"),
                   quiet = FALSE)
inv_metric(matrix = TRUE)
loo(variables = "log_lik", r_eff = FALSE, moment_match = FALSE, ...)
```

Arguments

<code>matrix</code>	(logical) For <code>\$inv_metric()</code> : return each chain's inverse metric as a matrix? For a diagonal metric, <code>TRUE</code> (the default) wraps the adapted diagonal in <code>diag()</code> ; <code>FALSE</code> returns it as a vector. For a dense metric, the matrix is always returned and <code>matrix = FALSE</code> is an error.
<code>diagnostics</code>	(character vector) For <code>\$diagnostic_summary()</code> : which diagnostics to check. One or more of "divergences", "treedepth", and "ebfmi". The default checks all three; <code>NULL</code> or "" checks none.
<code>quiet</code>	(logical) For <code>\$diagnostic_summary()</code> : should messages about problems found in the requested diagnostics be suppressed? The default, <code>FALSE</code> , prints them in addition to returning the values. Diagnostics that could not be computed at all (rather than computed and found problematic) always warn, regardless of quiet.

Value

- `$sampler_diagnostics()` returns sampler diagnostics (e.g., `divergent__`, `treedepth__`, `accept__`) as a posterior draws object.
- `$num_chains()` returns the number of MCMC chains.
- `$diagnostic_summary()` returns a list with one element per requested diagnostic: `num_divergent` (divergent transitions per chain), `num_max_treedepth` (iterations per chain that hit the max treedepth), and/or `ebfmi` (E-BFMI per chain). An element is `NA` for every chain if the corresponding diagnostic was not collected (e.g. `divergent__`/ `treedepth__`/ `energy__` are unavailable for the static and walnuts engines, or are all-`NA` for `fixed_param` runs).

- `$inv_metric()` returns a list (one element per chain) of the inverse mass matrix adapted during sampling. Errors if the fit was not sampled with adaptation.
- `$loo()` returns a LOO-CV object from the **loo** package.

See Also

`$draws()`, `$loo()`

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-mle

Extract point estimate

Description

Extract the maximum likelihood or maximum a posteriori estimate from a [StanMLE](#) object as a named numeric vector.

Arguments

`variables` (character vector) Which variables to extract. If NULL (the default), all variables are returned.

Value

A named numeric vector of parameter estimates. Always reflects the final optimization iteration, regardless of whether `$optimize()` was run with `save_iterations = TRUE`.

See Also

`$draws()`, which under `save_iterations = TRUE` returns the full optimization path (one row per saved iteration) instead of a single row for the final estimate.

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-model-methods](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-model-methods

Compute log probability and transformations

Description

These methods compute the log probability, gradients, Hessians, and parameter transformations using the compiled Stan model. They require the model to be available, and lazily initialize the native model pointer on first use.

```
log_prob(unconstrained_variables, jacobian = TRUE)
grad_log_prob(unconstrained_variables, jacobian = TRUE)
hessian(unconstrained_variables, jacobian = TRUE)
constrain_variables(unconstrained_variables,
                    transformed_parameters = TRUE,
                    generated_quantities = TRUE)
unconstrain_variables(variables)
unconstrain_draws(draws = NULL, format = NULL, inc_warmup = FALSE)
variable_skeleton(transformed_parameters = TRUE,
                  generated_quantities = TRUE)
```

Arguments

unconstrained_variables	(numeric vector) The unconstrained parameter values at which to evaluate the function.
jacobian	(logical) Should the log density be adjusted by the abs-determinant of the Jacobian of the inverse transformation?
variables	(named list) Constrained parameter values to unconstrain.
draws	A posterior draws object, or NULL to use the fit's draws.
format	(string) The format of the returned draws. Must be a valid format from the posterior package. Defaults depend on the fitting method: "draws_array" for MCMC and generated quantities, "draws_matrix" for all other methods.
inc_warmup	(logical) Should warmup draws be included? Defaults to FALSE. Errors unless the fit is a <code>\$sample()</code> run with <code>save_warmup = TRUE</code> .
transformed_parameters	(logical) Include transformed parameters?
generated_quantities	(logical) Include generated quantities?

Value

- `$log_prob()` returns the log probability as a numeric scalar.
- `$grad_log_prob()` returns the gradient as a numeric vector, with the log probability attached as the `log_prob` attribute.

- `$hessian()` returns a list with elements `log_prob`, `grad_log_prob`, and `hessian`.
- `$constrain_variables()` returns a named list of constrained parameter values.
- `$unconstrain_variables()` returns a numeric vector of unconstrained parameter values.
- `$unconstrain_draws()` returns a posterior draws object with unconstrained parameter values.
- `$variable_skeleton()` returns a named list with the structure of the constrained parameter space.

See Also

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-print](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-print	<i>Print fit summary</i>
------------------	--------------------------

Description

Print a summary table of posterior draws or point estimates.

Arguments

<code>digits</code>	(integer) Number of digits for numeric summary statistics.
<code>max_rows</code>	(integer) Maximum number of rows to print.

Value

The fitted model object, invisibly.

See Also

[\\$summary\(\)](#)

Other StanFit methods: [fit-method-diagnose](#), [fit-method-draws](#), [fit-method-expose-stan-functions](#), [fit-method-fit-info](#), [fit-method-gq](#), [fit-method-laplace-mode](#), [fit-method-loo](#), [fit-method-lp](#), [fit-method-mcmc](#), [fit-method-mle](#), [fit-method-model-methods](#), [fit-method-save](#), [fit-method-summary](#)

fit-method-save	<i>Save fitted model object</i>
-----------------	---------------------------------

Description

Methods for saving fitted model objects.

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

Arguments

file	(string) Path where the file should be saved.
...	Additional arguments passed to <code>base::saveRDS()</code> .

Value

- `$save_object()` returns the normalized file path, invisibly.

See Also

`$draws()`

Other StanFit methods: `fit-method-diagnose`, `fit-method-draws`, `fit-method-expose-stan-functions`, `fit-method-fit-info`, `fit-method-gq`, `fit-method-laplace-mode`, `fit-method-loo`, `fit-method-lp`, `fit-method-mcmc`, `fit-method-mle`, `fit-method-model-methods`, `fit-method-print`, `fit-method-summary`

fit-method-summary	<i>Summarize posterior draws</i>
--------------------	----------------------------------

Description

Compute summary statistics for posterior draws or point estimates using `posterior::summarise_draws()`.

Arguments

...	Additional arguments passed to <code>posterior::summarise_draws()</code> .
-----	--

Value

A data frame of summary statistics.

See Also

`$draws()`, `$print()`

Other StanFit methods: `fit-method-diagnose`, `fit-method-draws`, `fit-method-expose-stan-functions`, `fit-method-fit-info`, `fit-method-gq`, `fit-method-laplace-mode`, `fit-method-loo`, `fit-method-lp`, `fit-method-mcmc`, `fit-method-mle`, `fit-method-model-methods`, `fit-method-print`, `fit-method-save`

model-method-check-syntax

Check Stan program syntax

Description

The `$check_syntax()` method of a [StanModel](#) object runs the Stan program through `stanc` without compiling the generated C++. It is a cheap way to validate a program before calling `$compile()`.

Arguments

<code>pedantic</code>	(logical) Should <code>stanc</code> 's pedantic-mode warnings be requested? The default is FALSE.
<code>quiet</code>	(logical) Should the success message be suppressed? The default is FALSE.

Value

TRUE, invisibly. Errors if the program has a syntax error.

See Also

[\\$compile\(\)](#)

Other `StanModel` methods: [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-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](#)

model-method-compile *Compile a Stan program*

Description

The `$compile()` method of a [StanModel](#) object compiles the Stan program using the in-process backend. In most cases the user does not need to explicitly call `$compile()` as compilation occurs automatically when calling [stan_model\(\)](#). However it is possible to set `compile = FALSE` in the call to `stan_model()` and subsequently call `$compile()` directly.

Arguments

<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 FALSE.
<code>quiet</code>	(logical) Should verbose output from compilation be suppressed? The default is TRUE.

compile_standalone
(logical) Should the Stan program’s functions block be exposed as part of this compilation? Defaults to whatever was set at `stan_model()` construction time. See `$expose_stan_functions()`.

Value

The `StanModel` object, invisibly.

See Also

`$is_compiled()` and `stan_model()`
Other `StanModel` methods: `model-method-check-syntax`, `model-method-diagnose`, `model-method-expose-stan-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`

model-method-diagnose	<i>Run gradient diagnostics</i>
-----------------------	---------------------------------

Description

The `$diagnose()` method of a `StanModel` object runs Stan’s “diagnose” method to check the correctness of gradients computed by Stan. Returns a `StanDiagnose` object.

Arguments

data (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type	R shape
complex	complex scalar (e.g. 1+2i)
complex_vector[n], complex_row_vector[n], array[n]	complex vector of length n
complex_matrix[r, c]	complex matrix, r x c
tuple(T1, ..., Tk)	unnamed list of length k, element j shaped as Tj
array[n] tuple(...)	list of n tuple-shaped lists
array[n, m] tuple(...)	list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (`as.array(x)` or `dim(x) <- 1L`) to match its declared dimension.
Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

<code>seed</code>	(integer) The random seed for reproducibility.
<code>init</code>	Initial values for parameters. Either <code>NULL</code> (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (<code>Uniform(-init, init)</code> on the unconstrained scale; the default is 2), or a named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for <code>tuple(...)/complex</code> parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.
<code>epsilon</code>	(number) The finite difference step size.
<code>error</code>	(number) The maximum allowed relative error.
<code>show_messages</code>	(logical) When <code>TRUE</code> (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to <code>FALSE</code> to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
<code>show_exceptions</code>	(logical) When <code>TRUE</code> (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to <code>FALSE</code> to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.

Value

A `StanDiagnose` object containing gradient check results.

See Also

`$sample()`

Other StanModel methods: `model-method-check-syntax`, `model-method-compile`, `model-method-expose-stan-funct`, `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`

Description

The `$expose_stan_functions()` method of a [StanModel](#) object compiles the functions declared in the Stan program's functions block and makes them callable from R. `$expose_functions()` is an alias (`cmdstanr`'s name for the same method).

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

Exposed functions are always assigned into the `$functions` member environment (e.g. `mod$functions$my_fun(...)`); `global = TRUE` additionally assigns each one into the global environment, so it can also be called directly (`my_fun(...)`). Repeat calls are cheap: once a model's functions have been compiled – whether by an earlier `$expose_stan_functions()` call, or automatically via `compile_standalone = TRUE` at [stan_model\(\)](#) time – a later call just performs the requested `$functions/global` assignments, without recompiling.

A Stan function whose name ends in `_rng` gets a trailing `seed = NULL` argument on its R wrapper. All exposed `_rng` functions share one underlying RNG, seeded once per `$expose_stan_functions()` call from R's own RNG stream (so calling `set.seed()` before exposing makes draws reproducible); passing `seed` explicitly to a call reseeds the generator immediately before that call.

Stan `tuple(...)` arguments/returns map to/from **unnamed** R lists (one element per slot; arrays of tuples become lists of such lists), and `complex` / `complex_vector` / `complex_row_vector` / `complex_matrix` map to/from R's native complex type. An overloaded Stan function (same name, different signature) exposes only the first-defined overload, with a warning.

Arguments

<code>global</code>	(logical) Should the exposed functions also be assigned into the global environment? The default, <code>FALSE</code> , only populates <code>\$functions</code> .
<code>verbose</code>	(logical) Should compiler progress messages be printed? No compilation happens (so this has no effect) if the functions were already compiled, e.g. via <code>compile_standalone = TRUE</code> .

Value

The [StanModel](#) object's `$functions` environment, invisibly.

Caching

Compiled functions are cached on disk the same way compiled models are (see [stan_model\(\)](#)) – a warm cache skips recompilation entirely, including across R sessions.

Serialization

Exposed function objects in `$functions` are compiled bindings and, like any compiled function from this package, do not survive `saveRDS()/readRDS()`. After restoring a saved [StanModel](#) or [StanFit](#), call `$expose_stan_functions()` again to repopulate `$functions` – this rebuilds from the on-disk cache and does not recompile.

See Also

`stan_model()` for the `compile_standalone` argument, which exposes functions as part of the model's own compilation.

Other StanModel methods: `model-method-check-syntax`, `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-variables`, `model-method-variational`

model-method-format	<i>Format a Stan program</i>
---------------------	------------------------------

Description

The `$format()` method of a `StanModel` object reformats the Stan program using `stanc`'s auto-formatter and returns the result as a string. Unlike `$check_syntax()`, it does not work on programs with unresolved `#include` directives, since formatting would inline their contents.

Arguments

<code>overwrite_file</code>	(logical) Should the formatted code be written back to <code>\$stan_file()</code> ? The default is <code>FALSE</code> . Requires a model created with <code>stan_file</code> , and does not update this object's in-memory code – construct a new <code>StanModel</code> to pick up the change.
<code>canonicalize</code>	(logical or character) <code>FALSE</code> (the default) formats without canonicalizing, <code>TRUE</code> also canonicalizes deprecated syntax, and a character vector requests specific canonicalizations (e.g. <code>c("braces", "parentheses")</code>).
<code>backup</code>	(logical) When <code>overwrite_file = TRUE</code> , should the original file be backed up first? The default is <code>TRUE</code> .
<code>max_line_length</code>	(integer) Maximum output line width. The default, <code>NULL</code> , uses <code>stanc</code> 's default.
<code>quiet</code>	(logical) Should the backup message be suppressed? The default is <code>FALSE</code> .

Value

The formatted Stan code as a string.

See Also

`$check_syntax()`, `$code()`

Other StanModel methods: `model-method-check-syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose-stan-functions`, `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`

model-method-generate-quantities

Run generated quantities

Description

The `$generate_quantities()` method of a [StanModel](#) object runs Stan's "generate quantities" method, which evaluates the generated quantities block of the Stan program for a set of parameter values. Returns a [StanGQ](#) object.

Arguments

fitted_params A [StanFit](#) object, or anything accepted by `posterior::as_draws_matrix()`, containing draws with every model parameter present by name (e.g. `beta[1]`, not a positional column).

data (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type

complex

complex_vector[n], complex_row_vector[n], array[n] complex

complex_matrix[r, c]

tuple(T1, ..., Tk)

array[n] tuple(...)

array[n, m] tuple(...)

R shape

complex scalar (e.g. `1+2i`)

complex vector of length n

complex matrix, r x c

unnamed list of length k, element j shaped as `Tj`

list of n tuple-shaped lists

list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (`as.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

seed (integer) The random seed for reproducibility.

num_threads (integer) The total number of threads to use. Defaults to all available threads.

show_messages (logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's `$output()` method.

`show_exceptions` (logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's `$output()` method.

`opengl_ids` (integer vector) `c(platform_id, device_id)` identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with `use_opengl = TRUE` (see [stan_model\(\)](#)); errors if the model was not compiled with OpenCL support. Defaults to NULL, meaning `select_opengl_device()` is never called and the platform/device baked in at compile time (0/0) is used.

Value

A [StanGQ](#) object containing the generated quantities.

See Also

[\\$sample\(\)](#)

Other StanModel methods: [model-method-check-syntax](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-functions](#), [model-method-format](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-sample](#), [model-method-variables](#), [model-method-variational](#)

model-method-laplace *Run Laplace approximation*

Description

The `$laplace()` method of a [StanModel](#) object runs Stan's "laplace" method to draw from a Gaussian approximation to the posterior centered at the mode. Returns a [StanLaplace](#) object.

Arguments

`data` (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type	R shape
<code>complex</code>	complex scalar (e.g. <code>1+2i</code>)
<code>complex_vector[n]</code> , <code>complex_row_vector[n]</code> , <code>array[n]</code>	complex vector of length <code>n</code>
<code>complex_matrix[r, c]</code>	complex matrix, <code>r</code> x <code>c</code>
<code>tuple(T1, ..., Tk)</code>	unnamed list of length <code>k</code> , element <code>j</code> shaped as <code>Tj</code>
<code>array[n]</code>	list of <code>n</code> tuple-shaped lists
<code>array[n, m]</code>	list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (as `.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

<code>seed</code>	(integer) The random seed for reproducibility.
<code>init</code>	Initial values for parameters. Either NULL (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (<code>Uniform(-init, init)</code>) on the unconstrained scale; the default is 2), or a named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for <code>tuple(...)/complex</code> parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.
<code>mode</code>	A numeric vector of parameter values at the mode, or a StanMLE object from <code>\$optimize()</code> . If NULL, optimization is run first.
<code>opt_args</code>	(list) Additional arguments to pass to <code>\$optimize()</code> when finding the mode.
<code>jacobian</code>	(logical) Should the log density be adjusted by the abs-determinant of the Jacobian of the inverse transformation?
<code>draws</code>	(integer) The number of draws from the Laplace approximation.
<code>calculate_lp</code>	(logical) Should the log density of the Laplace approximation be calculated?
<code>num_threads</code>	(integer) The total number of threads to use. Defaults to all available threads.
<code>show_messages</code>	(logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
<code>show_exceptions</code>	(logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.
<code>opencl_ids</code>	(integer vector) <code>c(platform_id, device_id)</code> identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with <code>use_opencl = TRUE</code> (see stan_model()); errors if the model was not compiled with OpenCL support. Defaults to NULL, meaning <code>select_opencl_device()</code> is never called and the platform/device baked in at compile time (0/0) is used.

Value

A [StanLaplace](#) object containing approximate posterior draws.

See Also

`$optimize()`, `$variational()`

Other StanModel methods: `model-method-check-syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose-stan-functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-model-info`, `model-method-optimize`, `model-method-pathfinder`, `model-method-sample`, `model-method-variables`, `model-method-variational`

model-method-model-info

Access information from a StanModel object

Description

These methods access information stored in a `StanModel` object and print its Stan program.

```
stan_file()
has_stan_file()
code()
print()
model_name()
include_paths()
stan_version()
is_compiled()
cpp_options()
stanc_options()
use_opengl()
```

Value

- `$stan_file()` returns a path as a string, or `character()` if the model was created from code (not a file).
- `$has_stan_file()` returns TRUE if the model was created with a Stan file and FALSE otherwise.
- `$code()` returns the Stan program as a single string.
- `$print()` prints the Stan program and returns the `StanModel` object invisibly.
- `$model_name()` returns the model name as a string.
- `$include_paths()` returns a character vector of absolute paths.
- `$stan_version()` returns the Stan version bundled with the package as a string.
- `$is_compiled()` returns TRUE if the model has been compiled.
- `$cpp_options()` returns the `cpp_options` list the model was created with (see `stan_model()`).
- `$stanc_options()` returns a named list of stanc options.
- `$use_opengl()` returns TRUE if the model was created with `use_opengl = TRUE` and FALSE otherwise.

See Also

`$compile()` and `stan_model()`

Other StanModel methods: `model-method-check-syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose-stan-functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-optimize`, `model-method-pathfinder`, `model-method-sample`, `model-method-variables`, `model-method-variational`

model-method-optimize *Run optimization*

Description

The `$optimize()` method of a `StanModel` object runs Stan's "optimize" method to find the maximum a posteriori (MAP) estimate or maximum likelihood estimate (MLE). Returns a `StanMLE` object.

Arguments

`data` (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type	R shape
<code>complex</code>	complex scalar (e.g. <code>1+2i</code>)
<code>complex_vector[n]</code> , <code>complex_row_vector[n]</code> , <code>array[n]</code>	complex vector of length <code>n</code>
<code>complex_matrix[r, c]</code>	complex matrix, <code>r</code> x <code>c</code>
<code>tuple(T1, ..., Tk)</code>	unnamed list of length <code>k</code> , element <code>j</code> shaped as <code>Tj</code>
<code>array[n]</code>	list of <code>n</code> tuple-shaped lists
<code>array[n, m]</code>	list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (`as.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

`seed` (integer) The random seed for reproducibility.

<code>init</code>	Initial values for parameters. Either NULL (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (<code>Uniform(-init, init)</code>) on the unconstrained scale; the default is 2), or a named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for <code>tuple(...)/complex</code> parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.
<code>algorithm</code>	(string) The optimization algorithm: "lbfgs", "bfgs", or "newton".
<code>jacobian</code>	(logical) Should the log density be adjusted by the abs-determinant of the Jacobian of the inverse transformation?
<code>init_alpha</code>	(number) Initial step size for LBFGS.
<code>iter</code>	(integer) Maximum number of optimization iterations.
<code>tol_obj</code>	(number) Absolute tolerance for changes in objective value.
<code>tol_rel_obj</code>	(number) Relative tolerance for changes in objective value.
<code>tol_grad</code>	(number) Absolute tolerance for the norm of the gradient.
<code>tol_rel_grad</code>	(number) Relative tolerance for the norm of the gradient.
<code>tol_param</code>	(number) Absolute tolerance for changes in parameter values.
<code>history_size</code>	(integer) Number of corrections in LBFGS approximation.
<code>save_iterations</code>	(logical) Should optimization iterations be saved? When TRUE, <code>\$draws()</code> on the returned <code>StanMLE</code> exposes the full optimization path (one row per saved iteration, including the initial point) instead of a single row for the final estimate. <code>\$mle()</code> is unaffected either way and always reflects the final iteration.
<code>num_threads</code>	(integer) The total number of threads to use. Defaults to all available threads.
<code>show_messages</code>	(logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
<code>show_exceptions</code>	(logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.
<code>opencl_ids</code>	(integer vector) <code>c(platform_id, device_id)</code> identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with <code>use_opencl = TRUE</code> (see <code>stan_model()</code>); errors if the model was not compiled with OpenCL support. Defaults to NULL, meaning <code>select_opencl_device()</code> is never called and the platform/device baked in at compile time (0/0) is used.

Value

A `StanMLE` object containing the point estimate.

See Also

`$sample()`, `$laplace()`

Other StanModel methods: `model-method-check-syntax`, `model-method-compile`, `model-method-diagnose`, `model-method-expose-stan-functions`, `model-method-format`, `model-method-generate-quantities`, `model-method-laplace`, `model-method-model-info`, `model-method-pathfinder`, `model-method-sample`, `model-method-variables`, `model-method-variational`

model-method-pathfinder

Run Pathfinder

Description

The `$pathfinder()` method of a `StanModel` object runs Stan's "pathfinder" method, which uses a parallel iterative optimization approach to approximate the posterior distribution. Returns a `StanPathfinder` object.

Arguments

data (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type

`complex`

`complex_vector[n]`, `complex_row_vector[n]`, `array[n]` `complex`

`complex_matrix[r, c]`

`tuple(T1, ..., Tk)`

`array[n]` `tuple(...)`

`array[n, m]` `tuple(...)`

R shape

complex scalar (e.g. `1+2i`)

complex vector of length `n`

complex matrix, `r x c`

unnamed list of length `k`, element `j` shaped as `Tj`

list of `n` tuple-shaped lists

list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (`as.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

seed

(integer) The random seed for reproducibility.

<code>init</code>	Initial values for parameters. Either NULL (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (<code>Uniform(-init, init)</code>) on the unconstrained scale; the default is 2), or a named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for <code>tuple(...)/complex</code> parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.
<code>num_paths</code>	(integer) The number of paths to use.
<code>single_path_draws</code>	(integer) Number of draws per path.
<code>draws</code>	(integer) Total number of draws from the approximation.
<code>max_lbfgs_iters</code>	(integer) Maximum LBFGS iterations per path.
<code>num_elbo_draws</code>	(integer) Number of draws for ELBO estimation.
<code>save_single_paths</code>	(logical) Should single path results be saved?
<code>psis_resample</code>	(logical) Should Pareto smoothed importance sampling resampling be used?
<code>calculate_lp</code>	(logical) Should the log density be calculated?
<code>num_threads</code>	(integer) The total number of threads to use. Defaults to all available threads.
<code>show_messages</code>	(logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
<code>show_exceptions</code>	(logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.
<code>openccl_ids</code>	(integer vector) <code>c(platform_id, device_id)</code> identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with <code>use_openccl = TRUE</code> (see stan_model()); errors if the model was not compiled with OpenCL support. Defaults to NULL, meaning <code>select_openccl_device()</code> is never called and the platform/device baked in at compile time (0/0) is used.

Value

A [StanPathfinder](#) object containing approximate posterior draws.

See Also

[\\$sample\(\)](#), [\\$variational\(\)](#)

Other StanModel methods: [model-method-check-syntax](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-optimize](#), [model-method-sample](#), [model-method-variables](#), [model-method-variational](#)

Description

The `$sample()` method of a [StanModel](#) object runs Stan's "sample" method, which uses Hamiltonian Monte Carlo (HMC) with the No-U-Turn Sampler (NUTS) to draw from the posterior distribution. Returns a [StanMCMC](#) object.

Arguments

data (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type	R shape
<code>complex</code>	complex scalar (e.g. <code>1+2i</code>)
<code>complex_vector[n]</code> , <code>complex_row_vector[n]</code> , <code>array[n]</code>	complex vector of length <code>n</code>
<code>complex_matrix[r, c]</code>	complex matrix, <code>r</code> x <code>c</code>
<code>tuple(T1, ..., Tk)</code>	unnamed list of length <code>k</code> , element <code>j</code> shaped as <code>Tj</code>
<code>array[n] tuple(...)</code>	list of <code>n</code> tuple-shaped lists
<code>array[n, m] tuple(...)</code>	list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (as `.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

seed (integer) The random seed for reproducibility.

refresh (integer) How often (in iterations) to print progress.

init Initial values for parameters. Either `NULL` (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (`Uniform(-init, init)` on the unconstrained scale; the default is 2), or a named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for `tuple(...)`/complex parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.

chains	(integer) The number of MCMC chains.
chain_ids	(integer vector) The IDs for each chain.
num_threads	(integer) The total number of threads to use. Defaults to all available threads.
iter_warmup	(integer) The number of warmup iterations.
iter_sampling	(integer) The number of sampling iterations.
save_warmup	(logical) Should warmup samples be saved? Ignored when <code>fixed_param = TRUE</code> , since that mode runs no warmup and so has nothing to save.
thin	(integer) Thin interval.
max_treedepth	(integer) Maximum tree depth for NUTS.
adapt_engaged	(logical) Should step size adaptation be used?
adapt_delta	(number) Target acceptance statistic for HMC.
step_size	(number) Step size for HMC.
metric	Character string indicating the metric to use: "diag_e", "dense_e", or "unit_e".
inv_metric	(numeric) Initial inverse mass matrix values.
init_buffer	(integer) Adaptation phase: initial buffer length.
term_buffer	(integer) Adaptation phase: terminal buffer length.
window	(integer) Adaptation phase: window length.
fixed_param	(logical) Treat all parameters as fixed (no adaptation).
show_messages	(logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
show_exceptions	(logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.
diagnostics	(character vector) Which sampler diagnostics to check immediately after sampling, printing a warning message for any problems found – see \$diagnostic_summary() . One or more of "divergences", "treedepth", and "ebfmi". The default checks all three; NULL or "" skips the check entirely. Ignored when <code>fixed_param = TRUE</code> .
engine	(string) The sampling engine: "nuts", "static", or "walnuts" (see walnut-pie). walnuts only supports <code>metric = "diag_e"</code> and <code>adapt_engaged = TRUE</code> . It reuses <code>max_treedepth</code> as the maximum trajectory doublings, <code>adapt_delta</code> as the step size acceptance target, and <code>step_size</code> as the initial step size; <code>refresh</code> , <code>step_size_jitter</code> , <code>init_buffer</code> , <code>term_buffer</code> , <code>window</code> , and <code>adapt_gamma/adapt_kappa/adapt_t0</code> are ignored, and no sampler diagnostics or progress output are produced.
int_time	(number) Integration time for static HMC.
step_size_jitter	(number) Jitter for step size after adaptation.
adapt_gamma	(number) Adaptation hyperparameter for dual averaging.

<code>adapt_kappa</code>	(number) Adaptation hyperparameter for dual averaging.
<code>adapt_t0</code>	(number) Adaptation hyperparameter for dual averaging.
<code>max_step_halvings</code>	(integer) walnuts only: maximum number of step size halvings per macro step.
<code>min_micro_steps</code>	(integer) walnuts only: minimum number of micro steps per macro step.
<code>max_hamiltonian_error</code>	(number) walnuts only: maximum allowed error in the Hamiltonian at a macro step.
<code>mass_init_count</code>	(number) walnuts only: initial count for the mass matrix estimator.
<code>mass_additive_smoothing</code>	(number) walnuts only: additive smoothing for the mass matrix estimator.
<code>max_macro_steps_target</code>	(number) walnuts only: target number of macro steps per iteration.
<code>step_learning_rate</code>	(number) walnuts only: learning rate for the Adam step size adaptation.
<code>step_gradient_decay</code>	(number) walnuts only: gradient decay rate for the Adam step size adaptation.
<code>step_sq_gradient_decay</code>	(number) walnuts only: squared gradient decay rate for the Adam step size adaptation.
<code>step_stabilization</code>	(number) walnuts only: stabilization term for the Adam step size adaptation.
<code>step_learn_rate_decay</code>	(number) walnuts only: learning rate decay exponent for the Adam step size adaptation.
<code>opengl_ids</code>	(integer vector) <code>c(platform_id, device_id)</code> identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with <code>use_opengl = TRUE</code> (see stan_model()); errors if the model was not compiled with OpenCL support. Defaults to <code>NULL</code> , meaning <code>select_opengl_device()</code> is never called and the platform/device baked in at compile time (0/0) is used.

Value

A [StanMCMC](#) object containing posterior draws and diagnostics.

See Also

[\\$optimize\(\)](#), [\\$variational\(\)](#)

Other StanModel methods: [model-method-check-syntax](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-functions](#), [model-method-format](#), [model-method-generate-quantities](#), [model-method-laplace](#), [model-method-model-info](#), [model-method-optimize](#), [model-method-pathfinder](#), [model-method-variables](#), [model-method-variational](#)

model-method-variables

Input and output variables of a Stan program

Description

The `$variables()` method of a [StanModel](#) 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.

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

Value

A list with information on input and output variables for each of the Stan model blocks.

See Also

Other StanModel methods: [model-method-check-syntax](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-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-variational](#)

Examples

```
## Not run:
mod <- stan_model(
  code = "
    data {
      int N;
      array[2, 3] int y;
    }
    parameters {
      real alpha;
      vector[N] beta;
      array[2] matrix[3, 4] theta;
    }
  ",
  compile = FALSE
)

vars <- mod$variables()
```

```
str(vars)

## End(Not run)
```

model-method-variational

Run variational inference (ADVI)

Description

The `$variational()` method of a [StanModel](#) object runs Stan's "variational" method, which uses Automatic Differentiation Variational Inference (ADVI) to approximate the posterior distribution. Returns a [StanVB](#) object.

Arguments

data (named list) Values for all or part of the data block in the Stan program. Each element is named after the corresponding Stan variable and shaped as follows:

Stan type

complex

complex_vector[n], complex_row_vector[n], array[n] complex

complex_matrix[r, c]

tuple(T1, ..., Tk)

array[n] tuple(...)

array[n, m] tuple(...)

R shape

complex scalar (e.g. 1+2i)

complex vector of length n

complex matrix, r x c

unnamed list of length k, element j shaped as Tj

list of n tuple-shaped lists

list (over the first index) of lists (over the second index)

A length-1 R vector is read as a scalar, so a `vector[1]`, `array[1]` ..., or length-1 tuple-slot container value must be given an explicit shape (as `.array(x)` or `dim(x) <- 1L`) to match its declared dimension.

Real-valued input is accepted wherever a complex type is declared – it is coerced automatically, including inside tuple slots. All other Stan types use their ordinary R representation (numeric/integer scalars, vectors, matrices, arrays).

As a power-user escape hatch, tuple variables can instead be supplied pre-flattened using dotted names (e.g. `list(t.1 = ..., t.2 = ...)` for a tuple `t`), bypassing the list-shape conversion above; this is not the primary documented path. One caveat applies: a dotted entry for a complex tuple slot is assumed to hold exactly one complex value per enclosing array element, which is only correct when that slot is itself a plain (non-container) complex – for anything else (e.g. a `complex_vector` tuple slot), use the list shapes above instead.

seed (integer) The random seed for reproducibility.

init Initial values for parameters. Either NULL (equivalent to the default radius of 2), a non-negative number, used as the range of CmdStan-style random initialization (`Uniform(-init, init)` on the unconstrained scale; the default is 2), or a

	named list / named numeric vector of constrained parameter values. Supplied values are shared by all chains; parameters not covered by the supplied values are randomly initialized with the default range. A named list may include tuple-shaped list entries and R complex values for <code>tuple(...)/complex</code> parameters, using the same canonical shapes (and the same dotted-name escape hatch) documented for data.
<code>algorithm</code>	(string) The variational inference algorithm: "meanfield" or "fullrank".
<code>iter</code>	(integer) The number of ADVI iterations.
<code>grad_samples</code>	(integer) Number of samples to use for gradient estimation.
<code>elbo_samples</code>	(integer) Number of samples for ELBO evaluation.
<code>eta</code>	(number) Learning rate for ADVI.
<code>adapt_engaged</code>	(logical) Should the learning rate be adapted?
<code>adapt_iter</code>	(integer) Number of iterations for learning rate adaptation.
<code>tol_rel_obj</code>	(number) Relative tolerance for ELBO convergence.
<code>eval_elbo</code>	(integer) How often to evaluate the ELBO.
<code>draws</code>	(integer) The number of draws from the variational approximation.
<code>num_threads</code>	(integer) The total number of threads to use. Defaults to all available threads.
<code>show_messages</code>	(logical) When TRUE (the default), print progress and informational output (e.g. iteration and timing messages) to the console. Set to FALSE to silence it. Suppressed output is still recorded and available via the fit's <code>\$output()</code> method.
<code>show_exceptions</code>	(logical) When TRUE (the default), print informational messages about numerical exceptions – e.g. Metropolis proposal rejections and rejected initial values. Set to FALSE to silence them. Suppressed messages are still recorded and available via the fit's <code>\$output()</code> method.
<code>opencl_ids</code>	(integer vector) <code>c(platform_id, device_id)</code> identifying the OpenCL platform/device to run on. Only meaningful for a model compiled with <code>use_opencl = TRUE</code> (see stan_model()); errors if the model was not compiled with OpenCL support. Defaults to NULL, meaning <code>select_opencl_device()</code> is never called and the platform/device baked in at compile time (0/0) is used.

Value

A [StanVB](#) object containing approximate posterior draws.

See Also

[\\$sample\(\)](#), [\\$pathfinder\(\)](#)

Other StanModel methods: [model-method-check-syntax](#), [model-method-compile](#), [model-method-diagnose](#), [model-method-expose-stan-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](#)

stan_model	Create a Stan model object
------------	----------------------------

Description

Create a new [StanModel](#) object from a Stan program file or from Stan code as a string. The [StanModel](#) object stores the Stan program source and compiled model, and provides methods for fitting the model using Stan's inference algorithms.

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

Usage

```
stan_model(
  stan_file = NULL,
  code = NULL,
  compile = TRUE,
  model_name = NULL,
  include_paths = NULL,
  user_header = NULL,
  cpp_options = list(),
  stanc_options = list(),
  force_recompile = getOption("stanr_force_recompile", FALSE),
  precompiled_headers = TRUE,
  quiet = TRUE,
  external_cpp = NULL,
  use_opengl = FALSE,
  compile_standalone = FALSE,
  backend = "compiled"
)
```

Arguments

<code>stan_file</code>	(string) The path to a <code>.stan</code> file containing a Stan program. If <code>stan_file</code> is not specified then <code>code</code> must be specified.
<code>code</code>	(string) A Stan program as a single string. If <code>code</code> is not specified then <code>stan_file</code> must be specified.
<code>compile</code>	(logical) Should the model be compiled? The default is <code>TRUE</code> . If <code>FALSE</code> compilation can be done later via the \$compile() method.
<code>model_name</code>	(string) The name to use for the model. If <code>NULL</code> (the default), the model name is derived from the Stan file name (if provided) or set to <code>"model"</code> .
<code>include_paths</code>	(character vector) Paths to directories where Stan should look for files specified in <code>#include</code> directives.
<code>user_header</code>	(string) A path to a single C++ header file to prepend to the generated model code. A legacy alias for <code>external_cpp</code> ; supplied headers are appended to <code>external_cpp</code> and treated identically.

cpp_options	(list) C++ compilation options, merged into the Makevars flags used to compile the model. Each element is either: • A named element, e.g. <code>list(CXX = "g++")</code>. This <i>overrides</i> any value computed internally for that name (e.g. CXXFLAGS), replacing it rather than adding to it. • An unnamed string of the form "<code><NAME> = <value></code>", e.g. <code>"CXX = g++"</code>. Equivalent to the named form above – also overrides. • An unnamed string of the form "<code><NAME> += <value></code>", e.g. <code>"CXXFLAGS += -Wno-psabi"</code>. <i>Appends</i> <code><value></code> to any existing value for <code><NAME></code> instead of replacing it (space-separated), matching how <code>+=</code> behaves in a Makevars file. Entries are applied in list order, so the same name may appear more than once, e.g. <code>list(CXXFLAGS = "-O3", "CXXFLAGS += -Wall")</code> first overrides CXXFLAGS, then appends to it.
stanc_options	(list) Stan-to-C++ transpiler options. Not yet supported.
force_recompile	(logical) Should the model be recompiled even if it has not been modified? The default is FALSE, but can be set via the <code>stanr_force_recompile</code> option.
precompiled_headers	(logical) Should precompiled headers be used to speed up compilation? The default is TRUE. Automatically disabled when <code>cpp_options</code> overrides compiler flags (e.g. CXXFLAGS), since the precompiled header is not built with those flags.
quiet	(logical) Should verbose output from compilation be suppressed? The default is TRUE.
external_cpp	(character vector) Paths to C++ files to prepend to the generated model code. Useful for defining custom functions.
use_opengl	(logical) Should the model be compiled with OpenCL support? The default is FALSE. When TRUE, <code>stanc</code> generates OpenCL-accelerated code for the functions that support it (most notably <code>bernoulli_logit_glm</code> and other GLM likelihoods), and the compiled model can run its computation on an OpenCL device. Which platform/device is used at runtime is controlled by the <code>opengl_ids</code> argument of the fit methods (e.g. <code>\$sample()</code>), not by this argument; <code>opengl_ids = NULL</code> (the default there) means <code>select_opengl_device()</code> is simply never called, so the platform/device baked in at compile time (0/0) is used. The default OpenCL link flags are <code>"-framework OpenCL"</code> on macOS and <code>"-lOpenCL"</code> elsewhere.
compile_standalone	(logical) Should the Stan program's functions block be exposed as part of this compilation? The default is FALSE. When TRUE, <code>\$functions</code> is already populated when <code>stan_model()</code> returns – equivalent to FALSE here plus calling <code>\$expose_stan_functions()</code> immediately after, but without a second compile. See <code>\$expose_stan_functions()</code> for what gets exposed and how.
backend	(string) Either <code>"compiled"</code> (the default), which compiles the model to a native shared library, or <code>"stanli"</code> , which interprets the model instead of compiling it. The <code>"stanli"</code> backend supports constrained-scale initialization, but not <code>use_opengl</code> , <code>external_cpp</code> , or non-empty <code>cpp_options</code> . Its exposed-function

backend supports deterministic integer/real scalar and container functions, but not RNG, `_lp`, void, complex, or tuple functions.

Value

A `StanModel` object.

The compiled model is cached persistently on disk as the compiled shared library itself, next to `stan_file` (named after it, e.g. `mymodel.stan` gets a sibling `.mymodel.so/.mymodel.dll`), or under `tempdir()` when the model was created from a code string, or when `stan_file`'s directory isn't writable. The library embeds a hash of the generated C++, so it's reused across R sessions as long as the Stan program, `include_paths`, `external_cpp`, and installed `stanr`/`Stan` versions are unchanged, and is silently recompiled and overwritten otherwise – there is no separate cache-clearing step. On a cache hit, the Stan-to-C++ transpiler is skipped entirely, so any transpiler warnings (e.g. from pedantic mode or deprecated syntax) are only surfaced the first time a given model is compiled, not on subsequent cache hits.

See Also

`StanModel`, `$compile()`, `$sample()`

Examples

```
## Not run:
# Create a StanModel from Stan code
mod <- stan_model(
  code = "
    parameters {
      real theta;
    }
    model {
      theta ~ normal(0, 1);
    }
  "
)
mod$model_name()
mod$variables()

# Run MCMC sampling
fit <- mod$sample(data = list(), chains = 2)
fit$summary()

## End(Not run)
```

Description

Compiles a Stan program with the bundled stanc3 compiler. Before compilation, `#include` directives are replaced with the contents of the matching file from `include_directories`. Includes may be nested; directories are searched in the supplied order.

Usage

```
stanc(
  model_code,
  include_directories = character(),
  standalone_functions = FALSE,
  use_opengl = FALSE,
  optim_level = 0,
  allow_undefined = FALSE,
  warn_pedantic = FALSE,
  warn_uninitialized = FALSE,
  external_cpp = NULL
)
```

Arguments

<code>model_code</code>	Stan model code as a single string.
<code>include_directories</code>	Character vector of directories to search for Stan include files. Defaults to no include directories.
<code>standalone_functions</code>	Generate standalone function definitions.
<code>use_opengl</code>	Enable OpenGL code generation.
<code>optim_level</code>	stanc optimization level.
<code>allow_undefined</code>	Allow undefined functions.
<code>warn_pedantic</code>	Request pedantic compiler warnings.
<code>warn_uninitialized</code>	Request uninitialized-variable warnings.
<code>external_cpp</code>	NULL (the default), or a character vector of paths to C++ source or header files. Their contents are prepended to the generated C++, in order. Supplying this argument enables <code>allow_undefined</code> so Stan functions implemented by the supplied C++ can be declared in the model.

Value

A string containing the generated C++ model implementation.

StanDiagnose	<i>StanDiagnose objects</i>
--------------	-----------------------------

Description

A StanDiagnose object is returned by `$diagnose()` and contains the results of Stan's gradient checking diagnostics.

Methods

In addition to the methods inherited from `StanFit`, StanDiagnose objects have:

Method	Description
<code>\$gradients()</code>	Return the gradient check results.
<code>\$lp()</code>	Return the log probability evaluated at the initial parameter values.
<code>\$num_failed()</code>	Return the number of failed gradient checks.

StanFit	<i>StanFit objects</i>
---------	------------------------

Description

StanFit is the base class for fitted model objects returned by the inference methods of `StanModel`. Subclasses include `StanMCMC` (MCMC sampling), `StanMLE` (optimization), `StanLaplace` (Laplace approximation), `StanVB` (variational inference), `StanPathfinder` (Pathfinder), `StanGQ` (generated quantities), and `StanDiagnose` (gradient diagnostics).

Methods

All StanFit objects share the following methods:

Draws and summaries:

Method	Description
<code>\$draws()</code>	Extract posterior draws or point estimates.
<code>\$summary()</code>	Summarize posterior draws or point estimates.
<code>\$print()</code>	Print a summary of the fit.

Fit information:

Method	Description
<code>\$return_codes()</code>	Return the Stan return codes.

<code>\$metadata()</code>	Return fit metadata.
<code>\$output()</code>	Return Stan output messages.
<code>\$time()</code>	Return timing information.
<code>\$init()</code>	Return user-specified initial values.
<code>\$code()</code>	Return the Stan program code.

Log density:

Method	Description
<code>\$lp()</code>	Extract the log density (lp__) draws.

Model methods:

Method	Description
<code>\$log_prob()</code>	Compute the log probability.
<code>\$grad_log_prob()</code>	Compute the gradient of the log probability.
<code>\$hessian()</code>	Compute the Hessian of the log probability.
<code>\$constrain_variables()</code>	Constrain unconstrained parameters.
<code>\$unconstrain_variables()</code>	Unconstrain parameters.
<code>\$unconstrain_draws()</code>	Unconstrain posterior draws.
<code>\$variable_skeleton()</code>	Return a skeleton of the variable structure.

Function exposure:

Method	Description
<code>\$expose_stan_functions()</code>	Expose the model's functions block as R functions.
<code>\$functions</code>	Environment holding the exposed functions (shared with the model).

Saving:

Method	Description
<code>\$save_object()</code>	Save the fitted model object to a file.

StanGQ

StanGQ objects

Description

A StanGQ object is returned by `$generate_quantities()` and contains the generated quantities computed from posterior draws.

Methods

In addition to the methods inherited from [StanFit](#), StanGQ objects have:

Method	Description
\$num_chains()	Return the number of chains in the draws.

StanLaplace	<i>StanLaplace objects</i>
-------------	----------------------------

Description

A StanLaplace object is returned by [\\$laplace\(\)](#) and contains draws from a Gaussian approximation to the posterior centered at the mode.

Methods

In addition to the methods inherited from [StanFit](#), StanLaplace objects have:

Method	Description
\$mode()	Return the mode used for the approximation.
\$lp_approx()	Extract the log density approximation draws.

StanMCMC	<i>StanMCMC objects</i>
----------	-------------------------

Description

A StanMCMC object is returned by [\\$sample\(\)](#) and contains MCMC posterior draws, sampler diagnostics, and fit metadata.

Methods

In addition to the methods inherited from [StanFit](#), StanMCMC objects have:

Method	Description
\$sampler_diagnostics()	Extract sampler diagnostics.
\$num_chains()	Return the number of chains.
\$diagnostic_summary()	Summarize sampler diagnostics.
\$inv_metric()	Return the inverse mass matrix.

StanMLE

StanMLE objects

Description

A StanMLE object is returned by `$optimize()` and contains a point estimate (MAP or MLE) and optimization metadata.

Methods

In addition to the methods inherited from `StanFit`, StanMLE objects have:

Method	Description
<code>\$mle()</code>	Extract the point estimate as a numeric vector.

StanModel

StanModel objects

Description

A StanModel object is an `R6` object created by `stan_model()`. The object stores the Stan program source code and compiled model environment, and provides methods for fitting the model using Stan's inference algorithms.

Methods

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

Stan code:

Method	Description
<code>\$code()</code>	Return Stan program as a string.
<code>\$print()</code>	Print the Stan program.

Model information:

Method	Description
<code>\$model_name()</code>	Return the model name.
<code>\$stan_file()</code>	Return the path to the Stan file.
<code>\$has_stan_file()</code>	Check whether the model was created with a Stan file.
<code>\$include_paths()</code>	Return the Stan include paths.
<code>\$stan_version()</code>	Return the Stan version used by the package.

<code>\$variables()</code>	Return input and output variable information.
<code>\$cpp_options()</code>	Return the C++ options associated with the model.
<code>\$stanc_options()</code>	Return the stanc options associated with the model.

Compilation:

Method	Description
<code>\$check_syntax()</code>	Check the Stan program's syntax without compiling.
<code>\$format()</code>	Reformat the Stan program using stanc's auto-formatter.
<code>\$compile()</code>	Compile the Stan program.
<code>\$is_compiled()</code>	Check whether the model has been compiled.

Diagnostics:

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

Function exposure:

Method	Description
<code>\$expose_stan_functions()</code>	Expose the program's functions block as R functions.
<code>\$functions</code>	Environment holding the exposed functions.

Model fitting:

Method	Description
<code>\$sample()</code>	Run Stan's "sample" method (HMC/NUTS MCMC), return StanMCMC object.
<code>\$optimize()</code>	Run Stan's "optimize" method, return StanMLE object.
<code>\$laplace()</code>	Run Stan's "laplace" method, return StanLaplace object.
<code>\$variational()</code>	Run Stan's "variational" method (ADVI), return StanVB object.
<code>\$pathfinder()</code>	Run Stan's "pathfinder" method, return StanPathfinder object.
<code>\$generate_quantities()</code>	Run Stan's "generate quantities" method, return StanGQ object.

StanPathfinder

*StanPathfinder objects***Description**

A `StanPathfinder` object is returned by `$pathfinder()` and contains approximate posterior draws from the Pathfinder algorithm.

Methods

In addition to the methods inherited from [StanFit](#), StanPathfinder objects have:

Method	Description
\$lp_approx()	Extract the log density approximation draws.

StanVB	<i>StanVB objects</i>
--------	-----------------------

Description

A StanVB object is returned by [\\$variational\(\)](#) and contains approximate posterior draws from Automatic Differentiation Variational Inference (ADVI).

Methods

In addition to the methods inherited from [StanFit](#), StanVB objects have:

Method	Description
\$lp_approx()	Extract the log density approximation draws.

Index

* StanFit methods

- fit-method-diagnose, [3](#)
- fit-method-draws, [4](#)
- fit-method-expose-stan-functions,
[5](#)
- fit-method-fit-info, [5](#)
- fit-method-gq, [6](#)
- fit-method-laplace-mode, [7](#)
- fit-method-loo, [7](#)
- fit-method-lp, [8](#)
- fit-method-mcmc, [9](#)
- fit-method-mle, [10](#)
- fit-method-model-methods, [11](#)
- fit-method-print, [12](#)
- fit-method-save, [13](#)
- fit-method-summary, [13](#)

* StanModel methods

- model-method-check-syntax, [14](#)
- model-method-compile, [14](#)
- model-method-diagnose, [15](#)
- model-method-expose-stan-functions,
[16](#)
- model-method-format, [18](#)
- model-method-generate-quantities,
[19](#)
- model-method-laplace, [20](#)
- model-method-model-info, [22](#)
- model-method-optimize, [23](#)
- model-method-pathfinder, [25](#)
- model-method-sample, [27](#)
- model-method-variables, [30](#)
- model-method-variational, [31](#)

\$check_syntax(), [18](#), [41](#)

\$code(), [18](#), [38](#), [40](#)

\$compile(), [14](#), [23](#), [33](#), [35](#), [41](#)

\$constrain_variables(), [38](#)

\$cpp_options(), [41](#)

\$diagnose(), [37](#), [41](#)

\$diagnostic_summary(), [28](#), [39](#)

\$draws(), [4](#), [6](#), [7](#), [9](#), [10](#), [13](#), [24](#), [37](#)

\$expose_stan_functions(), [5](#), [15](#), [34](#), [38](#),
[41](#)

\$format(), [41](#)

\$functions, [38](#), [41](#)

\$generate_quantities(), [38](#), [41](#)

\$grad_log_prob(), [38](#)

\$gradients(), [37](#)

\$has_stan_file(), [40](#)

\$hessian(), [38](#)

\$include_paths(), [40](#)

\$init(), [38](#)

\$inv_metric(), [39](#)

\$is_compiled(), [15](#), [41](#)

\$laplace(), [25](#), [39](#), [41](#)

\$log_prob(), [38](#)

\$loo(), [10](#)

\$lp(), [37](#), [38](#)

\$lp_approx(), [39](#), [42](#)

\$metadata(), [38](#)

\$mle(), [24](#), [40](#)

\$mode(), [39](#)

\$model_name(), [40](#)

\$num_chains(), [39](#)

\$num_failed(), [37](#)

\$optimize(), [7](#), [10](#), [21](#), [22](#), [29](#), [40](#), [41](#)

\$output(), [38](#)

\$pathfinder(), [32](#), [41](#)

\$print(), [13](#), [37](#), [40](#)

\$return_codes(), [37](#)

\$sample(), [4](#), [11](#), [16](#), [20](#), [25](#), [26](#), [32](#), [34](#), [35](#),
[39](#), [41](#)

\$sampler_diagnostics(), [39](#)

\$save_object(), [38](#)

\$stan_file(), [18](#), [40](#)

\$stan_version(), [40](#)

\$stanc_options(), [41](#)

\$summary(), [4](#), [12](#), [37](#)

\$time(), [38](#)

- `$unconstrain_draws()`, 38
- `$unconstrain_variables()`, 38
- `$variable_skeleton()`, 38
- `$variables()`, 41
- `$variational()`, 22, 26, 29, 41, 42
- `base::saveRDS()`, 13
- `check_syntax`
 - `(model-method-check-syntax)`, 14
- `compile (model-method-compile)`, 14
- `diagnose (model-method-diagnose)`, 15
- `draws (fit-method-draws)`, 4
- `expose_functions`
 - `(model-method-expose-stan-functions)`, 16
- `expose_stan_functions`
 - `(model-method-expose-stan-functions)`, 16
- `fit-method-diagnose`, 3
- `fit-method-draws`, 4
- `fit-method-expose-stan-functions`, 5
- `fit-method-fit-info`, 5
- `fit-method-gq`, 6
- `fit-method-laplace-mode`, 7
- `fit-method-loo`, 7
- `fit-method-lp`, 8
- `fit-method-mcmc`, 9
- `fit-method-mle`, 10
- `fit-method-model-methods`, 8, 11
- `fit-method-print`, 12
- `fit-method-save`, 13
- `fit-method-summary`, 13
- `format (model-method-format)`, 18
- `generate_quantities`
 - `(model-method-generate-quantities)`, 19
- `laplace (model-method-laplace)`, 20
- `loo (fit-method-loo)`, 7
- `loo::loo.array()`, 7, 8
- `loo::loo_moment_match()`, 8
- `loo::loo_moment_match.default()`, 8
- `loo::relative_eff()`, 7
- `mle (fit-method-mle)`, 10
- `mode (fit-method-laplace-mode)`, 7
- `model-method-check-syntax`, 14
- `model-method-compile`, 14
- `model-method-diagnose`, 15
- `model-method-expose-stan-functions`, 16
- `model-method-format`, 18
- `model-method-generate-quantities`, 19
- `model-method-laplace`, 20
- `model-method-model-info`, 22
- `model-method-optimize`, 23
- `model-method-pathfinder`, 25
- `model-method-sample`, 27
- `model-method-variables`, 30
- `model-method-variational`, 31
- `moment-matching`, 8
- `optimize (model-method-optimize)`, 23
- `pathfinder (model-method-pathfinder)`, 25
- `posterior::as_draws_matrix()`, 19
- `posterior::summarise_draws()`, 13
- `print (fit-method-print)`, 12
- R6, 40
- `sample (model-method-sample)`, 27
- `stan_model`, 33
- `stan_model()`, 14, 15, 17, 18, 20–24, 26, 29, 32, 40
- `stanc`, 35
- `StanDiagnose`, 3, 15, 16, 37, 37, 41
- `StanFit`, 5, 17, 19, 37, 37, 39, 40, 42
- `StanGQ`, 6, 19, 20, 37, 38, 41
- `StanLaplace`, 8, 20, 21, 37, 39, 41
- `StanMCMC`, 9, 27, 29, 37, 39, 41
- `StanMLE`, 7, 10, 21, 23, 24, 37, 40, 41
- `StanModel`, 5, 14, 15, 17–20, 22, 23, 25, 27, 30, 31, 33, 35, 37, 40
- `StanPathfinder`, 8, 25, 26, 37, 41, 41
- `stanr (stanr-package)`, 3
- `stanr-package`, 3
- `StanVB`, 8, 31, 32, 37, 41, 42
- `summary (fit-method-summary)`, 13
- `tempdir()`, 35
- `variables (model-method-variables)`, 30
- `variational (model-method-variational)`, 31