Package 'shinystan'

October 14, 2022

Title Interactive Visual and Numerical Diagnostics and Posterior Analysis for Bayesian Models

Version 2.6.0

Date 2022-03-02

Maintainer Jonah Gabry <`jsg2201@columbia.edu`>

Description A graphical user interface for interactive Markov chain Monte Carlo (MCMC) diagnostics and plots and tables helpful for analyzing a posterior sample. The interface is powered by the 'Shiny' web application framework from 'RStudio' and works with the output of MCMC programs written in any programming language (and has extended functionality for 'Stan' models fit using the 'rstan' and 'rstanarm' packages).

URL `https://mc-stan.org/shinystan/`, `https://discourse.mc-stan.org`

BugReports `https://github.com/stan-dev/shinystan/issues/`

License GPL (>= 3)

Depends R (>= 3.1.0), shiny (>= 1.0.3)

Imports bayesplot (>= 1.5.0), colourpicker, DT (>= 0.2), dygraphs (>= 1.1.1.2), ggplot2 (>= 2.1.1), gridExtra, gtools, markdown (>= 0.7.4), methods, reshape2, rstan (>= 2.17.1), stats, shinyjs (>= 0.6.0), shinythemes (>= 1.0.1), threejs (>= 0.2.1), utils, xtable, xts (>= 0.9-7)

Suggests cmdstanr (>= 0.4.0), coda, knitr (>= 1.9), posterior (>= 1.0.0), rmarkdown (>= 0.8.1), rsconnect (>= 0.4.2), rstanarm (>= 2.17.4), testthat

Additional_repositories `https://mc-stan.org/r-packages/`

LazyData true

Encoding UTF-8

VignetteBuilder knitr

RoxygenNote 7.1.2

NeedsCompilation no

Author Jonah Gabry [aut, cre],
Duco Veen [aut],
Stan Development Team [ctb],
Michael Andreae [ctb],
Michael Betancourt [ctb],
Bob Carpenter [ctb],
Yuanjun Gao [ctb],
Andrew Gelman [ctb],
Ben Goodrich [ctb],
Daniel Lee [ctb],
Dongying Song [ctb],
Rob Trangucci [ctb]

Repository CRAN

Date/Publication 2022-03-03 00:10:02 UTC

R topics documented:

`shinystan-package` **shinystan** *R package ('ShinyStan' graphical user interface)*

Description

Applied Bayesian data analysis is primarily implemented through the Markov chain Monte Carlo (MCMC) algorithms offered by various software packages. When analyzing a posterior sample obtained by one of these algorithms the first step is to check for signs that the chains have converged to the target distribution and and also for signs that the algorithm might require tuning or might be ill-suited for the given model. There may also be theoretical problems or practical inefficiencies with the specification of the model. The ShinyStan app provides interactive plots and tables helpful for analyzing a posterior sample, with particular attention to identifying potential problems with the performance of the MCMC algorithm or the specification of the model. ShinyStan is powered by the Shiny web application framework by RStudio (`https://shiny.rstudio.com/`) and works with the output of MCMC programs written in any programming language (and has extended functionality for models fit using the **rstan** package and the No-U-Turn sampler).

ShinyStan has extended functionality for Stan models

Stan (https://mc-stan.org/) models can be run in R using the **rstan** package. Other packages like **rstanarm** and **brms** provide higher-level interfaces to Stan that use **rstan** internally.

Saving and sharing

The **shinystan** package allows you to store the basic components of an entire project (code, posterior samples, graphs, tables, notes) in a single object, a `shinystan object` (sso, for short). Users can save many of the plots as **ggplot2** objects for further customization and easy integration in reports or post-processing for publication.

The `deploy_shinystan` function lets you easily deploy your own ShinyStan apps online for any of your models using the shinyapps.io service from 'RStudio'. Each of your apps (each of your models) will have a unique url and will be compatible with most web browsers.

License

The **shinystan** package is open source licensed under the GNU Public License, version 3 (GPLv3).

Demo

Check out the demo using `launch_shinystan_demo` or try it with one of your own models using `launch_shinystan`.

Resources

- Web page with online documentation (https://mc-stan.org/shinystan/)
- Stan Forums on Discourse (https://discourse.mc-stan.org)
- GitHub issue tracker (https://github.com/stan-dev/shinystan/issues)

References

Muth, C., Oravecz, Z., and Gabry, J. (2018) User-friendly Bayesian regression modeling: A tutorial with rstanarm and shinystan. *The Quantitative Methods for Psychology*. 14(2), 99–119. https://www.tqmp.org/RegularArticles/vol14-2/p099/p099.pdf

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378 ([journal version](https://rss.onlinelibrary.wiley.com/doi/full/10.1111/rssa.12378), [preprint arXiv:1709.01449](https://arxiv.org/abs/170
[code on GitHub](https://github.com/jgabry/bayes-vis-paper))

See Also

`as.shinystan` for creating `shinystan` objects.

`launch_shinystan_demo` to try a demo.

`launch_shinystan` to launch the 'ShinyStan' interface using a particular `shinystan` object.

as.shinystan *Create and test* shinystan *objects*

Description

The as.shinystan function creates shinystan objects that can be used with launch_shinystan and various other functions in the **shinystan** package. as.shinystan is a generic for which the **shinystan** package provides several methods. Currently methods are provided for creating shinystan objects from arrays, lists of matrices, stanfit objects (**rstan**), stanreg objects (**rstanarm**), and mcmc.list objects (**coda**).

is.shinystan tests if an object is a shinystan object.

Usage

```
as.shinystan(X, ...)

is.shinystan(X)

## S4 method for signature 'array'
as.shinystan(
  X,
  model_name = "unnamed model",
  warmup = 0,
  burnin = 0,
  param_dims = list(),
  model_code = NULL,
  note = NULL,
  sampler_params = NULL,
  algorithm = NULL,
  max_treedepth = NULL,
  ...
)

## S4 method for signature 'list'
as.shinystan(
  X,
  model_name = "unnamed model",
  warmup = 0,
  burnin = 0,
  param_dims = list(),
  model_code = NULL,
  note = NULL,
  sampler_params = NULL,
  algorithm = NULL,
  max_treedepth = NULL,
  ...
)

## S4 method for signature 'mcmc.list'
as.shinystan(
  X,
  model_name = "unnamed model",
  warmup = 0,
  burnin = 0,
  param_dims = list(),
  model_code = NULL,
  note = NULL,
  ...
)

## S4 method for signature 'stanfit'
as.shinystan(X, pars, model_name = X@model_name, note = NULL, ...)

## S4 method for signature 'stanreg'
as.shinystan(X, ppd = TRUE, seed = 1234, model_name = NULL, note = NULL, ...)

## S4 method for signature 'CmdStanMCMC'
as.shinystan(X, pars = NULL, model_name = NULL, note = NULL, ...)
```

Arguments

`X`	For `as.shinystan`, an object to be converted to a `shinystan` object. See the Methods section below. For `is.shinystan`, an object to check.
`...`	Arguments passed to the individual methods.
`model_name`	A string giving a name for the model.
`warmup`	The number of iterations to treat as warmup. Should be `0` if warmup iterations are not included in `X`.
`burnin`	Deprecated. Use `warmup` instead. The `burnin` argument will be removed in a future release.
`param_dims`	Rarely used and never necessary. A named list giving the dimensions for all parameters. For scalar parameters use `0` as the dimension. See Examples.
`model_code`	Optionally, a character string with the code used to run the model. This can also be added to your `shinystan` object later using the `model_code` function. See `model_code` for additional formatting instructions. After launching the app the code will be viewable in the **Model Code** tab. For `stanfit` (**rstan**) and `stanreg` (**rstanarm**) objects the model code is automatically retrieved from the object.
`note`	Optionally, text to display on the **Notepad** page in the 'ShinyStan' GUI (stored in `user_model_info` slot of the `shinystan` object).
`sampler_params, algorithm, max_treedepth`	Rarely used and never necessary. If using the `as.shinystan` method for arrays or lists, these arguments can be used to manually provide information that is automatically retrieved from a stanfit object when using the `as.shinystan`

	method for stanfit objects. If specified, sampler_params must have the same structure as an object returned by get_sampler_params (**rstan**), which is a list of matrices, with one matrix per chain. algorithm, if specified, must be either "NUTS" or "HMC" (static HMC). If algorithm is "NUTS" then max_treedepth (an integer indicating the maximum allowed treedepth when the model was fit) must also be provided.
pars	For stanfit objects (**rstan**), an optional character vector specifying which parameters should be included in the shinystan object.
ppd	For stanreg objects (**rstanarm**), ppd (logical) indicates whether to draw from the posterior predictive distribution before launching the app. The default is TRUE, although for very large objects it can be convenient to set it to FALSE as drawing from the posterior predictive distribution can be time consuming. If ppd is TRUE then graphical posterior predictive checks are available when 'ShinyStan' is launched.
seed	Passed to pp_check (**rstanarm**) if ppd is TRUE.

Value

as.shinystan returns a shinystan object, which is an instance of S4 class "shinystan".

is.shinystan returns TRUE if the tested object is a shinystan object and FALSE otherwise.

Functions

- as.shinystan,array-method: Create a shinystan object from a 3-D array of simulations. The array should have dimensions corresponding to iterations, chains, and parameters, in that order.
- as.shinystan,list-method: Create a shinystan object from a list of matrices. Each matrix (or 2-D array) should contain the simulations for an individual chain and all of the matrices should have the same number of iterations (rows) and parameters (columns). Parameters should have the same names and be in the same order.
- as.shinystan,mcmc.list-method: Create a shinystan object from an mcmc.list object (**coda**).
- as.shinystan,stanfit-method: Create a shinystan object from a stanfit object (**rstan**). Fewer optional arguments are available for this method because all important information can be taken automatically from the stanfit object.
- as.shinystan,stanreg-method: Create a shinystan object from a stanreg object (**rstanarm**).
- as.shinystan,CmdStanMCMC-method: Create a shinystan object from a CmdStanMCMC object (**cmdstanr**).

See Also

launch_shinystan to launch the 'ShinyStan' interface using a particular shinystan object.

drop_parameters to remove parameters from a shinystan object.

generate_quantity to add a new quantity to a shinystan object.

Examples

```
## Not run:
sso <- as.shinystan(X, ...) # replace ... with optional arguments or omit it
launch_shinystan(sso)

## End(Not run)

## Not run:
########################
### list of matrices ###
########################

# Generate some fake data
chain1 <- cbind(beta1 = rnorm(100), beta2 = rnorm(100), sigma = rexp(100))
chain2 <- cbind(beta1 = rnorm(100), beta2 = rnorm(100), sigma = rexp(100))
sso <- as.shinystan(list(chain1, chain2))
launch_shinystan(sso)

# We can also specify some or all of the optional arguments
# note: in order to use param_dims we need to rename 'beta1' and 'beta2'
# to 'beta[1]' and 'beta[2]'
colnames(chain1) <- colnames(chain2) <- c(paste0("beta[",1:2,"]"), "sigma")
sso2 <- as.shinystan(list(chain1, chain2),
                     model_name = "Example", warmup = 0,
                     param_dims = list(beta = 2, sigma = 0))
launch_shinystan(sso2)

## End(Not run)

## Not run:
######################
### stanfit object ###
######################
library("rstan")
fit <- stan_demo("eight_schools")
sso <- as.shinystan(fit, model_name = "example")

## End(Not run)

## Not run:
######################
### stanreg object ###
######################
library("rstanarm")
example("example_model")
sso <- as.shinystan(example_model)
launch_shinystan(sso)

## End(Not run)
```

`deploy_shinystan` *Deploy a 'ShinyStan' app on the web using 'shinyapps.io' by 'RStudio'*

Description

Requires a (free or paid) 'ShinyApps' account. Visit `https://www.shinyapps.io/` to sign up.

Usage

```
deploy_shinystan(sso, appName, account = NULL, ..., deploy = TRUE)
```

Arguments

`sso`	A `shinystan object`.
`appName`	The name to use for the application. Application names must be at least four characters long and may only contain letters, numbers, dashes and underscores.
`account`	shinyapps.io account username. Only required if more than one account is configured on the system.
`...`	Optional arguments. See Details.
`deploy`	Should the app be deployed? The only reason for this to be `FALSE` is if you just want to check that the preprocessing before deployment is successful.

Details

In `...`, the arguments `ppcheck_data` and `ppcheck_yrep` can be specified. `ppcheck_data` should be a vector of observations to use for graphical posterior predictive checking and `ppcheck_yrep` should be a character string naming the parameter in `sso` containing the posterior predictive simulations/replications. The value of `ppcheck_yrep` is only used to preselect the appropriate parameter/generated quantity to use for the posterior predictive checking. `ppcheck_yrep` (but not `ppcheck_data`) can also be set interactively on shinyapps.io when using the app.

Value

Invisibly, `TRUE` if deployment succeeded (did not encounter an error) or, if `deploy` argument is set to `FALSE`, the path to the temporary directory containing the app ready for deployment (also invisibly).

See Also

The example in the *Deploying to shinyapps.io* vignette that comes with this package.

`https://www.shinyapps.io/` to sign up for a free or paid 'ShinyApps' account and for details on how to configure your account on your local system using the **rsconnect** package from 'RStudio'.

Examples

```
## Not run:
# For this example assume sso is the name of the \code{shinystan} object for
# the model you want to use. Assume also that you want to name your app
# 'my-model' and that your shinyapps.io username is 'username'.

deploy_shinystan(sso, appName = "my-model", account = "username")

# If you only have one ShinyApps account configured then you can also omit
# the 'account' argument.

deploy_shinystan(sso, appName = "my-model")

## End(Not run)
```

`drop_parameters` *Drop parameters from a* `shinystan` *object*

Description

Remove selected parameters from a `shinystan` object. This is useful if you have a very large `shinystan` object when you only want to look at a subset of parameters. With a smaller `shinystan` object, `launch_shinystan` will be faster and you should experience better performance (responsiveness) after launching when using the 'ShinyStan' app.

Usage

```
drop_parameters(sso, pars)
```

Arguments

`sso`	A `shinystan object`.
`pars`	A character vector of parameter names. If the name of a non-scalar (e.g. vector, matrix) parameter is included in `pars` all of its elements will be removed. Currently it is not possible to remove only a subset of the elements of a non-scalar parameter.

Value

`sso`, with `pars` dropped.

See Also

`generate_quantity` to add a new quantity to a `shinystan` object.

Examples

```
# Using example shinystan object 'eight_schools'
print(eight_schools@param_names)

# Remove the scalar parameters mu and tau
sso <- drop_parameters(eight_schools, pars = c("mu", "tau"))
print(sso@param_names)

# Remove all elements of the parameter vector theta
sso <- drop_parameters(sso, pars = "theta")
print(sso@param_names)
```

---

generate_quantity *Add new quantity to shinystan object*

Description

Add to shinystan object a new parameter as a function of one or two existing parameters.

Usage

```
generate_quantity(sso, param1, param2, fun, new_name)
```

Arguments

sso	A shinystan object.
param1	Name of first parameter as character string.
param2	Optional. Name of second parameter as character string.
fun	Function to call, i.e. `function(param1)` or `function(param1,param2)`. See Examples, below.
new_name	Name for the new parameter as character string.

Value

sso, updated. See Examples.

See Also

`drop_parameters` to remove parameters from a `shinystan` object.

Examples

```
# Using example shinystan object 'eight_schools'
sso <- eight_schools
sso <- generate_quantity(sso, fun = function(x) x^2,
                         param1 = "tau", new_name = "tau_sq")
sso <- generate_quantity(sso, fun = "-",
                         param1 = "theta[1]", param2 = "theta[2]",
                         new_name = "theta1minus2")
```

`launch_shinystan` *Launch the 'ShinyStan' app*

Description

Launch the 'ShinyStan' app in the default web browser. 'RStudio' users also have the option of launching the app in the pop-up Viewer.

Usage

```
launch_shinystan(object, ...)

## Default S3 method:
launch_shinystan(object, ..., rstudio = getOption("shinystan.rstudio"))

## S3 method for class 'shinystan'
launch_shinystan(object, ..., rstudio = getOption("shinystan.rstudio"))
```

Arguments

`object`	The object to use. For the default method this can be an object of class `"shinystan"`, `"stanfit"`, or `"stanreg"`. To use other types of objects first create a shinystan object using `as.shinystan`.
`...`	Optional arguments passed to `runApp`.
`rstudio`	Only relevant for 'RStudio' users. The default (`FALSE`) is to launch the app in the user's default web browser rather than the pop-up Viewer provided by 'RStudio'. Users can change the default to `TRUE` by setting the global option `options(shinystan.rstudio = TRUE)`.

Value

The `launch_shinystan` function is used for the side effect of starting the 'ShinyStan' app, but it also returns a `shinystan` object, an instance of S4 class `"shinystan"`.

References

Muth, C., Oravecz, Z., and Gabry, J. (2018) User-friendly Bayesian regression modeling: A tutorial with rstanarm and shinystan. *The Quantitative Methods for Psychology*. 14(2), 99–119. `https://www.tqmp.org/RegularArticles/vol14-2/p099/p099.pdf`

Gabry, J. , Simpson, D. , Vehtari, A. , Betancourt, M. and Gelman, A. (2019), Visualization in Bayesian workflow. *J. R. Stat. Soc. A*, 182: 389-402. doi:10.1111/rssa.12378 ([journal version](https://rss.onlinelibrary.wiley.com/doi/full/10.1111/rssa.12378), [preprint arXiv:1709.01449](https://arxiv.org/abs/1709
[code on GitHub](https://github.com/jgabry/bayes-vis-paper))

See Also

`as.shinystan` for creating `shinystan` objects.

`update_sso` to update a `shinystan` object created by a previous version of the package.

`launch_shinystan_demo` to try a demo.

Examples

```
## Not run:
########################################
# Example 1: 'sso' is a shinystan object
########################################

# Just launch shinystan
launch_shinystan(sso)

# Launch shinystan and replace sso with an updated version of itself
# if any changes are made to sso while using the app
sso <- launch_shinystan(sso)

# Launch shinystan but save any changes made to sso while running the app
# in a new shinystan object sso2. sso will remained unchanged.
sso2 <- launch_shinystan(sso)

########################################
# Example 2: 'sf' is a stanfit object
########################################

# Just launch shinystan
launch_shinystan(sf)

# Launch shinystan and save the resulting shinystan object
sf_sso <- launch_shinystan(sf)

# Now sf_sso is a shinystan object and so Example 1 (above) applies when
# using sf_sso.

########################################
# Example 3: 'fit' is an mcmc.list, array or list of matrices
########################################
```

```
# First create shinystan object (see ?as.shinystan) for full details)

## End(Not run)
```

`launch_shinystan_demo` *'ShinyStan' demo*

Description

'ShinyStan' demo

Usage

```
launch_shinystan_demo(
  demo_name = "eight_schools",
  rstudio = getOption("shinystan.rstudio"),
  ...
)
```

Arguments

`demo_name`	The name of the demo. Currently `"eight_schools"` is the only option, but additional demos may be available in future releases. `eight_schools` Hierarchical meta-analysis model. See *Meta Analysis* chapter of the 'Stan' manual `https://mc-stan.org/users/documentation/`.
`rstudio`	Only relevant for 'RStudio' users. The default (`FALSE`) is to launch the app in the user's default web browser rather than the pop-up Viewer provided by 'RStudio'. Users can change the default to `TRUE` by setting the global option `options(shinystan.rstudio = TRUE)`.
`...`	Optional arguments passed to `runApp`.

Value

An S4 shinystan object.

See Also

`launch_shinystan` to launch the 'ShinyStan' interface using a particular `shinystan` object.

`as.shinystan` for creating `shinystan` objects.

Examples

```
## Not run:
# launch demo but don't save a shinystan object
launch_shinystan_demo()

# launch demo and save the shinystan object for the demo
sso_demo <- launch_shinystan_demo()

## End(Not run)
```

---

retrieve *Get summary statistics from shinystan object*

Description

From a shinystan object get rhat, effective sample size, posterior quantiles, means, standard deviations, sampler diagnostics, etc.

Usage

```
retrieve(sso, what, ...)
```

Arguments

`sso`	A `shinystan object`.
`what`	What do you want to get? See Details, below.
`...`	Optional arguments, in particular `pars` to specify parameter names (by default all parameters will be used). For NUTS sampler parameters only (e.g. stepsize, treedepth) `inc_warmup` can also be specified to include/exclude warmup iterations (the default is `FALSE`). See Details, below.

Details

The argument `what` can take on the values below. 'Args: `arg`' means that `arg` can be specified in `...` for this value of `what`.

`"rhat"`, `"Rhat"`, `"r_hat"`, or `"R_hat"` returns: Rhat statistics. Args: `pars`

`"N_eff"`, `"n_eff"`, `"neff"`, `"Neff"`, `"ess"`, or `"ESS"` returns: Effective sample sizes. Args: `pars`

`"mean"` returns: Posterior means. Args: `pars`

`"sd"` returns: Posterior standard deviations. Args: `pars`

`"se_mean"` or `"mcse"` returns: Monte Carlo standard error. Args: `pars`

`"median"` returns: Posterior medians. Args: `pars`.

`"quantiles"` or any string with `"quant"` in it (not case sensitive) returns: 2.5%, 25%, 50%, 75%, 97.5% posterior quantiles. Args: `pars`.

`"avg_accept_stat"` or any string with `"accept"` in it (not case sensitive) returns: Average value of "accept_stat" (which itself is the average acceptance probability over the NUTS subtree). Args: `inc_warmup`

`"prop_divergent"` or any string with `"diverg"` in it (not case sensitive) returns: Proportion of divergent iterations for each chain. Args: `inc_warmup`

`"max_treedepth"` or any string with `"tree"` or `"depth"` in it (not case sensitive) returns: Maximum treedepth for each chain. Args: `inc_warmup`

`"avg_stepsize"` or any string with `"step"` in it (not case sensitive) returns: Average stepsize for each chain. Args: `inc_warmup`

Note

Sampler diagnostics (e.g. `"avg_accept_stat"`) only available for models originally fit using Stan.

Examples

```
# Using example shinystan object 'eight_schools'
sso <- eight_schools
retrieve(sso, "rhat")
retrieve(sso, "mean", pars = c('theta[1]', 'mu'))
retrieve(sso, "quantiles")
retrieve(sso, "max_treedepth")  # equivalent to retrieve(sso, "depth"), retrieve(sso, "tree"), etc.
retrieve(sso, "prop_divergent")
retrieve(sso, "prop_divergent", inc_warmup = TRUE)
```

`shinystan-class` *S4* `shinystan` *objects*

Description

See `as.shinystan` for documentation on creating `shinystan` objects and `eight_schools` for an example object.

Slots

`model_name` `("character")` Model name.

`param_names` `("character")` Parameter names.

`param_dims` `("list")` Parameter dimensions.

`posterior_sample` `("array")` MCMC sample.

`summary` `("matrix")` Summary stats for `posterior_sample`.

`sampler_params` `("list")` Sampler parameters (for certain Stan models only).

`n_chain` `("integer")` Number of chains.

`n_iter` `("integer")` Number of iterations per chain.

`n_warmup` `("integer")` Number of warmup iterations per chain.

`user_model_info` (`"character"`) Notes to display on the **Notepad** page in the 'ShinyStan' GUI.

`model_code` (`"character"`) Model code to display on the **Model Code** page in the 'ShinyStan' GUI.

`misc` (`"list"`) Miscellaneous, for internal use.

References

Muth, C., Oravecz, Z., and Gabry, J. (2018) User-friendly Bayesian regression modeling: A tutorial with rstanarm and shinystan. *The Quantitative Methods for Psychology*. 14(2), 99–119. https://www.tqmp.org/RegularArticles/vol14-2/p099/p099.pdf

See Also

`as.shinystan` for creating `shinystan` objects.

`drop_parameters` to remove parameters from a `shinystan` object.

`generate_quantity` to add a new quantity to a `shinystan` object.

`shinystan-metadata` to view or change metadata associated with a `shinystan` object.

---

shinystan-metadata — *View or change metadata associated with a* `shinystan` *object*

Description

View or change metadata associated with a `shinystan` object

Usage

```
sso_info(sso)

model_code(sso, code = NULL)

notes(sso, note = NULL, replace = FALSE)

model_name(sso, name = NULL)
```

Arguments

`sso`	A `shinystan object`.
`code`	A string, containing model code to be added, that can be used as an argument to `cat`. See **Examples**.
`note`	A string containing a note to add to any existing notes or replace existing notes, depending on the value of `replace`.
`replace`	If `TRUE` the existing notes are overwritten by `note` if `note` is specified. If `FALSE` (the default) if `note` is specified then its content is appended to the existing notes.
`name`	A string giving the new model name to use.

Value

sso_info prints basic metadata including number of parameters, chains, iterations, warmup iterations, etc. It does not return anything.

model_code returns or replaces model code stored in a shinystan object. If code is NULL then any existing model code stored in sso is returned as a character string. If code is specified then an updated shinystan object is returned with code added. For shinystan objects created from stanfit (**rstan**) and stanreg (**rstanarm**) objects, model code is automatically taken from that object and does not need to be added manually. From within the 'ShinyStan' interface model code can be viewed on the **Model Code** page.

notes returns, amends, or replaces notes stored in a shinystan object. If note is NULL then any existing notes stored in sso are returned as a character string. If note is specified then an updated shinystan object is returned with either note added to the previous notes (if replace=FALSE) or overwritten by note (if replace = TRUE). From within the 'ShinyStan' interface, notes are viewable on the **Notepad** page.

model_name returns or replaces the model name associated with a shinystan object. If name is NULL then the current model name is returned. If name is specified then sso is returned with an updated model name.

See Also

as.shinystan for creating shinystan objects.

drop_parameters to remove parameters from a shinystan object.

generate_quantity to add a new quantity to a shinystan object.

Examples

```
# use eight_schools example object
sso <- eight_schools

################
### sso_info ###
################

sso_info(sso)

##################
### model_code ###
##################

# view model code in example shinystan object 'eight_schools'
cat(model_code(sso))

# change the model code in sso
# some jags style code
my_code <- "
 model {
   for (i in 1:length(Y)) {
     Y[i] ~ dpois(lambda[i])
     log(lambda[i]) <- inprod(X[i,], theta[])
```

```
    }
    for (j in 1:J) {
      theta[j] ~ dt(0.0, 1.0, 1.0)
    }
  }
"
sso <- model_code(sso, my_code)
cat(model_code(sso))

#############
### notes ###
#############

# view existing notes
notes(sso)

# add a note to the existing notes
sso <- notes(sso, "New note")
notes(sso)
cat(notes(sso))

# replace existing notes
sso <- notes(sso, "replacement note", replace = TRUE)
notes(sso)

##################
### model_name ###
##################

# view model name
model_name(sso)

# change model name
sso <- model_name(sso, "some other name")
identical(model_name(sso), "some other name")
```

`update_sso` *Update an object created by the previous version of shinystan*

Description

If you encounter any errors when using a shinystan object (`sso`) created by a previous version of **shinystan**, you might need to run `update_sso`. If `update_sso` does not resolve the problem and you still have the object (e.g. stanfit, stanreg, mcmc.list) from which `sso` was originally created, you can create a new shinystan object using `as.shinystan`.

Usage

```
update_sso(sso)
```

Arguments

`sso`	A `shinystan object`.

Value

If `sso` is already compatible with your version of **shinystan** then `sso` itself is returned and a message is printed indicating that `sso` is already up-to-date. Otherwise an updated version of `sso` is returned unless an error is encountered.

See Also

`as.shinystan` for creating `shinystan` objects.

Examples

```
## Not run:
sso_new <- update_sso(sso)

## End(Not run)
```

Index