Package ‘chkptstanr’

October 12, 2022

Title Checkpoint MCMC Sampling with 'Stan'

Version 0.1.1

Date 2022-04-27

Description Fit Bayesian models in Stan <doi:10.18637/jss.v076.i01>
with checkpointing, that is, the ability to stop the MCMC sampler at
will, and then pick right back up where the MCMC sampler left off.
Custom 'Stan' models can be fitted, or the popular package 'brms'
<doi:10.18637/jss.v080.i01> can be used to generate the 'Stan' code. This
package is fully compatible with the R packages 'brms', 'posterior', 'cmdstanr',
and 'bayesplot'.

License Apache License 2.0 | file LICENSE

Depends R (>= 4.1.0)

Imports brms (>= 2.16.1), abind, methods, Rdpack, rstan

Suggests cmdstanr, rmarkdown, knitr, posterior

Encoding UTF-8

RoxygenNote 7.1.2

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

RdMacros Rdpack

VignetteBuilder knitr

NeedsCompilation no

Author Donald Williams [aut, cre],
Tyler Matta [aut],
NWEA [cph]

Maintainer Donald Williams <`donald.williams@nwea.org`>

Repository CRAN

Date/Publication 2022-04-29 15:30:02 UTC

R topics documented:

`chkptstanr-package` *chkptstanr: Checkpoint MCMC Sampling with 'Stan'*

Description

Fit Bayesian models in **Stan** (Carpenter et al. 2017) with checkpointing, that is, the ability to stop the MCMC sampler at will, and then pick right back up where the MCMC sampler left off. Custom **Stan** models can be fitted, or the popular package **brms** (Bürkner 2017) can be used to generate the **Stan** code. This package is fully compatible with the R packages **brms**, **posterior**, **cmdstanr**, and **bayesplot**.

There are a variety of use cases for **chkptstanr**, including (but not limited to) the following:

- The primary motivation for developing **chkptstanr** is to reduce the cost of fitting models with **Stan** when using, say, AWS, and in particular by taking advantage of so-called *spot instances*. These instances are "a cost-effective choice if you can be flexible about when your applications run and if your applications can be *interrupted* [emphasis added]" (AWS website).

  chkptstanr thus allows for taking advantage of spot instances by enabling "interruptions" during model fitting. This can reduce the cost by 90 %.

- **Stan** allows for fitting complex models. This often entails iteratively improving the model to ensure that the MCMC algorithm has converged. Typically this requires waiting until the model has *finished sampling*, and then assessing MCMC diagnostics (e.g., R-hat).

  chkptstanr can be used to make iterative model building more efficient, e.g., by having the ability to pause sampling and examine the model (e.g., convergence diagnostics), and then deciding to stop sampling or to continue on.

- Computationally intensive models can sometimes take several days to finish up. When using a personal computer, this can take up all the computing resources.

  chkptstanr can be used with scheduling, such that the model is fitted during certain windows (e.g., at night, weekends, etc.)

- Those familiar with Bayesian methods will know all too well that a model can take longer than expected. This can be problematic when there is another task that needs to be completed, because one is faced with waiting it out or stopping the model (and loosing all of the progress). **chkptstanr** makes it so that models can be conveniently stopped if need be, while not loosing any of the progress.

References

Bürkner P (2017). “brms: An R package for Bayesian multilevel models using Stan.” *Journal of statistical software*, **80**, 1–28.

Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). “Stan: A probabilistic programming language.” *Journal of statistical software*, **76**(1).

`chkpt_brms` *Checkpoint Sampling: brms*

Description

Fit Bayesian generalized (non-)linear multivariate multilevel models using brms with checkpointing.

Usage

```
chkpt_brms(
  formula,
  data,
  iter_warmup = 1000,
  iter_sampling = 1000,
  iter_per_chkpt = 100,
  iter_typical = 150,
  parallel_chains = 2,
  threads_per = 1,
  chkpt_progress = TRUE,
  control = NULL,
  brmsfit = TRUE,
  seed = 1,
  path,
  ...
)
```

Arguments

`formula`	An object of class `formula`, `brmsformula`, or `brms`{`mvbrmsformula`}. Further information can be found in `brmsformula`.

`data`	An object of class `data.frame` (or one that can be coerced to that class) containing data of all variables used in the model.
`iter_warmup`	(positive integer) The number of warmup iterations to run per chain (defaults to 1000).
`iter_sampling`	(positive integer) The number of post-warmup iterations to run per chain (defaults to 1000).
`iter_per_chkpt`	(positive integer). The number of iterations per checkpoint. Note that `iter_sampling` is divided by `iter_per_chkpt` to determine the number of checkpoints. This must result in an integer (if not, there will be an error).
`iter_typical`	(positive integer) The number of iterations in the initial warmup, which finds the so-called typical set. This is an initial phase, and not included in `iter_warmup`. Note that a large enough value is required to ensure convergence (defaults to 150).
`parallel_chains`	(positive integer) The *maximum number* of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option `mc.cores`, which can be set for an entire R session by `options(mc.cores=value)`. If the `mc.cores` option has not been set then the default is `1`.
`threads_per`	(positive integer) Number of threads to use in within-chain parallelization (defaults to `1`).
`chkpt_progress`	logical. Should the `chkptstanr` progress be printed (defaults to `TRUE`) ? If set to `FALSE`, the standard `cmdstanr` progress bar is printed for each checkpoint (which does not actually keep track of checkpointing progress)
`control`	A named list of parameters to control the sampler's behavior. It defaults to NULL so all the default values are used. For a comprehensive overview see `stan`.
`brmsfit`	Logical. Should a `brmsfit` object be returned (defaults to `TRUE`).
`seed`	(positive integer). The seed for random number generation to make results reproducible.
`path`	Character string. The path to the folder, that is used for saving the checkpoints.
`...`	Additional arguments based to `make_stancode`, including, for example, user-defined prior distributions and the `brmsfamily` (e.g., `family = poisson()`).

Value

An object of class `brmsfit` (with `brmsfit = TRUE`) or `chkpt_brms` (with `brmsfit = FALSE`)

Examples

```
## Not run:
library(brms)
library(cmdstanr)

# path for storing checkpoint info
path <- create_folder(folder_name  = "chkpt_folder_fit1")
```

```
# "random" intercept
fit1 <- chkpt_brms(bf(formula = count ~ zAge + zBase * Trt + (1|patient),
                      family = poisson()),
                   data = epilepsy, ,
                   iter_warmup = 1000,
                   iter_sampling = 1000,
                   iter_per_chkpt = 250,
                   path = path)

# brmsfit output
fit1

# path for storing checkpoint info
 path <- create_folder(folder_name  = "chkpt_folder_fit2")

# remove "random" intercept (for model comparison)
fit2 <- chkpt_brms(bf(formula = count ~ zAge + zBase * Trt,
                      family = poisson()),
                   data = epilepsy, ,
                   iter_warmup = 1000,
                   iter_sampling = 1000,
                   iter_per_chkpt = 250,
                   path = path)

# brmsfit output
fit2

# compare models
loo(fit1, fit2)


# using custom priors
path <- create_folder(folder_name = "chkpt_folder_fit3")

# priors
bprior <- prior(constant(1), class = "b") +
  prior(constant(2), class = "b", coef = "zBase") +
  prior(constant(0.5), class = "sd")

# fit model
fit3 <-
  chkpt_brms(
    bf(
      formula = count ~ zAge + zBase + (1 | patient),
      family = poisson()
    ),
    data = epilepsy,
    path  = path,
    prior = bprior,
    iter_warmup = 1000,
    iter_sampling = 1000,
    iter_per_chkpt = 250,
  )
```

```
# check priors
prior_summary(fit3)


## End(Not run)
```

`chkpt_setup` *Checkpoint Setup*

Description

Deterimine the number of checkpoints for the warmup and sampling, given the desired number of iterations for each and the iterations per checkpoint.

Usage

```
chkpt_setup(iter_sampling, iter_warmup, iter_per_chkpt)
```

Arguments

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

Value

A list with the following:

- `warmup_chkpts`: Number of warmup checkpoints
- `sample_chkpts`: Number of sampling checkpoints
- `total_chkpts`: Total number of checkpoints (warmup_chkpts + sample_chkpts)
- `iter_per_chkpt`: Iterations per checkpoint

Examples

```
chkpt_setup <- chkpt_setup(
  iter_sampling = 5000,
  iter_warmup = 2000,
  iter_per_chkpt = 10
)

chkpt_setup
```

chkpt_stan *Checkpoint Sampling: Stan*

Description

Fit Bayesian models using Stan with checkpointing.

Usage

```
chkpt_stan(
  model_code,
  data,
  iter_warmup = 1000,
  iter_sampling = 1000,
  iter_per_chkpt = 100,
  iter_typical = 150,
  parallel_chains = 2,
  threads_per = 1,
  chkpt_progress = TRUE,
  control = NULL,
  seed = 1,
  path,
  ...
)
```

Arguments

`model_code`	Character string corresponding to the Stan model.
`data`	A named list of R objects (like for RStan). Further details can be found in `sample`.
`iter_warmup`	(positive integer) The number of warmup iterations to run per chain (defaults to 1000).
`iter_sampling`	(positive integer) The number of post-warmup iterations to run per chain (defaults to 1000).
`iter_per_chkpt`	(positive integer). The number of iterations per checkpoint. Note that `iter_sampling` is divided by `iter_per_chkpt` to determine the number of checkpoints. This must result in an integer (if not, there will be an error).
`iter_typical`	(positive integer) The number of iterations in the initial warmup, which finds the so-called typical set. This is an initial phase, and not included in `iter_warmup`. Note that a large enough value is required to ensure converge (defaults to 150).
`parallel_chains`	(positive integer) The *maximum number* of MCMC chains to run in parallel. If parallel_chains is not specified then the default is to look for the option `mc.cores`, which can be set for an entire R session by `options(mc.cores=value)`. If the `mc.cores` option has not been set then the default is `1`.

threads_per	(positive integer) Number of threads to use in within-chain parallelization (defaults to 1).
chkpt_progress	logical. Should the chkptstanr progress be printed (defaults to TRUE) ? If set to FALSE, the standard cmdstanr progress bar is printed for each checkpoint (which does not actually keep track of checkpointing progress)
control	A named list of parameters to control the sampler's behavior. It defaults to NULL so all the default values are used. For a comprehensive overview see stan.
seed	(positive integer). The seed for random number generation to make results reproducible.
path	Character string. The path to the folder, that is used for saving the checkpoints.
...	Currently ignored.

Value

An objet of class chkpt_stan

Examples

```
## Not run:

# path for storing checkpoint info
path <- create_folder(folder_name = "chkpt_folder_fit1")

stan_code <- make_stancode(bf(formula = count ~ zAge + zBase * Trt + (1|patient),
                              family = poisson()),
                           data = epilepsy)
stan_data <- make_standata(bf(formula = count ~ zAge + zBase * Trt + (1|patient),
                              family = poisson()),
                           data = epilepsy)

# "random" intercept
fit1 <- chkpt_stan(model_code = stan_code,
                   data = stan_data,
                   iter_warmup = 1000,
                   iter_sampling = 1000,
                   iter_per_chkpt = 250,
                   path = path)

draws <- combine_chkpt_draws(object = fit1)

posterior::summarise_draws(draws)


# eight schools example

# path for storing checkpoint info
path <- create_folder(parent_folder = "chkpt_folder_fit2")

stan_code <- "
```

```
data {
 int<lower=0> n;
  real y[n];
  real<lower=0> sigma[n];
}
parameters {
  real mu;
  real<lower=0> tau;
  vector[n] eta;
}
transformed parameters {
  vector[n] theta;
  theta = mu + tau * eta;
}
model {
  target += normal_lpdf(eta | 0, 1);
  target += normal_lpdf(y | theta, sigma);
}
"
stan_data <- schools.data <- list(
  n = 8,
  y = c(28,  8, -3,  7, -1,  1, 18, 12),
  sigma = c(15, 10, 16, 11,  9, 11, 10, 18)
)

fit2 <- chkpt_stan(model_code = stan_code,
                   data = stan_data,
                   iter_warmup = 1000,
                   iter_sampling = 1000,
                   iter_per_chkpt = 250,
                   path = path)

draws <- combine_chkpt_draws(object = fit2)

posterior::summarise_draws(draws)

## End(Not run)
```

`combine_chkpt_draws` *Combine Checkpoint Draws*

Description

Combine Checkpoint Draws

Usage

```
combine_chkpt_draws(object, ...)
```

Arguments

`object`	An object of class `brmsfit` or `chkpt_stan`.
`...`	Currently ignored.

Value

An object of class `draws_array`.

Examples

```
## Not run:
path <- create_folder(folder_name = "chkpt_folder_fit1")

stan_code <- "
data {
 int<lower=0> n;
  real y[n];
  real<lower=0> sigma[n];
}
parameters {
  real mu;
  real<lower=0> tau;
  vector[n] eta;
}
transformed parameters {
  vector[n] theta;
  theta = mu + tau * eta;
}
model {
  target += normal_lpdf(eta | 0, 1);
  target += normal_lpdf(y | theta, sigma);
}
"

stan_data <- schools.data <- list(
  n = 8,
  y = c(28,  8, -3,  7, -1,  1, 18, 12),
  sigma = c(15, 10, 16, 11,  9, 11, 10, 18)
)

fit2 <- chkpt_stan(model_code = stan_code,
                   data = stan_data,
                   iter_warmup = 1000,
                   iter_sampling = 1000,
                   iter_per_chkpt = 250,
                   path = path)

draws <- combine_chkpt_draws(object = fit2)

draws
```

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

`create_folder` *Create Folder for Checkpointing*

Description

Create the folder for checkingpointing, which will "house" additional folders for the `.stan` model, checkpointing information, and draws from the posterior distribution.

Usage

```
create_folder(folder_name = "cp_folder", path = NULL)
```

Arguments

`folder_name`	Character string. Desired name for the "parent" folder (defaults to `checkpoint`).
`path`	Character string, when specified. Defaults to `NULL`, which then makes the folder in the working directory.

Value

No return value, and instead creates a directory with folders that will contain the checkpointing samples and other information.

Note

This creates a directory with four folders:

- **cmd_fit**: The cmdstanr fittted models (one for each checkpoint).
- **cp_info**: Mass matrix, step size, and initial values for next checkpoint (last iteration from previous checkpoint).
- **cp_samples**: Samples from the posterior distribution (post warmup)
- **stan_model**: Complied **Stan** model

Examples

```
path <- create_folder(folder_name = "cp_folder")

# remove folder
unlink("cp_folder", recursive = TRUE)
```

extract_chkpt_draws *Extract Draws from* CmdStanMCMC *Objects*

Description

A convenience function for extracting the draws from a CmdStanMCMC object.

Usage

```
extract_chkpt_draws(object, phase)
```

Arguments

object	An object of class CmdStanMCMC.
phase	Character string. Which phase during checkpointing? The options included warmup and sample. The latter extracts the draws with inc_warmup = FALSE, which is the default in draws

Value

A 3-D draws_array object (iteration *x* chain *x* variable).

Note

This can be used to extract the draws in general by setting phase = "sample" which then only includes the post-warmup draws.

Examples

```
## Not run:
 library(cmdstanr)

# eight schools example
fit_schools_ncp_mcmc <- cmdstanr_example("schools_ncp")

drws <- extract_chkpt_draws(object = fit_schools_ncp_mcmc,
                            phase = "sample")

# compare to cmdstanr
all.equal(drws, fit_schools_ncp_mcmc$draws())

## End(Not run)
```

`extract_hmc_info` *Extract HMC Sampler Information*

Description

Extract the inverse metric and step size adaption from `CmdStanMCMC` objects.

Usage

```
extract_hmc_info(object)
```

Arguments

`object`	An object of class `CmdStanMCMC`

Value

A list including

- `inv_metric`: Inverse metric for each chain (with `matrix = FALSE`).
- `step_size_adapt`: Step size adaptation for each chain.

Note

This is primarily used internally.

Examples

```
## Not run:

library(cmdstanr)

fit_schools_ncp_mcmc <- cmdstanr_example("schools_ncp")

extract_hmc_info(fit_schools_ncp_mcmc)


## End(Not run)
```

extract_stan_state *Extract Stan State*

Description

Extract Stan State

Usage

```
extract_stan_state(object, phase)
```

Arguments

object	An object of class cmdstanr
phase	Character string indicating the current phase. Options include wormup and sample/

Value

A list containing the inverse metric, step size, and last MCMC draw (to be used as the initial value for the next checkpoint)

Examples

```
## Not run:
library(cmdstanr)

# eight schools example
fit_schools_ncp_mcmc <- cmdstanr_example("schools_ncp")

extract_stan_state(fit_schools_ncp_mcmc, "sample")

## End(Not run)
```

make_brmsfit *Make* brmsfit *Object*

Description

This is primarily used internally, wherein the cmdstanr object is converted into a brmsfit object.

Usage

```
make_brmsfit(object, formula = NULL, data = NULL, prior = NULL, path)
```

Arguments

object	An object of class chkpt_brms
formula	An object of class formula, brmsformula, or brms{mvbrmsformula}. Further information can be found in brmsformula.
data	An object of class data.frame (or one that can be coerced to that class) containing data of all variables used in the model.
prior	An object of class brmsprior.
path	Character string. The path to the folder, that is used for saving the checkpoints.

Value

An object of class brmsfit

Note

This is primarily an internal function that constructs a brmsfit object.

print.chkpt_brms *Print* chkpt_brms *Objects*

Description

Print chkpt_brms Objects

Usage

```
## S3 method for class 'chkpt_brms'
print(x, ...)
```

Arguments

x	Object of class chkpt_brms
...	Currently ignored

Value

No return value, and used to print the chkpt_brms object.

Note

This function mainly avoids printing out a list, and it is only used when brmsfit = "FALSE" in chkpt_brms.

Typically, after fitting, the posterior draws should be summarized with combine_chkpt_draws (assuming brmsfit = "FALSE").

`print.chkpt_setup` *Print* `chkpt_setup` *Object*

Description

Print `chkpt_setup` Object

Usage

```
## S3 method for class 'chkpt_setup'
print(x, ...)
```

Arguments

`x`	An object of class `chkpt_setup`.
`...`	Currently ignored.

Value

No return value, and used to print the `chkpt_setup` object.

Examples

```
chkpt_setup <- chkpt_setup(
  iter_sampling = 5000,
  iter_warmup = 2000,
  iter_per_chkpt = 10
)


chkpt_setup
```

`print.chkpt_stan` *Print* `chkpt_stan` *Objects*

Description

Print `chkpt_stan` Objects

Usage

```
## S3 method for class 'chkpt_stan'
print(x, ...)
```

Arguments

`x`	Object of class `chkpt_stan`
`...`	Currently ignored

Value

No return value, and used to print the `chkpt_stan` object.

Note

This function mainly avoids printing out a list.

Typically, after fitting, the posterior draws should be summarized with `combine_chkpt_draws`.

Index