Package 'pmwg'

September 2, 2020

Title Particle Metropolis Within Gibbs

Version 0.1.9

Description Provides an R implementation of the Particle Metropolis within Gibbs sampler for model parameter, covariance matrix and random effect estimation. A more general implementation of the sampler based on the paper by Gunawan, D., Hawkins, G. E., Tran, M. N., Kohn, R., & Brown, S. D. (2020) <doi:10.1016/j.jmp.2020.102368>. An HTML tutorial document describing the package is available at <https://newcastlecl.github.io/samplerDoc/> and includes several detailed examples, some background and troubleshooting steps.

License GPL-3

URL https://github.com/newcastlecl/pmwg

BugReports https://github.com/newcastlecl/pmwg/issues

Depends R (>= 3.5.0)

Imports coda, condMVNorm, MASS, MCMCpack, mvtnorm, stats

Suggests covr, rtdists, testthat

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Gavin Cooper [aut, cre, trl] (Package creator and maintainer),
Reilly Innes [aut],
Caroline Kuhne [aut],
Jon-Paul Cavallaro [aut],
David Gunawan [aut] (Author of original MATLAB code),
Guy Hawkins [aut],
Scott Brown [aut, trl] (Original translation from MATLAB to R)

Maintainer Gavin Cooper <gavin@gavincooper.net>

Repository CRAN

Date/Publication 2020-09-02 08:00:08 UTC

R topics documented:

`as_mcmc` *Return a CODA mcmc object with the required samples*

Description

Given a sampler object and a specification of the samples required, return either an individual coda mcmc object, or a list of mcmc objects.

Usage

```
as_mcmc(sampler, selection = "theta_mu", filter = stages)
```

Arguments

`sampler`	The pmwgs object containing samples to extract.
`selection`	The selection of sample types to return.
`filter`	A filter that defines which stage to draw samples from.

Value

An mcmc object or list containing the selected samples.

Selecting sample types

The values that can be chosen for the `selection` argument can be one of the following list:

`"theta_mu"` the model parameter estimate samples

`"theta_sig"` the covariance matrix estimates, returns a list of mcmc objects, one for each model parameter.

`"alpha"` the random effect estimates, returns a list of mcmc objects, one for each subject.

The default value for `selection` is `"theta_mu"`

Filtering samples

The `filter` argument can take one of two forms:

- An integer vector, usually a sequence of integers, that must fall within the range 1:end.
- A character vector, where each element corresponds to a stage of the sampling process, ie one or more of "init", "burn", "adapt" or "sample".

The default value for `filter` is all stages.

Examples

```
# No example yet
```

`forstmann` *Forstmann et al.'s data*

Description

A dataset containing the speed or accuracy manipulation for a Random Dot Motion experiment.

Usage

```
forstmann
```

Format

A data frame with 15818 rows and 5 variables:

subject integer ID for each subject

rt reaction time for each trial as a double

condition Factor with 3 levels for Speed, Accuracy and Neutral

stim Factor with 2 levels for Left and Right trials

resp Factor with 2 levels for Left and Right responses

Details

Details on the dataset can be found in the following paper:

Striatum and pre-SMA facilitate decision-making under time pressure

Birte U. Forstmann, Gilles Dutilh, Scott Brown, Jane Neumann, D. Yves von Cramon, K. Richard Ridderinkhof, Eric-Jan Wagenmakers.

Proceedings of the National Academy of Sciences Nov 2008, 105 (45) 17538-17542; DOI: 10.1073/pnas.0805903105

Source

https://www.pnas.org/content/105/45/17538

`init` *Initialise values for the random effects*

Description

Initialise the random effects for each participant using MCMC.

Usage

```
init(
  pmwgs,
  start_mu = NULL,
  start_sig = NULL,
  display_progress = TRUE,
  particles = 1000
)
```

Arguments

`pmwgs`	The sampler object that provides the parameters.
`start_mu`	An array of starting values for the group means
`start_sig`	An array of starting values for the group covariance matrix
`display_progress`	Display a progress bar during sampling
`particles`	The number of particles to generate in initialisation

Details

Before sampling can start the Particle Metropolis within Gibbs sampler needs initial values for the random effects. The `init` function generates these values using a Monte Carlo algorithm. One alternative methods would be setting the initial values randomly.

Optionally takes starting values for the model parameters and the variance / covariance matrix. All arrays must match the appropriate shape.

For example, with 5 parameters and 10 subjects, the model parameter start means must be a vector of length 5 and the covariance matrix must be an array of 5 x 5.

Alternatively the if argument values for the starting points are left at the default (NULL) then starting points will be sampled from the prior for group level values (model parameters and covariance matrix)

Value

The sampler object but with initial values set for `theta_mu`, `theta_sig`, `alpha` and other values for the first sample.

Examples

```
lba_ll <- function(x, data) {
  x <- exp(x)
  if (any(data$rt < x["t0"])) {
    return(-1e10)
  }
  sum(
    log(
      rtdists::dLBA(
        rt = data$rt,
        response = data$correct,
        A = x["A"],
        b = x["A"] + x[c("b1", "b2", "b3")][data$condition],
        t0 = x["t0"],
        mean_v = x[c("v1", "v2")],
        sd_v = c(1, 1),
        silent = TRUE
      )
    )
  )
}
sampler <- pmwgs(
  forstmann,
  c("b1", "b2", "b3", "A", "v1", "v2", "t0"),
  lba_ll
)
sampler <- init(sampler, start_mu = rnorm(7), start_sig = diag(rep(0.01, 7)))
```

`is.pmwgs` *Test whether object is a pmwgs*

Description

Checks whether object is a Particle Metropolis with Gibbs sampler

Usage

```
is.pmwgs(x)
```

Arguments

`x`	An object to test

Value

logical, whether object inherits from pmwgs

pmwgs *Create a PMwG sampler and return the created object*

Description

This function takes a few necessary elements for creating a PMwG sampler. Each pmwgs object is required to have a dataset, a list of parameter names, a log likelihood function and optionally a prior for the model parameters.

Usage

```
pmwgs(data, pars, ll_func, prior = NULL)
```

Arguments

data	A data frame containing empirical data to be modelled. Assumed to contain at least one column called subject whose elements are unique identifiers for each subject. Can be any of data.frame, data.table or tibble, or any other data frame like object that can have subsets created in an identical way.
pars	The list of parameter names to be used in the model
ll_func	A log likelihood function that given a list of parameter values and a data frame (or other data store) containing subject data will return the log likelihood of x given data.
prior	Specification of the prior distribution for the model parameters. It should be a list with two elements, theta_mu_mean and theta_mu_var which fully specify the prior distribution. If left as the default (NULL) then the theta_mu_mean will be all zeroes and theta_mu_var will be 1 on the diagonal and zero elsewhere.

Value

A pmwgs object that is ready to be initialised and sampled.

Examples

```
# Specify the log likelihood function
lba_loglike <- function(x, data) {
  x <- exp(x)
  if (any(data$rt < x["t0"])) {
    return(-1e10)
  }
  # This is faster than "paste".
  bs <- x["A"] + x[c("b1", "b2", "b3")][data$condition]

  out <- rtdists::dLBA(
    rt = data$rt, # nolint
    response = data$correct,
    A = x["A"],
```

```
    b = bs,
    t0 = x["t0"],
    mean_v = x[c("v1", "v2")],
    sd_v = c(1, 1),
    distribution = "norm",
    silent = TRUE
  )
  bad <- (out < 1e-10) | (!is.finite(out))
  out[bad] <- 1e-10
  out <- sum(log(out))
  out
}

# Specify parameter names and priors
pars <- c("b1", "b2", "b3", "A", "v1", "v2", "t0")
priors <- list(
  theta_mu_mean = rep(0, length(pars)),
  theta_mu_var = diag(rep(1, length(pars)))
)

# Create the Particle Metropolis within Gibbs sampler object
sampler <- pmwgs(
  data = forstmann,
  pars = pars,
  ll_func = lba_loglike,
  prior = priors
)
```

`run_stage` *Run a stage of the PMwG sampler*

Description

Run one of burnin, adaptation or sampling phase from the PMwG sampler. Each stage involves slightly different processes, so for the full PMwG sampling we need to run this three times.

Usage

```
run_stage(
  pmwgs,
  stage,
  iter = 1000,
  particles = 1000,
  display_progress = TRUE,
  n_cores = 1,
  ...
)
```

Arguments

pmwgs	A Particle Metropolis within Gibbs sampler which has been set up and initialised
stage	The sampling stage to run. Must be one of 'burn', 'adapt' or 'sample'.
iter	The number of iterations to run for the sampler. For 'burn' and 'sample' all iterations will run. However for 'adapt' if all subjects have enough unique samples to create the conditional distribution then the stage will finish early.
particles	The default here is 1000 particles to be generated for each iteration, however during the sample phase this should be reduced.
display_progress	Display a progress bar during sampling.
n_cores	Set to more than 1 to use mclapply. Setting n_cores greater than 1 is only permitted on Linux and Mac OS X machines.
...	Further arguments used to fine tune the sampling process. Accepted additional arguments include: • epsilon which should be a value between 0 and 1. **epsilon** controls the extent to which the covariance matrix is scaled when generating particles from the previous random effect. • mix controls the mixture of different sources for particles. The function numbers_from_proportion is passed this value and includes extra details on what is accepted. • n_unique is a number representing the number of unique samples to check for on each iteration of the sampler. Only used during the 'adapt' stage.

Details

The **burnin** stage by default selects 50 parameter sample (selected through a Gibbs step) and 50 the previous random effect of each participant. It assesses each particle with the log-likelihood function and samples from all particles weighted by their log-likelihood.

The **adaptation** stage selects and assesses particle in the same was as burnin, however on each iteration it also checks whether each participant has enough unique random effect samples to attempt to create a conditional distribution for efficient sampling in the next stage. If the attempt at creating a conditional distribution fails, then the number of unique samples is increased and sampling continues. If the attempt succeeds then the stage is finished early.

The **final** stage (sampling) by default samples predominantly from the conditional distribution created at the end of adaptation. This is more efficient and allows the number of particles to be reduced whilst still getting a high enough acceptance rate of new samples.

Once complete each stage will return a sampler object with the new samples stored within it.

Value

A pmwgs object with the newly generated samples in place.

Examples

```
# No example yet
```

Index