Package 'runMCMCbtadjust'

November 10, 2023

Type Package

Title Runs Monte Carlo Markov Chain - With Either 'JAGS', 'nimble' or 'greta' - While Adjusting Burn-in and Thinning Parameters

Version 1.0.4

Description The function runMCMC_btadjust() returns a mcmc.list object which is the output of a Markov Chain Monte Carlo obtained - from either 'JAGS', 'nimble' or 'greta' - after adjusting burn-in and thinning parameters to meet pre-specified criteria in terms of convergence & effective sample size.

License CECILL-2.1

Encoding UTF-8

RoxygenNote 7.2.3

Imports coda

Suggests nimble (>= 1.0.0), rjags, runjags, greta, R6, tensorflow, ggmcmc, rstan, knitr, markdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Frédéric Gosselin [cre, aut] (<https://orcid.org/0000-0003-3737-106X>, INRAE), Institut national de recherche pour l'agriculture, l'alimentation et l'environnement [cph] (INRAE)

Maintainer Frédéric Gosselin <frederic.gosselin@inrae.fr>

Repository CRAN

Date/Publication 2023-11-10 11:20:02 UTC

R topics documented:

runMCMC_btadjust *runMCMC_btadjust*

Description

returns a mcmc.list object which is the output of a Markov Chain Monte Carlo obtained after adjusting burn-in & thinning parameters to meet pre-specified criteria in terms of convergence & effective sample size - i.e. sample size adjusted for autocorrelation - of the MCMC output

Usage

```
runMCMC_btadjust(
  code = NULL,
  data = NULL,
  constants = NULL,
  model = NULL,
  MCMC_language = "Nimble",
  Nchains,
  inits = NULL,
  params = NULL,
  params.conv = NULL,
  params.save = NULL,
  niter.min = 100,
  niter.max = Inf,
  nburnin.min = 10,
  nburnin.max = Inf,
  thin.min = 1,
  thin.max = Inf,
  neff.min = NULL,
  neff.med = NULL,
  neff.mean = NULL,
  conv.max = NULL,
  conv.med = NULL,
  conv.mean = NULL,
 control = list(time.max = NULL, check.convergence = TRUE, check.convergence.firstrun =
    NULL, recheck.convergence = TRUE, convtype = "Gelman", convtype.Gelman = 2,
    convtype.Geweke = c(0.1, 0.5), convtype.alpha = 0.05, neff.method = "Stan",
   Ncycles.target = 2, props.conv = c(0.25, 0.5, 0.75), min.Nvalues = 300, min.thinmult
   = 1.1, safemultiplier.Nvals = 1.2, round.thinmult = TRUE, identifier.to.print = "",
   print.diagnostics = FALSE, print.thinmult = TRUE, innerprint = FALSE, seed = 1,
    remove.fixedchains = TRUE,
     check.installation = TRUE),
 control.MCMC = list(confModel.expression.toadd = NULL, sampler = expression(hmc()),
    warmup = 1000, n.adapt = 1000, RNG.names = c("base::Wichmann-Hill",
  "base::Marsaglia-Multicarry", "base::Super-Duper", "base::Mersenne-Twister"), n_cores
    = NULL, showCompilerOutput = TRUE, buildDerivs = FALSE)
)
```

Arguments

code	R object: code for the model that will be used to build the MCMC when MCMC_language is "Nimble" or "Jags". If "Nimble", must be the name (in R) of the object which is the result of the function nimbleCode. If "Jags", should be either: (i) a character string which is the name of a txt file that contains the code of the model (as used in the function jags.model): should then end up by ".txt"; or (ii) a character string that contains the text of the Jags code.
data	R list: a list that will contain the data when MCMC_language is "Nimble" or "Jags". If "Nimble", will be sent to the data argument of the nimbleModel function in nimble package, i.e. the data that have a random distribution in the model. If MCMC_language is "Greta", can be used just to document the summary of data in the output.
constants	R list: a list that will contain the rest of the data (in addition to data) when MCMC_language is "Nimble". Will be sent to the constants argument of the nimbleModel function in nimble package, i.e. the data that do not have a random distribution in the model. If MCMC_language is "Greta", can be used just to document the summary of other data in the output.
model	R object: should be the result of the model command of Greta when MCMC_language is "Greta".
MCMC_language	character value: designates the MCMC_language used to write & fit the Bayesian model in R. Current choices are "Nimble" - the default-, "Greta" or "Jags". Note that in case it is "Nimble", package nimble should be loaded in your search list.
Nchains	integer value : the number of Markov chains to run in the MCMC.
inits	an R list, with Nchains components. Each component is a named list that contains the initial values of the parameters for the MCMC. In case MCMC_language=="Greta", each component should be the result of the initials function in greta package.
params	character vector: contains the names of the parameters to save at the end of the MCMC and to monitor for convergence and effective sample size; inactive for convergence/effective sample size if params.conv is specified; inactive for saving if params.save is specified.
params.conv	character vector: contains the names of the parameters to monitor for convergence and effective sample size.
params.save	character vector: contains the names of the parameters to be saved at the end of the MCMC.
niter.min	integer value: the minimum number of iterations in each chain of the MCMC.
niter.max	integer value: the maximum number of iterations in each chain of the MCMC. Will stop the MCMC once the number of iterations will reach this limit.
nburnin.min	integer value: the minimum number of burn-in (=transitory) iterations in each chain of the MCMC.
nburnin.max	integer value: the maximum number of burn-in (=transitory) iterations in each chain of the MCMC. Will stay at this burn-in value once this limit is reached.
thin.min	integer value: the minimum value of the thin parameter of the MCMC.

`thin.max` integer value: the maximum value of the thin parameter of the MCMC. Will stay at this thin value once this limit is reached.

`neff.min` positive real number: minimum effective sample size - over parameters used to diagnose convergence & effective sample size- , as calculated with `neff.method` (specified in `Control`). The algorithm will not stop if the minimum number of efficient values is not above this value (unless another limit - e.g. `niter.max` - is reached).

`neff.med` positive real number: median effective sample size - over parameters used to diagnose convergence & effective sample size- , as calculated with `neff.method` (specified in `Control`). The algorithm will not stop if the median number of efficient values is not above this value (unless another limit - e.g. `niter.max` - is reached).

`neff.mean` positive real number: mean effective sample size - over parameters used to diagnose convergence & effective sample size-, as calculated with `neff.method` (specified in `Control`). The algorithm will not stop if the mean number of efficient values is not above this value (unless another limit - e.g. `niter.max` - is reached).

`conv.max` positive real number: maximum - over parameters used to diagnose convergence & effective sample size - convergence diagnostic, as calculated with `convtype` method (specified in `Control`). The algorithm will not stop if the maximum convergence diagnostic is not below this value (unless another limit - e.g. `niter.max` - is reached).

`conv.med` positive real number: median - over parameters used to diagnose convergence & effective sample size - convergence diagnostic, as calculated with `convtype` method(specified in `Control`). The algorithm will not stop if the median convergence diagnostic is not below this value (unless another limit - e.g. `niter.max` - is reached).

`conv.mean` positive real number: mean - over parameters used to diagnose convergence & effective sample size - convergence diagnostic, as calculated with `convtype` method (specified in `Control`). The algorithm will not stop if the mean convergence diagnostic is not below this value (unless another limit - e.g. `niter.max` - is reached).

`control` list of `runMCMC_btadjust` control parameters: with the following components:

- `time.max`: positive number (units: seconds): maximum time of the process in seconds; the program will organize itself to stop before `0.95*time.max`. Default to NULL, corresponding to no time constraint.
- `check.convergence`: logical value: should the program check convergence at all? Default to TRUE. See Details.
- `check.convergence.firstrun`: logical value: should we check convergence after the first run? Default to NULL in which case will depend on `MCMC_language`: if "Greta", will be TRUE because warmup phase separated from the rest; otherwise will be FALSE.
- `recheck.convergence`: logical value: should the algorithm recheck convergence once convergence has been found in a previous run? Default to TRUE.

- contype: character value: specifies the type of convergence diagnostic used. Currently implemented: "Gelman" for original Gelman-Rubin diagnostic (only possible if Nchains>=2), "Gelman_new" for the version of the Gelman-Rubin diagnostic in the second version of "Bayesian Data Analysis" (Gelman, Carlin, Stern and Rubin)(only possible if Nchains>=2), "Geweke" for Geweke diagnostic (at present applied only in case Nchains==1) and "Heidleberger" for the reciprocal of Heidleberger-Welch first part of convergence diagnostic based on the Cramer-von Mises test statistic.
- convtype.Gelman: integer value: when convtype=="Gelman", do we target the Point estimate diagnostic (value 1) or the Upper C.I. diagnostic (value 2). Default to 2.
- convtype.Geweke: real vector with two components between 0 and 1: (i) the fraction of samples to consider as the beginning of the chain (frac1 in geweke.diag); (ii) the fraction of samples to consider as the end of the chain (frac2 in gewke.diag). Default to c(0.1,0.5) as in geweke.diag.
- convtype.alpha: real value between 0 and 1: significance level used in case convtype=="Gelman" and convtype.Gelman==2, or convtype=="Heidleberger"
- neff.method: character value: method used to calculate the effective sample sizes. Current choice between "Stan" (the default) and "Coda". If "Stan", uses the function monitor in package rstan. If "Coda", uses the function effectiveSize in package coda.
- Ncycles.target: integer value: targeted number of MCMC runs. Default to 2.
- props.conv: numeric vector: in case of non convergence: quantiles of number of iterations removed to recheck convergence. Values should be between 0 and 1.
- min.Nvalues: integer value: minimum number of values to diagnose convergence or level of autocorrelation.
- round.thinmult: logical value: should the thin multiplier be rounded to the nearest integer so that past values are precisely positioned on the modified iteration sequence? Default to TRUE. Value of FALSE may not be rigorous or may not converge well.
- min.thinmult: numeric value: minimum value of thin multiplier: if diagnostics suggest to multiply by less than this, this is not done and the current situation of autocorrelation is considered OK.
- seed: integer number: seed for the pseudo-random number generator inside runMCMC_btadjust.
- identifier.to.print: character string: printed each time an MCMC update is ran to identify the model (esp. if multiple successive calls to runMCMC_btadjust are made).
- safemultiplier.Nvals: positive number: number bigger than 1 used to multiply the targeted number of efficient values in calculations of additional number of iterations.
- print.diagnostics: logical value: should diagnostics be printed each time they are calculated? Default to FALSE.
- print.thinmult: logical value: should the raw multiplier of thin be printed each time it is calculated? Default to TRUE.

- innerprint: logical value: should printings be done inside the function monitor of rstan in case neff.method=="Stan"? Default to FALSE.
- remove.fixedchains: logical value: should we remove Markov chains that do not vary (i.e. whose all parameters have zero variances)? Default to TRUE.
- check.installation: logical value: should the function check installation of packages and programs? Default to TRUE.

control.MCMC list of MCMC control parameters: with the following components - that depend on MCMC_language:

- confModel.expression.toadd (only for MCMC_language=="Nimble"): expression to add to confModel to specify samplers, remove nodes... confModel should be referred to by confModel[[i]]. See Details for an example.
- sampler (only for MCMC_language=="Greta"): expression used to specify the sampler used.
- warmup (only for MCMC_language=="Greta"): integer value used as warmup parameter in the mcmc.
- n.adapt (only for MCMC_language=="Jags"): integer value: number of iterations used for adaptation (in function jags.model in rjags package).
- RNG.names (only for MCMC_language=="Jags"): character vector: name of pseudo-random number generators for each chain. Each component of the vector should be among "base::Wichmann-Hill", "base::Marsaglia-Multicarry", "base::Super-Duper", "base::Mersenne-Twister". If less values than Nchains are provided, they are specified periodically.
- n_cores (only for MCMC_language=="Greta"): integer or NULL: maximum number of cores to use by each sampler.
- showCompilerOutput (only for MCMC_language=="Nimble"): logical value indicating whether details of C++ compilation should be printed. Default to TRUE.
- buildDerivs (only for MCMC_language=="Nimble"): logical value indicating derivatives should be prepared when preparing Nimble model (will esp. allow to use HMC sampler). Default to FALSE.

Details

Recap:
If MCMC_language=="Nimble", the code, data and constants arguments should be specified according to the requirements of nimble package.
If MCMC_language=="Jags", the code and data arguments need to be specified as required by rjags package.
If MCMC_language=="Greta", the model argument must be specified and should be the result of the model command in greta package.

Details on check.convergence:
If FALSE, no check of convergence at all, after nburnin.min (& recheck.convergence is put to FALSE & check.convergence.firstrun is dominated by check.convergence).
If TRUE, the convergence behavior is governed by check.convergence.firstrun & recheck.convergence.

Example for confModel.expression.toadd component of control.MCMC:
confModel.expression.toadd<-expression({ConfModel[[i]]$removeSamplers(c("alpha","dzetad","beta","ex
ConfModel[[i]]$addSampler(target = c("alpha","dzetad","beta"),type = "RW_block") ConfModel[[i]]$addSam
= c("exper_bias[2]","exper_bias[3]"),type = "RW_block") ConfModel[[i]]$addSampler(target
= c("exper_precision[2]","exper_precision[3]"),type = "RW_block") })

Remark for params, params.conv, params.save:
in cases of parameters that are vectors, matrices... the params vector can contain only the name of the vector or matrix... in which case all its components will be used. It can also contain the names of individual components.

Value

a mcmc.list object with attributes with the following components:

- call.params: a list containing most of the important arguments of the runMCMC_btadjust call as well as a summary of dimensions/lengths and mean of components of data and constants arguments.
- final.params: a list with the parameters of the MCMC at the end of fitting:
  - burnin: number of iterations of the transient (burn-in) period
  - thin: number of iterations used for thinning the final output
  - niter.tot: total number of iterations (of each MCMC chain)
  - duration: total duration (elapsed time) of the fit (in seconds)
  - duration.MCMC.preparation: duration (elapsed time) of MCMC preparation (in seconds)
  - duration.MCMC.transient: duration (elapsed time) of the MCMC transient (burn-in) phase (in seconds)
  - duration.MCMC.asymptotic: duration (elapsed time) of the MCMC asymptotic phase (in seconds)
  - duration.btadjust: duration (elapsed time) outside MCMC preparation & fitting (in seconds)
  - CPUduration: total CPU duration (user+system) of the fit (in seconds)
  - CPUduration.MCMC.preparation: CPU duration (user+system) of MCMC preparation (in seconds)
  - CPUduration.MCMC.transient: CPU duration (user+system) of the MCMC transient (burn-in) phase (in seconds)
  - CPUduration.MCMC.asymptotic: CPU duration (user+system) of the MCMC asymptotic phase (in seconds)
  - CPUduration.btadjust: CPU duration (user+system) outside MCMC preparation & fitting (in seconds)
  - time: time (from Sys.time) at the end of model fitting
- final.diags: a list with final diagnostics of the fit:
  - params: parameters of the MCMC (burn-in, thin, niter...)
  - conv_synth: synthetic output of convergence diagnostics
  - neff_synth: synthetic output for calculations of effective sample sizes
  - conv: raw convergence values for all the parameters being diagnosed

  - neff: raw effective sample size values for all the parameters being diagnosed
- package.versions: a named vector with the versions of the packages used during MCMC fitting
- R.version: a list giving the different details of the R setup during MCMC fitting
- warnings: a list of the warning messages issued during fitting; unsure it still works with this version
- error: a list with the error messages issued during fitting; unsure it still works with this version

Examples

```
 #\code{
# for examples with Nimble or Greta, see the Vignette.
# condition variable of whether installation is OK with Jags to avoid error durong package check
condition_jags<-TRUE
if (nchar(system.file(package='rjags'))==0) {condition_jags<-FALSE}
if (nchar(system.file(package='runjags'))==0) {condition_jags<-FALSE}
if (condition_jags)
{suppressWarnings(temp<-runjags::testjags(silent=TRUE))
 if(!(temp$JAGS.available&temp$JAGS.found&temp$JAGS.major==4)) {condition_jags<-FALSE}}

if (condition_jags) {
#generating data
set.seed(1)
y1000<-rnorm(n=1000,mean=600,sd=30)
ModelData <-list(mass = y1000,nobs = length(y1000))

#writing the Jags code as a character chain in R
modeltotransfer<-"model {

# Priors
population.mean ~ dunif(0,5000)
population.sd ~ dunif(0,100)

# Precision = 1/variance: Normal distribution parameterized by precision in Jags
population.variance <- population.sd * population.sd
precision <- 1 / population.variance

# Likelihood
for(i in 1:nobs){
  mass[i] ~ dnorm(population.mean, precision)
 }
 }"

#specifying the initial values
ModelInits <- function()
{list (population.mean = rnorm(1,600,90), population.sd = runif(1, 1, 30))}
params <- c("population.mean", "population.sd", "population.variance")
K<-3
set.seed(1)
Inits<-lapply(1:K,function(x){ModelInits()})
```

```
# running runMCMC_btadjust with MCMC_language="Jags":
set.seed(1)
out.mcmc.Coda<-runMCMC_btadjust(MCMC_language="Jags", code=modeltotransfer,
data=ModelData,
Nchains=K, params=params, inits=Inits,
niter.min=1000, niter.max=300000,
nburnin.min=100, nburnin.max=200000,
thin.min=1, thin.max=1000,
neff.min=1000, conv.max=1.05,
control=list(print.diagnostics=TRUE, neff.method="Coda"))

summary(out.mcmc.Coda)
}
#}
```

Index