Package ‘runMCMCbtadjust’

April 25, 2024

Type Package

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

Version 1.1.0

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), nimbleAPT (>= 1.0.6), parallel, Hmisc

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 2024-04-25 15:00:02 UTC

R topics documented:

findMCMC_strong_corrs *findMCMC_strong_corrs*

Description

finds the couples of parameters of a MCMC.list object that have at least a minCorr level of (absolute) correlation

Usage

```
findMCMC_strong_corrs(
  mcmcList,
  corrMethod = "pearson",
  minCorr = 0.3,
  namesToRemove = NULL
)
```

Arguments

`mcmcList`	R object of type mcmc.list that contains the MCMC output
`corrMethod`	character: designates the kind of correlation calculated among "pearson" (the default, for linear relationships), "spearman" (for monotone relationships) or "hoeffd" (for general associations - i.e. dependencies - between parameters)
`minCorr`	double, between 0 and 1: minimum level of (absolute) correlation to report.
`namesToRemove`	R object (can be a vector, matrix, array, list...) all components of which must be of character type: will remove parameters whose names partially match one of tghese components.

Details

In case corrMethod equals "hoeffd", the hoeffd function in the Hmisc package will be used. This can be very slow. Therefore a warning message is printed in this case.

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"))

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

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 = NULL, convtype.Gelman = 2,
   convtype.Geweke = c(0.1, 0.5), convtype.alpha = 0.05, ip.nc = 0, 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, max.prop.decr.neff = 0.1,
   round.thinmult = TRUE, thinmult.in.resetMV.temporary = TRUE, check.thinmult = FALSE,
    only.final.adapt.thin = FALSE, identifier.to.print = "",
     print.diagnostics =
   FALSE, conv.thorough.check = FALSE, print.thinmult = TRUE, innerprint = FALSE, seed =
   NULL, remove.fixedchains = TRUE, check.installation = TRUE, save.data = FALSE),
 control.MCMC = list(confModel.expression.toadd = NULL, sampler = expression(hmc()),
    warmup = 1000, n.adapt = -1, RNG.names = c("base::Wichmann-Hill",
   "base::Marsaglia-Multicarry", "base::Super-Duper", "base::Mersenne-Twister"), n_cores
   = NULL, showCompilerOutput = FALSE, buildDerivs = FALSE, resetMV = FALSE, parallelize
    = FALSE, parallelizeInitExpr = expression(if (MCMC_language == "Nimble") {

    library(nimble)
     if (control.MCMC$APT) {
         library(nimbleAPT)
     }
 }
    else {
```

```
    NULL
 }), useConjugacy = FALSE, WAIC = FALSE, WAIC.Nsamples = 2000,
   WAIC.control = list(online = TRUE, dataGroups = NULL, marginalizeNodes = NULL,
  niterMarginal = 1000, convergenceSet = c(0.25, 0.5, 0.75), thin = TRUE, nburnin_extra
  = 0), APT = FALSE, APT.NTemps = 7, APT.initTemps = NULL, APT.tuneTemps = c(10, 0.7),
    APT.thinPrintTemps = expression(niter/5), includeAllStochNodes = FALSE,
  saveAllStochNodes = FALSE, includeParentNodes = FALSE, saveParentNodes = FALSE,
    extraCalculations = NULL)
)
```

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`	either NULL, a function or an R list, with `Nchains` components. Each component is then 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. If a function, it will generate values for one chain.
`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 effective 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 effective 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 effective 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 or NULL 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 "Heidelberger" for the reciprocal of Heidelberger-Welch first part of convergence diagnostic based on the Cramer-von Mises test statistic. If NULL (the default), chooses "Geweke" in case Nchains==1 and "Gelman" in case Nchains>1.
- `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=="Heidelberger"`
- `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.
- `ip.nc`: real value: inflexion point for log(scaleconvs-1); if (very) negative will tend to double the number of iterations in case of non convergence (i.e. add niter iterations) ; if (very) positive will tend to add niter/Ncycles iterations. Default to 0.
- `conv.thorough.check`: logical value: whether one goes through all the `props.conv` proportions to find the best one in terms first of convergence and then of neffs (if TRUE) or stops at the first `props.conv` corresponding to convergence (if FALSE, the default).
- `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.
- `min.Nvalues`: integer value or expression giving an 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.
- `thinmult.in.resetMV.temporary`: logical value: should the thin multiplier be taken into account in resetting parameter collection in case MCMC_language is "Nimble". Important mainly if control.MCMC$WAIC is TRUE. If TRUE, resetting will be more frequent, and WAIC calculation will be longer and more rigorous. Default to TRUE.
- `check.thinmult`: logical value: should we check thinmult value after thinmult calculation? If TRUE, it is tested whether thinmult meets specific criteria -relative to convergence reaching conservation, number of effective value reaching conservation, minimum number of output values - min.Nvalues- and proportional reduction of number of effective values - and if not decreased values are tested with the same criteria. If FALSE, only the min.Nvalues criterion is taken into account. Default to FALSE. TRUE values should produce shorter MCMCs, more values in the output, with more autocorrelation.
- `only.final.adapt.thin`: logical value: should the thin parameter be adapted only at the end - so that during running of the MCMC we conserve a sufficient number of values - esp. with respect to min.Nvalues. Default to FALSE.
- `min.thinmult`: numeric value: minimum value of thin multiplier: if diagnostics suggest to multiply by less than this, this multiplication is not done. Default to 1.1.
- `seed`: integer number or NULL value: seed for the pseudo-random number generator inside runMCMC_btadjust. Default to NULL in which case here is no control of this seed.
- `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 effective values in calculations of additional number of iterations. Default to 1.2.
- `max.prop.decr.neff`: number between 0 and 1 used, if check.thinmult, to decide if we accept dimension reduction - through augmentation of thin parameter with thinmult -: maximum acceptable Proportional Decrease of the number of effective values: guaranties that at least (1-max.prop.decr.neff) times the number of effective values estimated prior to dimension reduction are kept. Default to 0.1.
- `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.
- `save.data`: logical value: should the program save the entire data in the call.params section of the attributes? Default to FALSE, in which case only a summary of data is saved.

`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"` or `MCMC_language=="Nimble"`): integer value: number of iterations used for adaptation (in function `jags.model` in `rjags` package in case `MCMC_language=="Jags"` and otherwise in Nimble - added to burnin: first iterations that are not saved).
- `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 FALSE to get default printings of limited size.
- `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.
- `resetMV` (only for `MCMC_language=="Nimble"`): logical value to be passed to $run specifying whether previous parameter samples should be reset or not. Default to FALSE to speed up WAIC calculations. You can turn it to TRUE if you wish to speed up runs of MCMC (cf. https://groups.google.com/g/nimble-users/c/RHH9Ybh7bSI/m/Su40lgNRBgAJ).
- `parallelize` (only for `MCMC_language=="Jags"` and `MCMC_language=="Nimble"`): logical value specifying whether the MCMC should be parallelized within the `runMCMC_btadjust` function with the `parallel` package (and for the moment default settings of this package). Default to FALSE. In case `MCMC_language=="Greta"`, parallelization is managed directly by Greta.
- `parallelizeInitExpr` (only for `MCMC_language=="Jags"` and `MCMC_language=="Nimble"`): expression to add in each cluster created by parallelization. Default to

`expression(if(MCMC_language=="Nimble"){library(nimble);if(control.MCMC$APT)` `{library(nimbleAPT)}}` else `{NULL})`.

- `useConjugacy` (only for `MCMC_language=="Nimble"`): logical value specifying whether Nimble should search for conjugate priors in the model. Default to FALSE. If TRUE, can render model configuration shorter (https://groups.google.com/g/nimbl users/c/a6DFCefYfjU/m/kqUWx9UXCgAJ) at the expense of not allowing any conjugate sampler
- `WAIC` (only for `MCMC_language=="Nimble"`): logical value specifying whether WAIC should be calculated online within Nimble. Default to FALSE.
- `WAIC.Nsamples` (only for `MCMC_language=="Nimble"`): integer number: number of samples overs which to calculate WAIC. Default to 2000.
- `WAIC.control` (only for `MCMC_language=="Nimble"`): named list or list of such named lists: list (or lists) specifying the control parameters to calculate WAIC online within Nimble. Default to list(online=TRUE,dataGroups=NULL,marginalizeNodes=N Given the way WAIC is here calculated (after convergence and last sample outputs), components thin will be turned to TRUE and nburnin_extra to 0. If several lists are used, only at most the first Nchains lists will be taken into account to calculate WAICs differently on different Markov Chains.
- `APT` (only for `MCMC_language=="Nimble"`): logical value specifying whether NimbleAPT should be used as a sampler. Default to FALSE. Note that if TRUE, WAIC is turned to FALSE since it seems this method does not work with nimbleAPT.
- `APT.NTemps` (only for `MCMC_language=="Nimble"`): integer number: number of temperatures for NimbleAPT. Default to 7.
- `APT.initTemps` (only for `MCMC_language=="Nimble"`): NULL or double vector of length APT.NTemps: initial temperatures for Nimble APT. Default to NULL in which case initial temperatures will be 1:APT.NTemps. Should all be >1?
- `APT.tuneTemps` (only for `MCMC_language=="Nimble"`): numerical vector of length 2: values to feed the parameters tuneTemper1 and tuneTemper2 in NimbleAPT. See documentation of NimbleAPT. Default to c(10,0.7).
- `APT.thinPrintTemps` (only for `MCMC_language=="Nimble"`): expression or numerical value : thinning parameter for printing temperatures in case APT. Default to expression(niter/5).
- `includeAllStochNodes` (only for `MCMC_language=="Nimble"`): logical value specifying whether all stochastic nodes should be made available in runMCMC_btadjust. Default to FALSE. Maybe useful for the extraCalculations component of control.MCMC.
- `saveAllStochNodes` (only for `MCMC_language=="Nimble"`): logical value specifying whether all stochastic nodes should be made available in runMCMC_btadjust and kept in the result of runMCMC_btadjust. Default to FALSE. Note that if TRUE, will change the content of params_saved.
- `includeParentNodes` (only for `MCMC_language=="Nimble"`): logical value specifying whether parent stochastic nodes of data should be made available in runMCMC_btadjust. Default to FALSE. Maybe useful for the extraCalculations component of control.MCMC - for example ofr "offline" calculations of WAIC.

- `saveParentNodes` (only for `MCMC_language=="Nimble"`): logical value specifying whether parent stochastic nodes of data should be made available in runMCMC_btadjust and kept in the result of runMCMC_btadjust. Default to FALSE. Note that if TRUE, will change the content of params_saved.
- `extraCalculations` (mainly useful for `MCMC_language=="Nimble"` but can be used with other languages as well): NULL value or expression that will be evaluated at tyhe end of runMCMC_btadjust. The value of this expression will be saved in the extraResults component of the final.params component of the attributes of the result of runMCMC_btadjust. See the vignette devoted to the use of it. Default to NULL.

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 either a summary of dimensions/lengths and mean of components of `data` and `constants` arguments or their entire values.
- `final.params`: a list with the parameters of the MCMC at the end of fitting:
  - converged: logical: TRUE if model has converged when stopping the MCMC, FALSE otherwise
  - `neffs.reached`: logical: TRUE if model has converged and reached the objectives in terms of effective sample size, FALSE otherwise

- `final.Nchains`: number of Markov chains finally retained - since some chains could be excluded if invariable.
- `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)
- `WAIC`: results of the calculus of online WAIC (only if control.MCMC$WAIC and `MCMC_language=="Nimble"`). One component per component of control.MCMC$control.WAIC. Each component has a WAIC component and then a WAICDetails component.
- `extraResults`: results of the implementation of control.MCMC$extraCalculations. Can be any kind of R object.
- `Temps`: results of the series of Temperatures met in APT (only if control.MCMC$APT and `MCMC_language=="Nimble"`). One line for each end of $run (=MCMC run).
- `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.MCMC.after`: duration (elapsed time) of the MCMC phase after the asymptotic phase of sampling (e.g. to calculate WAIC) (in seconds)
- `duration.btadjust`: duration (elapsed time) outside MCMC preparation & fitting (in seconds)
- `CPUduration`: total CPU duration (user+system, self+child if not NA - otherwise only self) of the fit (in seconds)
- `CPUduration.MCMC.preparation`: CPU duration (user+system, self+child if not NA - otherwise only self) of MCMC preparation (in seconds)
- `CPUduration.MCMC.transient`: CPU duration (user+system, self+child if not NA - otherwise only self) of the MCMC transient (burn-in) phase (in seconds)
- `CPUduration.MCMC.asymptotic`: CPU duration (user+system, self+child if not NA - otherwise only self) of the MCMC asymptotic phase (in seconds)
- `CPUduration.MCMC.after`: CPU duration (user+system, self+child if not NA - otherwise only self) of the MCMC phase after the asymptotic phase of sampling (e.g. to calculate WAIC) (in seconds)
- `CPUduration.btadjust`: CPU duration (user+system, self+child if not NA - otherwise only self) outside MCMC preparation & fitting (in seconds)
- `childCPUduration`: total child CPU duration (user+system) of the fit (in seconds)
- `childCPUduration.MCMC.preparation`: child CPU duration (user+system) of MCMC preparation (in seconds)
- `childCPUduration.MCMC.transient`: child CPU duration (user+system) of the MCMC transient (burn-in) phase (in seconds)
- `childCPUduration.MCMC.asymptotic`: child CPU duration (user+system) of the MCMC asymptotic phase (in seconds)
- `childCPUduration.MCMC.after`: child CPU duration (user+system) of the MCMC phase after the asymptotic phase of sampling (e.g. to calculate WAIC) (in seconds)

  - `childCPUduration.btadjust`: child 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
- `sessionInfo`: a list containing the result of the call to sessionInfo() function at the end of runMCMC_btadjust function; contains info on the platform, versions of packages, R version...;
- `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 during 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