Package 'nimbleAPT'

July 22, 2025

Type Package

Title Adaptive Parallel Tempering for 'NIMBLE'

Version 1.0.7

Encoding UTF-8

Date 2025-02-17

Maintainer David Pleydell <david.pleydell@inrae.fr>

URL https://github.com/DRJP/nimbleAPT

Description Functions for adaptive parallel tempering (APT) with NIMBLE models. Adapted from 'Lacki' & 'Miasojedow' (2016) <DOI:10.1007/s11222-015-9579-0> and 'Miasojedow, Moulines and Vihola' (2013) <DOI:10.1080/10618600.2013.778779>.

License BSD_3_clause + file LICENSE

Copyright See COPYRIGHTS file.

RoxygenNote 7.3.2

Depends R (>= 3.1.2), nimble

Imports methods

Suggests knitr, rmarkdown, coda

VignetteBuilder knitr

Note The APT samplers within this package were adapted from MCMC samplers provided in the NIMBLE package (https://github.com/nimble-dev/nimble) and developed by the cited members of the NIMBLE Development Team. The adaptations for APT included in this package were the work of the package developer. Many thanks to the NIMBLE Development Team for their support during the development of this package.

NeedsCompilation no

Author David Pleydell [aut, cre, cph] (Package developer, ORCID: <https://orcid.org/0000-0002-6450-1475>),
Daniel Turek [cph] (Member of the NIMBLE Developent Team),
Perry de Valpine [cph] (Member of the NIMBLE Developent Team),
Christopher Paciorek [cph] (Member of the NIMBLE Developent Team),
Nick Michaud [cph] (Member of the NIMBLE Developent Team)

Repository CRAN

Date/Publication 2025-02-17 13:10:02 UTC

Contents

`buildAPT` *Create an APT function, from an MCMCconf object*

Description

Adapted from nimble::buildMCMC. Accepts a single required argument, which may be of class MCMCconf, or inherit from class modelBaseClass (a NIMBLE model object). Returns an APT function; see details section.

Usage

```
buildAPT(conf, Temps, monitorTmax = TRUE, ULT = 1e+06, thinPrintTemps = 1, ...)
```

Arguments

`conf`	An object of class MCMCconf that specifies the model, samplers, monitors, and thinning intervals for the resulting MCMC function. See `configureMCMC` for details of creating MCMCconf objects. Alternatively, `MCMCconf` may a NIMBLE model object, in which case an MCMC function corresponding to the default MCMC configuration for this model is returned.
`Temps`	A numeric vector giving the initial temperature ladder.
`monitorTmax`	A logical indicator (default = TRUE) controlling if MCMC output should be stored at the hottest rung of the temperature ladder. Useful when monitoring the behaviour of APT. When TRUE mvSamples and mvSamples2 monitor T=1 and mvSamplesTmax and mvSamples2Tmax provide identically defined monitors (i.e. for exactly the same nodes) for T=Tmax.
`ULT`	A numeric value (default = 1E6) that provides an upper limit to temperature during APT.
`thinPrintTemps`	A numeric value controlling how often temperatures of the temperature ladder should be printed to screen when runtime parameter printTemps is TRUE. The default value of 1 is often too verbose. A good value to use is niter/10.
`...`	Additional arguments to be passed to `configureMCMC` if `conf` is a NIMBLE model object

Details

Calling buildAPT(conf,Temps,monitorTmax,ULT,thinPrintTemps) will produce an uncompiled (R) APT function object, say 'myAPT'.

The uncompiled MCMC function will have arguments:

`niter` The number of iterations to run the MCMC.

`reset` Boolean specifying whether to reset the internal MCMC sampling algorithms to their initial state (in terms of self-adapting tuning parameters), and begin recording posterior sample chains anew. Specifying `reset=FALSE` allows the MCMC algorithm to continue running from where it left off, appending additional posterior samples to the already existing sample chains. Generally, `reset=FALSE` should only be used when the MCMC has already been run (default = TRUE).

`resetMV`: Boolean specifying whether to begin recording posterior sample chains anew. This argument is only considered when using `reset = FALSE`. Specifying `reset = FALSE, resetMV = TRUE` allows the MCMC algorithm to continue running from where it left off, but without appending the new posterior samples to the already existing samples, i.e. all previously obtained samples will be erased. This option can help reduce memory usage during burn-in (default = FALSE).

`resetTempering` Boolean specifying whether to reset the flexibility of the temperature ladder's adaptation process.

`simulateAll` Boolean specifying whether to simulate into all stochastic nodes. This will overwrite the current values in all stochastic nodes (default = FALSE).

`time` Boolean specifying whether to record runtimes of the individual internal MCMC samplers. When `time=TRUE`, a vector of runtimes (measured in seconds) can be extracted from the MCMC using the method `mcmc$getTimes()` (default = FALSE).

`adaptTemps` Boolean specifying whether the temperature ladder will be adapted or not.

`printTemps` Boolean specifying whether the temperature ladder will be printed during the MCMC. The print frequency is controlled by thinPrintTemps. The output includes (in order): iteration number of current MCMC run; total number of iterations of the tempering scheme - this will include iterations from previous MCMC runs unless resetTempering=TRUE; the number of rows in mvSamples; the number of rows in mvSamples2; the temperature ladder.

`tuneTemper1` Numeric tuning parameter of the adaptation process of the temperature ladder. See source code for buildAPT. Defaults to 10.

`tuneTemper2` Numeric tuning parameter of the adaptation process of the temperature ladder. See source code for buildAPT. Defaults to 1.

`progressBar` Boolean specifying whether to display a progress bar during MCMC execution (default = TRUE). The progress bar can be permanently disabled by setting the system option `nimbleOptions(MCMCprogressBa = FALSE)`.

`thin` Thinning to be applied to monitor.

`thin2` Thinning to be applied to monitor2

Samples corresponding to the `monitors` and `monitors2` from the MCMCconf are stored into the interval variables `mvSamples` and `mvSamples2`, respectively. These may be accessed and converted into R matrix objects via: `as.matrix(mcmc$mvSamples)` `as.matrix(mcmc$mvSamples2)`

The uncompiled (R) MCMC function may be compiled to a compiled MCMC object, taking care to compile in the same project as the R model object, using: `Cmcmc <- compileNimble(Rmcmc, project=Rmodel)`

The compiled function will function identically to the uncompiled object, except acting on the compiled model object.

Value

Calling buildAPT returns an uncompiled APT function object. This is very similar to how NIMBLE's buildMCMC function returns an uncompiled MCMC function object. See ?buildMCMC. Users should be familiar with the chapter 'MCMC' of the NIMBLE manual.

Author(s)

David Pleydell, Daniel Turek

Examples

```
## See the nimbleAPT vignette for more details.
bugsCode <- nimbleCode({
  for (ii in 1:nObs) {
    y[ii,1:2] ~ dmnorm(mean=absCentroids[1:2], cholesky=cholCov[1:2,1:2], prec_param=0)
  }
  absCentroids[1:2] <- abs(centroids[1:2])
  for (ii in 1:2) {
    centroids[ii] ~ dnorm(0, sd=1E3)
  }
})

nObs      <- 100
centroids <- rep(-3, 2)
covChol   <- chol(diag(2))

rModel <- nimbleModel(bugsCode,
                      constants=list(nObs=nObs, cholCov=covChol),
                      inits=list(centroids=centroids))
simulate(rModel, "y")
rModel <- nimbleModel(bugsCode,
                      constants=list(nObs=nObs, cholCov=covChol),
                      data=list(y=rModel$y),
                      inits=list(centroids=centroids))

conf <- configureMCMC(rModel, nodes="centroids", monitors="centroids", enableWAIC = TRUE)
conf$removeSamplers()
conf$addSampler("centroids[1]", type="sampler_RW_tempered", control=list(temperPriors=TRUE))
conf$addSampler("centroids[2]", type="sampler_RW_tempered", control=list(temperPriors=TRUE))
aptR <- buildAPT(conf, Temps=1:5, ULT= 1000, print=TRUE)
```

`nimbleAPT` *nimbleAPT*

Description

A collection of NIMBLE functions for adaptive parallel tempering.

Adaptive parallel tempering (APT) for NIMBLE. Adapted from NIMBLE'S MCMC suite to enable APT specific features (i.e. an adaptive temperature ladder).

NA

See Also

https://r-nimble.org/ _PACKAGE

`plotTempTraj` *plot.tempTraj*

Description

Plot the trajectories of a temperature ladder of an adaptive parallel tempering algorithm

Usage

```
plotTempTraj(cAPT)
```

Arguments

`cAPT`	An APT object generated by buildAPT and compiled by compileNimble.

Details

plotTempTraj Returns two plots, one with T~iterations, the other log10(T)~iterations.

Value

A plot of the trajectories.

Author(s)

David Pleydell

See Also

An example is provided in the documentation of buildAPT.

`sampler_APT` *A virtual function to use as a contains argument when writing APT samplers*

Description

Modified from NIMBLE's samplers_BASE to include a setTemp method

Details of the adaptive parallel tempering (APT) samplers adapted from NIMBLE's MCMC samplers.

Usage

```
sampler_APT()

sampler_RW_tempered(model, mvSaved, target, control)

sampler_RW_block_tempered(model, mvSaved, target, control)

sampler_slice_tempered(model, mvSaved, target, control)

sampler_RW_multinomial_tempered(model, mvSaved, target, control)
```

Arguments

`model`	(uncompiled) model on which the APT algorithm is to be run
`mvSaved`	`modelValues` object to be used to store MCMC samples
`target`	node(s) on which the sampler will be used
`control`	named list that controls the precise behavior of the sampler, with elements specific to `samplertype`. The default values for control list are specified in the setup code of each sampling algorithm. Descriptions of each sampling algorithm, and the possible customizations for each sampler (using the `control` argument) appear below.

Details

Set up functions for this class should include the following arguments

APT samplers must include "contains = sampler_APT" and include a setTemp method

Value

These functions are called from the `addSampler` function and return an uncompiled APT sampler object that can be included in an APT sampling scheme.

`sampler_APT`

base class for APT samplers

When you write a new sampler for use in a NIMBLE MCMC with APT, you must include `contains = sampler_APT`.

RW sampler

The RW sampler executes adaptive Metropolis-Hastings sampling with a normal proposal distribution (Metropolis, 1953), implementing the adaptation routine given in Shaby and Wells, 2011. This sampler can be applied to any scalar continuous-valued stochastic node, and can optionally sample on a log scale.

The RW sampler accepts the following control list elements:

- logScale. A logical argument, specifying whether the sampler should operate on the log scale. (default = FALSE)
- reflective. A logical argument, specifying whether the normal proposal distribution should reflect to stay within the range of the target distribution. (default = FALSE)
- adaptive. A logical argument, specifying whether the sampler should adapt the scale (proposal standard deviation) throughout the course of MCMC execution to achieve a theoretically desirable acceptance rate. (default = TRUE)
- adaptInterval. The interval on which to perform adaptation. Every adaptInterval MCMC iterations (prior to thinning), the RW sampler will perform its adaptation procedure. This updates the scale variable, based upon the sampler's achieved acceptance rate over the past adaptInterval iterations. (default = 200)
- scale. The initial value of the normal proposal standard deviation. If adaptive = FALSE, scale will never change. (default = 1)
- temperPriors. Logical indicator determining if tempering should apply to prior likelihoods. Usually can be set to TRUE. But setting to FALSE can help avoid degeneracy issues for complex problems where bounded uniform priors have been transformed to other (e.g. logit) scales.

The RW sampler cannot be used with options log=TRUE and reflective=TRUE, i.e. it cannot do reflective sampling on a log scale.

RW_block sampler

The RW_block sampler performs a simultaneous update of one or more model nodes, using an adaptive Metropolis-Hastings algorithm with a multivariate normal proposal distribution (Roberts and Sahu, 1997), implementing the adaptation routine given in Shaby and Wells, 2011. This sampler may be applied to any set of continuous-valued model nodes, to any single continuous-valued multivariate model node, or to any combination thereof.

The RW_block sampler accepts the following control list elements:

- adaptive. A logical argument, specifying whether the sampler should adapt the scale (a coefficient for the entire proposal covariance matrix) and propCov (the multivariate normal proposal covariance matrix) throughout the course of MCMC execution. If only the scale should undergo adaptation, this argument should be specified as TRUE. (default = TRUE)

- adaptScaleOnly. A logical argument, specifying whether adaption should be done only for scale (TRUE) or also for propCov (FALSE). This argument is only relevant when adaptive = TRUE. When adaptScaleOnly = FALSE, both scale and propCov undergo adaptation; the sampler tunes the scaling to achieve a theoretically good acceptance rate, and the proposal covariance to mimic that of the empirical samples. When adaptScaleOnly = TRUE, only the proposal scale is adapted. (default = FALSE)
- adaptInterval. The interval on which to perform adaptation. Every adaptInterval MCMC iterations (prior to thinning), the RW_block sampler will perform its adaptation procedure, based on the past adaptInterval iterations. (default = 200)
- scale. The initial value of the scalar multiplier for propCov. If adaptive = FALSE, scale will never change. (default = 1)
- propCov. The initial covariance matrix for the multivariate normal proposal distribution. This element may be equal to the character string 'identity', in which case the identity matrix of the appropriate dimension will be used for the initial proposal covariance matrix. (default = 'identity')
- temperPriors. Logical indicator determining if tempering should apply to prior likelihoods. Usually can be set to TRUE. But setting to FALSE can help avoid degeneracy issues for complex problems where bounded uniform priors have been transformed to other (e.g. logit) scales.

slice sampler

The slice sampler performs slice sampling of the scalar node to which it is applied (Neal, 2003). This sampler can operate on either continuous-valued or discrete-valued scalar nodes. The slice sampler performs a 'stepping out' procedure, in which the slice is iteratively expanded to the left or right by an amount sliceWidth. This sampler is optionally adaptive, governed by a control list element, whereby the value of sliceWidth is adapted towards the observed absolute difference between successive samples.

The slice sampler accepts the following control list elements:

- adaptive. A logical argument, specifying whether the sampler will adapt the value of sliceWidth throughout the course of MCMC execution. (default = TRUE)
- adaptInterval. The interval on which to perform adaptation. (default = 200)
- width. The initial value of the width of each slice, and also the width of the expansion during the iterative 'stepping out' procedure. (default = 1)
- maxSteps. The maximum number of expansions which may occur during the 'stepping out' procedure. (default = 100)

RW_multinomial sampler

This sampler is designed for sampling multinomial target distributions. The sampler performs a series of Metropolis-Hastings steps between pairs of groups. Proposals are generated via a draw from a binomial distribution, whereafter the proposed number density is moved from one group to another group. The acceptance or rejection of these proposals follows a standard Metropolis-Hastings procedure. Probabilities for the random binomial proposals are adapted to a target acceptance rate of 0.5.

The `RW_multinomial` sampler accepts the following control list elements:

- adaptive. A logical argument, specifying whether the sampler should adapt the binomial proposal probabilities throughout the course of MCMC execution. (default = TRUE)
- adaptInterval. The interval on which to perform adaptation. A minimum value of 100 is required. (default = 200)
- useTempering. A logical argument to optionally turn tempering off (i.e. assume all temperatures are 1) for this sampler.

Author(s)

David Pleydell, Daniel Turek

References

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6), 1087-1092.

Neal, Radford M. (2003). Slice Sampling. *The Annals of Statistics*, 31(3), 705-741.

Roberts, G. O. and S. K. Sahu (1997). Updating Schemes, Correlation Structure, Blocking and Parameterization for the Gibbs Sampler. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(2), 291-317.

Shaby, B. and M. Wells (2011). *Exploring an Adaptive Metropolis Algorithm*. 2011-14. Department of Statistics, Duke University.

See Also

configureMCMC addSampler buildMCMC buildAPT runMCMC

Examples

```
## This example is taken from the nimbleAPT vignette. See the vignette for more details.

bugsCode <- nimbleCode({
   for (ii in 1:nObs) {
      y[ii,1:2] ~ dmnorm(mean=absCentroids[1:2], cholesky=cholCov[1:2,1:2], prec_param=0)
   }
   absCentroids[1:2] <- abs(centroids[1:2])
   for (ii in 1:2) {
       centroids[ii] ~ dnorm(0, sd=1E3)
   }
})

nObs      <- 100
centroids <- rep(-3, 2)
covChol   <- chol(diag(2))

rModel <- nimbleModel(bugsCode,
                      constants=list(nObs=nObs, cholCov=covChol),
                      inits=list(centroids=centroids))

simulate(rModel, "y") ## Use model to simulate data
```

```
rModel <- nimbleModel(bugsCode,
                      constants=list(nObs=nObs, cholCov=covChol),
                      data=list(y=rModel$y),
                      inits=list(centroids=centroids))

conf <- configureMCMC(rModel, nodes="centroids", monitors="centroids", enableWAIC = TRUE)

conf$removeSamplers()
conf$addSampler("centroids[1]", type="sampler_RW_tempered", control=list(temperPriors=TRUE))
conf$addSampler("centroids[2]", type="sampler_RW_tempered", control=list(temperPriors=TRUE))
aptR <- buildAPT(conf, Temps=1:5, ULT= 1000, print=TRUE)
```

Index