Package 'nimbleHMC'

July 20, 2026

Title Hamiltonian Monte Carlo and Other Gradient-Based MCMC Sampling Algorithms for 'nimble'

Version 0.2.5

Date 2026-07-20

Maintainer Daniel Turek <danielturek@gmail.com>

Description Provides gradient-based MCMC sampling algorithms for use with the MCMC engine provided by the 'nimble' package. This includes two versions of Hamiltonian Monte Carlo (HMC) No-U-Turn (NUTS) sampling, and (under development) Langevin samplers. The `NUTS_classic` sampler implements the original HMC-NUTS algorithm as described in Hoffman and Gelman (2014) <doi:10.48550/arXiv.1111.4246>. The `NUTS` sampler is a modern version of HMC-NUTS sampling matching the HMC sampler available in version 2.32.2 of Stan (Stan Development Team, 2023). In addition, convenience functions are provided for generating and modifying MCMC configuration objects which employ HMC sampling. Functionality of the 'nimbleHMC' package is described further in Turek, et al (2024) <doi:10.21105/joss.06745>.

Depends R (>= 3.5.0), nimble (>= 1.4.0)

Imports methods

Suggests testthat

License BSD_3_clause + file LICENSE | GPL (>= 2)

Copyright See COPYRIGHTS file.

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Daniel Turek [aut, cre],
Perry de Valpine [aut],
Christopher Paciorek [aut]

Repository CRAN

Date/Publication 2026-07-20 17:20:02 UTC

Contents

addHMC 2
buildHMC 4
configureHMC 6
nimbleHMC 8
sampler_NUTS 11
sampler_NUTS_classic 14
stateNL_NUTS 17
treebranchNL_NUTS 18

Index **19**

---

addHMC *Add HMC sampler*

Description

Add a No-U-Turn (NUTS) Hamiltonian Monte Carlo (HMC) sampler to an existing nimble MCMC configuration object

Usage

```
addHMC(
  conf,
  target = character(),
  type = "NUTS",
  control = list(),
  replace = FALSE,
  print = TRUE,
  ...
)
```

Arguments

conf	A nimble MCMC configuration object, as returned by ‘configureMCMC‘.
target	A character vector of continuous-valued stochastic node names to sample. If this argument contains any discrete-valued nodes, an error is produced and no HMC sampler is added. If this argument is omitted, then no HMC sampler is added.
type	A character string specifying the type of HMC sampler to add, either "NUTS" or "NUTS_classic". See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details of each sampler. The default sampler type is "NUTS".
control	Optional named list of control parameters to be passed as the ‘control‘ argument to the HMC sampler. See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details of the control list elements accepted by each sampler.

replace	Logical argument. If 'TRUE', any existing samplers operating on the specified nodes will be removed, prior to adding the HMC sampler. Default value is 'FALSE'.
print	Logical argument whether to print the newly added HMC sampler. Default value is 'TRUE'.
...	Additional named arguments passed through ... will be used as additional control list elements.

Details

This function adds an HMC sampler to an MCMC configuration object. Use this function if you have already created an MCMC configuration and want to add an HMC sampler. Optionally, using 'replace = TRUE', this function will also remove any existing samplers operating on the target node(s).

Either the 'NUTS_classic' or the 'NUTS' sampler can be added. Both implement variants of No-U-Turn HMC sampling, however the 'NUTS' sampler uses more modern adapatation techniques. See 'help(NUTS)' or 'help(NUTS_classic)' for details.

Use 'conf$addSampler' instead if you need more fine-grained control. See 'help(configureMCMC)' in nimble.

Value

Invisibly returns an object of class 'MCMCconf', but this function is primary called for its side effect.

Author(s)

Daniel Turek

See Also

configureHMC buildHMC configureMCMC addSampler sampler_NUTS sampler_NUTS_classic

Examples

```
code <- nimbleCode({
    b0 ~ dnorm(0, 0.001)
    b1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 10000)
    for(i in 1:N) {
        mu[i] <- b0 + b1 * x[i]
        y[i] ~ dnorm(mu[i], sd = sigma)
    }
})

N <- 10
constants <- list(N = N, x = 1:N)
data <- list(y = 1:N)
inits <- list(b0 = 1, b1 = 0.1, sigma = 1)
```

```
Rmodel <- nimbleModel(code, constants, data, inits, buildDerivs = TRUE)

## create default MCMC configuration object
conf <- configureMCMC(Rmodel)

## remove default samplers operating on b0 and b1
conf$removeSamplers(c("b0", "b1"))

## add an HMC sampler operating on b0 and b1
addHMC(conf, target = c("b0", "b1"))

Rmcmc <- buildMCMC(conf)

# Cmodel <- compileNimble(Rmodel)
# Cmcmc <- compileNimble(Rmcmc, project = Rmodel)
# samples <- runMCMC(Cmcmc)
```

`buildHMC` *Build HMC*

Description

Build an MCMC algorithm which applies HMC sampling to continuous-valued dimensions

Usage

```
buildHMC(
  model,
  nodes = character(),
  type = "NUTS",
  control = list(),
  print = TRUE,
  ...
)
```

Arguments

`model`	A nimble model, as returned by ‘nimbleModel‘. Alternatively, an MCMC configuration object, as returned by either ‘configureHMC‘ or ‘configureHMC‘. See details.
`nodes`	A character vector of stochastic node names to be sampled. If an empty character vector is provided (the default), then all stochastic non-data nodes will be sampled. An HMC sampler will be applied to all continuous-valued non-data nodes, and nimble's default sampler will be assigned for all discrete-valued nodes.
`type`	A character string specifying the type of HMC sampling to apply, either "NUTS" or "NUTS_classic". See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details of each sampler. The default sampler type is "NUTS".

control	Optional named list of control parameters to be passed as the 'control' argument to the HMC sampler. See 'help(NUTS)' or 'help(NUTS_classic)' for details of the control list elements accepted by each sampler.
print	Logical argument specifying whether to print the montiors and samplers. Default is TRUE.
...	Other arguments that will be passed to 'configureHMC'.

Details

This is the most direct way to create an MCMC algorithm using HMC sampling in nimble. This will create a compilable, executable MCMC algorithm, with HMC sampling assigned to all continuous-valued model dimensions, and nimble's default sampler assigned to all discrete-valued dimensions. The 'nodes' argument can be used to control which model nodes are assigned samplers. Use this if you don't otherwise need to modify the MCMC configuration.

Either the 'NUTS_classic' or the 'NUTS' sampling can be applied, which is controled by the 'type' argument. Both implement variants of No-U-Turn HMC sampling, however the 'NUTS' sampler uses more modern adapatation techniques. See 'help(NUTS)' or 'help(NUTS_classic)' for details.

Note that when an MCMC configuration object is provided as the 'model' argument, then an executable MCMC algorithm is generated using the MCMC configuration that was provided - regardless of whether or not it specifies any HMC samplers.

Value

An object of class 'MCMC'.

Author(s)

Daniel Turek

See Also

addHMC configureHMC configureMCMC addSampler sampler_NUTS sampler_NUTS_classic

Examples

```
code <- nimbleCode({
    b0 ~ dnorm(0, 0.001)
    b1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 10000)
    for(i in 1:N) {
        mu[i] <- b0 + b1 * x[i]
        y[i] ~ dnorm(mu[i], sd = sigma)
    }
})

N <- 10
constants <- list(N = N, x = 1:N)
data <- list(y = 1:N)
inits <- list(b0 = 1, b1 = 0.1, sigma = 1)
Rmodel <- nimbleModel(code, constants, data, inits, buildDerivs = TRUE)
```

```
Rmcmc <- buildHMC(Rmodel)

# Cmodel <- compileNimble(Rmodel)
# Cmcmc <- compileNimble(Rmcmc, project = Rmodel)
# samples <- runMCMC(Cmcmc)
```

`configureHMC` *Configure HMC*

Description

Create a nimble MCMC configuration object which applies HMC sampling to continuous-valued dimensions

Usage

```
configureHMC(
  model,
  nodes = character(),
  type = "NUTS",
  control = list(),
  print = TRUE,
  ...
)
```

Arguments

`model`	A nimble model, as returned by ‘nimbleModel‘
`nodes`	A character vector of stochastic node names to be sampled. If an empty character vector is provided (the default), then all stochastic non-data nodes will be sampled. An HMC sampler will be applied to all continuous-valued non-data nodes, and nimble’s default sampler will be assigned for all discrete-valued samplin to apply, either "NUTS" or "NUTS_classic".
`type`	A character string specifying the type of HMC sampling to apply, either "NUTS" or "NUTS_classic". See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details of each sampler. The default sampler type is "NUTS".
`control`	Optional named list of control parameters to be passed as the ‘control‘ argument to the HMC sampler. See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details of the control list elements accepted by each sampler.
`print`	Logical argument specifying whether to print the montiors and samplers. Default is TRUE.
`...`	Other arguments that will be passed to ‘configureMCMC‘

Details

This function can be used like ‘configureMCMC‘ in nimble to create an MCMC configuration object. It will return an MCMC configuration with an HMC sampler assigned to continuous-valued model dimensions, and nimble’s default sampler assigned for discrete-valued dimensions (or, only for the nodes specified in the ‘nodes‘ argument). The resulting MCMC configuration object can be used as an argument to ‘buildMCMC‘ to generate an executable MCMC algorithm.

Either the ‘NUTS_classic‘ or the ‘NUTS‘ sampler can be applied. Both implement variants of No-U-Turn HMC sampling, however the ‘NUTS‘ sampler uses more modern adapatation techniques. See ‘help(NUTS)‘ or ‘help(NUTS_classic)‘ for details.

Use this function if you want to create an MCMC configuration, and then modify it further before building the MCMC algorithm. ‘buildHMC‘ provides a more direct route to a compilable MCMC algorithm with HMC sampling applied to all continuous-valued dimensions.

Value

An object of class ‘MCMCconf‘.

Author(s)

Daniel Turek

See Also

addHMC buildHMC configureMCMC addSampler sampler_NUTS sampler_NUTS_classic

Examples

```
code <- nimbleCode({
    b0 ~ dnorm(0, 0.001)
    b1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 10000)
    for(i in 1:N) {
        mu[i] <- b0 + b1 * x[i]
        y[i] ~ dnorm(mu[i], sd = sigma)
    }
})

N <- 10
constants <- list(N = N, x = 1:N)
data <- list(y = 1:N)
inits <- list(b0 = 1, b1 = 0.1, sigma = 1)

Rmodel <- nimbleModel(code, constants, data, inits, buildDerivs = TRUE)

## create MCMC configuration object with only an HMC sampler
conf <- configureHMC(Rmodel)

Rmcmc <- buildMCMC(conf)

# Cmodel <- compileNimble(Rmodel)
```

```
# Cmcmc <- compileNimble(Rmcmc, project = Rmodel)
# samples <- runMCMC(Cmcmc)
```

`nimbleHMC` *Builds and executes NIMBLE's HMC sampler*

Description

`nimbleHMC` is the most direct entry point to using NIMBLE's HMC sampler. HMC sampling is applied to all unobserved dimensions of a hierarchical model. Discrete-valued model dimensions cannot be sampled using HMC, and will produce an error. See `help(HMC)` for details of the HMC algorithm.

Usage

```
nimbleHMC(
  code,
  constants = list(),
  data = list(),
  inits,
  dimensions = list(),
  model,
  type = "NUTS",
  monitors,
  thin = 1,
  niter = 10000,
  nburnin = 0,
  nchains = 1,
  check = TRUE,
  setSeed = FALSE,
  progressBar = getNimbleOption("MCMCprogressBar"),
  samples = TRUE,
  samplesAsCodaMCMC = FALSE,
  summary = FALSE,
  WAIC = FALSE,
  userEnv = parent.frame()
)
```

Arguments

`code`	The quoted code expression representing the model, such as the return value from a call to `nimbleCode`). Not required if `model` is provided.
`constants`	Named list of constants in the model. Constants cannot be subsequently modified. For compatibility with JAGS and BUGS, one can include data values with constants and `nimbleModel` will automatically distinguish them based on what appears on the left-hand side of expressions in `code`.

`data` Named list of values for the data nodes. Values that are NA will not be flagged as data.

`inits` Argument to specify initial values for each MCMC chain. See details.

`dimensions` Named list of dimensions for variables. Only needed for variables used with empty indices in model code that are not provided in constants or data.

`model` A compiled or uncompiled NIMBLE model object. When provided, this model will be used to configure the MCMC algorithm to be executed, rather than using the `code`, `constants`, `data` and `inits` arguments to create a new model object. However, if also provided, the `inits` argument will still be used to initialize this model prior to running each MCMC chain.

`type` A character string specifying the type of HMC sampling to apply, either "NUTS" or "NUTS_classic". See 'help(NUTS)' or 'help(NUTS_classic)' for details of each sampler. The default sampler type is "NUTS".

`monitors` A character vector giving the node names or variable names to monitor. The samples corresponding to these nodes will returned, and/or will have summary statistics calculated. Default value is all top-level stochastic nodes of the model.

`thin` Thinning interval for collecting MCMC samples. Thinning occurs after the initial nburnin samples are discarded. Default value is 1.

`niter` Number of MCMC iterations to run. Default value is 10000.

`nburnin` Number of initial, pre-thinning, MCMC iterations to discard. Default value is 0.

`nchains` Number of MCMC chains to run. Default value is 1.

`check` Logical argument, specifying whether to check the model object for missing or invalid values. Default value is `TRUE`.

`setSeed` Logical or numeric argument. If a single numeric value is provided, R's random number seed will be set to this value at the onset of each MCMC chain. If a numeric vector of length `nchains` is provided, then each element of this vector is provided as R's random number seed at the onset of the corresponding MCMC chain. Otherwise, in the case of a logical value, if `TRUE`, then R's random number seed for the ith chain is set to be `i`, at the onset of each MCMC chain. Note that specifying the argument `setSeed = 0` does not prevent setting the RNG seed, but rather sets the random number generation seed to `0` at the beginning of each MCMC chain. Default value is `FALSE`.

`progressBar` Logical argument. If `TRUE`, an MCMC progress bar is displayed during execution of each MCMC chain. Default value is defined by the nimble package option MCMCprogressBar.

`samples` Logical argument. If `TRUE`, then posterior samples are returned from each MCMC chain. These samples are optionally returned as coda `mcmc` objects, depending on the `samplesAsCodaMCMC` argument. Default value is `TRUE`. See details.

`samplesAsCodaMCMC`

Logical argument. If `TRUE`, then a coda `mcmc` object is returned instead of an R matrix of samples, or when `nchains > 1` a coda `mcmc.list` object is returned containing `nchains` `mcmc` objects. This argument is only used when `samples` is `TRUE`. Default value is `FALSE`. See details.

summary — Logical argument. When TRUE, summary statistics for the posterior samples of each parameter are also returned, for each MCMC chain. This may be returned in addition to the posterior samples themselves. Default value is FALSE. See details.

WAIC — Logical argument. When TRUE, the WAIC (Watanabe, 2010) of the model is calculated and returned. If multiple chains are run, then a single WAIC value is calculated using the posterior samples from all chains. Default value is FALSE. Note that the version of WAIC used is the default WAIC conditional on random effects/latent states and without any grouping of data nodes. See help(waic) for more details.

userEnv — Environment in which if-then-else statements in model code will be evaluated if needed information not found in constants; intended primarily for internal use only.

Details

nimbleHMC provides capability for running multiple MCMC chains, specifying the number of MCMC iterations, thinning, and burn-in, and which model variables should be monitored. It also provides options to return the posterior samples, to return summary statistics calculated from the posterior samples, and to return a WAIC value.

The entry point for this function is providing the code, constants, data and inits arguments, to create a new NIMBLE model object, or alternatively providing an exisiting NIMBLE model object as the model argument.

At least one of samples, summary or WAIC must be TRUE, since otherwise, nothing will be returned. Any combination of these may be TRUE, including possibly all three, in which case posterior samples, summary statistics, and WAIC values are returned for each MCMC chain.

When samples = TRUE, the form of the posterior samples is determined by the samplesAsCodaMCMC argument, as either matrices of posterior samples, or coda mcmc and mcmc.list objects.

Posterior summary statistics are returned individually for each chain, and also as calculated from all chains combined (when nchains > 1).

The inits argument can be one of three things:

(1) a function to generate initial values, which will be executed once to initialize the model object, and once to generate initial values at the beginning of each MCMC chain, or (2) a single named list of initial values which, will be used to initialize the model object and for each MCMC chain, or (3) a list of length nchains, each element being a named list of initial values. The first element will be used to initialize the model object, and once element of the list will be used for each MCMC chain.

The inits argument may also be omitted, in which case the model will not be provided with initial values. This is not recommended.

The niter argument specifies the number of pre-thinning MCMC iterations, and the nburnin argument specifies the number of pre-thinning MCMC samples to discard. After discarding these burn-in samples, thinning of the remaining samples will take place. The total number of posterior samples returned will be floor((niter-nburnin)/thin).

Value

A list is returned with named elements depending on the arguments, unless only one among samples, summary, and WAIC are requested, in which case only that element is returned. These elements

may include samples, summary, and WAIC. When nchains = 1, posterior samples are returned as a single matrix, and summary statistics as a single matrix. When nchains > 1, posterior samples are returned as a list of matrices, one matrix for each chain, and summary statistics are returned as a list containing nchains+1 matrices: one matrix corresponding to each chain, and the final element providing a summary of all chains, combined. If samplesAsCodaMCMC is TRUE, then posterior samples are provided as coda mcmc and mcmc.list objects. When WAIC is TRUE, a WAIC summary object is returned.

Author(s)

Daniel Turek

See Also

configureHMC buildHMC configureMCMC buildMCMC runMCMC

Examples

```
code <- nimbleCode({
    mu ~ dnorm(0, sd = 1000)
    sigma ~ dunif(0, 1000)
    for(i in 1:10) {
        x[i] ~ dnorm(mu, sd = sigma)
    }
})
data <- list(x = c(2, 5, 3, 4, 1, 0, 1, 3, 5, 3))
inits <- function() list(mu = rnorm(1,0,1), sigma = runif(1,0,10))
mcmc.output <- nimbleHMC(code, data = data, inits = inits,
                         monitors = c("mu", "sigma"), thin = 10,
                         niter = 20000, nburnin = 1000, nchains = 3,
                         summary = TRUE, WAIC = TRUE)
```

---

sampler_NUTS *No-U-Turn (NUTS) Hamiltonian Monte Carlo (HMC) Sampler*

Description

The NUTS sampler implements No-U-Turn (NUTS) Hamiltonian Monte Carlo (HMC) sampling following the algorithm of version 2.32.2 of Stan. Internally, any posterior dimensions with bounded support are transformed, so sampling takes place on an unconstrained space. In contrast to standard HMC (Neal, 2011), the NUTS algorithm removes the tuning parameters of the leapfrog step size and the number of leapfrog steps, thus providing a sampling algorithm that can be used without hand-tuning or trial runs.

Usage

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

Arguments

model	An uncompiled nimble model object on which the MCMC will operate.
mvSaved	A nimble modelValues object to be used to store MCMC samples.
target	A character vector of node names on which the sampler will operate.
control	A named list that controls the precise behavior of the sampler. The default values for control list elements are specified in the setup code of the sampler. A description of the possible control list elements appear in the details section.

Details

The NUTS sampler accepts the following control list elements:

- messages. A logical argument, specifying whether to print informative messages (default = TRUE)
- numWarnings. A numeric argument, specifying how many warnings messages to emit (for example, when NaN values are encountered). See additional details below. (default = 0)
- epsilon. A positive numeric argument, specifying the initial step-size value. If not provided, an appropriate initial value is selected.
- gamma. A positive numeric argument, specifying the degree of shrinkage used during the initial period of step-size adaptation. (default = 0.05)
- t0. A non-negative numeric argument, where larger values stabilize (attenuate) the initial period of step-size adaptation. (default = 10)
- kappa. A numeric argument between zero and one, where smaller values give a higher weighting to more recent iterations during the initial period of step-size adaptation. (default = 0.75)
- delta. A numeric argument, specifying the target acceptance probability used during the initial period of step-size adaptation. (default = 0.8)
- deltaMax. A positive numeric argument, specifying the maximum allowable divergence from the Hamiltonian value. Paths which exceed this value are considered divergent, and will not proceed further. (default = 1000)
- M. A vector of positive real numbers, with length equal to the number of dimensions being sampled. Elements of M specify the diagonal elements of the diagonal mass matrix (or the metric) used for the auxiliary momentum variables in sampling. Sampling may be improved if the elements of M approximate the marginal inverse variance (precision) of the (potentially transformed) parameters. (default: a vector of ones).
- warmupMode. A character string, specifying the behavior for choosing the number of warmup iterations. Four values are possible. The value 'default' (the default) sets the number of warmup iterations as the number of burnin iterations (if a positive value for nburnin is used) or half the number of MCMC iterations in each chain (if nburnin = 0). The value 'burnin' sets the number of warmup iterations as the number of burnin iterations regardless of the length of the burnin period. The value 'fraction' sets the number of warmup iterations as fraction*niter, where fraction is the value of the warmup control argument, and niter is the number of MCMC iterations in each chain; in this case, the value of the warmup control argument must be between 0 and 1. The value 'iterations' sets the number of warmup iterations as the value of the warmup control argumnet, regardless of the length of the burnin period or the number of MCMC iterations; in this case the value of warmup must be a non-negative

integer. In all cases, the number of (pre-thinning) samples discarded equals nburnin, as is always the case for MCMC in NIMBLE.

- warmup. Numeric value used in determining the number of warmup iterations. This control argument is only used when warmupMode is 'fraction' or 'iterations'.
- maxTreeDepth. The maximum allowable depth of the binary leapfrog search tree for generating candidate transitions. (default = 10)
- adaptWindow. Number of iterations in the first adaptation window used for adapting the mass matrix (M). Subsequent adaptation windows double in length, so long as enough warmup iterations are available. (default = 25)
- initBuffer. Number of iterations in the initial warmup window, which occurs prior to the first adaptation of the metric M. (default = 75)
- termBuffer. Number of iterations in the final (terminal) warmup window, before which the metric M is not adjusted(default = 50)
- adaptive. A logical argument, specifying whether to do any adaptation whatsoever. When TRUE, specific adaptation routines are controlled by the adaptEpsilon and adaptM control list elements. (default = TRUE)
- adaptEpsilon. A logical argument, specifying whether to perform stepsize adaptation. Only used when adaptive = TRUE. (default = TRUE)
- adaptM. A logical argument, specifying whether to perform adaptation of the mass matrix (metric) M. Only used when adaptive = TRUE. (default = TRUE)
- initializeEpsilon. A logical argument, specifying whether to perform the epsilon (stepsize) initialization routine at the onset of each adaptation window. (default = TRUE)

NaN values may be encountered in the course of the leapfrog procedure. In particular, when the stepsize (epsilon) is too large, the leapfrog procedure can step too far and arrive at an invalid region of parameter space, thus generating a NaN value in the likelihood evaluation or in the gradient calculation. These situation are handled by the sampler by rejecting the NaN value, and reducing the stepsize.

Value

A object of class 'sampler_NUTS'.

Author(s)

Perry de Valpine and Daniel Turek

References

Hoffman, Matthew D., and Gelman, Andrew (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1): 1593-1623.

Stan Development Team. 2023. Stan Modeling Language Users Guide and Reference Manual, 2.32.2. https://mc-stan.org.

Examples

```
code <- nimbleCode({
    b0 ~ dnorm(0, 0.001)
    b1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 10000)
    for(i in 1:N) {
        mu[i] <- b0 + b1 * x[i]
        y[i] ~ dnorm(mu[i], sd = sigma)
    }
})

set.seed(0)
N <- 100
x <- rnorm(N)
y <- 1 + 0.3*x + rnorm(N)
constants <- list(N = N, x = x)
data <- list(y = y)
inits <- list(b0 = 1, b1 = 0.1, sigma = 1)

Rmodel <- nimbleModel(code, constants, data, inits, buildDerivs = TRUE)

conf <- configureMCMC(Rmodel, nodes = NULL)

conf$addSampler(target = c('b0', 'b1', 'sigma'), type = 'NUTS')

Rmcmc <- buildMCMC(conf)
```

---

sampler_NUTS_classic *Classic No-U-Turn (NUTS_classic) Hamiltonian Monte Carlo (HMC) Sampler*

Description

The NUTS_classic sampler implements the original No-U-Turn (NUTS classic) sampler as put forth in Hoffman and Gelman (2014) for performing joint updates of multiple continuous-valued posterior dimensions. This is done by introducing auxiliary momentum variables and using first-order derivatives to simulate Hamiltonian dynamics on this augmented paramter space. Internally, any posterior dimensions with bounded support are transformed, so sampling takes place on an unconstrained space. In contrast to standard HMC (Neal, 2011), the NUTS_classic algorithm removes the tuning parameters of the leapfrog step size and the number of leapfrog steps, thus providing a sampling algorithm that can be used without hand tuning or trial runs.

Usage

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

Arguments

model	An uncompiled nimble model object on which the MCMC will operate.
mvSaved	A nimble modelValues object to be used to store MCMC samples.
target	A character vector of node names on which the sampler will operate.
control	A named list that controls the precise behavior of the sampler. The default values for control list elements are specified in the setup code of the sampler. A description of the possible control list elements appear in the details section.

Details

The NUTS_classic sampler accepts the following control list elements:

- messages. A logical argument, specifying whether to print informative messages. (default = TRUE)
- numWarnings. A numeric argument, specifying how many warnings messages to emit (for example, when NaN values are encountered). See additional details below. (default = 0)
- epsilon. A positive numeric argument, specifying the initial step-size value. If not provided, an appropriate initial value is selected.
- gamma. A positive numeric argument, specifying the degree of shrinkage used during the initial period of step-size adaptation. (default = 0.05)
- t0. A non-negative numeric argument, where larger values stabilize (attenuate) the initial period of step-size adaptation. (default = 10)
- kappa. A numeric argument between zero and one, where smaller values give a higher weighting to more recent iterations during the initial period of step-size adaptation. (default = 0.75)
- delta. A numeric argument, specifying the target acceptance probability used during the initial period of step-size adaptation. (default = 0.65)
- deltaMax. A positive numeric argument, specifying the maximum allowable divergence from the Hamiltonian value. Paths which exceed this value are considered divergent and will not proceed further. (default = 1000)
- M. A vector of positive real numbers, with length equal to the number of dimensions being sampled. Elements of M specify the diagonal elements of the diagonal mass matrix (or the metric) used for the auxiliary momentum variables in sampling. Sampling may be improved if the elements of M approximate the marginal inverse variance (precision) of the (potentially transformed) parameters. (default: a vector of ones).
- warmupMode. A character string, specifying the behavior for choosing the number of warmup iterations. Four values are possible. The value 'default' (the default) sets the number of warmup iterations as the number of burnin iterations (if a positive value for nburnin is used) or half the number of MCMC iterations in each chain (if nburnin = 0). The value 'burnin' sets the number of warmup iterations as the number of burnin iterations regardless of the length of the burnin period. The value 'fraction' sets the number of warmup iterations as fraction*niter, where fraction is the value of the warmup control argument, and niter is the number of MCMC iterations in each chain; in this case, the value of the warmup control argument must be between 0 and 1. The value 'iterations' sets the number of warmup iterations as the value of the warmup control argumnet, regardless of the length of the burnin period or the number of MCMC iterations; in this case the value of warmup must be a non-negative

integer. In all cases, the number of (pre-thinning) samples discarded equals nburnin, as is always the case for MCMC in NIMBLE.

- warmup. Numeric value used in determining the number of warmup iterations. This control argument is only used when warmupMode is 'fraction' or 'iterations'.
- maxTreeDepth. The maximum allowable depth of the binary leapfrog search tree for generating candidate transitions. (default = 10)
- adaptWindow. Number of iterations in the first adaptation window used for adapting the mass matrix (M). Subsequent adaptation windows double in length, so long as enough warmup iterations are available. (default = 25)
- initBuffer. Number of iterations in the initial warmup window, which occurs prior to the first adaptation of the metric M. (default = 75)
- termBuffer. Number of iterations in the final (terminal) warmup window, before which the metric M is not adjusted. (default = 50)
- adaptive. A logical argument, specifying whether to do any adaptation whatsoever. When TRUE, specific adaptation routines are controlled by the adaptEpsilon and adaptM control list elements. (default = TRUE)
- adaptEpsilon. A logical argument, specifying whether to perform stepsize adaptation. Only used when adaptive = TRUE. (default = TRUE)
- adaptM. A logical argument, specifying whether to perform adaptation of the mass matrix (metric) M. Only used when adaptive = TRUE. (default = TRUE)
- initializeEpsilon. A logical argument, specifying whether to perform the epsilon (stepsize) initialization routine at the onset of each adaptation window. (default = TRUE)

NaN values may be encountered in the course of the leapfrog procedure. In particular, when the stepsize (epsilon) is too large, the leapfrog procedure can step too far and arrive at an invalid region of parameter space, thus generating a NaN value in the likelihood evaluation or in the gradient calculation. These situation are handled by the sampler by rejecting the NaN value, and reducing the stepsize.

Value

A object of class 'sampler_NUTS_classic'.

Author(s)

Daniel Turek

References

Hoffman, Matthew D., and Gelman, Andrew (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1): 1593-1623.

Examples

```
code <- nimbleCode({
    b0 ~ dnorm(0, 0.001)
    b1 ~ dnorm(0, 0.001)
    sigma ~ dunif(0, 10000)
    for(i in 1:N) {
        mu[i] <- b0 + b1 * x[i]
        y[i] ~ dnorm(mu[i], sd = sigma)
    }
})

set.seed(0)
N <- 100
x <- rnorm(N)
y <- 1 + 0.3*x + rnorm(N)
constants <- list(N = N, x = x)
data <- list(y = y)
inits <- list(b0 = 1, b1 = 0.1, sigma = 1)

Rmodel <- nimbleModel(code, constants, data, inits, buildDerivs = TRUE)

conf <- configureMCMC(Rmodel, nodes = NULL)

conf$addSampler(target = c('b0', 'b1', 'sigma'), type = 'NUTS_classic')

Rmcmc <- buildMCMC(conf)
```

---

stateNL_NUTS *nimbleList definition used internally in NUTS sampler.*

Description

nimbleList definition used internally in NUTS sampler.

Usage

```
stateNL_NUTS
```

Format

An object of class `list` of length 1.

`treebranchNL_NUTS` *nimbleList definition used internally in NUTS sampler.*

Description

nimbleList definition used internally in NUTS sampler.

Usage

```
treebranchNL_NUTS
```

Format

An object of class `list` of length 1.

Index

∗ **datasets**
    stateNL_NUTS, 17
    treebranchNL_NUTS, 18

addHMC, 2, *5*, *7*
addSampler, *3*, *5*, *7*

buildHMC, *3*, 4, *7*, *11*
buildMCMC, *11*

configureHMC, *3*, *5*, 6, *11*
configureMCMC, *3*, *5*, *7*, *11*

HMC *(*sampler_NUTS*)*, 11
hmc *(*sampler_NUTS*)*, 11

nimbleHMC, 8
NUTS *(*sampler_NUTS*)*, 11
nuts *(*sampler_NUTS*)*, 11
NUTS-classic *(*sampler_NUTS_classic*)*, 14
nuts-classic *(*sampler_NUTS_classic*)*, 14
NUTS_classic *(*sampler_NUTS_classic*)*, 14
nuts_classic *(*sampler_NUTS_classic*)*, 14

runMCMC, *11*

sampler_NUTS, *3*, *5*, *7*, 11
sampler_NUTS_classic, *3*, *5*, *7*, 14
stateNL_NUTS, 17

treebranchNL_NUTS, 18