Package 'EntropyMCMC'

January 20, 2025

Type Package

Title MCMC Simulation and Convergence Evaluation using Entropy and Kullback-Leibler Divergence Estimation

Version 1.0.4

Date 2019-03-08

Description

Tools for Markov Chain Monte Carlo (MCMC) simulation and performance analysis. Simulate MCMC algorithms including adaptive MCMC, evaluate their convergence rate, and compare candidate MCMC algorithms for a same target density, based on entropy and Kullback-Leibler divergence criteria. MCMC algorithms can be simulated using provided functions, or imported from external codes. This package is based upon work starting with Chauveau, D. and Vandekerkhove, P. (2013) <doi:10.1051/ps/2012004> and next articles.

Depends R (>= 3.0)

Imports RANN, parallel, mixtools

Suggests Rmpi, snow

License GPL (>= 3)

LazyLoad yes

Author Didier Chauveau [aut, cre],
Houssam Alrachid [ctb]

Maintainer Didier Chauveau <didier.chauveau@univ-orleans.fr>

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-03-08 17:22:51 UTC

Contents

`EntropyMCMC-package` *(A)MCMC Simulation and Convergence Evaluation using Entropy and Kullback-Leibler Divergence Estimation*

Description

Contains functions to analyse (Adaptive) Markov Chain Monte Carlo (MCMC) algorithms, evaluate their convergence rate, and compare candidate MCMC algorithms for a same target density, based on entropy and Kullback-Leibler divergence criteria. MCMC algorithms can be simulated using provided functions, or imported from external codes. The diagnostics are based on consistent estimates of entropy and Kulback distance between the density at iteration t and the target density f, based on iid (parallel) chains.

Details

Package:	EntropyMCMC
Type:	Package
Version:	1.0.4
Date:	2019-03-08
License:	GPL (>= 3)
LazyLoad:	yes

Statistical background:

This package allows for simulation of standard or adaptive MCMC samplers for a user-defined target density, and provides statistical tools to evaluate convergence of MCMC's and compare performance of algorithms for the same target density (typically against benchmark samplers).

The criteria are graphical and based on plots against iterations (time) t, of the *Kullback divergence* $K(p^t, f)$ between the density p^t of the MCMC algorithm at time t, and the target density f, for $t = 1$ up to the number of iterations that have been simulated. This requires estimation of the entropy of p^t,

$$E_{p^t}[\log(p^t)],$$

and of the external entropy

$$E_{p^t}[\log(f)].$$

Consistent estimates are computed based on N iid (parallel) chains, since the N positions of the chains at iterations t forms a N-iid sample from the density p^t.

Computational considerations:

The simulation of iid chains can be performed in this package, which provides a mechanism for defining (A)MCMC algorithms and building the iid chains required for convergence evaluation. Each MCMC algorithm is defined by a list with five elements. Each user can define its own MCMC, starting from the standard MCMC algorithms that are already defined:

- `RWHM`: a standard Randow-Walk Hastings-Metropolis (HM) algorithm.
- `HMIS_norm`: an Independence Sampler HM with gaussian proposal
- `AMHaario`: the Haario (2001) Adaptive Hastings-Metropolis algorithm, provided as an example of a standard AMCMC.
- `IID_norm`: a "fake" MCMC that is just a gaussian IID sampler, used mostly for testing purpose. Simulation of N iid chains for n iterations using this algorithm just returns $N \times n$ gaussian d-dimensional vectors.

Functions for doing the simulations and the convergence evaluation automatically using these algorithms in their first argument are provided. Two strategies are available:

- *Simulation and Kullback estimation separately:* A "cube" of N chains for n iterations in a space of dimension d is first simulated and stored using `MCMCcopies` or its multicore or cluser versions, then the entropy and Kullback divergence are estimated from that object using `EntropyMCMC` or its multicore version.
- *Simulation and Kullback estimation simultaneously:* For each iteration t, the next step of all the `N` chains are generated, then the Entropy and Kullback divergence $K(p^t, f)$ are estimated, and the past of the parallel chains is discarded so that the amount of memory requirement is kept small, and only entropy-related estimates are stored and returned. Functions for this strategy are `EntropyParallel` and its multicore and cluster version.

See the Examples section of `plot_Kblist` for an illustration of these two methods.

Doing the simulations outside from this package

A third hybrid strategy is also available: the simulation of iid chains can be done using an external code (in R, `C` or any language) and imported in the **EntropyMCMC** package (defining an object of the appropriate class `"plMCMC"` and structure, see `MCMCcopies`).

Then the Kullback divergence criterion can be computed using `EntropyMCMC` or its multicore version, and convergence/comparison diagnostics can be displayed using the associated `plot` method.

About High Performance Computing

The required simulations can be done using singlecore or HCP (multicore computers, snow or clusters using the **parallel** or **Rmpi** pakages). Note that the **parallel** package using socket cluster is not available on Windows machines.

Author(s)

Didier Chauveau, Institut Denis Poisson, University of Orleans, CNRS, Orleans France. https://www.idpoisson.fr/chauveau/

Maintainer: Didier Chauveau <didier.chauveau@univ-orleans.fr>

Contributor: Houssam Alrachid

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

`accept_ratio` *Acceptance ratio for Hastings-Metropolis simulated MCMC chains*

Description

Internal function for the package `EntropyMCMC`, computes the acceptance ratio required in the definition of any Hastings-Metropolis algorithm.

Usage

```
accept_ratio(x, y, target, q_pdf, f_param, q_param, symmetric = FALSE)
```

Arguments

`x`	The current position.
`y`	The next (proposal) position.
`target`	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. Target must be a function such as the multidimensional gaussian `target_norm(x,param)`.
`q_pdf`	The density of the proposal.
`f_param`	A list holding all the necessary target parameters, consistent with the target definition.
`q_param`	A list holding all the necessary parameters for the proposal density of the MCMC algorithm `mcmc_algo`.
`symmetric`	If `TRUE`, the proposal `q_pdf` is symmetric which simplifies the acceptance ratio compuatation

Details

The `accept_ratio` is used to decide whether to accept or reject a candidate y. The acceptance ratio indicates how probable the new proposed candidate is with respect to the current candidate x, according to the distribution `target`.

Value

`accept_ratio` returns a real value `alpha`, which indicates the computed value of the current `accept_ratio`.

Author(s)

Didier Chauveau, Houssam Alrachid.

`CollectChains` *Collect MCMC chains in a single object*

Description

Utility function for the package **EntropyMCMC**.

Usage

```
CollectChains(s)
```

Arguments

`s`	An object of class `plMCMC`, such as the one returned by `MCMCcopies`, containing in particular an array of dimension (n, d, nmc) holding the simulation of n steps of nmc parallel chains in dimension d.

Details

Concatenates all simulated copies together in one matrix of dimension `(n*nmc,d)`.

Value

Returns a matrix of dimension `(n*nmc,d)`.

Author(s)

Didier Chauveau.

DrawInit *Random draws for initialization*

Description

Utility function for the package **EntropyMCMC**, for generating random starting positions for the parallel Markov chains, used by, e.g., MCMCcopies or EntropyParallel.

Usage

```
DrawInit(nmc, d, initpdf="rnorm", ...)
```

Arguments

nmc	Number of parallel chains = initial points.
d	Space dimension.
initpdf	Random generator. Generators currently implemented are: "rnorm" as the Normal distribution and "runif" as the uniform distribution.
...	Parameters passed to initpdf

Value

DrawInit returns a matrix of dimension (nmc,d) where each row is a d-dimensional point.

Note

It is better for mixing properties to use diffuse initial distributions, such as the one proposed here. However Dirac initial points can also be used, precisely to evaluate the efficiency of a MCMC to escape from a wrong initial position (e.g., in the tails of the target density).

Author(s)

Didier Chauveau.

See Also

MCMCcopies and MCMCcopies.mc for iid MCMC simulations, EntropyParallel and EntropyParallel.cl for simultaneous simulation and entropy estimation.

Examples

```
Ptheta0 <- DrawInit(10, 5, initpdf="rnorm", mean=0, sd=5)
```

EntropyMCMC *Kullback and entropy estimation from MCMC simulation output - single and multicore versions*

Description

These functions return estimates of the entropy of the density p^t of a MCMC algorithm at time t, $E_{p^t}[\log(p^t)]$, and of the Kullback divergence between p^t and the target density, for $t = 1$ up to the number of iterations that have been simulated. The MCMC simulations must be computed before or externally, and passed as a "plMCMC" object in the first argument (see details). The target may be known only up to a multiplicative constant (see details).

EntropyMCMC.mc is a parallel computing version that uses the **parallel** package to split the task between the available (virtual) cores on the computer. This version using socket cluster is not available for Windows computers.

Usage

```
EntropyMCMC(plmc1, method = "A.Nearest.Neighbor", k=1, trim = 0.02, eps=0,
        all.f = TRUE, verb = FALSE, EntVect = FALSE,
        uselogtarget = FALSE, logtarget = NULL)

EntropyMCMC.mc(plmc1, method = "A.Nearest.Neighbor", k = 1, trim = 0.02, eps=0,
        all.f = TRUE, verb = FALSE, EntVect = FALSE, nbcores=detectCores(),
    uselogtarget = FALSE, logtarget = NULL)
```

Arguments

plmc1	an objects of class plMCMC (for parallel MCMC), like the output of MCMCcopies, which contains all the simulations plus target f definition and parameters.
method	The method for estimating the entropy $E_{p^t}[\log(p^t)]$. Methods currently implemented are : "NearestNeighbor" as in Kozachenko and Leonenko (1987), "k.NearestNeighbor" as in Leonenko et al. (2005), "A.Nearest.Neighbor" (the default) which is as "k.NearestNeighbor" but uses the **RANN** package for (Approximate) fast computation of nearest neighbors, "Gyorfi.trim" subsampling method as defined in Gyorfi and Vander Mulen (1989), plus a tuning parameter trim for trimming the data (see Chauveau and Vandekerkhove (2011)).
k	The k-nearest neighbor index, the default is $k = 1$.
trim	Parameter controlling the percentage of smallest data from one subsample that is removed, only for method = "Gyorfi.trim".
eps	A parameter controlling precision in the "A.Nearest.Neighbor"" method, the default means no approximation, see the **RANN** package.
all.f	If TRUE (the default) relative entropy is computed over the whole sample. Should be removed in next version.

verb	Verbose mode
EntVect	If FALSE (the default), the entropy is computed only on the kth-nearest neighbor. If TRUE, the entropy is computed for all j-NN's for $j = 1$ to k (the latter being mostly for testing purposes).
nbcores	Number of required (virtual) cores, defaults to all as returned by detectCores().
uselogtarget	Set to FALSE by default; useful in some cases where $log(f(\theta))$ returns -Inf values in Kullback computations because $f(\theta)$ itself returns too small values for some θ far from modal regions. In these case using a function computing the logarithm of the target can remove the infinity values.
logtarget	The function defining $log(f(theta))$, NULL by default, required if uselogtarget equals TRUE. This option and uselogtarget are currently implemented only for the "A.Nearest.Neighbor" method, and for the default EntVect = FALSE option.

Details

Methods based on Nearest Neighbors (NN) should be preferred since these require less tuning parameters. Some options, as uselogtarget are in testing phase and are not implemented in all the available methods (see Arguments).

Value

An object of class KbMCMC (for Kullback MCMC), containing:

Kullback	A vector of estimated divergences $K(p^t, f)$, for $t = 1$ up to the number of iterations that have been simulated. This is the convergence/comparison criterion.
Entp	A vector of estimated entropies $E_{p^t}[\log(p^t)]$, for $t = 1$ up to the number of iterations that have been simulated.
nmc	The number of iid copies of each single chain.
dim	The state space dimension of the MCMC algorithm.
algo	The name of the MCMC algorithm that have been used to simulate the copies of chains, see MCMCcopies.
target	The target density for which the MCMC algorithm is defined; ususally given only up to a multiplicative constant for MCMC in Bayesian models. target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in the example below.
method	The method input parameter (see above).
f_param	A list holding all the necessary target parameters, consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm that have been used.

Note

The method "Resubst" is implemented for testing, without theoretical guarantee of convergence.

Author(s)

Didier Chauveau, Houssam Alrachid.

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

`MCMCcopies` and `MCMCcopies.mc` for iid MCMC simulations (single core and multicore), `EntropyParallel` and `EntropyParallel.cl` for simultaneous simulation and entropy estimation (single core and multicore).

Examples

```
## Toy example using the bivariate gaussian target
## with default parameters value, see target_norm_param
n = 150; nmc = 50; d=2 # bivariate example
varq=0.1 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)
# simulation of the nmc iid chains, singlecore
s1 <- MCMCcopies(RWHM, n, nmc, Ptheta0, target_norm,
                 target_norm_param, q_param, verb = FALSE)
summary(s1) # method for "plMCMC" object
e1 <- EntropyMCMC(s1) # computes Entropy and Kullback divergence estimates
par(mfrow=c(1,2))
plot(e1) # default plot.plMCMC method, convergence after about 80 iterations
plot(e1, Kullback = FALSE) # Plot Entropy estimates over time
abline(normEntropy(target_norm_param), 0, col=8, lty=2) # true E_f[log(f)]
```

`EntropyParallel` *Parallel simulation and Entropy estimation of MCMC's - single core and cluster versions*

Description

This function simulates "parallel chains" (iid copies) of a MCMC algorithm, i.e. for each "time" iteration t the next step of all the nmc chains are generated, then the Entropy of the density p^t of the algorithm at iteration t, $E_{p^t}[\log(p^t)]$, and the Kullback divergence between p^t and the target density are estimated, based on these nmc steps iid from p^t. By default keep.all = FALSE i.e. the past of the parallel chains is discarded so that the amount of memory requirement is kept small, and only entropy-related estimates are returned. If keep.all = TRUE, the entire set of chains trajectories is saved in an array of dimensions (n,d,nmc), such as the one returned by MCMCcopies or MCMCcopies.cl.

A version of this function implementing several HPC (parallel) computing strategies is available (see details).

Usage

```
EntropyParallel(mcmc_algo, n = 100, nmc = 10, Ptheta0, target, f_param, q_param,
          method = "A.Nearest.Neighbor",k=1, trim = 0.02, keep.all = FALSE,
          verb = TRUE, EntVect = FALSE)

EntropyParallel.cl(mcmc_algo, n = 100, nmc = 10, Ptheta0, target, f_param, q_param,
          method = "A.Nearest.Neighbor",k=1, eps = 0, trim=0.02,
          verb=TRUE, EntVect = FALSE, cltype="PAR_SOCK", nbnodes = 4,
          par.logf = FALSE, uselogtarget = FALSE, logtarget = NULL)
```

Arguments

mcmc_algo	a list defining an MCMC algorithm in terms of the functions it uses, such as RWHM, see details below.
n	The number of (time) iterations of each single chain to run.
nmc	The number of iid copies of each single chain.
Ptheta0	A (nmc,d) matrix, with the ith row giving a d-dimensional initial theta values for the ith chain.
target	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in the example below.
f_param	A list holding all the necessary target parameters, including the data in an actual Bayesian model, and consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm mcmc_algo.
method	The method for estimating the entropy $E_{p^t}[\log(p^t)]$. The methods currently implemented for this function are "Nearest.Neighbor" as in Kozachenko and Leonenko (1987), "k.Nearest.Neighbor" as in Leonenko et al. (2005) (the default in the single core version), and "A.Nearest.Neighbor" which is as "k.NearestNeighbor" using the **RANN** package for (Approximate) fast computation of nearest neighbors, instead of R code (the default for the cluster version). Other methods such as "Gyorfi.trim" subsampling method as defined

	in Gyorfi and Vander Mulen (1989) are available as well (see Chauveau and Vandekerkhove (2012)).
k	The k-nearest neighbor index, the default is $k = 1$.
eps	Error bound: default of 0.0 implies exact nearest neighbour search, see the **RANN** package.
trim	not used in this implementation, only for `method="Gyorfi.trim"`
keep.all	If `TRUE`, all the simulated chains are stored in a 3-dimensional array of dimensions `(n,d,nmc)`, such as the one returned by `MCMCcopies`
verb	Verbose mode for summarizing output during the simulation.
EntVect	If `FALSE` (the default), the entropy is computed only on the kth-nearest neighbor. If `TRUE`, the entropy is computed for all j-NN's for $j = 1$ to k (the latter being mostly for testing purposes).
cltype	Character string specifying the type of cluster; currently implemented types are: "PAR_SOCK" for socket cluster with `parallel` library, the default; "SNOW_SOCK" for socket cluster with `snow` library, and "SNOW_RMPI" for `snow` MPI cluster with `Rmpi` library.
nbnodes	The number of nodes or virtual cores requested to run the `nmc` simulations in parallel. For the snow version, defaults to all; for the cluster version, defaults to 4.
par.logf	if `TRUE`, then the computation of the log of the target density at each of the `nmc` chain locations, needed for the NN procedure is also executed in parallel using `parRapply()`. This may speed up the process if the target is complicated i.e. takes some time to evaluate. If the target is simple enough (like `target_norm`), then communications between nodes are slower than computations, in which case `par.logf = FALSE` (the default) should be preferred.
uselogtarget	Set to `FALSE` by default; useful in some cases where $log(f(\theta))$ returns `-Inf` values in Kullback computations because $f(\theta)$ itself returns too small values for some θ far from modal regions. In these case using a function computing the logarithm of the target can remove the infinity values.
logtarget	The function defining $log(f(theta))$, `NULL` by default, required if `uselogtarget` equals `TRUE`. This option and `uselogtarget` are currently implemented only for the "A.Nearest.Neighbor" method, and for the default `EntVect = FALSE` option.

Details

About parallel computing:

For the HPC (parallel) version, the computation of the `nmc` chains next step are done by the cluster nodes: `EntropyParallel.cl` is a generic *cluster* version implementing several types of cluster for running on a single, multicore computer or on a true cluster using MPI communications. It is under development and may not work on all platform/OS. For instance the parallel socket cluster version does not work on Windows machines (see the **parallel** package documentation). Currently tested under LINUX, Mac OSX, and a cluster using OpenMPI and Sun Grid Engine.

Note that the parallel computing for this approach is less efficient than the two-steps procedure consisting in (i) parallel simulation of the iid chains using `MCMCcopies.cl` to generate the "cube" of simulated values, and then (ii) entropy and Kullback estimation using `EntropyMCMC.mc`. This

is because each node computes only one iteration at a time for the nmc chains here, whereas it computes all the n iterations once for the nmc chains when the entire cube is saved first. This is a trade-off between memory and speed.

Note also that the Rmpi option is less efficient than the default option using **parallel** if you are running on a single computer. MPI communication are required only for running on a true cluster/grid.

About passing your MCMC algorithm:

The list mcmc_algo must contain the named elements:

- name, the name of the MCMC, such as "RWHM"
- chain, the function for simulation of n steps of a single chain
- step, the function for simulation of 1 step of that algorithm
- q_pdf, the density of the proposal
- q_proposal, the function that simulates a proposal

For examples, see the algorithms currently implemented: RWHM, the Random Walk Hasting-Metropolis with gaussian proposal; HMIS_norm, an Independence Sampler HM with gaussian proposal; IID_norm, a gaussian iid sampler which is merely a "fake" MCMC for testing purposes.

Currently only non-adaptive Markov chains or adaptive chains for which the past can be summarized by some sufficient statistics are eligible for this computation forgetting the past of the nmc chains. Adaptive chains such as AMHaario, the Adaptive-Metropolis (AM) from Haario (2001) are not yet tested for this function.

Value

An object of class "KbMCMC", containing

Kullback	A vector of estimated $K(p^t, f)$, for $t = 1$ up to the number of iterations n. This is the convergence/comparison criterion.
Entp	A vector of estimated $E_{p^t}[\log(p^t)]$, for $t = 1$ up to the number of iterations that have been simulated.
nmc	The number of iid copies of each single chain.
dim	The state space dimension of the MCMC algorithm.
algo	The name of the MCMC algorithm that have been used to simulate the copies of chains, see MCMCcopies.
target	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in this example.
method	The method input parameter (see above).
f_param	A list holding all the necessary target parameters, consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm that have been used.
prob.accept	Estimated rate of acceptation (meaningful for accept/reject-type algorithms).
Ptheta	The nmc copies of chains in an array(n,d,nmc) of simulated values, where 1st value (1,d,nmc) is Ptheta0.

Author(s)

Didier Chauveau, Houssam Alrachid.

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

MCMCcopies, MCMCcopies.mc and MCMCcopies.cl for just simulating the iid chains, and EntropyMCMC or EntropyMCMC.mc for computing entropy and Kullback estimates from an already simulated set of iid chains (internally or from external code).

Examples

```
## Toy example using the bivariate gaussian target
## same as for MCMCcopies
n = 150; nmc = 50; d=2 # bivariate example
varq=0.1 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)
# simulations and entropy + Kullback using the singlecore version
e1 <- EntropyParallel(RWHM, n, nmc, Ptheta0, target_norm,
                target_norm_param, q_param, verb = FALSE)
par(mfrow=c(1,2))
plot(e1) # default plot.plMCMC method, convergence after about 80 iterations
plot(e1, Kullback = FALSE) # Plot Entropy estimates over time
abline(normEntropy(target_norm_param), 0, col=8, lty=2) # true E_f[log(f)]

# Another example using multicore version, (not available on Windows)
varq=0.05 # variance of the proposal, even smaller
q_param=list(mean=rep(0,d),v=varq*diag(d))
n=300 # requires more iterations to show convergence
e1 <- EntropyParallel.cl(RWHM, n, nmc, Ptheta0, target_norm,
                         target_norm_param, q_param, cltype="PAR_SOCK",
                         verb = FALSE, nbnodes = 2)
plot(e1) # convergence after about 150 iterations
```

gaussian_pdf *Proposal density evaluation and simulation*

Description

Functions for proposal density evaluation and random generation in MCMC algorithms, in the case where these are Gaussian.

Usage

```
gaussian_pdf(y, x, param)

gaussian_proposal(x, param)
```

Arguments

y	Candidate for next move, a vector of dimension d
x	Current position of a chain, a vector of dimension d
param	The proposal parameters, that must contains the $d \times d$ variance matrix in `param$v`.

Details

The Gaussian proposal density $q(y|x)$ used in, e.g., random walk Hastings-Metropolis algorithm `RWHM` is the multivariate Gaussian $N(x, v)$ density evaluated at point y. Similarly, the Gaussian proposal (next move) is a random draw $y \sim N(x, v)$ when the chain is at position x.

Value

The value of the density, or the random draw, both in dimension d

Note

These functions are calling multivariate Gaussian density and random generation functions imported from the **mixtools** package (chosen for efficiency) and wrapped in the format required by the **EntropyMCMC** package.

Author(s)

Didier Chauveau.

MCMCcopies *Simulates iid copies of a MCMC algorithm*

Description

Simulates nmc iid copies of a MCMC algorithm mcmc_algo for n (time) iterations and returns an object of class plMCMC (for parallel MCMC) holding an array of the trajectories and running information.

Usage

```
MCMCcopies(mcmc_algo, n = 100, nmc = 10, Ptheta0, target, f_param, q_param, verb = TRUE)
```

Arguments

mcmc_algo	a list defining an MCMC algorithm in terms of the functions it uses, such as RWHM, see details below.
n	The number of (time) iterations of each single chain to run.
nmc	The number of iid copies of each single chain.
Ptheta0	A (nmc x d) matrix, with the ith row giving a d-dimensional initial theta values for the ith chain.
target	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. Target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in this example.
f_param	A list holding all the necessary target parameters, consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm mcmc_algo.
verb	Verbose mode for summarizing output during the simulation.

Details

MCMCcopies sequentially simulates nmc iid copies of the MCMC algorithm passed in the list mcmc_algo, for n (time) iterations, and returns an object of class plMCMC holding an array of the trajectories and running information. The list mcmc_algo must contain the named elements:

- name, the name of the MCMC, such as "RWHM"
- chain, the function for simulation of n steps of a single chain
- step, the function for simulation of 1 step of that algorithm
- q_pdf, the density of the proposal
- q_proposal, the function that simulates a proposal

For examples, see the algorithms currently implemented: RWHM, the Random Walk Hasting-Metropolis with gaussian proposal; HMIS_norm, an Independence Sampler HM with gaussian proposal; AMHaario, the Adaptive-Metropolis (AM) from Haario (2001); IID_norm, a gaussian iid sampler which is merely a "fake" MCMC for testing purposes.

Value

`MCMCcopies` returns a list of class `plMCMC` with items:

`Ptheta`	The `nmc` copies of chains in an array(n,d,nmc) of simulated values, where 1st value (1,d,nmc) is `Ptheta0`.
`prob.accept`	The estimated rate of acceptation over all simulations.
`algo`	The MCMC algorithm name i.e. `mcmc_algo$name`.
`target`	The target density.
`f_param`	The list holding all the target parameters.
`q_param`	The list holding all the proposal density parameters.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.

See Also

Two multicore and cluster version `MCMCcopies.mc` and `MCMCcopies.cl`, and functions doing simulation and entropy and Kullback estimation simultaneously: `EntropyParallel` and `EntropyParallel.cl`

Examples

```
## Toy example using the bivariate gaussian target
## with default parameters value, see target_norm_param
n = 150; nmc = 20; d=2 # bivariate example
varq=0.1 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)
# simulation
s1 <- MCMCcopies(RWHM, n, nmc, Ptheta0, target_norm,
                 target_norm_param, q_param, verb = FALSE)
summary(s1) # method for "plMCMC" object
par(mfrow=c(1,2))
plot(s1) # just a path of the iid chains, method for "plMCMC" object
hist(s1$Ptheta[,1,], col=8) # marginal 1
```

MCMCcopies.cl *Parallel simulation of iid copies of a MCMC algorithm - cluster versions*

Description

This function simulates "parallel chains" (iid copies) of a MCMC algorithm for n (time) iterations, i.e. for each chain k, the whole trajectory of the chain is generated. It returns an object of class plMCMC (for parallel MCMC) holding an array of the trajectories and running information. This functions is similar to MCMCcopies and MCMCcopies.mc except that it uses HPC in a more generic way, implementing several types of HPC for running on a single, multicore computer or on a true cluster using MPI communications.

Usage

```
MCMCcopies.cl(mcmc_algo, n=100, nmc=10, Ptheta0, target, f_param, q_param,
              cltype="PAR_SOCK", nbnodes=4)
```

Arguments

mcmc_algo	a list defining an MCMC algorithm in terms of the functions it uses, such as RWHM, see details below.
n	The number of (time) iterations of each single chain to run.
nmc	The number of iid copies of each single chain.
Ptheta0	A $(nmcxd)$ matrix, with the ith row giving a d-dimensional initial theta values for the ith chain.
target	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in this example.
f_param	A list holding all the necessary target parameters, consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm mcmc_algo.
cltype	Character string specifying the type of cluster; currently implemented types are: "PAR_SOCK" for socket cluster with parallel library, the default; "SNOW_SOCK" for socket cluster with snow library, and "SNOW_RMPI" for snow MPI cluster with Rmpi library.
nbnodes	The number of nodes or virtual cores requested to run the nmc simulations in parallel. For the snow version, defaults to all; for the cluster version, defaults to 4.

Details

MCMCcopies.cl simulates in parallel nmc iid copies of the MCMC algorithm passed in the list mcmc_algo, for n (time) iterations, and returns an object of class plMCMC holding an array of the trajectories and running information.

About parallel computing:

The Rmpi option is less efficient than the default option using **parallel** if you are running on a single computer. MPI communication are required only for running on a true cluster/grid.

This generic *cluster* version implementing several types of cluster for running on a single, multicore computer or on a true cluster using MPI communications may not work on all platform/OS. For instance the parallel socket cluster version does not work on Windows machines (see the **parallel** package documentation).

About passing your MCMC algorithm:

The list mcmc_algo must contain the named elements:

- name, the name of the MCMC, such as "RWHM"
- chain, the function for simulation of n steps of a single chain
- step, the function for simulation of 1 step of that algorithm
- q_pdf, the density of the proposal
- q_proposal, the function that simulates a proposal

For examples, see the algorithms currently implemented: RWHM, the Random Walk Hasting-Metropolis with gaussian proposal; HMIS_norm, an Independence Sampler HM with gaussian proposal; AMHaario, the Adaptive-Metropolis (AM) from Haario (2001); IID_norm, a gaussian iid sampler which is merely a "fake" MCMC for testing purposes.

Value

MCMCcopies.cl returns a list of class plMCMC with items:

Ptheta	The nmc copies of chains in an array(n,d,nmc) of simulated values, where 1st value (1,d,nmc) is Ptheta0.
prob.accept	The estimated rate of acceptation over all simulations.
algo	The MCMC algorithm name i.e. mcmc_algo$name.
target	The target density.
f_param	The list holding all the target parameters.
q_param	The list holding all the proposal density parameters.

Author(s)

Houssam Alrachid and Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

A simpler cluster version `MCMCcopies.mc`, a single core version `MCMCcopies`, and functions doing simulation and entropy and Kullback estimation simultaneously: `EntropyParallel` and `EntropyParallel.cl`

Examples

```
## Toy example using the bivariate gaussian target

n = 150; nmc = 20; d=2 # bivariate example
varq=0.1 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)
# simulations (may be compared with the singlecore version using system.time)
s1 <- MCMCcopies.cl(RWHM, n, nmc, Ptheta0, target_norm,
              target_norm_param, q_param, nbnodes = 2)
summary(s1) # method for "plMCMC" object

## see MCMCcopies example for plots
```

`MCMCcopies.mc` *Simulates iid copies of a MCMC algorithm - multicore version*

Description

Simulates `nmc` iid copies of a MCMC algorithm `mcmc_algo` for `n` (time) iterations and returns an object of class `plMCMC` (for parallel MCMC) holding an array of the trajectories and running information. This functions is similar to `MCMCcopies` except that it uses the **parallel** package (available in the main distribution, but not for Windows machines) to split the task between the available virtual cores on the computer.

Usage

```
MCMCcopies.mc(mcmc_algo, n = 100, nmc = 10, Ptheta0, target, f_param, q_param,
verb = TRUE, nbcores=detectCores())
```

Arguments

mcmc_algo	a list defining an MCMC algorithm in terms of the functions it uses, such as RWHM, see details below.
n	The number of (time) iterations of each single chain to run.
nmc	The number of iid copies of each single chain.
Ptheta0	A $(nmcxd)$ matrix, with the ith row giving a d-dimensional initial theta values for the ith chain.
target	The target density for which the MCMC algorithm is defined; may be given only up to a multiplicative constant for most MCMC. target must be a function such as the multidimensional gaussian target_norm(x,param) with argument and parameters passed like in this example.
f_param	A list holding all the necessary target parameters, consistent with the target definition.
q_param	A list holding all the necessary parameters for the proposal density of the MCMC algorithm mcmc_algo.
verb	Verbose mode for summarizing output during the simulation.
nbcores	Number of required (virtual) cores, defaults to all as returned by detectCores().

Details

MCMCcopies.mc, like MCMCcopies, sequentially simulates nmc iid copies of the MCMC algorithm passed in the list mcmc_algo, for n (time) iterations, and returns an object of class plMCMC holding an array of the trajectories and running information. The list mcmc_algo must contain the named elements:

- name, the name of the MCMC, such as "RWHM"
- chain, the function for simulation of n steps of a single chain
- step, the function for simulation of 1 step of that algorithm
- q_pdf, the density of the proposal
- q_proposal, the function that simulates a proposal

For examples, see the algorithms currently implemented: RWHM, the Random Walk Hasting-Metropolis with gaussian proposal; HMIS_norm, an Independence Sampler HM with gaussian proposal; AMHaario, the Adaptive-Metropolis (AM) from Haario (2001); IID_norm, a gaussian iid sampler which is merely a "fake" MCMC for testing purposes.

Value

MCMCcopies returns a list of class plMCMC with items:

Ptheta	The nmc copies of chains in an array(n,d,nmc) of simulated values, where 1st value (1,d,nmc) is Ptheta0.
prob.accept	The estimated rate of acceptation over all simulations.
algo	The MCMC algorithm name i.e. mcmc_algo$name.
target	The target density.
f_param	The list holding all the target parameters.
q_param	The list holding all the proposal density parameters.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2013), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

A more general cluster version `MCMCcopies.cl`, a single core version `MCMCcopies`, and functions doing simulation and entropy and Kullback estimation simultaneously: `EntropyParallel` and `EntropyParallel.cl`

Examples

```
## Toy example using the bivariate gaussian target

## not working on Windows since socket cluster not implemented
n = 150; nmc = 20; d=2 # bivariate example
varq=0.1 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)
# simulations (may be compared with the singlecore version using system.time)
s1 <- MCMCcopies.mc(RWHM, n, nmc, Ptheta0, target_norm,
                    target_norm_param, q_param, nbcores = 2)
summary(s1) # method for "plMCMC" object

## see MCMCcopies example for plots
```

---

normEntropy *Theoretical value of the entropy for the multivariate gaussian*

Description

This function computes the entropy $E_f[\log(f)]$ of the density of the multivariate gaussian, with parameters in a list, as it is the case for MCMC target density parameters. This function is used mostly for benchmarking entropy estimation performed by the package (using, e.g., the iid algorithm `IID_norm`).

Usage

```
normEntropy(target_param)
```

Arguments

target_param	A list of two elements: the mean target_param$mean and the covariance matrix target_param$v.

Value

The entropy of the Gaussian with these parameters.

Author(s)

Didier Chauveau.

Examples

```
d=2 # model dimension
mu=rep(0,d); v = diag(d) # mean and variance
target_param = list(mean=mu, v=v) # parameters
normEntropy(target_param) # the entropy
```

---

plot.KbMCMC *Plot sequences of estimates of Kullback distance or Entropy against iterations*

Description

This S3 method for plot plots by default sequences of estimates of the Kullback distance $K(p^t, f)$ between the (estimated) pdf of the MCMC algorithm at time t, p^t, and the target density f, for $t = 1$ up to the number of iterations that have been provided/computed. It can also plot the first term in the Kullback distance, i.e. the Entropy $E_{p^t}[\log(p^t)]$. Its argument is an object of class KbMCMC such as the one returned by, e.g., EntropyMCMC.

Usage

```
## S3 method for class 'KbMCMC'
plot(x, Kullback = TRUE, lim = NULL, ylim = NULL,
            new.plot = TRUE, title = NULL, ...)
```

Arguments

x	An object of class KbMCMC, such as the one returned by EntropyMCMC.
Kullback	TRUE to plot the Kullback distance, FALSE to plot the Entropy.
lim	for zooming over 1:lim iterations only.
ylim	y limits, passed to plot.

new.plot	set to FALSE to add the plot to an existing plot.
title	The title; if NULL, then a default title is displayed.
...	Further parameters passed to plot or lines.

Value

The graphic to plot.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2012), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, (2013) 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

EntropyMCMC, EntropyMCMC.mc

Examples

```
## See the EntropyMCMC Examples.
```

---

plot.plMCMC *Plot paths of copies of Markov chains*

Description

This function plots 2d-projections of the paths of i.i.d. copies of Markov chains output by an MCMC algorithm and stored in an object of class plMCMC (for parallel MCMC) such as the one returned by, e.g., MCMCcopies or the multicore version MCMCcopies.mc.

Usage

```
## S3 method for class 'plMCMC'
plot(x, xax = 1, yax = 2, title = NULL, cname = NULL, ...)
```

Arguments

x	An object of class plMCMC, such as output from MCMCcopies.
xax	Coordinate for the horizontal axis.
yax	Coordinate for the vertical axis.
title	The title; if NULL, then a default title is displayed.
cname	Coordinate base name; "var" is the default, so that coordinates are named "var1", "var2", and so on.
...	Further parameters except pch which is already used, passed to plot.

Details

This function is currently limited to a 2D projection path of all the i.i.d. chains for the two selected coordinates. The copies of the Markov chain must be in the 3-dimensional array s$Ptheta.

Value

The graphic to plot.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2012), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, (2013) 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

MCMCcopies, MCMCcopies.mc, MCMCcopies.cl

Examples

```
## See MCMCcopie Example
```

plottarget3d *3D plot of a two-dimensional MCMC target, or any function*

Description

Utility function for the package **EntropyMCMC**, to visualize a 2-dimensional target of a MCMC algorithm, mostly for testing purpose. This uses the function `persp` from package **graphics**.

Usage

```
plottarget3d(zft, l, r, ms, theta, phi, ...)
```

Arguments

`zft`	a function, typically a 2-dimensional target of a MCMC.
`l`, `r`, `ms`	mesh boundaries and size.
`theta`, `phi`	angles defining the viewing direction. *theta* gives the azimuthal direction and *phi* the colatitude.
`...`	additional graphical parameters.

Value

Returns a 3D plot on a mesh of size (`l`, `r`, `ms`).

Author(s)

Didier Chauveau.

plot_Kblist *Plot sequences of Kullback distance estimates for comparison of several MCMC algorithms for a same target density*

Description

This function draws on a same plot several sequences of estimates of Kullback distances $K(p^t, f)$, i.e. the convergence criterion vs. time (iteration t), for each MCMC algorithm for which the convergence criterion has been computed.

Usage

```
plot_Kblist(Kb, which = 1, lim = NULL, ylim = NULL)
```

Arguments

Kb	A list of objects of class "KbMCMC", such as the ones returned by EntropyMCMC or EntropyParallel, or their HPC versions.
which	Controls the level of details in the legend added to the plot (see details)
lim	for zooming over 1:lim iterations only.
ylim	limits on the y axis for zooming, passed to plot.

Details

The purpose of this plot if to compare K MCMC algorithms (typically based on K different simulation strategies or kernels) for convergence or efficiency in estimating a same target density f. For the kth algorithm, the user has to generate the convergence criterion, i.e. the sequence $K(p^t(_k)k), f)$ for $t = 1$ up to the number of iterations that has been chosen, and where $p^t(k)$ is the estimated pdf of the algorithm at time t.

For the legend, which=1 displays the MCMC's names together with some technical information depending on the algorithms definition (e.g. the proposal variance for the RWHM algorithm) and the method used for entropy estimation. The legend for which=2 is shorter, only displaying the MCMC's names together with the number of parallel chains used for each, typically to compare the effect of that number for a single MCMC algorithm.

Value

The graphic to plot.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2012), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, (2013) 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.
- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

EntropyMCMC, EntropyMCMC.mc

Examples

```
## Toy example using the bivariate centered gaussian target
## with default parameters value, see target_norm_param
d = 2           # state space dimension
n=300; nmc=100  # number of iterations and iid Markov chains
## initial distribution, located in (2,2), "far" from target center (0,0)
Ptheta0 <- DrawInit(nmc, d, initpdf = "rnorm", mean = 2, sd = 1)

## MCMC 1: Random-Walk Hasting-Metropolis
varq=0.05 # variance of the proposal (chosen too small)
q_param=list(mean=rep(0,d),v=varq*diag(d))

## using Method 1: simulation with storage, and *then* entropy estimation
# simulation of the nmc iid chains, single core here
s1 <- MCMCcopies(RWHM, n, nmc, Ptheta0, target_norm,
                 target_norm_param, q_param)
summary(s1) # method for "plMCMC" object
e1 <- EntropyMCMC(s1) # computes Entropy and Kullback divergence

## MCMC 2: Independence Sampler with large enough gaussian proposal
varq=1; q_param <- list(mean=rep(0,d),v=varq*diag(d))

## using Method 2: simulation & estimation for each t, forgetting the past
## HPC with 2 cores here (using parallel socket cluser, not available on Windows machines)
e2 <- EntropyParallel.cl(HMIS_norm, n, nmc, Ptheta0, target_norm,
                      target_norm_param, q_param,
                      cltype="PAR_SOCK", nbnodes=2)

## Compare these two MCMC algorithms
plot_Kblist(list(e1,e2)) # MCMC 2 (HMIS, red plot) converges faster.
```

RWHM_chain *Simulating MCMC single chains using MCMC algorithms*

Description

These functions are used to define the elements `$chain` of the MCMC algorithms that are (and must be) implemented as lists in **EntropyMCMC**. These functions are usually only called by higher-level functions, see details below.

Usage

```
RWHM_chain(theta0, it = 100, target, f_param, q_param, q_pdf = gaussian_pdf,
                q_proposal = gaussian_proposal)
HMIS_norm_chain(theta0, it = 100, target, f_param, q_param, q_pdf = q_pdf_ISnorm,
                q_proposal = q_proposal_ISnorm)
AMHaario_chain(theta0, it = 100, target, f_param, q_param, q_pdf = gaussian_pdf,
                q_proposal = gaussian_proposal)
```

```
IID_chain(theta0 = NULL, it = 100, target, f_param, q_param = NULL, q_pdf = NULL,
                q_proposal = NULL)
```

Arguments

it	the number of iterations to simulate
theta0	the initial position of the chain, a d-dim vector
target	the user-defined target density
f_param	the parameters (hyperparameters, data) of the user-defined target density
q_param	the parameters of the proposal density, which structure depends on the algorithm and the proposal density chosen by the user. Defaults are for RWHM: a list with the mean and covariance matrix of the proposal. For AMHaario: a list that must contain three elements: v the initial covariance matrix, t0 the iteration of the end of initial stage with that matrix, and epsi the epsilon parameter (for the nondegenerate matrix part), see Haario et. al.(2001).
q_pdf	the proposal density
q_proposal	the function simulating the proposal for the next move

Details

Each MCMC algorithm is defined as a list with five elements, see the object RWHM for an example. The element $chain must provide the name of the function performing simulation of a single chain and returning that chain, with arguments that must follow the definition above. Each user can define its own MCMC starting with the algorithms provided (see also section below). These functions are thus usually called by higher-level functions like MCMCcopies, EntropyParallel, or their multicore versions, for simulating copies of MCMC chains in an automatic manner.

- RWHM_chain is used in RWHM, a standard Randow-Walk Hastings-Metropolis algorithm.
- HMIS_norm_chain is used in HMIS_norm, an Independence Sampler HM with gaussian proposal
- AMHaario_chain is used in AMHaario, the Haario Adaptive Hastings-Metropolis algorithm (Haario 2001), and is provided as an example of a benchmark AMCMC.
- IID_chain is used in IID_norm, a "fake" MCMC that is just a gaussian IID sampler.

Value

A list with elements:

theta	the simulated chain in an array of it rows and d columns (the dimension)
paccept	the empirical acceptance rate
finalcov	the last covariance matrix
algo	the name of the algorithm (for plot methods)

Author(s)

Didier Chauveau.

References

H. Haario, E. Saksman, and J. Tamminen (2001), An adaptive Metropolis algorithm. Bernoulli 7, 223–242.

See Also

The algorithm already implemented, listed in `EntropyMCMC-package`.

The higher level functions that use these functions for simulation: `MCMCcopies`, `EntropyParallel` and their multicore versions.

`summary.plMCMC` *Summarizes content of a* `plMCMC` *object holding iid copies of MCMC's*

Description

This S3 method for `summary` summarizes the content of an object of class `plMCMC` (for parallel MCMC) as returned by, e.g., `MCMCcopies`, containing the trajectories of iid copies of trajectories from a MCMC algorithm, and its associated kernel, target and proposal densities.

Usage

```
## S3 method for class 'plMCMC'
summary(object, stats = FALSE, ...)
```

Arguments

`object`	An object of class `plMCMC` as returned by, e.g.,`MCMCcopies`.
`stats`	print additional summary statistics for the variables over all chains.
`...`	additional arguments passed to other methods

Value

Returns the object associated dimensions, the overall rate of acceptation, and descriptive statistics over the variable coordinates if `stats = TRUE`.

Author(s)

Didier Chauveau.

References

- Chauveau, D. and Vandekerkhove, P. (2012), Smoothness of Metropolis-Hastings algorithm and application to entropy estimation. *ESAIM: Probability and Statistics*, **17**, (2013) 419–431. DOI: http://dx.doi.org/10.1051/ps/2012004
- Chauveau D. and Vandekerkhove, P. (2014), Simulation Based Nearest Neighbor Entropy Estimation for (Adaptive) MCMC Evaluation, In *JSM Proceedings, Statistical Computing Section*. Alexandria, VA: American Statistical Association. 2816–2827.

- Chauveau D. and Vandekerkhove, P. (2014), The Nearest Neighbor entropy estimate: an adequate tool for adaptive MCMC evaluation. *Preprint HAL* http://hal.archives-ouvertes.fr/hal-01068081.

See Also

MCMCcopies, MCMCcopies.mc

Examples

```
## See Example for MCMCcopies
```

Index