Package 'PAWL'

July 2, 2014

Type Package

Title Implementation of the PAWL algorithm

Version 0.5

Date 2012-11-15

Author Luke Bornn, Pierre E. Jacob

Maintainer Pierre Jacob <pierre.jacob.work@gmail.com>

Description Implementation of the Parallel Adaptive Wang-Landau algorithm. Also implemented for comparison: parallel adaptive Metropolis-Hastings,SMC sampler.

Depends methods, mvtnorm, foreach, reshape, ggplot2

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2012-11-15 13:08:48

NeedsCompilation yes

R topics documented:

`PAWL-package` *PARALLEL ADAPTIVE WANG-LANDAU*

Description

The package implements the Parallel Adaptive Wang-Landau algorithm on various examples. The provided demos allow to reproduce the figures of the article.

Details

Package:	PAWL
Type:	Package
Version:	1.0
Date:	2011-08-11
License:	GPL (>= 2)
LazyLoad:	yes
Depends:	mvtnorm
Suggests:	ggplot2

The main function is `pawl`. It takes algorithmic parameters in arguments (see the help of the `pawl` function), as well a target distribution. Look at the demos to learn how to specify a target distribution.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

Examples

```
demo(discreteexample)
demo(gaussianexample)
demo(mixture2kexample)
```

---

adaptiveMH *Adaptive Metropolis-Hastings*

Description

Adaptive Metropolis-Hastings algorithm, with parallel chains. The adaptation is such that it targets an acceptance rate.

Usage

```
adaptiveMH(target, AP, proposal, verbose)
```

Arguments

target	Object of class target: specifies the target distribution. See the help of target. If the target is discrete, target must contain the slots dproposal, rproposal and proposalparam that specify the proposal kernel in the Metropolis-Hastings step. Otherwise the default is an adaptive gaussian random walk.
AP	Object of class "tuningparameters": specifies the number of chains, the number of iterations, and what should be stored during along the run. See the help of tuningparameters.
proposal	Object of class "proposal": specifies the proposal distribution to be used to propose new values and to compute the acceptance rate. See the help of proposal. If this is not specified and the target is continuous, then the default is an adaptive gaussian random walk.
verbose	Object of class "logical": if TRUE (default) then prints some indication of progress in the console.

Value

The function returns a list holding various information:

finalchains	The last point of each chain.
acceptrates	The vector of acceptance rates at each step.
sigma	The vector of the standard deviations used by the MH kernel along the iterations. If the proposal was adaptive, this allows to check how the adaptation behaved.
allchains	If asked in the tuning parameters, the chain history.
alllogtarget	If asked in the tuning parameters, the associated log density evaluations.
meanchains	If asked in the tuning parameters, the mean (component-wise) of each chain.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

preexplorationAMH

---

binning *Class* "binning"

Description

This class holds all the parameters of the Parallel Adaptive Wang-Landau algorithm that are related to the bins: it includes the functions that take points and return the point locations with respect to the bins, parameters related to the number of bins, the split mechanism, the adaptation rate of the stochastic approximation schedule, etc.

Objects from the Class

Objects should created by calls of the function `binning`. Examples are provided that should help understanding this class. Essentially it is a list of parameters, most of which have a reasonable default value so you do not need to think about it too much.

Important slots

`position`: Object of class `"function"`: should be a function taking points and associated log density values, and returning a "reaction coordinate", that is, a value that will be associated with bins. Typically, it can be the log density itself, or one component of a d-dimensional point. See the example below.

`binrange`: Object of class `"numeric"`: it should be a vector of size 2, holding the minimum and the maximum on the reaction coordinate scale. The bins are going to be between those two (inner bins), while a bin will go from - infinity to the minimum, and a bin will go from maximum to + infinity (outer bins).

`ncuts`: Object of class `"numeric"`: how many cuts will be made in the bin range specified by the previous argument. This induce the number of initial bins. Bins are automatically created by the following line:

```
bins <- c(-Inf, seq(from = binrange[1], to = binrange[2], length.out = ncuts))
```

There are then (ncuts +1) bins. The default for ncuts is 9, resulting in 10 bins.

Optional slots

`bins`: Object of class `"numeric"`: you can specify the bins directly, in which case you do not need to specify `binrange`.

`name`: Object of class `"character"`: ... if you want to name the instance (default is "unspecified").

`autobinning`: Object of class `"logical"`: activate or not the splitting mechanism, to create new inner bins automatically. This does not create new bins outside the specified bin range, it just add new bins inside to help reaching the Flat Histogram criteria more quickly.

`desiredfreq`: Object of class `"numeric"`: you can specify the desired frequency of each bin. The default is 1 / nbins in each bin, where nbins is the number of bins. Note that if autobinning is enable, when a bin is split into two bins, the desired frequencies of the new bins are equal to half of the desired frequency of the former bin.

`useLearningRate`: Object of class `"logical"`: active or not the stochastic approximation schedule. That is, if it is not activated, then no schedule are used in the update of theta (the penalty associated to the bins). Default is TRUE.

`useFH`: Object of class `"logical"`: active or not the Flat Histogram checks. If it is not activated, then the stochastic approximation decreases at each step. Default is TRUE, unless useLearningRate is FALSE, in which case there is no point checking for Flat Histograms.

`fhthreshold`: Object of class `"numeric"`: specifies the threshold to accept Flat Histogram. The default is 0.5. Smaller values make the Flat Histogram criterion harder to reach.

`minSimEffort`: Object of class `"numeric"`: specifies the minimum number of iterations after a Flat Histogram, for a new Flat Histogram criterion to be accepted. It prevents the criterion to be accepted at every iteration when using a large number of parallel chains. Default is 200.

`learningrate`: Object of class `"function"`: specifies the learning rate, that is, the rate at which the stochastic schedule decreases. It should be a function defined on [0, + infty[such that it is not integrable but its square is integrable, e.g. t -> 1/t for instance. The default is t -> t^-0.6.

`splitThreshold`: Object of class `"numeric"`: specifies the threshold to split a bin into two new bins. The default is 0.1 (read 10%), which means that a bin is split if at least 90% of the points in that bin are on the half right (or left) side of the bin. Larger values (e.g. 25%) result in more splits, and hence more final bins.

Methods

show `signature(object = "binning")`: provides a little summary of a binning object when called (or when `print` is called).

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

Examples

```
showClass("binning")
getPos <- function(points, logdensity) points
positionbinning <- binning(position = getPos,
                      name = "position",
                      binrange = c(-4, 0),
                      ncuts = 4,
                      autobinning = TRUE,
                      useLearningRate = TRUE)
```

ConvertResults *Convert Results*

Description

Convert results from pawl and adaptiveMH. The result is a data set that is more convenient to use with "ggplot2" functions.

Usage

```
ConvertResults(results, verbose)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
verbose	Object of class "logical": if TRUE (default) then prints some indication of progress in the console.

Details

Essentially it concatenates the parallel chains in a single column, and adds a column with the associated log density values. If more than 1000 parallel chains are used, the function can take some time to return its output.

Value

The function returns an object of class "data.frame", with columns for the chain indices, the chain values, the iteration indices, and the associated log density values.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

adaptiveMH, pawl

createAdaptiveRandomWalkProposal
Adaptive Random Walk proposal distribution for MCMC algorithms

Description

Create the adaptive gaussian random walk proposal that is used as a default in adaptiveMH and pawl, whenever the target distribution is continuous.

Usage

```
createAdaptiveRandomWalkProposal(nchains, targetdimension, adaptiveproposal, adaptationrate, sigm
```

Arguments

nchains	Object of class "numeric": it should be an integer representing the desired number of parallel chains.
targetdimension	Object of class "numeric": it should be an integer representing the dimension of the target distribution.
adaptiveproposal	Object of class "logical": specifies whether an adaptive proposal (Robbins-Monroe type of adaptation) should be used. Default is FALSE.
adaptationrate	Object of class "function": specifies the rate at which the adaptation of the proposal is performed. It should be a function defined on [0, + infty[such that it is not integrable but its square is integrable, e.g. t -> 1/t for instance. The default is t -> t^-0.6.
sigma_init	Object of class "numeric": it should be a positive real number specifying the standard deviation of the proposal distribution at the first iteration. If the proposal is adaptive, it acts as a starting point for the adaptation. If it is not adaptive, then this value is used throughout all the iterations. Default is 1.

Value

The function returns an object of class proposal-class, to be used in calls to adaptiveMH and pawl.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

proposal-class, adaptiveMH, pawl

`createMixtureTarget` *Mixture target distribution*

Description

Create the posterior distribution of the parameters of a mixture of univariate gaussian distributions, with a fixed (known) number of components.

Usage

```
createMixtureTarget(mixturesample, mixturesize, ncomponents, mixtureparameters)
```

Arguments

All the arguments are optional, since if none is given, a mixture distribution with 4 components will be created, as in Jasra, Holmes, Stephens, "MCMC and label switching problem in Bayesian mixture models", published in Statistical Science (2005).

Object of class `"vector"`: data set to be used. If not provided, a synthetic data set is generated.

`mixturesample`

`mixturesize` Object of class `"numeric"`: represents the data set size if a data set is to be generated.

`ncomponents` Object of class `"numeric"`: represents the fixed number of components to be used.

`mixtureparameters`

Object of class `"list"`: provides the parameters to be used if a data set has to be generated. The parameters include the number of components, the component weights, means and variances.

Value

The function returns an object of class `target-class`, with a name, a dimension, a function giving the log density, a function to generate sample from the distribution, parameters of the distribution, and a function to draw init points for the MCMC algorithms. The log density involves a likelihood and a prior, and the prior is as in Richardson and Green, "On Bayesian analysis of mixtures with an unknown number of components", published in JRSS B, 1997.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

References

Jasra, Holmes, Stephens, "MCMC and label switching problem in Bayesian mixture models", published in Statistical Science (2005). Richardson and Green, "On Bayesian analysis of mixtures with an unknown number of components", published in JRSS B, 1997.

See Also

target-class, createTrimodalTarget

---

createTrimodalTarget *Trimodal target distribution*

Description

Create the trimodal distribution as in Liang, Liu and Caroll, 2007: Stochastic approximation in Monte Carlo computation.

Usage

```
createTrimodalTarget()
```

Details

This distribution is a mixture of three bivariate Gaussian distributions. Their covariance matrices are such that an adaptive MCMC algorithm which proposal variance adapts to one of the component, will likely fail to explore the others.

Value

The function returns an object of class target, with a name, a dimension, a function giving the log density, a function to generate sample from the distribution, parameters of the distribution, and a function to draw init points for the MCMC algorithms.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

References

Liang, Liu and Caroll: Stochastic approximation in Monte Carlo computation. Published in JASA, 2007.

See Also

target, createMixtureTarget

getFrequencies *Observed Frequencies in each bin.*

Description

This function provides a convenient way to check whether the target frequencies have been reached. Since new bins can be created during the algorithm, this function aggregates them in the right way so that the user can know if the initial bins (on which the desired frequencies were specified) have been visited enough.

Usage

```
getFrequencies(results, binning)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
binning	Object of class binning: the binning on which the frequencies have to be computed.

Value

The function is supposed to be used for the lines that it prints in the console. However it also returns a vector of sampling frequencies corresponding to the initial bins.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

pawl

IceFloe *Image of ice floes*

Description

This data represents a binary matrix, representing an image of ice floes.

Usage

```
IceFloe
```

Format

A matrix containing 40 rows and 40 columns

Source

Banfield, J. and Raftery, A. (1992). Ice floe identification in satellite images using mathematical morphology and clustering about principal curves. Journal of the American Statistical Association, 87(417):7-16.

`normalizeweight` *Normalize weights*

Description

Simple function that normalize vectors (ie takes log weights and returns normalized weights, in the SMC context).

Usage

```
normalizeweight(log_weights)
```

Arguments

`log_weights`	Object of class `"numeric"`: a real-valued vector.

Details

Simple function that takes log weights (ie any real-valued vector), computes the exponential of it, divides it by its sum and returns it.

Value

The function returns an object of class `"data.frame"`, with columns for the chain indices, the chain values, the iteration indices, and the associated log density values.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

`smc`

pawl *Parallel Adaptive Wang-Landau*

Description

Implements the Parallel Adaptive Wang-Landau algorithm.

Usage

```
pawl(target, binning, AP, proposal, verbose)
```

Arguments

target	Object of class target: specifies the target distribution. See the help of target. If the target is discrete, target must contain the slots dproposal, rproposal and proposalparam that specify the proposal kernel in the Metropolis-Hastings step. Otherwise the default is an adaptive gaussian random walk.
binning	Object of class binning, defining the initial bins used by the Wang-Landau algorithm. The binning object also contains some parameters specifying if the automatic binning mechanism is active or not, for instance.
AP	Object of class tuningparameters: specifies the number of chains, the number of iterations, and what should be stored during along the run. See the help of tuningparameters.
proposal	Object of class proposal: specifies the proposal distribution to be used to propose new values and to compute the acceptance rate. See the help of proposal. If this is not specified and the target is continuous, then the default is an adaptive gaussian random walk.
verbose	Object of class "logical": if TRUE (default) then prints some indication of progress in the console.

Value

The function returns a list holding various information:

finalchains	The last point of each chain.
acceptrates	The vector of acceptance rates at each step.
sigma	The vector of the standard deviations used by the MH kernel along the iterations. If the proposal was adaptive, this allows to check how the adaptation behaved.
allchains	If asked in the tuning parameters, the chain history.
alllogtarget	If asked in the tuning parameters, the associated log density evaluations.
meanchains	If asked in the tuning parameters, the mean (component-wise) of each chain.
logthetahistory	If asked in the tuning parameters, all the log theta penalties.

and other quantities, that you can browse by calling "names(results)" where "results" is the output of the function.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

adaptiveMH, binning

PlotAllVar *Trace plot of all the variables*

Description

This function takes the result of adaptiveMH or of pawl, and draws a trace plot for each component of the chains

Usage

```
PlotAllVar(results)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotComp1vsComp2 *Plot one component versus another in a scatter plot*

Description

This function takes the result of adaptiveMH or of pawl, and component indices, and draws a cloud of points with the first component on the x-axis and the second on the y-axis.

Usage

```
PlotComp1vsComp2(results, comp1, comp2)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
comp1	Object of class "numeric": specifies the index of the component to plot on the x-axis.
comp2	Object of class "numeric": specifies the index of the component to plot on the y-axis.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotDensComp1vsComp2 *Plot one component versus another in a density plot*

Description

This function takes the result of adaptiveMH or of pawl, and component indices, and draws a 2D density plot with the first component on the x-axis and the second on the y-axis.

Usage

```
PlotDensComp1vsComp2(results, comp1, comp2)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
comp1	Object of class "numeric": specifies the index of the component to plot on the x-axis.
comp2	Object of class "numeric": specifies the index of the component to plot on the y-axis.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotFH *Plot of the Flat Histogram occurrences*

Description

This function takes the result of pawl, and draws a plot of the occurrences of the Flat Histogram criteria along the iterations.

Usage

```
PlotFH(results)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotHist *Plot a histogram of one component of the chains*

Description

This function takes the result of adaptiveMH or of pawl, and a component index, and draws a histogram of it.

Usage

```
PlotHist(results, component)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
component	Object of class "numeric": specifies the index of the component to plot on the x-axis.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotHistBin *Plot a histogram of the binning coordinate*

Description

This function takes the result of adaptiveMH or of pawl, and a binning object, and draws a histogram of the chains according to the binning coordinate.

Usage

```
PlotHistBin(results, binning)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.
binning	Object of class binning, defining the initial bins used by the Wang-Landau algorithm.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

`PlotLogTheta` *Plot of the log theta penalties*

Description

This function takes the result of pawl, and draws a trace plot of the log theta penalties along the iterations.

Usage

```
PlotLogTheta(results)
```

Arguments

`results`	Object of class `"list"`: either the output of pawl or of adaptiveMH.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

PlotNbins *Plot of the increase of the number of bins along the iterations*

Description

This function takes the result of pawl, and draws a plot of the increase of the number of bins along the iterations.

Usage

```
PlotNbins(results)
```

Arguments

results	Object of class "list": either the output of pawl or of adaptiveMH.

Value

The function returns a ggplot2 object.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

ggplot

Pollution *Pollution Data*

Description

This data contains 1 response (mortality, normalized to have mean zero) along with 15 pollution-related explanatory variables.

Usage

```
Pollution
```

Format

A matrix containing 60 rows and 16 columns

Source

McDonald, G.C. and Schwing, R.C. (1973) 'Instabilities of regression estimates relating air pollution to mortality', Technometrics, vol.15, 463-482.

preexplorationAMH *Pre exploration Adapative Metropolis-Hastings*

Description

This function takes a target distribution, an integer representing the number of parallel chains, and an integer representing a number of iterations, and runs adaptive Metropolis-Hastings algorithm using them. The chains are then used to create a range called SuggestedRange, to be used to bin the state space according to the energy levels. The energy is here defined as minus the log density of the target distribution.

Usage

```
preexplorationAMH(target, nchains, niterations, proposal, verbose)
```

Arguments

target	Object of class "target": this argument describes the target distribution. See target for details.
nchains	Object of class "numeric": specifies the number of parallel chains.
niterations	Object of class "numeric": specifies the number of iterations.
proposal	Object of class "proposal": specifies the proposal distribution to be used to propose new values and to compute the acceptance rate. See the help of proposal. If this is not specified and the target is continuous, then the default is an adaptive gaussian random walk.
verbose	Object of class "logical": if TRUE (default) then prints some indication of progress in the console.

Details

The adaptive Metropolis-Hastings algorithm used in the function is described in more details in the help page of adaptiveMH

Value

The function returns a list holding the following entries:

LogEnergyRange	This holds the minimum and maximum energy values seen by the chains during the exploration.
LogEnergyQtile	Returns the first 10% quantile of the energy values seen by the chains during the exploration.
SuggestedRange	This holds the suggested range, that is, the first 10% quantile and the maximum value of the energy values seen during the exploration. This can be passed as the binrange argument of the binning class, see the trimodal example.
finalchains	The last point of each chain.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

adaptiveMH

---

proposal *Class* "proposal"

Description

This class holds a proposal distribution to be used in a Metropolis-Hastings kernel.

Objects from the Class

Objects should created by calls of the function proposal.

Important slots

rproposal: Object of class "function":

dproposal: Object of class "function":

Optional slots

proposalparam: Object of class "list":

adaptiveproposal: Object of class "logical":

adaptationrate: Object of class "function":

sigma_init: Object of class "numeric":

Methods

show `signature(object = "proposal")`: provides a little summary of a proposal object when called (or when print is called).

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

smc *Sequential Monte Carlo*

Description

Sequential Monte Carlo samplers, using a sequence of tempered distributions.

Usage

```
smc(target, AP, verbose)
```

Arguments

target	Object of class target: specifies the target distribution. See the help of target. The target must be defined on a continuous state space.
AP	Object of class "smcparameters": specifies the number of particles, the ESS threshold, the sequence of distributions, etc. See the help of smcparameters.
verbose	Object of class "logical": if TRUE (default) then prints some indication of progress in the console.

Value

The function returns a list holding various information:

particles	a matrix with rows representing particles and columns components of each particle.
weights	a vector of weights associated to each particle. See also the convenience function normalizeweight.
ESSarray	a vector of the ESS computed at each iteration.
resamplingtimes	a vector indicating the iteration at which resampling was performed.

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

smcparameters

smcparameters *SMC Tuning Parameters*

Description

This class holds parameters for the Sequential Monte Carlo sampler.

Objects from the Class

Objects can be created by calls of the function "smcparameters".

Slots

nparticles: Object of class "numeric": an integer representing the desired number of particles.

temperatures: Object of class "numeric": a vector of temperatures, default being "seq(from = 0.01, to = 1, length.o

nmoves: Object of class "numeric": number of move steps to be performed after each resampling step, default being 1.

ESSthreshold: Object of class "numeric": resampling occurs when the Effective Sample Size goes below "ESSthreshold" multiplied by the number of particles "nparticles".

movetype: Object of class "character": type of Metropolis-Hastings move step to be performed; can be either set to "independent" or "randomwalk", default being "independent".

movescale: Object of class "numeric": if movetype is set to "randomwalk", this parameter specifies the amount by which the estimate of the standard deviation of the target distribution is multiplied; the product being used to propose new points in the random-walk MH step. Default is 10%, ie a new point is proposed from a Normal distribution, centered on the latest point, with standard deviation equal to 10% of the standard deviation of the already-generated chain.

resamplingscheme: Object of class "character": type of resampling to be used; either "multinomial", "residual" or "systematic", the default being "systematic".

Methods

show signature(object = "smcparameters"): provides a little summary of a binning object when called (or when print is called).

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

smc

Examples

```
showClass("smcparameters")
smcparam<- smcparameters(nparticles=5000,
                         temperatures = seq(from = 0.0001, to = 1, length.out= 100),
                         nmoves = 5, ESSthreshold = 0.5, movetype = "randomwalk",
                         movescale = 0.1)
```

---

target *Class: target distribution*

Description

This class represents target distributions, that is, probability distributions from which we want to sample using MCMC or Wang-Landau.

Objects from the Class

Objects should created by calls of the function `target`. Examples are provided that should help implementing any continuous probability distributions.

Important slots

`dimension`**:** Object of class `"numeric"`: should be an integer specifying the dimension of the state space on which the target distribution is defined.

`logdensity`**:** Object of class `"function"` : should be a function taking n points in the state space and parameters, and returning a vector of n real values. See the example below. This function is in most cases the most time-consuming part in a MCMC algorithm, so make sure it runs reasonably fast!

`rinit`**:** Object of class `"function"` : this function should take an integer as argument, say n. Then the function should return a matrix of dimension n times d (where d is the dimension of the state space), representing n points in the state space. These n points will be used as starting points of a parallel MCMC algorithm.

Optional slots

`parameters`**:** Object of class `"list"` : you can put anything in that list (and nothing, which is the defaults), the important thing is that calls to `logdensity(x, parameters)`return sensible values. For example, for a gaussian target distribution, you can put the mean and the variance in the `parameters` list (see example below). If need be, you can put a whole data set in there.

`type`**:** Object of class `"character"` : could be "continuous" or "discrete"; default is "continuous".

`name`**:** Object of class `"character"` : ... if you want to name your distribution (default is "unspecified").

`generate`**:** Object of class `"function"` : does not have to be specified, but if it is specified it should be a function to generate from the distribution (like rnorm is to the standard normal distribution).

Methods

show `signature(object = "target")`: provides a little summary of a target object when called (or when `print` is called).

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

Examples

```
showClass("target")
# starting points for MCMC algorithms
rinit <- function(size) rnorm(size)
# target log density function: a gaussian distribution N(mean = 2, sd = 3)
parameters <- list(mean = 2, sd = 3)
logdensity <- function(x, parameters) dnorm(x, parameters$mean, parameters$sd, log = TRUE)
# creating the target object
gaussiantarget <- target(name = "gaussian", dimension = 1,
                  rinit = rinit, logdensity = logdensity,
                  parameters = parameters)
print(gaussiantarget)
```

---

tuningparameters *MCMC Tuning Parameters*

Description

This class holds tuning parameters for the Metropolis-Hastings and Wang-Landau algorithms.

Objects from the Class

Objects can be created by calls of the function `"tuningparameters"`.

Slots

`nchains`**:** Object of class `"numeric"`: it should be an integer representing the desired number of parallel chains.

`niterations`**:** Object of class `"numeric"`: it should be an integer representing the desired number of iterations.

`computemean`**:** Object of class `"logical"`: specifies whether the mean of all chains should be computed at each iteration (useful if the chains are not to be stored).

`computemeanburnin`**:** Object of class `"numeric"`: if `computemean` is set to TRUE, specifies after which iteration the mean of the chain has to be computed. Default is 0 (no burnin).

`saveeverynth`**:** Object of class `"numeric"`: specifies when the chains are to be stored: for instance at every iteration (=1), every 10th iteration (=10), etc. Default is -1, meaning the chains are not stored.

Methods

show `signature(object = "tuningparameters")`: provides a little summary of a binning object when called (or when `print` is called).

Author(s)

Luke Bornn <bornn@stat.harvard.edu>, Pierre E. Jacob <pierre.jacob.work@gmail.com>

See Also

`adaptiveMH` `preexplorationAMH` `pawl`

Examples

```
showClass("tuningparameters")
mhparameters <- tuningparameters(nchains = 10, niterations = 1000, adaptiveproposal = TRUE)
```

Index