

Package ‘EMC’

April 17, 2009

Type Package

Title Evolutionary Monte Carlo (EMC) algorithm

Version 1.1

Date 2008-02-04

Author Gopi Goswami <goswami@stat.harvard.edu>

Maintainer Gopi Goswami <rgoswami@gmail.com>

Depends R (>= 1.9.0), mvtnorm

Description Random walk Metropolis, Metropolis Hastings, parallel tempering, evolutionary Monte Carlo, temperature ladder construction and placement.

License GPL (>= 2)

Repository CRAN

Date/Publication 2008-02-05 09:35:11

R topics documented:

evolMonteCarlo	2
findMaxTemper	6
MetropolisHastings	11
parallelTempering	13
placeTempers	17
print	21
randomWalkMetropolis	22
utilsForExamples	24

Index	26
--------------	-----------

evolMonteCarlo *evolutionary Monte Carlo algorithm*

Description

Given a multi-modal and multi-dimensional target density function, a (possibly asymmetric) proposal distribution and a temperature ladder, this function produces samples from the target using the evolutionary Monte Carlo algorithm.

Usage

```
evolMonteCarlo(nIters,
               temperLadder,
               startingVals,
               logTarDensFunc,
               MHPPropNewFunc,
               logMHPPropDensFunc = NULL,
               MHBlocks            = NULL,
               MHBlockNTimes      = NULL,
               moveProbsList      = NULL,
               moveNTimesList     = NULL,
               SCRWMNTimes       = NULL,
               SCRWMPPropSD       = NULL,
               levelsSaveSampFor = NULL,
               saveFitness        = FALSE,
               verboseLevel       = 0,
               ...)
```

Arguments

Below `sampDim` refers to the dimension of the sample space, `temperLadderLen` refers to the length of the temperature ladder, and `levelsSaveSampForLen` refers to the length of the `levelsSaveSampFor`.

`nIters` integer > 0.

`temperLadder` double vector with all *positive* entries, in *decreasing* order.

`startingVals` double matrix of dimension `temperLadderLen` × `sampDim` or vector of length `sampDim`, in which case the same starting values are used for every temperature level.

`logTarDensFunc`
function of two arguments (`draw`, ...) that returns the target density evaluated in the log scale.

`MHPPropNewFunc`
function of four arguments (`temperature`, `block`, `currentDraw`, ...) that returns new Metropolis-Hastings proposals. *See details below on the argument `block`.*

`logMHPpropDensFunc` function of five arguments (`temperature`, `block`, `currentDraw`, `proposalDraw`, ...) that returns the proposal density evaluated in the log scale. *See details below on the argument block.*

`MHBlocks` list of integer vectors giving dimensions to be blocked together for sampling. It defaults to `as.list(1:sampDim)`, i.e., each dimension is treated as a block on its own. *See details below for an example.*

`MHBlockNTimes` integer vector of number of times each block given by `MHBlocks` should be sampled in each iteration. It defaults to `rep(1, length(MHBlocks))`. *See details below for an example.*

`moveProbsList` named list of probabilities adding upto 1.

`moveNTimesList` named list of integers ≥ 0 .

`SCRWMNTimes` integer > 0 .

`SCRWMPpropSD` double > 0 .

`levelsSaveSampFor` integer vector with *positive* entries.

`saveFitness` logical.

`verboseLevel` integer, a value ≥ 2 produces a lot of output.

... optional arguments to be passed to `logTarDensFunc`, `MHPpropNewFunc` and `logMHPpropDensFunc`.

Details

MHPpropNewFunc and logMHPpropDensFunc The `MHPpropNewFunc` and the `logMHPpropDensFunc` are called multiple times by varying the `block` argument over `1:length(MHBlocks)`, so these functions should know how to generate a proposal from the `currentDraw` or to evaluate the proposal density depending on which block was passed as the argument. *See the example section for sample code.*

MHBlocks and MHBlockNTimes Blocking is an important and useful tool in MCMC that helps speed up sampling and hence mixing. Example: Let `sampDim = 6`. Let we want to sample dimensions 1, 2, 4 as one block, dimensions 3 and 5 as another and treat dimension 6 as the third block. Suppose we want to sample the three blocks mentioned above 1, 5 and 10 times in each iteration, respectively. Then we could set `MHBlocks = list(c(1, 2, 4), c(3, 5), 6)` and `MHBlockNTimes = c(1, 5, 10)`.

The EMC and the TOEMC algorithm The evolutionary Monte Carlo (EMC; Liang and Wong, 2001) algorithm is composed of the following moves:

- MH Metropolis-Hastings or mutation
- RC real crossover
- SC snooker crossover
- RE (random) exchange

The target oriented EMC (TOEMC; Goswami and Liu, 2007) algorithm has the following

additional moves on top of EMC:

BCE	best chromosome exchange
BIRE	best importance ratio exchange
BSE	best swap exchange
CE	cyclic exchange

The current function could be used to run both EMC and TOEMC algorithms by specifying what moves to employ using the following variables.

moveProbsList and **moveNTimesList** The allowed names for components of `moveProbsList` and `moveNTimesList` come from the abbreviated names of the moves above. For example, the following specifications are valid:

```
moveProbsList = list(MH = 0.4, RC = 0.3, SC = 0.3)
moveNTimesList = list(MH = 1, RC = floor(temperLadderLen / 2),
SC = floor(temperLadderLen / 2), RE = temperLadderLen)
```

SCRWMNTimes and **SCRWMPropSD** The conditional sampling step of the snooker crossover (SC) move is done using random walk Metropolis (RWM) with normal proposals; `SCRWMNTimes` and `SCRWMPropSD` are the number of RWM draws and the proposal standard deviation for the RWM step, respectively. Note these variables are only required if the SC move has positive probability in `moveProbsList` or a positive number of times in `moveNTimesList`.

levelsSaveSampFor By default, samples are saved and returned for temperature level `temperLadderLen`. The `levelsSaveSampFor` could be used to save samples from other temperature levels as well (e.g., `levelsSaveSampFor = 1:temperLadderLen` saves samples from all levels).

saveFitness The term *fitness* refers to the function $H(x)$, where the target density of interest is given by:

$$g(x) \propto \exp[-H(x)/\tau_{min}]$$

$H(x)$ is also known as the *energy* function. By default, the fitness values are not saved, but one can do so by setting `saveFitness = TRUE`.

Value

This function returns a list with the following components:

<code>draws</code>	array of dimension <code>nIters × sampDim × levelsSaveSampForLen</code> , if <code>saveFitness = FALSE</code> . If <code>saveFitness = TRUE</code> , then the returned array is of dimension <code>nIters × (sampDim + 1) × levelsSaveSampForLen</code> ; i.e., each of the <code>levelsSaveSampForLen</code> matrices contain the fitness values in their last column.
<code>acceptRatios</code>	matrix of the acceptance rates for various moves used.
<code>detailedAcceptRatios</code>	list of matrices with detailed summary of the acceptance rates for various moves used.

nIters the nIters argument.
 temperLadder the temperLadder argument.
 startingVals the startingVals argument.
 moveProbsList the moveProbsList argument.
 moveNTimesList the moveNTimesList argument.
 levelsSaveSampFor the levelsSaveSampFor argument.
 time the time taken by the run.

Note

The effect of leaving the default value NULL for some of the arguments above are as follows:

logMHPpropDensFunc the proposal density MHPpropNewFunc is deemed symmetric.
 MHBlocks as.list(1:sampDim).
 MHBlockNTimes rep(1, length(MHBlocks)).
 moveProbsList list(MH = 0.4, RC = 0.3, SC = 0.3).
 moveNTimesList list(MH = 1, RC = mm, SC = mm, RE = nn), where
 mm <- floor(nn / 2) and nn <- temperLadderLen.
 SCRWMNTimes 1, if SC is used.
 SCRWMPpropSD needs to be provided by the user, if SC is used.
 levelsSaveSampFor temperLadderLen.

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Gopi Goswami and Jun S. Liu (2007). On learning strategies for evolutionary Monte Carlo. Statistics and Computing 17:1:23-38.

Faming Liang and Wing H. Wong (2001). Real-Parameter Evolutionary Monte Carlo with Applications to Bayesian Mixture Models. Journal of the American Statistical Association 96:653-666.

See Also

[parallelTempering](#)

Examples

```

samplerObj <-
  with(VShapedFuncGenerator(-13579),
    {
      allMovesNTimesList <- list(MH = 1, RC = 2, SC = 2, RE = 4,
                                BIRE = 2, BCE = 2, BSE = 2, CE = 2)
      evolMonteCarlo(nIters = 2000,

```

```

                                temperLadder      = c(15, 6, 2, 1),
                                startingVals       = c(0, 0),
                                logTarDensFunc     = logTarDensFunc,
                                MHPpropNewFunc     = MHPpropNewFunc,
                                moveNTimesList    = allMovesNTimesList,
                                SCRWMNTimes       = 1,
                                SCRWMPropSD       = 3.0,
                                levelsSaveSampFor = seq_len(4),
                                verboseLevel      = 1)
    })
  print(samplerObj)
  print(names(samplerObj))
  with(samplerObj,
    {
      print(detailedAcceptRatios)
      print(dim(draws))
      par(mfcol = c(2, 2))
      for (ii in seq_along(levelsSaveSampFor)) {
        main <- paste('temper:', round(temperLadder[levelsSaveSampFor[ii]], 3))
        plot(draws[ , , ii],
             xlim = c(-5, 20),
             ylim = c(-8, 8),
             pch = '.',
             ask = FALSE,
             main = as.expression(main),
             xlab = as.expression(substitute(x[xii], list(xii = 1))),
             ylab = as.expression(substitute(x[xii], list(xii = 2))))
      }
    })
})

```

findMaxTemper

Find the maximum temperature for parallel MCMC chains

Description

Multiple MCMC chains based algorithms (e.g., parallel tempering, evolutionary Monte Carlo) need a temperature ladder. This function finds the maximum temperature for constructing the ladder.

Usage

```

findMaxTemper(nIters,
              statsFuncList,
              startingVals,
              logTarDensFunc,
              MHPpropNewFunc,
              logMHPpropDensFunc = NULL,
              temperLadder      = NULL,
              temperLimits     = NULL,
              ladderLen        = 10,

```

```

        scheme           = 'exponential',
        schemeParam      = 0.5,
        cutoffDStats     = 1.96,
        cutoffESS        = 50,
        guideMe          = TRUE,
        levelsSaveSampFor = NULL,
        saveFitness       = FALSE,
        doFullAnal       = TRUE,
        verboseLevel     = 0,
        ...)

```

Arguments

Below `sampDim` refers to the dimension of the sample space, `temperLadderLen` refers to the length of the temperature ladder, and `levelsSaveSampForLen` refers to the length of `levelsSaveSampFor`. Note, this function calls `evolMonteCarlo`, so some of the arguments below have the same name and meaning as the corresponding ones for `evolMonteCarlo`. See details below for explanation on the following arguments.

```

nIters           integer > 0.
statsFuncList    list of functions of one argument each, which return the value of the statistic
                 evaluated at one MCMC sample or draw.

startingVals     double matrix of dimension temperLadderLen × sampDim or vector of
                 length sampDim, in which case the same starting values are used for every
                 temperature level.

logTarDensFunc   function of two arguments (draw, ...) that returns the target density
                 evaluated in the log scale.

MHPPropNewFunc   function of four arguments (temperature, block, currentDraw,
                 ...) that returns new Metropolis-Hastings proposals. See details below on the
                 argument block.

logMHPPropDensFunc function of five arguments (temperature, block, currentDraw,
                 proposalDraw, ...) that returns the proposal density evaluated in the log
                 scale. See details below on the argument block.

temperLadder     double vector with all positive entries, in decreasing order.
temperLimits     double vector with two positive entries.
ladderLen        integer > 0.
scheme           character.
schemeParam      double > 0.
cutoffDStats     double > 0.
cutoffESS        double > 0.
guideMe          logical.

```

```

levelsSaveSampFor
    integer vector with positive entries.
saveFitness    logical.
doFullAnal     logical.
verboseLevel   integer, a value  $\geq 2$  produces a lot of output.
...            optional arguments to be passed to logTarDensFunc, MHPpropNewFunc
              and logMHPpropDensFunc.

```

Details

This function is based on the method to find the temperature range introduced in section 4.1 of Goswami and Liu (2007).

statsFuncList The user specifies this list of functions, each of which is known to be sensitive to the presence of modes. For example, if both dimension 1 and 3 are sensitive to presence of modes, then one could use:

```

coord1 <- function (xx) { xx[1] }
coord3 <- function (xx) { xx[3] }
statsFuncList <- list(coord1, coord3)

```

temperLadder This is the temperature ladder needed for the first stage preliminary run. One can either specify a temperature ladder via `temperLadder` or specify `temperLimits`, `ladderLen`, `scheme` and `schemeParam`. For details on the later set of parameters, see below. Note, `temperLadder` overrides `temperLimits`, `ladderLen`, `scheme` and `schemeParam`.

temperLimits `temperLimits = c(lowerLimit, upperLimit)` is a two-tuple of positive numbers, where the `lowerLimit` is usually 1 and `upperLimit` is a number in [100, 1000]. If stochastic optimization (via sampling) is the goal, then `lowerLimit` is taken to be in [0, 1]. Often the `upperLimit` is the maximum temperature as suggested by [findMaxTemper](#).

ladderLen, scheme and schemeParam These three parameters are required (along with `temperLimits`) if `temperLadder` is *not* provided. We recommend taking `ladderLen` in [15, 30]. The allowed choices for `scheme` and `schemeParam` are:

scheme	schemeParam
=====	=====
linear	NA
log	NA
geometric	NA
mult-power	NA
add-power	≥ 0
reciprocal	NA
exponential	≥ 0
tangent	≥ 0

We recommended using `scheme = 'exponential'` and `schemeParam` in [0.3, 0.5].

- cutoffDStats** This cutoff comes from $Normal_1(0, 1)$, the standard normal distribution (Goswami and Liu, 2007); the default value 1.96 is a conservative cutoff. Note if you have more than one statistic in `statsFuncList`, which is usually the case, using this cutoff may result in different suggested maximum temperatures (as can be seen by calling the `print` function on the result of `findMaxTemper`). A conservative recommendation is that you choose the maximum of the suggested temperatures as the final maximum temperature for use in `placeTemper`s and later in `parallelTempering` or `evolMonteCarlo`.
- cutoffESS** a cutoff for the effective sample size (ESS) of the underlying Markov chain ergodic estimator and the importance sampling estimators.
- guideMe** If `guideMe = TRUE`, then the function suggests different modifications to alter the setting towards a re-run, in case there are problems with the underlying MCMC run.
- doFullAnal** If `doFullAnal = TRUE`, then the search for the maximum temperature is conducted among *all* the levels of the `temperLadder`. In case this switch is turned off, the search for maximum temperature is done in a greedy (and faster) manner, namely, search is stopped as soon as all the statistic(s) in the `statsFuncList` find some maximum temperature(s). Note, the greedy search may result in much higher maximum temperature (and hence sub-optimal) than needed, so it is not recommended.
- levelsSaveSampFor** This is passed to `evolMonteCarlo` for the underlying MCMC run.

Value

This function returns a list with the following components:

- `temperLadder` the temperature ladder used for the underlying MCMC run.
- `DStats` the *D*-statistic (Goswami and Liu, 2007) values used to find the maximum temperature.
- `cutoffDStats` the `cutoffDStats` argument.
- `nIters` the post burn-in `nIters`.
- `levelsSaveSampFor` the `levelsSaveSampFor` argument.
- `draws` array of dimension `nIters × sampDim × levelsSaveSampForLen`, if `saveFitness = FALSE`. If `saveFitness = TRUE`, then the returned array is of dimension `nIters × (sampDim + 1) × levelsSaveSampForLen`; i.e., each of the `levelsSaveSampForLen` matrices contain the fitness values in their last column.
- `startingVals` the `startingVals` argument.
- `intermediate` statistics
a bunch of intermediate statistics used in the computation of `DStats`, namely, `MCEsts`, `MVarEsts`, `MCESS`, `ISEsts`, `ISVarEsts`, `ISESS`, each being computed for all the statistics provided by `statsFuncList` argument.
- `time` the time taken by the run.

Note

The effect of leaving the default value `NULL` for some of the arguments above are as follows:

logMHPpropDensFunc the proposal density MHPpropNewFunc is deemed symmetric.
 temperLadder valid temperLimits, ladderLen, scheme and schemeParam
 are provided, which are used to construct the temperLadder.
 temperLimits a valid temperLadder is provided.
 levelsSaveSampFor temperLadderLen.

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Gopi Goswami and Jun S. Liu (2007). On learning strategies for evolutionary Monte Carlo. Statistics and Computing 17:1:23-38.

See Also

[placeTempers](#), [parallelTempering](#), [evolMonteCarlo](#)

Examples

```

coord1      <- function (xx) { xx[1] }
ss          <- function (xx) { sum(xx) }
pp          <- function (xx) { prod(xx) }
statsFuncList <- list(coord1, ss, pp)
maxTemperObj <-
  with(VShapedFuncGenerator(-13579),
       findMaxTemper(nIters      = 15000,
                    statsFuncList = statsFuncList,
                    temperLadder  = c(20, 15, 10, 5, 1),
                    startingVals   = c(0, 0),
                    logTarDensFunc  = logTarDensFunc,
                    MHPpropNewFunc = MHPpropNewFunc,
                    levelsSaveSampFor = seq_len(5),
                    doFullAnal     = TRUE,
                    verboseLevel   = 1))

print(maxTemperObj)
print(names(maxTemperObj))
with(maxTemperObj,
     {
       par(mfcol = c(3, 3))
       for (ii in seq_along(levelsSaveSampFor)) {
         main <- paste('temper:', round(temperLadder[levelsSaveSampFor[ii]], 3))
         plot(draws[ , , ii],
              xlim = c(-10, 25),
              ylim = c(-10, 10),
              pch = '.',
              ask = FALSE,
              main = as.expression(main),
              xlab = as.expression(substitute(x[xii], list(xii = 1))),
              ylab = as.expression(substitute(x[xii], list(xii = 2))))
       }
     }

```

```
    })
```

MetropolisHastings *The Metropolis-Hastings algorithm*

Description

Given a target density function and an asymmetric proposal distribution, this function produces samples from the target using the Metropolis Hastings algorithm.

Usage

```
MetropolisHastings(nIters,
                  startingVal,
                  logTarDensFunc,
                  propNewFunc,
                  logPropDensFunc,
                  MHBlocks      = NULL,
                  MHBlockNTimes = NULL,
                  saveFitness   = FALSE,
                  verboseLevel  = 0,
                  ...)
```

Arguments

Below `sampDim` refers to the dimension of the sample space.

`nIters` integer > 0.

`startingVal` double vector of length `sampDim`.

`logTarDensFunc` function of two arguments (`draw`, ...) that returns the target density evaluated in the log scale.

`propNewFunc` function of three arguments (`block`, `currentDraw`, ...) that returns new Metropolis-Hastings proposals. *See details below on the argument block.*

`logPropDensFunc` function of four arguments (`block`, `currentDraw`, `proposalDraw`, ...) that returns the proposal density evaluated in the log scale. *See details below on the argument block.*

`MHBlocks` list of integer vectors giving dimensions to be blocked together for sampling. It defaults to `as.list(1:sampDim)`, i.e., each dimension is treated as a block on its own. *See details below for an example.*

`MHBlockNTimes` integer vector of number of times each block given by `MHBlocks` should be sampled in each iteration. It defaults to `rep(1, length(MHBlocks))`. *See details below for an example.*

`saveFitness` logical indicating whether fitness values should be saved. *See details below.*
`verboseLevel` integer, a value ≥ 2 produces a lot of output.
`...` optional arguments to be passed to `logTarDensFunc`, `propNewFunc` and `logPropDensFunc`.

Details

propNewFunc and logPropDensFunc The `propNewFunc` and the `logPropDensFunc` are called multiple times by varying the `block` argument over `1:length(MHBlocks)`, so these functions should know how to generate a proposal from the `currentDraw` or to evaluate the proposal density depending on which block was passed as the argument. *See the example section for sample code.*

MHBlocks and MHBlockNTimes Blocking is an important and useful tool in MCMC that helps speed up sampling and hence mixing. Example: Let `sampDim = 6`. Let we want to sample dimensions 1, 2, 4 as one block, dimensions 3 and 5 as another and treat dimension 6 as the third block. Suppose we want to sample the three blocks mentioned above 1, 5 and 10 times in each iteration, respectively. Then we could set `MHBlocks = list(c(1, 2, 4), c(3, 5), 6)` and `MHBlockNTimes = c(1, 5, 10)`

saveFitness The term *fitness* refers to the negative of the `logTarDensFunc` values. By default, the fitness values are not saved, but one can do so by setting `saveFitness = TRUE`.

Value

This function returns a list with the following components:

`draws` matrix of dimension `nIters × sampDim`, if `saveFitness = FALSE`.
 If `saveFitness = TRUE`, then the returned matrix is of dimension `nIters × (sampDim + 1)`, where the fitness values appear in its last column.
`acceptRatios` matrix of the acceptance rates.
`detailedAcceptRatios` matrix with detailed summary of the acceptance rates.
`nIters` the `nIters` argument.
`startingVal` the `startingVal` argument.
`time` the time taken by the run.

Note

The effect of leaving the default value `NULL` for some of the arguments above are as follows:

```

MHBlocks    as.list(1:sampDim).
MHBlockNTimes rep(1, length(MHBlocks)).
  
```

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Jun S. Liu (2001). Monte Carlo strategies for scientific computing. *Springer*.

See Also

[randomWalkMetropolis](#), [parallelTempering](#), [evolMonteCarlo](#)

Examples

```
samplerObj <-
  with(CigarShapedFuncGenerator2(-13579),
       MetropolisHastings(nIters      = 5000,
                          startingVal = c(0, 0),
                          logTarDensFunc = logTarDensFunc,
                          propNewFunc   = propNewFunc,
                          logPropDensFunc = logPropDensFunc,
                          verboseLevel  = 2))

print(samplerObj)
print(names(samplerObj))
with(samplerObj,
     {
       print(detailedAcceptRatios)
       print(dim(draws))
       plot(draws,
            xlim = c(-3, 5),
            ylim = c(-3, 4),
            pch = '.',
            ask = FALSE,
            main = as.expression(paste('# draws:', nIters)),
            xlab = as.expression(substitute(x[xii], list(xii = 1))),
            ylab = as.expression(substitute(x[xii], list(xii = 2))))
     })
```

parallelTempering *The parallel Tempering algorithm*

Description

Given a multi-modal and multi-dimensional target density function, a (possibly asymmetric) proposal distribution and a temperature ladder, this function produces samples from the target using the parallel tempering algorithm.

Usage

```
parallelTempering(nIters,
                  temperLadder,
                  startingVals,
                  logTarDensFunc,
                  MHPpropNewFunc,
```

```

logMHPpropDensFunc = NULL,
MHBlocks           = NULL,
MHBlockNTimes     = NULL,
moveProbsList     = NULL,
moveNTimesList    = NULL,
levelsSaveSampFor = NULL,
saveFitness       = FALSE,
verboseLevel      = 0,
...)
```

Arguments

Below `sampDim` refers to the dimension of the sample space, `temperLadderLen` refers to the length of the temperature ladder, and `levelsSaveSampForLen` refers to the length of the `levelsSaveSampFor`.

`nIters` integer > 0.

`temperLadder` double vector with all *positive* entries, in *decreasing* order.

`startingVals` double matrix of dimension `temperLadderLen` × `sampDim` or vector of length `sampDim`, in which case the same starting values are used for every temperature level.

`logTarDensFunc`
function of two arguments (`draw`, ...) that returns the target density evaluated in the log scale.

`MHPpropNewFunc`
function of four arguments (`temperature`, `block`, `currentDraw`, ...) that returns new Metropolis-Hastings proposals. *See details below on the argument block.*

`logMHPpropDensFunc`
function of five arguments (`temperature`, `block`, `currentDraw`, `proposalDraw`, ...) that returns the proposal density evaluated in the log scale. *See details below on the argument block.*

`MHBlocks` list of integer vectors giving dimensions to be blocked together for sampling. It defaults to `as.list(1:sampDim)`, i.e., each dimension is treated as a block on its own. *See details below for an example.*

`MHBlockNTimes`
integer vector of number of times each block given by `MHBlocks` should be sampled in each iteration. It defaults to `rep(1, length(MHBlocks))`. *See details below for an example.*

`moveProbsList`
named list of probabilities adding upto 1.

`moveNTimesList`
named list of integers ≥ 0.

`levelsSaveSampFor`
integer vector with *positive* entries.

`saveFitness` logical.

verboseLevel integer, a value ≥ 2 produces a lot of output.
 ... optional arguments to be passed to logTarDensFunc, MHPpropNewFunc
 and logMHPpropDensFunc.

Details

MHPpropNewFunc and logMHPpropDensFunc The MHPpropNewFunc and the logMHPpropDensFunc are called multiple times by varying the block argument over $1:\text{length}(\text{MHBlocks})$, so these functions should know how to generate a proposal from the currentDraw or to evaluate the proposal density depending on which block was passed as the argument. See the example section for sample code.

MHBlocks and MHBlockNTimes Blocking is an important and useful tool in MCMC that helps speed up sampling and hence mixing. Example: Let `sampDim = 6`. Let we want to sample dimensions 1, 2, 4 as one block, dimensions 3 and 5 as another and treat dimension 6 as the third block. Suppose we want to sample the three blocks mentioned above 1, 5 and 10 times in each iteration, respectively. Then we could set `MHBlocks = list(c(1, 2, 4), c(3, 5), 6)` and `MHBlockNTimes = c(1, 5, 10)`.

The parallel tempering algorithm The parallel tempering (PT; Liang and Wong, 2001) algorithm is composed of the following moves:

MH Metropolis-Hastings or mutation
 RE (random) exchange

The current function could be used to run the PT algorithm by specifying what moves to employ using the following variables.

moveProbsList and moveNTimesList The allowed names for components of `moveProbsList` and `moveNTimesList` come from the abbreviated names of the moves above. For example, the following specifications are valid:

```
moveProbsList = list(MH = 0.4, RE = 0.6)
moveNTimesList = list(MH = 1, RE = temperLadderLen)
```

levelsSaveSampFor By default, samples are saved and returned for temperature level `temperLadderLen`. The `levelsSaveSampFor` could be used to save samples from other temperature levels as well (e.g., `levelsSaveSampFor = 1:temperLadderLen` saves samples from all levels).

saveFitness The term *fitness* refers to the function $H(x)$, where the target density of interest is given by:

$$g(x) \propto \exp[-H(x)/\tau_{min}]$$

$H(x)$ is also known as the *energy* function. By default, the fitness values are not saved, but one can do so by setting `saveFitness = TRUE`.

Value

This function returns a list with the following components:

draws array of dimension `nIters × sampDim × levelsSaveSampForLen`, if `saveFitness = FALSE`. If `saveFitness = TRUE`, then the returned array is of dimension `nIters × (sampDim + 1) × levelsSaveSampForLen`; i.e., each of the `levelsSaveSampForLen` matrices contain the fitness values in their last column.

acceptRatios matrix of the acceptance rates for various moves used.

detailedAcceptRatios list of matrices with detailed summary of the acceptance rates for various moves used.

nIters the `nIters` argument.

temperLadder the `temperLadder` argument.

startingVals the `startingVals` argument.

moveProbsList the `moveProbsList` argument.

moveNTimesList the `moveNTimesList` argument.

levelsSaveSampFor the `levelsSaveSampFor` argument.

time the time taken by the run.

Note

The effect of leaving the default value `NULL` for some of the arguments above are as follows:

logMHPPropDensFunc the proposal density `MHPPropNewFunc` is deemed symmetric.

MHBlocks `as.list(1:sampDim)`.

MHBlockNTimes `rep(1, length(MHBlocks))`.

moveProbsList `list(MH = 0.4, RC = 0.3, SC = 0.3)`.

moveNTimesList `list(MH = 1, RC = mm, SC = mm, RE = nn)`, where `mm <- floor(nn / 2)` and `nn <- temperLadderLen`.

levelsSaveSampFor `temperLadderLen`.

Author(s)

Gopi Goswami (goswami@stat.harvard.edu)

References

Faming Liang and Wing H. Wong (2001). *Real-Parameter Evolutionary Monte Carlo with Applications to Bayesian Mixture Models*. *Journal of the American Statistical Association* 96:653-666.

See Also

[evolMonteCarlo](#)

Examples

```
samplerObj <-
```

```

with(VShapedFuncGenerator(-13579),
     parallelTempering(nIters           = 2000,
                       temperLadder    = c(15, 6, 2, 1),
                       startingVals    = c(0, 0),
                       logTarDensFunc  = logTarDensFunc,
                       MHPPropNewFunc  = MHPPropNewFunc,
                       levelsSaveSampFor = seq_len(4),
                       verboseLevel    = 1))

print(samplerObj)
print(names(samplerObj))
with(samplerObj,
     {
       print(detailedAcceptRatios)
       print(dim(draws))
       par(mfcol = c(2, 2))
       for (ii in seq_along(levelsSaveSampFor)) {
         main <- paste('temper:', round(temperLadder[levelsSaveSampFor[ii]], 3))
         plot(draws[ , , ii],
              xlim = c(-5, 20),
              ylim = c(-8, 8),
              pch = '.',
              ask = FALSE,
              main = as.expression(main),
              xlab = as.expression(substitute(x[xii], list(xii = 1))),
              ylab = as.expression(substitute(x[xii], list(xii = 2))))
       }
     })

```

placeTempers

Place the intermediate temperatures between the temperature limits

Description

Multiple MCMC chains based algorithms (e.g., parallel tempering, evolutionary Monte Carlo) need a temperature ladder. This function places the intermediate temperatures between the minimum and the maximum temperature for the ladder.

Usage

```

placeTempers(nIters,
             acceptRatioLimits,
             ladderLenMax,
             startingVals,
             logTarDensFunc,
             MHPPropNewFunc,
             logMHPPropDensFunc = NULL,
             temperLadder      = NULL,
             temperLimits     = NULL,
             ladderLen         = 15,

```


Details

This function is based on the temperature placement method introduced in section 4.2 of Goswami and Liu (2007).

acceptRatioLimits This is a range for the estimated acceptance ratios for the random exchange move for the consecutive temperature levels of the final ladder. It is recommended that specified range is between 0.3 and 0.6.

ladderLenMax It is preferred that one specifies `acceptRatioLimits` for constructing the final temperature ladder. However, If one has some computational limitations then one could also specify `ladderLenMax` which will limit the length of the final temperature ladder produced. This also serves as an upper bound on the number of temperature levels while placing the intermediate temperatures using the `acceptRatioLimits`.

temperLadder This is the temperature ladder needed for the second stage preliminary run. One can either specify a temperature ladder via `temperLadder` or specify `temperLimits`, `ladderLen`, `scheme` and `schemeParam`. For details on the later set of parameters, see below. Note, `temperLadder` overrides `temperLimits`, `ladderLen`, `scheme` and `schemeParam`.

temperLimits `temperLimits = c(lowerLimit, upperLimit)` is a two-tuple of positive numbers, where the `lowerLimit` is usually 1 and `upperLimit` is a number in [100, 1000]. If stochastic optimization (via sampling) is the goal, then `lowerLimit` is taken to be in [0, 1]. Often the `upperLimit` is the maximum temperature as suggested by `findMaxTemper`.

ladderLen, scheme and schemeParam These three parameters are required (along with `temperLimits`) if `temperLadder` is not provided. We recommend taking `ladderLen` in [15, 30]. The allowed choices for `scheme` and `schemeParam` are:

scheme	schemeParam
=====	=====
linear	NA
log	NA
geometric	NA
mult-power	NA
add-power	≥ 0
reciprocal	NA
exponential	≥ 0
tangent	≥ 0

We recommended using `scheme = 'exponential'` and `schemeParam` in [1.5, 2].

guideMe If `guideMe = TRUE`, then the function suggests different modifications to alter the setting towards a re-run, in case there are problems with the underlying MCMC run.

levelsSaveSampFor This is passed to `evolMonteCarlo` for the underlying MCMC run.

Value

This function returns a list with the following components:

`finalLadder` the final temperature ladder found by placing the intermediate temperatures to be used in `parallelTempering` or `evolMonteCarlo`.

`temperLadder` the temperature ladder used for the underlying MCMC run.
`acceptRatiosEst` the estimated acceptance ratios for the random exchange move for the consecutive temperature levels of `temperLadder`.
`CVSqWeights` this is the square of the coefficient of variation of the weights of the importance sampling estimators used to estimate the acceptance ratios, namely, `estAcceptRatios`.
`temperLimits` the sorted `temperLimits` argument.
`acceptRatioLimits` the sorted `acceptRatioLimits` argument.
`nIters` the post burn-in `nIters`.
`levelsSaveSampFor` the `levelsSaveSampFor` argument.
`draws` array of dimension `nIters × sampDim × levelsSaveSampForLen`, if `saveFitness = FALSE`. If `saveFitness = TRUE`, then the returned array is of dimension `nIters × (sampDim + 1) × levelsSaveSampForLen`; i.e., each of the `levelsSaveSampForLen` matrices contain the fitness values in their last column.
`startingVals` the `startingVals` argument.
`time` the time taken by the run.

Note

The effect of leaving the default value `NULL` for some of the arguments above are as follows:

`logMHPpropDensFunc` the proposal density `MHPpropNewFunc` is deemed symmetric.
`temperLadder` valid `temperLimits`, `ladderLen`, `scheme` and `schemeParam` are provided, which are used to construct the `temperLadder`.
`temperLimits` a valid `temperLadder` is provided.
`levelsSaveSampFor` `temperLadderLen`.

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Gopi Goswami and Jun S. Liu (2007). *On learning strategies for evolutionary Monte Carlo*. *Statistics and Computing* 17:1:23-38.

See Also

[findMaxTemper](#), [parallelTempering](#), [evolMonteCarlo](#)

Examples

```
placeTempersObj <-
  with(VShapedFuncGenerator(-13579),
```

```

        placeTempers(nIters          = 10000,
                    acceptRatioLimits = c(0.5, 0.6),
                    ladderLenMax     = 50,
                    startingVals     = c(0, 0),
                    logTarDensFunc   = logTarDensFunc,
                    MHPropNewFunc    = MHPropNewFunc,
                    temperLimits     = c(1, 5),
                    ladderLen        = 10,
                    levelsSaveSampFor = seq_len(10),
                    verboseLevel     = 1))
print(placeTempersObj)
print(names(placeTempersObj))
with(placeTempersObj,
{
  par(mfcol = c(3, 3))
  for (ii in seq_along(levelsSaveSampFor)) {
    main <- paste('temper:', round(temperLadder[levelsSaveSampFor[ii]], 3))
    plot(draws[ , , ii],
         xlim = c(-4, 20),
         ylim = c(-8, 8),
         pch = '.',
         ask = FALSE,
         main = as.expression(main),
         xlab = as.expression(substitute(x[xii], list(xii = 1))),
         ylab = as.expression(substitute(x[xii], list(xii = 2))))
  }
})

```

print

The printing family of functions

Description

The printing family of functions for this package.

Usage

```

## S3 method for class 'EMC':
print(x, ...)
## S3 method for class 'EMCMaxTemper':
print(x, ...)
## S3 method for class 'EMCPlaceTempers':
print(x, ...)

```

Arguments

x an object inheriting from class EMC (generated by functions `randomWalkMetropolis`, `MetropolisHastings`, `parallelTempering` and `evolMonteCarlo`), `EMCMaxTemper` (generated by function `findMaxTemper`) or `EMCPlaceTempers` (generated by function `placeTempers`).

... optional arguments passed to `print.default`; see its documentation.

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

See Also

[randomWalkMetropolis](#), [MetropolisHastings](#), [parallelTempering](#), [evolMonteCarlo](#), [findMaxTemper](#), [placeTempers](#)

randomWalkMetropolis

The Random Walk Metropolis algorithm

Description

Given a target density function and a symmetric proposal generating function, this function produces samples from the target using the random walk Metropolis algorithm.

Usage

```
randomWalkMetropolis(nIters,
                     startingVal,
                     logTarDensFunc,
                     propNewFunc,
                     MHBlocks      = NULL,
                     MHBlockNTimes = NULL,
                     saveFitness   = FALSE,
                     verboseLevel  = 0,
                     ...)
```

Arguments

Below `sampDim` refers to the dimension of the sample space.

`nIters` integer > 0.

`startingVal` double vector of length `sampDim`.

`logTarDensFunc` function of two arguments (`draw`, ...) that returns the target density evaluated in the log scale.

`propNewFunc` function of three arguments (`block`, `currentDraw`, ...) that returns new Metropolis-Hastings proposals. *See details below on the argument block.*

`MHBlocks` list of integer vectors giving dimensions to be blocked together for sampling. It defaults to `as.list(1:sampDim)`, i.e., each dimension is treated as a block on its own. *See details below for an example.*

MHBlockNTimes integer vector of number of times each block given by MHBlocks should be sampled in each iteration. It defaults to `rep(1, length(MHBlocks))`. See details below for an example.

saveFitness logical indicating whether fitness values should be saved. See details below.

verboseLevel integer, a value ≥ 2 produces a lot of output.

... optional arguments to be passed to `logTarDensFunc` and `propNewFunc`.

Details

propNewFunc The `propNewFunc` is called multiple times by varying the `block` argument over `1:length(MHBlocks)`, so this function should know how to generate a proposal from the `currentDraw` depending on which block was passed as the argument. See the example section for sample code.

MHBlocks and MHBlockNTimes Blocking is an important and useful tool in MCMC that helps speed up sampling and hence mixing. Example: Let `sampDim = 6`. Let we want to sample dimensions 1, 2, 4 as one block, dimensions 3 and 5 as another and treat dimension 6 as the third block. Suppose we want to sample the three blocks mentioned above 1, 5 and 10 times in each iteration, respectively. Then we could set `MHBlocks = list(c(1, 2, 4), c(3, 5), 6)` and `MHBlockNTimes = c(1, 5, 10)`

saveFitness The term *fitness* refers to the negative of the `logTarDensFunc` values. By default, the fitness values are not saved, but one can do so by setting `saveFitness = TRUE`.

Value

This function returns a list with the following components:

`draws` matrix of dimension `nIters × sampDim`, if `saveFitness = FALSE`. If `saveFitness = TRUE`, then the returned matrix is of dimension `nIters × (sampDim + 1)`, where the fitness values appear in its last column.

`acceptRatios` matrix of the acceptance rates.

`detailedAcceptRatios` matrix with detailed summary of the acceptance rates.

`nIters` the `nIters` argument.

`startingVal` the `startingVal` argument.

`time` the time taken by the run.

Note

The effect of leaving the default value `NULL` for some of the arguments above are as follows:

```
MHBlocks    as.list(1:sampDim).
MHBlockNTimes rep(1, length(MHBlocks)).
```

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Jun S. Liu (2001). Monte Carlo strategies for scientific computing. Springer.

See Also

[MetropolisHastings](#), [parallelTempering](#), [evolMonteCarlo](#)

Examples

```
samplerObj <-
  with(CigarShapedFuncGenerator1(-13579),
       randomWalkMetropolis(nIters      = 5000,
                            startingVal  = c(0, 0),
                            logTarDensFunc = logTarDensFunc,
                            propNewFunc   = propNewFunc,
                            verboseLevel  = 1))

print(samplerObj)
print(names(samplerObj))
with(samplerObj,
     {
       print(detailedAcceptRatios)
       print(dim(draws))
       plot(draws,
            xlim = c(-3, 5),
            ylim = c(-3, 4),
            pch  = '.',
            ask  = FALSE,
            main = as.expression(paste('# draws:', nIters)),
            xlab = as.expression(substitute(x[xii], list(xii = 1))),
            ylab = as.expression(substitute(x[xii], list(xii = 2))))
     })
```

utilsForExamples *The utility function(s) for examples*

Description

The utility function(s) that are used in the example sections of the exported functions in this package.

Usage

```
CigarShapedFuncGenerator1 (seed)  
CigarShapedFuncGenerator2 (seed)  
VShapedFuncGenerator (seed)  
WShapedFuncGenerator (seed)  
uniModeFuncGenerator (seed)  
twentyModeFuncGenerator (seed)
```

Arguments

seed the seed for random number generation.

Value

A list containing the objects to be used as arguments to the exported functions in the respective example sections of this package.

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

See Also

[randomWalkMetropolis](#), [MetropolisHastings](#), [parallelTempering](#), [evolMonteCarlo](#),
[findMaxTemper](#), [placeTempers](#)

Index

*Topic **datagen**

utilsForExamples, 26

*Topic **methods**

evolMonteCarlo, 1

findMaxTemper, 7

MetropolisHastings, 11

parallelTempering, 14

placeTempers, 19

randomWalkMetropolis, 23

*Topic **print**

print, 23

CigarShapedFuncGenerator1

(utilsForExamples), 26

CigarShapedFuncGenerator2

(utilsForExamples), 26

evolMonteCarlo, 1, 7, 10, 13, 18, 19, 22,
23, 25, 26

findMaxTemper, 7, 9, 20, 22, 23, 26

MetropolisHastings, 11, 23, 25, 26

parallelTempering, 6, 10, 13, 14, 22, 23,
25, 26

placeTempers, 10, 19, 23, 26

print, 23

randomWalkMetropolis, 13, 23, 23, 26

twentyModeFuncGenerator

(utilsForExamples), 26

uniModeFuncGenerator

(utilsForExamples), 26

utilsForExamples, 26

VShapedFuncGenerator

(utilsForExamples), 26

WShapedFuncGenerator

(utilsForExamples), 26