

Package ‘EMCC’

February 14, 2012

Type Package

Title Evolutionary Monte Carlo (EMC) methods for clustering

Version 1.2

Date 2011-12-10

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

Maintainer Gopi Goswami <grgoswami@gmail.com>

Depends R (>= 1.9.0), MASS, mclust, EMC

Description evolutionary Monte Carlo methods for clustering, temperature ladder construction and placement. For mclust see <http://cran.r-project.org/web/packages/mclust/LICENSE>

License GPL (>= 2) | file LICENSE

Repository CRAN

Date/Publication 2011-12-11 17:42:17

R topics documented:

evolMonteCarloClustering	2
findMaxTemper	5
placeTempers	11
print	15
utilsForExamples	16

Index	18
--------------	-----------

evolMonteCarloClustering

evolutionary Monte Carlo clustering algorithm

Description

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

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`.

Usage

```
evolMonteCarloClustering(nIters,
                        temperLadder,
                        startingVals,
                        logTarDensFunc,
                        MHMergeProb      = 0.5,
                        moveProbsList    = NULL,
                        moveNTimesList   = NULL,
                        levelsSaveSampFor = NULL,
                        saveFitness       = FALSE,
                        verboseLevel     = 0,
                        ...)
```

Arguments

<code>nIters</code>	integer > 0.
<code>temperLadder</code>	double vector with all <i>positive</i> entries, in <i>decreasing</i> order.
<code>startingVals</code>	double matrix of dimension <code>temperLadderLen</code> × <code>sampDim</code> or vector of length <code>sampDim</code> , in which case the same starting values are used for every temperature level.
<code>logTarDensFunc</code>	function of two arguments (<code>draw</code> , ...) that returns the target density evaluated in the log scale.
<code>MHMergeProb</code>	double in (0, 1). <i>See details below for the use of this argument.</i>
<code>moveProbsList</code>	named list of probabilities adding upto 1.
<code>moveNTimesList</code>	named list of integers ≥ 0.
<code>levelsSaveSampFor</code>	integer vector with <i>positive</i> entries.
<code>saveFitness</code>	logical.
<code>verboseLevel</code>	integer, a value ≥ 2 produces a lot of output.
...	optional arguments to be passed to <code>logTarDensFunc</code> .

Details

The EMCC algorithm The evolutionary Monte Carlo clustering (EMCC; Goswami and Liu, 2007) algorithm is composed of the following moves:

MH	Metropolis-Hastings or mutation
SCSC_ONE_NEW	sub-cluster swap crossover: one new
SCSC_TWO_NEW	sub-cluster swap crossover: two new
SCRC	sub-cluster reallocation crossover
RE	(random) exchange

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

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

```
moveProbsList = list(MH           = 0.5,
                    SCSC_TWO_NEW = 0.25,
                    SCRC          = 0.25)
```

```
moveTimesList = list(MH           = 1,
                    SCSC_TWO_NEW = floor(temperLadderLen / 2),
                    SCRC          = floor(temperLadderLen / 2),
                    RE            = temperLadderLen)
```

`MHMergeProb` In the MH or the mutation step, each of the `sampDim`-many objects are proposed to either merge with an existing cluster or split to form its own cluster with probability `MHMergeProb` and $(1 - \text{MHMergeProb})$, respectively (see Goswami and Liu, 2007).

`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:

```

moveProbsList list(MH = 0.5, RC = 0.25, 'SCSC_TWO_NEW' = 0.25).
moveNTimesList list(MH = 1, RC = mm, 'SCSC_TWO_NEW' = mm, RE = nn),
               where mm <- floor(nn / 2) and nn <- temperLadderLen.
levelsSaveSampFor temperLadderLen.
  
```

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

References

Gopi Goswami, Jun S. Liu and Wing H. Wong (2007). *Evolutionary Monte Carlo Methods for Clustering*. Journal of Computational and Graphical Statistics, 16:4:855-876.

Examples

```

## Not run:
## The following example is a simple stochastic optimization problem,
## the set up is same as that of findMaxTemper and placeTemper. Here
## no "heating up" is necessary, and hence the maximum temperature is
## the coldest one, namely, 0.5.
##
## However, we run evolMonteCarloClustering on this example with a
## temperature ladder that is the output of placeTemper, which
## assumes that the maximum temperature is 5.
KMeansObj <- KMeansFuncGenerator1(-97531)
samplerObj <-
  with(KMeansObj,
    {
      temperLadder <- c(5.0000000, 1.5593974, 1.1028349, 0.9220684,
                       0.7900778, 0.6496648, 0.5135825, 0.5000000)
    }
  )
  
```

```

nLevels      <- length(temperLadder)
sampDim      <- nrow(yy)
startingVals <- sample(c(0, 1),
                      size    = nLevels * sampDim,
                      replace = TRUE)

startingVals <- matrix(startingVals, nrow = nLevels, ncol = sampDim)
moveProbsList <- list(MH      = 0.4,
                     RC      = 0.3,
                     'SCSC_TWO_NEW' = 0.3)

mm          <- floor(nLevels / 2)
moveNTimesList <- list(MH      = 1,
                     RC      = mm,
                     'SCSC_TWO_NEW' = mm,
                     RE      = nLevels)

evolMonteCarloClustering(nIters      = 5000,
                          temperLadder = temperLadder,
                          startingVals = startingVals,
                          logTarDensFunc = logTarDensFunc,
                          moveProbsList = moveProbsList,
                          moveNTimesList = moveNTimesList,
                          levelsSaveSampFor = seq_len(nLevels),
                          saveFitness = TRUE,
                          verboseLevel = 1)

  })
print(samplerObj)
print(names(samplerObj))
with(c(samplerObj, KMeansObj),
{
  print(acceptRatios)
  print(detailedAcceptRatios)
  print(dim(draws))
  fitnessCol <- ncol(draws[ , , 1])
  sub      <- paste('uniform prior on # of clusters: DU[',
                  priorMinClusters, ', ',
                  priorMaxClusters, ']', sep = '')
  for (ii in rev(seq_along(levelsSaveSampFor))) {
    main <- paste('EMCC (MAP) clustering (temper = ',
                  round(temperLadder[levelsSaveSampFor[ii]], 3), ')',
                  sep = '')
    MAPRow <- which.min(draws[ , fitnessCol, ii])
    clusterPlot(clusterInd      = draws[MAPRow, -fitnessCol, ii],
                 data          = yy,
                 main          = main,
                 sub           = sub,
                 knownClusterMeans = knownClusterMeans)
  }
})

## End(Not run)

```

Description

The evolutionary Monte Carlo clustering (EMCC) algorithm needs a temperature ladder. This function finds the maximum temperature for constructing the ladder.

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 [evolMonteCarloClustering](#), so some of the arguments below have the same name and meaning as the corresponding ones for [evolMonteCarloClustering](#). See details below for explanation on the arguments.

Usage

```
findMaxTemper(nIters,
              statsFuncList,
              startingVals,
              logTarDensFunc,
              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

<code>nIters</code>	integer > 0.
<code>statsFuncList</code>	list of functions of one argument each, which return the value of the statistic evaluated at one MCMC sample or draw.
<code>startingVals</code>	double matrix of dimension <code>temperLadderLen</code> × <code>sampDim</code> or vector of length <code>sampDim</code> , in which case the same starting values are used for every temperature level.
<code>logTarDensFunc</code>	function of two arguments (<code>draw</code> , ...) that returns the target density evaluated in the log scale.
<code>temperLadder</code>	double vector with all <i>positive</i> entries, in <i>decreasing</i> order.
<code>temperLimits</code>	double vector with <i>two positive</i> entries.
<code>ladderLen</code>	integer > 0.
<code>scheme</code>	character.
<code>schemeParam</code>	double > 0.
<code>cutoffDStats</code>	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 (i.e., objects 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].

`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 `placeTemperers` 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:

<code>temperLadder</code>	the temperature ladder used for the underlying MCMC run.
<code>DStats</code>	the D -statistic (Goswami and Liu, 2007) values used to find the maximum temperature.
<code>cutoffDStats</code>	the <code>cutoffDStats</code> argument.
<code>nIters</code>	the post burn-in <code>nIters</code> .
<code>levelsSaveSampFor</code>	the <code>levelsSaveSampFor</code> argument.
<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>startingVals</code>	the <code>startingVals</code> argument.
<code>intermediate statistics</code>	a bunch of intermediate statistics used in the computation of <code>DStats</code> , namely, <code>MCEsts</code> , <code>MCVarEsts</code> , <code>MCESS</code> , <code>ISEsts</code> , <code>ISVarEsts</code> , <code>ISESS</code> , each being computed for all the statistics provided by <code>statsFuncList</code> argument.
<code>time</code>	the time taken by the run.

Note

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

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.

Gopi Goswami, Jun S. Liu and Wing H. Wong (2007). Evolutionary Monte Carlo Methods for Clustering. Journal of Computational and Graphical Statistics, 16:4:855-876.

See Also

[placeTempers](#), [evolMonteCarloClustering](#)

Examples

```
## Not run:
## The following example is a simple stochastic optimization problem,
## and thus it does not require any "heating up", and hence the
## maximum temperature turns out to be the coldest one, i.e, 0.5.
adjMatSum <-
  function (xx)
  {
    xx    <- as.integer(xx)
    adjMat <- outer(xx, xx, function (id1, id2) { id1 == id2 })
    sum(adjMat)
  }
modeSensitive1 <-
  function (xx)
  {
    with(partitionRep(xx),
      {
        rr  <- 1 + seq_along(clusterLabels)
        freq <- sapply(clusters, length)
        oo  <- order(freq, decreasing = TRUE)
        sum(sapply(clusters[oo], sum) * log(rr))
      })
  }
entropy <-
  function (xx)
  {
    yy <- table(as.vector(xx, mode = "numeric"))
    zz <- yy / length(xx)
    -sum(zz * log(zz))
  }
}
```

```

maxProp <-
  function (xx)
  {
    yy <- table(as.vector(xx, mode = "numeric"))
    oo <- order(yy, decreasing = TRUE)
    yy[oo][1] / length(xx)
  }
statsFuncList <- list(adjMatSum, modeSensitive1, entropy, maxProp)
KMeansObj <- KMeansFuncGenerator1(-97531)
maxTemperObj <-
  with(KMeansObj,
    {
      temperLadder <- c(20, 10, 5, 1, 0.5)
      nLevels <- length(temperLadder)
      sampDim <- nrow(yy)
      startingVals <- sample(c(0, 1),
                            size = nLevels * sampDim,
                            replace = TRUE)
      startingVals <- matrix(startingVals, nrow = nLevels, ncol = sampDim)
      findMaxTemper(nIters = 5000,
                    statsFuncList = statsFuncList,
                    temperLadder = temperLadder,
                    startingVals = startingVals,
                    logTarDensFunc = logTarDensFunc,
                    levelsSaveSampFor = seq_len(nLevels),
                    doFullAnal = TRUE,
                    saveFitness = TRUE,
                    verboseLevel = 1)
    })
print(maxTemperObj)
print(names(maxTemperObj))
with(c(maxTemperObj, KMeansObj),
  {
    fitnessCol <- ncol(draws[ , , 1])
    sub <- paste('uniform prior on # of clusters: DU[',
                 priorMinClusters, ', ',
                 priorMaxClusters, ']', sep = '')
    for (ii in rev(seq_along(levelsSaveSampFor))) {
      main <- paste('EMCC (MAP) clustering (temper = ',
                    round(temperLadder[levelsSaveSampFor[ii]], 3), ')',
                    sep = '')
      MAPRow <- which.min(draws[ , fitnessCol, ii])
      clusterPlot(clusterInd = draws[MAPRow, -fitnessCol, ii],
                  data = yy,
                  main = main,
                  sub = sub,
                  knownClusterMeans = knownClusterMeans)
    }
  })
## End(Not run)

```

placeTempers *Place the intermediate temperatures between the temperature limits*

Description

The evolutionary Monte Carlo clustering (EMCC) algorithm needs a temperature ladder. This function places the intermediate temperatures between the minimum and the maximum temperature for the ladder.

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 `evolMonteCarloClustering`, so some of the arguments below have the same name and meaning as the corresponding ones for `evolMonteCarloClustering`. *See details below for explanation on the arguments.*

Usage

```
placeTempers(nIters,
             acceptRatioLimits,
             ladderLenMax,
             startingVals,
             logTarDensFunc,
             temperLadder      = NULL,
             temperLimits     = NULL,
             ladderLen        = 15,
             scheme           = 'exponential',
             schemeParam      = 1.5,
             guideMe          = TRUE,
             levelsSaveSampFor = NULL,
             saveFitness      = FALSE,
             verboseLevel     = 0,
             ...)
```

Arguments

<code>nIters</code>	integer > 0.
<code>acceptRatioLimits</code>	double vector of <i>two probabilities</i> .
<code>ladderLenMax</code>	integer > 0.
<code>startingVals</code>	double matrix of dimension <code>temperLadderLen × sampDim</code> or vector of length <code>sampDim</code> , in which case the same starting values are used for every temperature level.
<code>logTarDensFunc</code>	function of two arguments (<code>draw</code> , ...) that returns the target density evaluated in the log scale.
<code>temperLadder</code>	double vector with all <i>positive</i> entries, in <i>decreasing</i> order.
<code>temperLimits</code>	double vector with <i>two positive</i> entries.

ladderLen integer > 0.
 scheme character.
 schemeParam double > 0.
 guideMe logical.
 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

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:

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

Note

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

<code>temperLadder</code>	valid <code>temperLimits</code> , <code>ladderLen</code> , <code>scheme</code> and <code>schemeParam</code> are provided, which are used to construct the <code>temperLadder</code> .
<code>temperLimits</code>	a valid <code>temperLadder</code> is provided.
<code>levelsSaveSampFor</code>	<code>temperLadderLen</code> .

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.

Gopi Goswami, Jun S. Liu and Wing H. Wong (2007). *Evolutionary Monte Carlo Methods for Clustering*. *Journal of Computational and Graphical Statistics*, 16:4:855-876.

See Also

[findMaxTemper](#), [evolMonteCarloClustering](#)

Examples

```
## Not run:
## The following example is a simple stochastic optimization problem,
## the set up is same as that of findMaxTemper. Here no "heating up"
## is necessary, and hence the maximum temprature is the coldest one,
## namely, 0.5.
##
## However, we do the temperature placement to show how placeTempers
## works, assuming the maximum temperature is 5.
KMeansObj <- KMeansFuncGenerator1(-97531)
placeTempersObj <-
  with(KMeansObj,
    {
      nLevels <- 15
      sampDim <- nrow(yy)
      startingVals <- sample(c(0, 1),
                            size = nLevels * sampDim,
                            replace = TRUE)
      startingVals <- matrix(startingVals, nrow = nLevels, ncol = sampDim)
      placeTempers(nIters = 5000,
                  acceptRatioLimits = c(0.5, 0.6),
                  ladderLenMax = 50,
                  startingVals = startingVals,
                  logTarDensFunc = logTarDensFunc,
                  temperLimits = c(0.5, 5),
                  ladderLen = nLevels,
                  scheme = 'geometric',
                  levelsSaveSampFor = seq_len(nLevels),
                  saveFitness = TRUE,
                  verboseLevel = 1)
    })
print(placeTempersObj)
print(names(placeTempersObj))
with(c(placeTempersObj, KMeansObj),
  {
    fitnessCol <- ncol(draws[ , , 1])
    sub <- paste('uniform prior on # of clusters: DU[',
                priorMinClusters, ', ',
                priorMaxClusters, ']', sep = '')
    for (ii in rev(seq_along(levelsSaveSampFor))) {
```

```

    main <- paste('EMCC (MAP) clustering (temper = ',
                 round(temperLadder[levelsSaveSampFor[ii]], 3), '), ',
                 sep = '')
    MAPRow <- which.min(draws[ , fitnessCol, ii])
    clusterPlot(clusterInd      = draws[MAPRow, -fitnessCol, ii],
                data            = yy,
                main            = main,
                sub              = sub,
                knownClusterMeans = knownClusterMeans)
  }
})

## End(Not run)

```

print

The printing family of functions

Description

The printing family of functions for this package.

Usage

```

## S3 method for class 'EMCC'
print(x, ...)
## S3 method for class 'EMCCMaxTemper'
print(x, ...)
## S3 method for class 'EMCCPlaceTempers'
print(x, ...)

```

Arguments

x an object inheriting from class EMCC (generated by function `evolMonteCarloClustering`), EMCCMaxTemper (generated by function `findMaxTemper`) or EMCCPlaceTempers (generated by function `placeTempers`).

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

Author(s)

Gopi Goswami <goswami@stat.harvard.edu>

See Also

[evolMonteCarloClustering](#), [findMaxTemper](#), [placeTempers](#)

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

```
partitionRep(clusterInd)
clusterPlot(clusterInd,
             data,
             main          = '',
             sub           = '',
             knownClusterMeans = NULL,
             ...)
KMeansFuncGenerator1(seed, plotIt = TRUE)
```

Arguments

clusterInd	vector of cluster indicators.
data	a matrix with two columns representing the two-dimensional data clustered by clusterInd.
main	the title of the plot.
sub	the sub-title of the plot.
knownClusterMeans	a matrix with two columns (for the two dimensions), the rows containing the cluster means. These are plotted when provided.
seed	the seed for random number generation.
plotIt	logical, controls the plotting of the generated data.
...	optional arguments to be passed to plot; see its documentation.

Value

partitionRep	this function returns a list with two components, namely, clusterLabels (the unique cluster identifiers) and clusters (the partitioning of the cluster identifiers), as a list.
KMeansFuncGenerator1	this function returns 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

[evolMonteCarloClustering](#), [findMaxTemper](#), [placeTempers](#)

Index

*Topic **datagen**

utilsForExamples, 16

*Topic **methods**

evolMonteCarloClustering, 2

findMaxTemper, 6

placeTempers, 11

*Topic **print**

print, 15

clusterPlot (utilsForExamples), 16

evolMonteCarloClustering, 2, 6, 9, 11, 14,
15, 17

findMaxTemper, 5, 12, 14, 15, 17

KMeansFuncGenerator1

(utilsForExamples), 16

partitionRep (utilsForExamples), 16

placeTempers, 9, 11, 15, 17

print, 15

utilsForExamples, 16