Package ‘pmclust’

December 19, 2016

Version 0.1-9
Date 2016-12-17
Title Parallel Model-Based Clustering using
   Expectation-Gathering-Maximization Algorithm for Finite Mixture
   Gaussian Model
Depends R (>= 3.0.0), pbdMPI (>= 0.3-1), pbdBASE (>= 0.4-3), pbdDMAT
   (>= 0.4-0)
Imports methods, MASS
Enhances MixSim
LazyLoad yes
LazyData yes
Description Aims to utilize model-based clustering (unsupervised)
   for high dimensional and ultra large data, especially in a distributed
   manner. The code employs pbdMPI to perform a
   expectation-gathering-maximization algorithm
   for finite mixture Gaussian
   models. The unstructured dispersion matrices are assumed in the
   Gaussian models. The implementation is default in the single program
   multiple data programming model. The code can be executed
   through pbdMPI and independent to most MPI applications.
   See the High Performance
   Statistical Computing website for more information, documents
   and examples.
License GPL (>= 2)
URL http://r-pbd.org/
BugReports http://group.r-pbd.org/
MailingList Please send questions and comments regarding pbdR to
   RBigData@gmail.com
NeedsCompilation yes
Author Wei-Chen Chen [aut, cre],
   George Ostrouchov [aut]
pmclust-package

Maintainer  Wei-Chen Chen <wccsnow@gmail.com>
Repository  CRAN
Date/Publication  2016-12-19 08:34:03

R topics documented:

pmclust-package ................................................. 2
as functions .................................................. 4
assign.N.sample ............................................. 5
EM-like algorithms ........................................... 6
generate.basic ............................................... 8
generate.MixSim ............................................ 10
get.N.CLASS .................................................. 12
Independent logL ............................................. 13
Initialization ............................................... 14
mb.print ....................................................... 16
One E-Step ................................................... 17
One M-Step ................................................... 18
One Step of EM algorithm ................................. 19
pmclust and pkmeans ....................................... 20
print.object .................................................. 22
Read Me First ............................................... 23
Set Global Variables .................................... 25
Set of CONTROL ........................................... 26
Set of PARAM ............................................... 28
Update Class of EM or Kmeans Results ............... 29

Index  31

pmclust-package  Parallel Model-Based Clustering

Description

The pmclust aims to utilize model-based clustering (unsupervised) for high dimensional and ultra large data, especially in a distributed manner. The package employs pbdMPI to perform a parallel version of expectation and maximization (EM) algorithm for finite mixture Gaussian models. The unstructured dispersion matrices are assumed in the Gaussian models. The implementation is default in the single program multiple data (SPMD) programming model. The code can be executed through pbdMPI and independent to most MPI applications. See the High Performance Statistical Computing (HPSC) website for more information, documents and examples.
Details

- **Package**: pmclust
- **Type**: Package
- **License**: GPL
- **LazyLoad**: yes

The main function is `pmclust` implementing the parallel EM algorithm for mixture multivariate Gaussian models with unstructured dispersions. This function groups a data matrix `X.gbd` or `X.spmd` into $k$ clusters where `X.gbd` or `X.spmd` is potentially huge and taken from the global environment `.GlobalEnv` or `.pmclustEnv`.

Other main functions `em.step`, `aecm.step`, `apecm.step`, and `apecma.step` may provide better performance than the `em.step` in terms of computing time and convergent iterations.

`kmeans.step` provides the fastest clustering among above algorithms, but it is restricted by Euclidean distance and spherical dispersions.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov

References

- Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

See Also

- `em.step`, `aecm.step`, `apecm.step`, `apecma.step`, `kmeans.step`
- `em.step.dmata`, `kmeans.step.dmata`
Examples

```r
## Not run:
### Under command mode, run the demo with 2 processors by
### (Use Rscript.exe for windows system)
mpiexec -np 2 Rscript -e 'demo(gbd_em,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(gbd_aecm,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(gbd_apecm,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(gbd_kmeans,"pmclust",ask=F,echo=F)'

mpiexec -np 2 Rscript -e 'demo(ex_em,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(ex_aecm,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(ex_apecm,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(ex_kmeans,"pmclust",ask=F,echo=F)'

mpiexec -np 2 Rscript -e 'demo(dmat_em,"pmclust",ask=F,echo=F)'
mpiexec -np 2 Rscript -e 'demo(dmat_kmeans,"pmclust",ask=F,echo=F)'
```

## End(Not run)

---

**as functions**

Convert between X.gbd (X.spmd) and X.dmat

**Description**

This function will convert a matrix of format from GBD row-major to ddmatrix vice versa.

**Usage**

```r
as.dmat(X.spmd, bldim = .pdb_env$BLDIM, ICTXT = .pdb_env$ICTXT, 
  comm = .pdb_env$SPMD.CT$comm)

as.gbd(X.dmat, comm = .pdb_env$SPMD.CT$comm)

as.spmd(X.dmat, comm = .pdb_env$SPMD.CT$comm)
```

**Arguments**

- `X.spmd` an input dataset of format GBD/SPMD row-major to be converted.
- `X.dmat` an input dataset of format ddmatrix to be converted.
- `bldim` block dimension, see `pdbBASE` and `pdbDMAT` for details.
- `ICTXT` block context, see `pdbBASE` and `pdbDMAT` for details.
- `comm` communicator, see `pdbMPI` for details.

**Details**

This function will provide a quick conversion to the input data `X.spmd` which is a default dataset to be clustered in `pmclust`. Usually, this matrix is distributed in GBD row-major format. For fully utilizing `pdbBASE` and `pdbDMAT`, it need to be converted to ddmatrix format.
assign.N.sample

Value
A ddmatrix is returned.

Author(s)
Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References
Programming with Big Data in R Website: http://r-pbd.org/

See Also
kmeans.step.dmat.

Examples
## Not run:
# Examples can be found in the help pages of
# kmeans.step.dmat().

## End(Not run)

assign.N.sample Obtain a Set of Random Samples for X.spmd

Description
This utility function samples data randomly from X.spmd to form a relatively small subset of original data. The EM algorithm on the smaller subset is topically performing fast and capturing rough structures of entire dataset.

Usage
assign.N.sample(total.sample = 5000, N.org.spmd)

Arguments
total.sample a total number of samples which will be selected from the original data X.spmd.
N.org.spmd the original data size, i.e. nrow(X.spmd).

Details
This utility function performs simple random sampling without replacement for the original dataset X.spmd. Different random seeds should be set before calling this function.

Value
A list variable will be returned and containing:
EM-like algorithms

N total sample size across all S processors
n.spmd sample size of given processor
N.allspmds a collection of sample sizes for all S processors
ID.spmd index of selected samples ranged from 1 to N.org.spmd

Note that N and N.allspmds are the same across all S processors, but N.spmd and ID.spmd are most likely all distinct. The lengths of these elements are 1 for N and N.spmd, S for N.allspmd, and N.spmd for ID.spmd.

Author(s)
Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References
Programming with Big Data in R Website: http://r-pbd.org/

See Also
set.global

Examples

```r
## Not run:
# Save code in a file "demo.r" and run in 4 processors by
# > mpiexec -np 4 Rscript demo.r

### Setup environment.
library(pmclust, quiet = TRUE)
comm.set.seed(123)

### Generate an example data.
N.org.spmd <- 5000 + sample(1:1000, 1)
ext.spmd <- assign.Nsample(total.sample = 5000, N.org.spmd)
cat("Rank:", comm.rank(), " Size:", ret.spmd$N.spmd,
   "\n", sep = "")

### Quit.
finalize()

## End(Not run)
```

Description

The EM-like algorithm for model-based clustering of finite mixture Gaussian models with unstructured dispersions.

*.dmat's are ddmatrix versions.
**EM-like algorithms**

**Usage**

```
em.step(PARAM.org)
aecm.step(PARAM.org)
apecm.step(PARAM.org)
apecma.step(PARAM.org)
kmeans.step(PARAM.org)
em.step.dmat(PARAM.org)
kmeans.step.dmat(PARAM.org)
```

**Arguments**

`PARAM.org` an original set of parameters generated by `set.global`.

**Details**

A global variable called `X.spm` should exist in the `.pmclustEnv` environment, usually the working environment. The `X.spm` is the data matrix to be clustered, and this matrix has a dimension `N.spm` by `p`.

A `PARAM.org` will be a local variable inside all EM-linke functions `em.step`, `aecm.step`, `apecm.step`, `apecma.step`, and `kmeans.step`. This variable is a list containing all parameters related to models. This function also updates in the parameters by the EM-like algorithms, and return the convergent results. The details of list elements are initially generated by `set.global`.

**Value**

A convergent results will be returned the other list variable containing all new parameters which represent the components of models. See the help page of `PARAM` or `PARAM.org` for details.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)


generate.basic

Generate Examples for Testing

Description
This function will generate a small set of data for testing algorithms.

Usage
generate.basic(N.allspmds, N.spmd, N.K.spmd, N, p, K)
### Arguments

- **N.allspmds**: a collection of sample sizes for all $S$ processors, i.e. a vector of length $S$.
- **N.spmd**: total sample size of given processor.
- **N.K.spmd**: sample size of each clusters given processor, i.e. sum over $N.K.spmd$ is $N.spmd$, a vector of length $K$.
- **N**: total sample size across all $S$ processors, i.e. sum over $N.spmd$ is $N$.
- **p**: dimension of data $X.spmd$, i.e. ncol($X.spmd$).
- **K**: number of clusters.

### Details

For all $S$ processors, this function will generate in total $N$ observations from $K$ clusters in $p$ dimensions.

The clusters centers and dispersions are generated automatically inside the code. Currently, it is not allowed for users to change, but it is not difficult to specify them by mimicking this code.

### Value

A set of simulated data and information will be returned in a list variable including:

- **K**: number of clusters, as the input
- **p**: dimension of data $X.spmd$, as the input
- **N**: total sample size, as the input
- **N.allspmds**: a collection of sample sizes for all $S$ processors, as the input
- **N.spmd**: total sample size of given processor, as the input
- **N.K.spmd**: sample size of each clusters given processor, as the input
- **X.spmd**: generated data set with dimension with dimension $N.spmd * p$
- **CLASS.spmd**: true id of each data, a vector of length $N.spmd$ and has values from 1 to $K$
- **N.CLASS.spmd**: true sample size of each clusters, a vector of length $K$

### Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

### References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

### See Also

- `generate.MixSim`

### Examples

```r
# Not run:
# Examples can be found in the help pages of em.step(),
# aecm.step(), apecm.step(), and apecma.step().
```
generate.MixSim  

*Generate MixSim Examples for Testing*

### Description

This function utilizes **MixSim** to generate sets of data for testing algorithms.

### Usage

```r
generate.MixSim(N, p, K, MixSim.obj = NULL, MaxOmega = NULL, 
                 BarOmega = NULL, PiLow = 1.0, sph = FALSE, hom = FALSE)
```

### Arguments

- `N` total sample size across all `S` processors, i.e. sum over `N.spmd` is `N`.
- `p` dimension of data `X.spmd`, i.e. `ncol(X.spmd)`.
- `K` number of clusters.
- `MixSim.obj` an object returned from `MixSim`.
- `MaxOmega` maximum overlap as in `MixSim`.
- `BarOmega` averaged overlap as in `MixSim`.
- `PiLow` lower bound of mixture proportion as in `MixSim`.
- `sph` sph as in `MixSim`.
- `hom` hom as in `MixSim`.

### Details

If `MixSim.obj` is NULL, then `BarOmega` and `MaxOmega` will be used in `MixSim` to obtain a new `MixSim.obj`.

### Value

A set of simulated data and information will be returned in a list variable including:

- `K` number of clusters, as the input
- `p` dimension of data `X.spmd`, as the input
- `N` total sample size, as the input
- `N.allspmds` a collection of sample sizes for all `S` processors, as the input
- `N.spmd` total sample size of given processor, as the input
- `X.spmd` generated data set with dimension with dimension `N.spmd * p`
- `CLASS.spmd` true id of each data, a vector of length `N.spmd` and has values from 1 to `K`
- `N.CLASS.spmd` true sample size of each clusters, a vector of length `K`
- `MixSim.obj` the true model where data `X.spmd` generated from
Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References


Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

See Also

generate.basic.

Examples

```r
## Not run:
# Save code in a file "demo.r" and run in 4 processors by
# > mpiexec -np 4 Rscript demo.r

### Setup environment.
library(pmclust, quiet = TRUE)

### Generate an example data.
N <- 5000
p <- 2
K <- 2
data.spmd <- generate.MixSim(N, p, K, BarOmega = 0.01)
X.spmd <- data.spmd$X.spmd

### Run clustering.
PARAM.org <- set.global(K = K) # Set global storages.
# PARAM.org <- initial.em(PARAM.org) # One initial.
PARAM.org <- initial.RndEM(PARAM.org) # Ten initials by default.
PARAM.new <- apecma.step(PARAM.org) # Run APECMa.
em.update.class() # Get classification.

### Get results.
N.CLASS <- get.N.CLASS(K)
comm.cat("# of class:", N.CLASS, "\n")
comm.cat("# of class (true):", data.spmd$N.CLASS.spmd, "\n")

### Quit.
finalize()

## End(Not run)
```
get.N.CLASS

**Obtain Total Elements for Every Clusters**

**Description**

This function will collect the total elements for every clusters from all processors that the all reduced calls with the sum operation will be performed.

get.N.CLASS.dmat is a ddmatrix version of get.N.CLASS.

The get.CLASS returns class ids.

**Usage**

```
get.N.CLASS(K)
get.N.CLASS.dmat(K)
get.CLASS(PARAM)
```

**Arguments**

- **K**
  - the total number of clusters.
- **PARAM**
  - a set of parameters.

**Details**

The final results are distributed in all processors including the total elements for each cluster. The global variable `CLASS.spmd` stores the identification for each observation on each processors. This function will first summary `CLASS.spmd` in K categories, then use the all reduce function with the sum operation to add the numbers by clusters. The `COMM.RANK 0` will be used to take care the printing.

**Value**

K numbers will be returned that are the total elements for each cluster. Sum of these K numbers should be equal to N the total number of observations.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)
Independent logL

See Also
em.step, aecm.step, apecm.step,
apecma.step, kmeans.step,
kmeans.step.dmat.

Examples

## Not run:
# Examples can be found in the help pages of em.step(),
# aecm.step(), apecm.step(), apecma.step(), and kmeans.step().
#
# Examples for ddmatrix version can be found in the help pages of
# kmeans.step.dmat().

## End(Not run)

Independent logL  Independent Function for Log Likelihood

Description

This function is for debugging only and for checking if the observed data log likelihood is consistent
for each EM iteration.

indep.logL.dmat is a ddmatrix version of indep.logL.

Usage

indep.logL(PARAM)

indep.logL.dmat(PARAM)

Arguments

PARAM  a set of parameters.

Details

This function will provide an observed data log likelihood based on the current parameter PARAM.

This function will take in information from global, but no global variables will be updated by this function.

This function also don’t take care the numerical issues, so the return value may be inaccurate some-
times.

Value

An observed data log likelihood will be returned. This value can quickly compare with the log
likelihood computed inside em.onestep. Small difference is allowed, but large difference indicates
bugs of code or illness of data.
Initialization

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References

Programming with Big Data in R Website: http://r-pbd.org/

See Also

set.global, em.onestep.

Examples

```
## Not run:
# This is a core function for em.estep()
# see the source code for details.
# Reset .pmclustEnv$CONTROL$debug to turn on this function
# automatically for each EM iteration.

## End(Not run)
```

---

Initialization for EM-like Algorithms

Description

These functions implement initialization of EM-like algorithms for model-based clustering based on `X.spmd`, and initialization of K-means algorithm by randomly picking samples from data based on `X.spmd`.

*.dmat's are ddmatrix versions.

Usage

```
initial.RndEM(PARAM)
initial.em(PARAM, MU = NULL)
initial.center(PARAM, MU = NULL)

initial.RndEM.dmat(PARAM)
initial.em.dmat(PARAM, MU = NULL)
initial.center.dmat(PARAM, MU = NULL)
```

Arguments

- PARAM: an original set of parameters generated by `set.global`.
- MU: a center matrix with dim = \( p \times K \).
Details

For initial.RndEM, the procedure is implemented by randomly picking .pmclustEnv$CONTROL$RndEM.iter starting points from data \texttt{X.spmd} and run one E-step to obtain the log likelihood. Then pick the starting point with the highest log likelihood as the best choice to pursue the MLEs in further EM iterations.

This function repeatedly run \texttt{initial.em} by .pmclustEnv$CONTROL$RndEM.iter random starts and pick the best initializations from the random starts.

For initial.em, it takes \texttt{X.spmd} from the global environment and randomly pick $K$ of them as the centers of $K$ groups. If \texttt{MU} is specified, then this \texttt{MU} will be the centers. The default identity dispersion in \texttt{PARAM}$\Sigma$ will be used. Then, one E-step will be called to obtain the log likelihood and new classification will be updated.

This function is used to implement the RndEM procedure for more elaborate initialization scheme in \texttt{initial.RndEM}. Potentially, several random starts should be tried before running EM algorithms. This can benefit in two aspects including: shorter convergent iterations and better classification results.

For initial.center, if \texttt{MU} is given, then the center will be assigned according.

Value

The best initial starting points \texttt{PARAM} will be returned among all random starting points. The number of random starting points is assigned by \texttt{set.global} to a list variable \texttt{CONTROL}. See the help page of \texttt{initial.em} and \texttt{set.global} for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References

Programing with Big Data in R Website: \url{http://r-pbd.org/}


See Also

\texttt{set.global.em.step, aecm.step, apecm.step.apecma.step, kmeans.step.}

\texttt{set.global.dmat, kmeans.step.dmat.}

Examples

```r
## Not run:
# Examples can be found in the help page of em.step(),
# aecm.step(), apecm.step(), apecma.step(), and kmeans.step().

# Examples for ddmatrix version can be found in the help pages of
# kmeans.step.dmat().

## End(Not run)
```
Print Results of Model-Based Clustering

Description
This function will print summarized messages for model-based clustering.

Usage
mb.print(PARAM, CHECK)

Arguments
- PARAM: a set of convergent parameters to be printed.
- CHECK: a set of checking parameters to be printed.

Details
This function will provide a quick summary from the PARAM and CHECK typically the output of clusterings when algorithms stop. The COMM.RANK 0 will be used to take care the printing.

Value
Summarized messages will print/cat on screen by default.

Author(s)
Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References
Programming with Big Data in R Website: http://r-pbd.org/

See Also
- em.step, aecm.step, apecm.step,
- apecma.step

Examples
## Not run:
# Examples can be found in the help pages of em.step(),
# aecm.step(), apecm.step(), and apecma.step().
## End(Not run)
One E-Step

Compute One E-step and Log Likelihood Based on Current Parameters

Description

This function will perform one E-step based on current parameters. This is a core function of `em.onestep`. `e.step.dmat` is a `ddmatrix` version of `e.step`.

Usage

```r
e.step(PARAM, update.logL = TRUE)
e.step.dmat(PARAM, update.logL = TRUE)
```

Arguments

- `PARAM` a set of parameters.
- `update.logL` TRUE for update observed data log likelihood.

Details

This function will base on the current parameter to compute the densities for all observations for all \( K \) components, and update the \( Z_{spmd} \) matrix. If the `update.logL` is true, then the log likelihood \( W.spmd.rowSums \) will be also updated before the end of this function.

Sum of \( W.spmd.rowSums \) of all processors will be the observed data log likelihood for the current iteration.

Value

Several global variables will be overwrote after this call including \( Z.spmd \), \( W.spmd.rowSums \), \( W.spmd \), \( U.spmd \), and \( Z.colSums \).

Computing Issues

Since the clusters can be degenerated or highly flat, these cause very large positive or negative exponents in densities. The log likelihood will tend to be inaccurate (not finite). Since the mixture structures can be over fit, this also cause very tiny mixing proportions. The poster probabilities can also unstable (NaN).

These can be solved by rescaling the range of exponents carefully and adjust the scaling factor on the log values. See `CONTROL` for details about constrains on E- and M-steps.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.
References

Programming with Big Data in R Website: http://r-pbd.org/

See Also

set.global, em.onestep.m.step.

Examples

## Not run:
# This is a core function for em.onestep()
# see the source code for details.

## End(Not run)

---

One M-Step

Compute One M-Step Based on Current Posterior Probabilities

Description

This function will perform one M-step based on current posterior probabilities. This is a core function of em.onestep.

m.step.dmat is a ddmatrix version of m.step.

Usage

m.step(PARAM)

m.step.dmat(PARAM)

Arguments

PARAM a set of parameters.

Details

This function will base on the current posterior probabilities \( z_{spmd} \) to estimate the parameters PARAM mainly including mixing proportions ETA, centers of clusters MU, and dispersions of clusters SIGMA.

Value

Returning a new PARAM which maximizes the complete data log likelihood for the current iteration.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.
One Step of EM algorithm

References

Programming with Big Data in R Website: http://r-pbd.org/

See Also

set.global, em.onestep.e.step.

Examples

```r
## Not run:
# This is a core function for em.onestep()
# see the source code for details.

## End(Not run)
```

---

One Step of EM algorithm

*One EM Step for GBD*

---

Description

One EM step only for model-based clustering of finite mixture Gaussian models with unstructured dispersions. This is a core function of `em.step`.

`em.onestep.dmat` is a `ddmatrix` version of `em.onestep`.

Usage

```r
em.onestep(PARAM)
em.onestep.dmat(PARAM)
```

Arguments

- `PARAM` an original set of parameters generated by `set.global`.

Details

A global variable called `X.spmd` should exist in the `.pmclustEnv` environment, usually the working environment. The `X.spmd` is the data matrix to be clustered, and this matrix has a dimension `N.spmd` by `p`.

The `PARAM` will be a local variable for the current iteration inside `em.onestep`, and this variable is a list containing all parameters related to models. This function also updates in the parameters by the EM algorithm, and return a new `PARAM` for the next iteration. The details of list elements are initially generated by `set.global`. 
pmclust and pkmeans

Value

This function is one EM step. The global variables will be updated and a new PARAM will be returned. See the help page of PARAM or PARAM.org for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References

Programming with Big Data in R Website: http://r-pbd.org/

See Also

set.global, e.step, m.step.

Examples

```r
## Not run:
# This is a core function for em.step()
# see the source code for details.

## End(Not run)
```

pmclust and pkmeans  Parallel Model-Based Clustering and Parallel K-means Algorithm

Description

Parallel Model-Based Clustering and Parallel K-means Algorithm

Usage

```r
pmclust(X = NULL, K = 2, MU = NULL,
  algorithm = .PMC.CT$algorithm, RndEM.iter = .PMC.CT$RndEM.iter,
  CONTROL = .PMC.CT$CONTROL, method.own.X = .PMC.CT$method.own.X,
  rank.own.X = .pdb_env$SPMD.CT$rank.source, comm = .pdb_env$SPMD.CT$comm)

pkmeans(X = NULL, K = 2, MU = NULL,
  algorithm = c("kmeans", "kmeans.dmat"),
  CONTROL = .PMC.CT$CONTROL, method.own.X = .PMC.CT$method.own.X,
  rank.own.X = .pdb_env$SPMD.CT$rank.source, comm = .pdb_env$SPMD.CT$comm)
```
Arguments

- **X**: a GBD row-major matrix or a ddmatrix.
- **k**: number of clusters.
- **mu**: pre-specified centers.
- **algorithm**: types of EM algorithms.
- **RndEM.iter**: number of Rand-EM iterations.
- **CONTROL**: a control for algorithms, see `CONTROL` for details.
- **method.own.X**: how X is distributed.
- **rank.own.X**: who own X if method.own.X = "single".
- **comm**: MPI communicator.

Details

These are high-level functions for several functions in `pmclust` including: data distribution, setting global environment `pmclustEnv`, initializations, algorithm selection, etc.

The input X is either in ddmatrix or gbd. It will be converted in gbd row-major format and copied into `pmclustEnv` for computation. By default, `pmclust` uses a GBD row-major format (gbdr). While common means that X is identical on all processors, and single means that X only exist on one processor `rank.own.X`.

Value

These functions return a list with class `pmclust` or `pkmeans`.

See the help page of `PARAM` or `PARAM.org` for details.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

See Also

`set.global.e.step.m.step`
`set.global.dmat.e.step.dmat.m.step.dmat`

Examples

```r
## Not run:
# Save code in a file "demo.r" and run in 4 processors by
# > mpiexec -np 4 Rscript demo.r

### Setup environment.
library(pmclust, quiet = TRUE)
```
### Load data
X <- as.matrix(iris[, -5])

### Distribute data
jid <- get.jid(nrow(X))
X.gbd <- X[jid,]

### Standardized
N <- allreduce(nrow(X.gbd))
p <- ncol(X.gbd)
mu <- allreduce(colSums(X.gbd / N))
X.std <- sweep(X.gbd, 2, mu, FUN = "-")
std <- sqrt(allreduce(colSums(X.std^2 / (N - 1))))
X.std <- sweep(X.std, 2, std, FUN = "/")

### Clustering
library(pmclust, quiet = TRUE)
comm.set.seed(123, diff = TRUE)

ret.mb1 <- pmclust(X.std, K = 3)
comm.print(ret.mb1)

ret.kms <- pkmeans(X.std, K = 3)
comm.print(ret.kms)

### Finish
finalize()

## End(Not run)

---

### print.object

*Functions for Printing or Summarizing Objects According to Classes*

#### Description

Several classes are declared in `pmclust`, and these are functions to print and summary objects.

#### Usage

```r
## S3 method for class 'pmclust'
print(x, ...)
```

```r
## S3 method for class 'pkmeans'
print(x, ...)
```

#### Arguments

- `x`: an object with the class attributes.
- `...`: other possible options.
Description

This function print the annotations of all variables used in this package.
*.dmat's are ddmatrix versions.

Usage

readme()

readme.dmat()

Details

This package is optimized in the way by pre-specifying several global variables in .pmclustEnv. These variables will be overwrote by EM algorithms. Users should use these names to access the results and utilize them with cautions.

readme.dmat is a ddmatrix version of readme.
Value

A readme message will print on screen by default and explain the global variables used in this package, including:

- **CHECK**
- **CLASS.spmd**
- **COMM.RANK**
- **COMM.SIZE**
- **CONTROL**
- **PARAM**
- **SAVE.param**
- **SAVE.iter**
- **U.spmd**
- **W.spmd**
- **W.spmd.rowSums**
- **X.spmd**
- **Z.colSums**
- **Z.spmd**
- **p.times.logtwopi**

Each variable may contain several elements if it is a list, some variables are used for temporary storages in order to optimize computing, and some variables are used for constant variables. These variables may be restricted, and only generated by the function `set.global`. One can access these variables via the global environment `.pmclustEnv` such as `.pmclustEnv$CONTROL`.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

References

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

See Also

- `set.global`
- `set.global.dmat`

Examples

```r
## Not run:
readme()
readme.dmat()

## End(Not run)
```
Set Global Variables

Set Global Variables According to the global matrix X.gbd (X.spmd) or X.dmat

Description

This function will set several sets of variables globally in the environment .pmclustEnv according to the global matrix X.gbd/X.spmd.

set.global.dmat is a ddmatrix version of set.global.gbd.

Usage

```
set.global.gbd(K = 2, X.gbd = NULL, PARAM = NULL,
algorithm = c("em", "aecm", "apecm", "apecm", "kmeans"),
RndEM.iter = 10)
set.global.dmat(K = 2, X.dmat = NULL, PARAM = NULL,
algorithm = c("em.dmat", "kmeans.dmat"),
RndEM.iter = 10)
set.global(K = 2, X.spmd = NULL, PARAM = NULL,
algorithm = c("em", "aecm", "apecm", "apecm", "kmeans"),
RndEM.iter = 10)
```

Arguments

- **K**: an original set of parameters generated by set.global.
- **X.gbd**: an input GBD matrix.
- **X.dmat**: an input ddmatrix.
- **X.spmd**: an input SPMD matrix.
- **PARAM**: an original set of parameters generated by set.global.
- **algorithm**: an original set of parameters generated by set.global.
- **RndEM.iter**: number of RndEM iterations.

Details

**WARNING**: A global variable named X.gbd/X.spmd should be set before calling set.global where X.gbd/X.spmd is a matrix containing data with dimension N.spmd * p. i.e. N.spmd observations and p variables.

**WARNING**: A global variable named X.dmat should be set before calling set.global.dmat where X.dmat is a ddmatrix (in block-cyclic format) containing data with dimension N * p.

X.gbd/X.spmd and X.dmat are supposed to exist in .GlobalEnv. If not, they should be as an input object and will be copied into .pmclustEnv which is less efficient.
Value
A new set of \texttt{PARAM} will be returned and several global variables will be set according to the data \texttt{X}.

Sets of global variables are store in the default environment \texttt{.pmclustEnv}.

Use \texttt{readme} to see all global variables set by this function.

Use \texttt{readme.dmat} to see all global variables of \texttt{ddmatrix} version set by this function.

Author(s)
Wei-Chen Chen \texttt{<wccsnow@gmail.com>} and George Ostrouchov.

References
Programming with Big Data in R Website: \texttt{http://r-pbd.org/}

See Also
\texttt{em.step}, \texttt{aecm.step}, \texttt{apecm.step},
\texttt{apecma.step}.

Examples
```r
## Not run:
# Examples can be found in the help pages of \texttt{em.step()},
# \texttt{aecm.step()}, \texttt{apecm.step()}, \texttt{apecma.step()}, and \texttt{kmeans.step()}.

# Examples for \texttt{ddmatrix} version can be found in the help pages of
# \texttt{kmeans.step.dmat()}.  
## End(Not run)
```

Description
This set of controls are used to guide all algorithms implemented in this package.

Format
A list variable contains several parameters for computing.
Set of CONTROL

Details

.NPM.C.T stores all default controls for pmclust and pkmeans including

- **algorithm** algorithms implemented
- **algorithm.gbd** algorithms implemented for gbd/spmd
- **algorithm.dmat** algorithms implemented for ddmatrix
- **method.own.X** how X is distributed
- **CONTROL** a CONTROL list as in next

The elements of CONTROL or .pmclustEnv$CONTROL are

- **max.iter** maximum number of iterations (1000)
- **abs.err** absolute error for convergence (1e-4)
- **rel.err** relative error for convergence (1e-6)
- **debug** debugging flag (0)
- **RndEM.iter** number of RndEM iterations (10)
- **exp.min** minimum exponent (log(.Machine$double.xmin))
- **exp.max** maximum exponent (log(.Machine$double.xmax))
- **U.min** minimum of diagonal of chol
- **U.max** maximum of diagonal of chol

These elements govern the computing including number of iterations, convergent criteria, ill conditions, and numerical issues. Some of them are machine dependent.

Currently, the algorithm could be em, aecm, apecm, apecma, and kmeans for GBD and ddmatrix (with *.dmat). The method.own.X could be gbdr, common, single, and ddmatrix.

Numerical Issues

For example, exp.min and exp.max will control the range of densities function before taking logarithm. If the density values were not in the range, they would be rescaled. The scaling factor will be also recorded for post adjustment for observed data log likelihood. This will provide more accurate posterior probabilities and observed data log likelihood.

Also, U.min and U.max will control the output of chol when decomposing SIGMA in every E-steps. If the diagonal terms were out of the range, a PARAM$U.check would be set to FALSE. Only the components with TRUE U.check will estimate and update the dispersions in M-steps for the rest of iterations.

These problems may cause wrong posteriors and log likelihood due to the degenerate and inflated components. Usually, this is a sign of overestimate the number of components K, or the initialization do not provide good estimations for parameters. See e.step for more information about computing.

Author(s)

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.
Set of Parameters in Model-Based Clustering.

Description

This set of parameters are used in initialization, EM iterations, and final convergent results. All share the same structure in a list variable.

Format

A list variable contains several parameters for computing.

Details

The elements of PARAM or PARAM.org are:

- **N**: number of observations
- **p**: dimension of each observation, total number of variables
- **K**: number of clusters
- **ETA**: mixing proportion
- **log.ETA**: log of mixing proportion
- **MU**: centers, dim = $p \times K$
- **SIGMA**: dispersions, a list containing $K$ elements, each element is a matrix, dim = $p \times p$
- **U**: Choleski of SIGMA, the same size of SIGMA
- **U.check**: checks of each elements of U, length $K$
- **logL**: log likelihood
- **min.N.CLASS**: minimum number of elements in a cluster (restrictions)

The model parameters are ETA, MU, and SIGMA, while log.ETA, U, U.check, and min.N.CLASS are only used in computing.
**Update Class of EM or Kmenas Results**

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.

**References**

Programming with Big Data in R Website: [http://r-pbd.org/](http://r-pbd.org/)

**See Also**

`set.global`.

**Examples**

```r
## Not run:
# Use set.global() to generate one of this.
# X.spmd should be pre-specified before calling set.global().

## End(Not run)
```

---

**Update Class of EM or Kmenas Results**

*Update CLASS.spmd Based on the Final Iteration*

**Description**

Update `CLASS.spmd` based on the final iteration of EM-like algorithms. 
*.dmat's are ddmatrix versions.

**Usage**

```r
dem.update.class()
kmeans.update.class()

dem.update.class.dmat()
kmeans.update.class.dmat()
```

**Details**

This function takes `Z.spmd` from the global environment `.pmclustEnv` and update `CLASS.spmd`, and provides the identification of groups for all data.

**Value**

`CLASS.spmd` will be updated.

**Author(s)**

Wei-Chen Chen <wccsnow@gmail.com> and George Ostrouchov.
References

Programming with Big Data in R Website: http://r-pbd.org/

See Also

em.step, aecm.step, apecm.step,
apecma.step, kmeans.step.
kmeans.step.dmat.

Examples

## Not run:
# Examples can be found in the help pages of em.step(),
# aecm.step(), apecm.step(), apecma.step(), and kmeans.step().

# Examples for ddmatrix version can be found in the help pages of
# kmeans.step.dmat().

## End(Not run)
Index

*Topic **algorithm**
EM-like algorithms, 6

*Topic **core function**
One E-Step, 17
One M-Step, 18
One Step of EM algorithm, 19

*Topic **debugging function**
Independent logl, 13

*Topic **global variables**
Read Me First, 23
Set Global Variables, 25
Set of CONTROL, 26
Set of PARAM, 28

*Topic **high-level function**
pmlust and pkmeans, 20
printNobject, 22

*Topic **initialization**
Initialization, 14

*Topic **package**
pmlust-package, 2

*Topic **programming**
as functions, 4
assignN.sample, 5
generate.basic, 8
generate.NixSim, 10
get.N.CLASS, 12
mb.print, 16
Update Class of EM or Kmenas Results, 29
.pmclust (Set of CONTROL), 26
.pmclustEnv, 3, 7, 19, 29
.pmclustEnv (Set Global Variables), 25

aem.c.step, 3, 7, 13, 15, 16, 26, 30
aem.c.step (EM-like algorithms), 6
aem.c.step, 3, 7, 13, 15, 16, 26, 30
aemcma.step, 3, 7, 13, 15, 16, 26, 30
aemcma.step (EM-like algorithms), 6
as.functions, 4
as.dmat (as functions), 4
as.gbd (as functions), 4
as.spmd (as functions), 4
as.g.Tsample, 5

CHECK (Read Me First), 23
chol, 27
CLASS.dmat (Read Me First), 23
CLASS.spmd, 9, 10, 12, 29
CLASS.spmd (Read Me First), 23
COMM.RANK, 12, 16
COMM.RANK (Read Me First), 23
COMM.SIZE (Read Me First), 23
CONTROL, 15, 17, 21, 24
CONTROL (Set of CONTROL), 26
e.step, 19–21, 27
e.step (One E-Step), 17
e.step.dmat, 21
EM-like algorithms, 6
em.onestep, 13, 14, 17–19
em.onestep (One Step of EM algorithm), 19
em.step, 3, 7, 13, 15, 16, 19, 26, 30
em.step (EM-like algorithms), 6
em.step.dmat, 3
em.update.class (Update Class of EM or Kmenas Results), 29
ETA, 18
ETA (Set of PARAM), 28
generate.basic, 8, 11
generate.MixSim, 9, 10
get.CLASS (get.N.CLASS), 12
get.N.CLASS, 12

indep.logL (Independent logL), 13
Independent logL, 13
initial.center (Initialization), 14
initial.em, 15
index

initial.em (Initialization), 14
initial.RndEM, 15
initial.RndEM (Initialization), 14
Initialization, 14

kmeans.step, 3, 7, 13, 15, 30
kmeans.step (EM-like algorithms), 6
kmeans.step.dmat, 3, 5, 13, 15, 30
kmeans.update.class (Update Class of EM or Kmenas Results), 29

m.step, 18, 20, 21
m.step (One M-Step), 18
m.step.dmat, 21
mb.print, 8, 16
MixSim, 10
MU, 15, 18
MU (Set of PARAM), 28

One E-Step, 17
One M-Step, 18
One Step of EM algorithm, 19

p.times.logtwopi (Read Me First), 23
PARAM, 7, 13, 15, 18–21, 24, 26
PARAM (Set of PARAM), 28
PARAM.org, 7, 20, 21
pkmeans, 23
pkmeans (pmclust and pkmeans), 20
pmclust, 3, 23
pmclust (pmclust and pkmeans), 20
pmclust and pkmeans, 20
pmclust-package, 2
print.object, 22
print.pkmeans (print.object), 22
print.pmclust (print.object), 22

Read Me First, 23
readme, 26
readme (Read Me First), 23
readme.dmat, 26

SAVE.iter (Read Me First), 23
SAVE.param (Read Me First), 23
Set Global Variables, 25
Set of CONTROL, 26
Set of PARAM, 28
set.global, 6–8, 14, 15, 18–21, 24, 28, 29
set.global (Set Global Variables), 25
set.global.dmat, 8, 15, 21, 24, 28

set.global.gbd, 28
SIGMA, 18, 27
SIGMA (Set of PARAM), 28

U.dmat (Read Me First), 23
U.spmd, 17
U.spmd (Read Me First), 23
Update Class of EM or Kmenas Results, 29

W.dmat (Read Me First), 23
W.spmd, 17
W.spmd (Read Me First), 23
W.spmd.rowSums, 17

X.dmat (Set Global Variables), 25
X.gbd, 3
X.gbd (Set Global Variables), 25
X.spmd, 3, 5, 7, 9, 10, 14, 15, 19, 24
X.spmd (Set Global Variables), 25

Z.colSums, 17
Z.colSums (Read Me First), 23
Z.dmat (Read Me First), 23
Z.spmd, 17, 18, 29
Z.spmd (Read Me First), 23