# MixAll: Clustering Heterogenous data with Missing Values

**Serge Iovleff**
University Lille 1

**Abstract**

The Clustering project is a part of the **STK++** library (Iovleff 2012) that can be accessed from R (R Development Core Team 2013) using the **MixAll** package. It is possible to cluster Gaussian, gamma, categorical, Poisson, kernel mixture models or a combination of these models in case of heterogeneous data. Moreover, if there is missing values in the original data set, these missing values will be imputed during the estimation process. These imputations can be biased estimators or Monte-Carlo estimators of the Maximum A Posteriori (MAP) values depending of the algorithm used.

*Keywords*: R, C++, STK++, Clustering, missing values.

## 1. Introduction

The Clustering project in STK++ implements a set of mixture model allowing to perform clustering on various data set using generative models. There is five kinds of generative models implemented:

1. the diagonal Gaussian mixture models (8 models), see sections 3.1 and 4.1,
2. the diagonal gamma mixture models (24 models), see sections 3.4 and 4.3,
3. the diagonal categorical mixture models (4 models), see sections 3.2 and 4.2,
4. the diagonal Poisson mixture models (6 models), see sections 3.3 and 4.4,
5. the kernel mixture models (4 models), see sections 3.5 and 4.5.

and a special model called ”heterogeneous” mixture model allowing to cluster heterogeneous data set using conditional independance between the different kind of data, see sections 3.6 and 4.6.

These models and the estimation algorithms can take into account missing values. It is thus possible to use these models in order to cluster, but also to complete data set with missing values.

The **MixAll** package provide an access in (R Development Core Team 2013) to the **STK++** (Iovleff 2012) C++ part of the library dedicated to clustering.

In this paper we will first give a general introduction about mixture models and the different algorithms, initialization methods and strategies that can be used in order to estimate parameters of mixture models (Section 2). In Section 3 we present the different mixture models implemented in STK++ that can be estimated using MixAll. Finally we give examples of clustering on real data set in Section 4.

## 2. MixAll Modeling and Estimation Tools

### 2.1. Short Introduction to Mixture Models

Let $\mathcal{X}$ be an arbitrary measurable space and let $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ be $n$ independent vectors in $\mathcal{X}$ such that each $\mathbf{x}_i$ arises from a probability distribution with density (a mixture model)

$$f(\mathbf{x}_i|\theta) = \sum_{k=1}^{K} p_k h(\mathbf{x}_i|\boldsymbol{\lambda}_k, \boldsymbol{\alpha}) \tag{1}$$

where the $p_k$'s are the mixing proportions ($0 < p_k < 1$ for all $k = 1, ..., K$ and $p_1 + ... + p_K = 1$), $h(\cdot|\boldsymbol{\lambda}_k, \boldsymbol{\alpha})$ denotes a $d$-dimensional distribution parameterized by $\boldsymbol{\lambda}_k$ and $\boldsymbol{\alpha}$. The parameters $\boldsymbol{\alpha}$ do not depend from $k$ and are common to all the components of the mixture. The vector parameter to be estimated is $\theta = (p_1, \dots, p_K, \boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_K, \boldsymbol{\alpha})$ and is chosen to maximize the observed log-likelihood

$$L(\theta|\mathbf{x}_1, \dots, \mathbf{x}_n) = \sum_{i=1}^{n} \ln \left( \sum_{k=1}^{K} p_k h(\mathbf{x}_i|\boldsymbol{\lambda}_k, \boldsymbol{\alpha}) \right). \tag{2}$$

In case there is missing data, that is some $\mathbf{x}_i$ are splited in observed values $\mathbf{x}_i^o$ and missing values $\mathbf{x}_i^m$, the log-likelihood to maximize should be the integrated log-likelihood

$$L(\theta|\mathbf{x}_1^o, \dots, \mathbf{x}_n^o) = \sum_{i=1}^{n} \int \ln \left( \sum_{k=1}^{K} p_k h(\mathbf{x}_i^o, \mathbf{x}_i^m|\boldsymbol{\lambda}_k, \boldsymbol{\alpha}) \right) d\mathbf{x}_i^m. \tag{3}$$

In the package **MixAll**, this quantity is approximated using a Monte-Carlo estimator by the `SEM` or the `SemiSEM` algorithms and by a biased estimator by the `EM` or the `CEM` algorithms.

It is well known that for a mixture distribution, a sample of indicator vectors or *labels* $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_n\}$, with $\mathbf{z}_i = (z_{i1}, \dots, z_{iK})$, $z_{ik} = 1$ or 0, according to the fact that $\mathbf{x}_i$ is arising from the $k$th mixture component or not, is associated to the observed data $\mathbf{x}$. The sample $\mathbf{z}$ is *unknown* so that the maximum likelihood estimation of mixture models is traditionally performed via the `EM` algorithm Dempster *et al.* (1997) or by a stochastic version of `EM` called SEM (see Mclachlan and Peel (2000)), or by a k-means like algorithm called `CEM`. In the **MixAll** package it is also possible to use an algorithm called `SemiSEM` which is an intermediate between the `EM` and `SEM` algorithm. In case there is no missing values, `SemiSEM` and `EM` are equivalents (except that the `SemiSEM` algorithm will run all the iterations as it does not stop using a tolerance).

### 2.2. Estimation Algorithms

#### *EM algorithm*

Starting from an initial arbitrary parameter $\theta^0$, the $m$th iteration of the `EM` algorithm consists of repeating the following I (if there exists missing values), E and M steps.

- **I step:** The missing values $\mathbf{x}_i^m$ are imputed using the current MAP value given by the current value $\theta^{m-1}$ of the parameter.
- **E step:** The current conditional probabilities that $z_{ik} = 1$ for $i = 1, \dots, n$ and $k = 1, \dots, K$ are computed using the current value $\theta^{m-1}$ of the parameter:

$$t_{ik}^m = t_k^m(\mathbf{x}_i|\theta^{m-1}) = \frac{p_k^{m-1} h(\mathbf{x}_i|\boldsymbol{\lambda}_k^{m-1}, \boldsymbol{\alpha}^{m-1})}{\sum_{l=1}^K p_l^{m-1} h(\mathbf{x}_i|\boldsymbol{\lambda}_l^{m-1}, \boldsymbol{\alpha}^{m-1})}. \tag{4}$$

- **M step:** The maximum likelihood estimate $\theta^m$ of $\theta$ is updated using the conditional probabilities $t_{ik}^m$ as conditional mixing weights. It leads to maximize

$$L(\theta|\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{t}^m) = \sum_{i=1}^n \sum_{k=1}^K t_{ik}^m \ln\left[p_k h(\mathbf{x}_i|\boldsymbol{\lambda}_k, \boldsymbol{\alpha})\right], \tag{5}$$

where $\mathbf{t}^m = (t_{ik}^m, i = 1, \dots, n, k = 1, \dots, K)$. Updated expression of mixture proportions are, for $k = 1, \dots, K$,

$$p_k^m = \frac{\sum_{i=1}^n t_{ik}^m}{n}. \tag{6}$$

Detailed formula for the updating of the $\boldsymbol{\lambda}_k$'s and $\boldsymbol{\alpha}$ are depending of the component parameterization and are detailed in section 3.

The `EM` algorithm may converge to a local maximum of the observed data likelihood function, depending on starting values.

*SEM algorithm*

The `SEM` algorithm is a stochastic version of `EM` incorporating between the E and M steps a restoration of the unknown component labels $\mathbf{z}_i$, $i = 1, \dots, n$, by drawing them at random from their current conditional distribution. Starting from an initial parameter $\theta^0$, an iteration of `SEM` consists of three steps.

- **I step:** The missing values are simulated using the current value $\theta^{m-1}$ of the parameter and current conditional probabilities $t_{ik}^{m-1}$.
- **E step:** The conditional probabilities $t_{ik}^m$ $(1 \leq i \leq n, 1 \leq k \leq K)$ are computed for the current value of $\theta^{m-1}$ as in the E step of `EM` algorithm (equation 4).
- **S step:** Generate labels $\mathbf{z}^m = \{\mathbf{z}_1^m, ..., \mathbf{z}_n^m\}$ by assigning each point $\mathbf{x}_i$ at random to one of the mixture components according to the categorical distribution with parameter $(t_{ik}^m, 1 \leq k \leq K)$.
- **M step:** The maximum likelihood estimate of $\theta$ is updated using the generated labels by maximizing

$$L(\theta|\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{t}^m) = \sum_{i=1}^n \sum_{k=1}^K z_{ik}^m \ln\left[p_k h(\mathbf{x}_i|\boldsymbol{\lambda}_k, \boldsymbol{\alpha})\right], \tag{7}$$

SEM does not converge point wise. It generates a Markov chain whose stationary distribution is more or less concentrated around the m.l. parameter estimator. A natural parameter estimate from a `SEM` sequence $\bar{\theta} = (\theta^r)_{r=1,\dots,R}$ is the mean $\sum_{r=1}^R \theta^r/R$ of the iterates values.

At the end of the algorithm, the missing values will be imputed using the MAP value given by the averaged estimator $\bar{\theta}$.

### *SemiSEM algorithm*

The `SemiSEM` algorithm is a stochastic version of `EM` incorporating a restoration of the missing values $\mathbf{x}_i^m$, $i = 1, \dots, n$ by drawing them at random from their current conditional distribution. Starting from an initial parameter $\theta^0$, an iteration of `SemiSEM` consists of three steps.

- **I step:** The missing values are simulated using the current value $\theta^{m-1}$ and current conditional probabilities $t_{ik}^{m-1}$ of the parameter as in the `SEM` algorithm.
- **E step:** The conditional probabilities $t_{ik}^m$ $(1 \leq i \leq n, 1 \leq k \leq K)$ are computed for the current value of $\theta^{m-1}$.
- **M step:** The maximum likelihood estimate of $\theta$ is updated by maximizing conditional probabilities $t_{ik}^m$ as conditional mixing weights as in the `EM` algorithm.

If there is no missing values, SemiSEM algorithm is equivalent to the EM algorithm. If there is missing values, SemiSEM does not converge point wise. It generates a Markov chain whose stationary distribution is more or less concentrated around the m.l. parameter estimator. A natural parameter estimate from a `SemiSEM` sequence $(\theta^r)_{r=1,\dots,R}$ is the mean $\bar{\theta} = \sum_{r=1}^R \theta^r/R$ of the iterates values.

At the end of the algorithm, the missing values are imputed using the MAP value given by the averaged estimator $\bar{\theta}$.

### *CEM algorithm*

This algorithm incorporates a classification step between the E and M steps of EM. Starting from an initial parameter $\theta^0$, an iteration of `CEM` consists of three steps.

- **I step:** The missing values are imputed using the current MAP value given by the current value $\theta^{m-1}$and current conditional probabilities $t_{ik}^{m-1}$ of the parameter as in the `EM` algorithm.
- **E step:** The conditional probabilities $t_{ik}^m$ $(1 \leq i \leq n, 1 \leq k \leq K)$ are computed for the current value of $\theta$ as done in the E step of EM.
- **C step:** Generate labels $\mathbf{z}^m = \{\mathbf{z}_1^m, ..., \mathbf{z}_n^m\}$ by assigning each point $\mathbf{x}_i$ to the component maximizing the conditional probability $(t_{ik}^m, 1 \leq k \leq K)$.
- **M step:** The maximum likelihood estimate of $\theta$ are computed as done in the M step of SEM.

CEM is a *K-means*-like algorithm and contrary to `EM`, it converges in a finite number of iterations. `CEM` is not maximizing the observed log-likelihood $L$ (2) but is maximizing in $\theta$

and $\mathbf{z}_1, \ldots, \mathbf{z}_n$ the complete data log-likelihood

$$CL(\theta, \mathbf{z}_1, \ldots, \mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{i=1}^{n} \sum_{k=1}^{K} z_{ik} \ln[p_k h(\mathbf{x}_i | \boldsymbol{\lambda}_k)]. \tag{8}$$

where the missing component indicator vector $\mathbf{z}_i$ of each sample point is included in the data set. As a consequence, CEM is not expected to converge to the maximum likelihood estimate of $\theta$ and yields inconsistent estimates of the parameters especially when the mixture components are overlapping or are in disparate proportions (see Mclachlan and Peel (2000), Section 2.21).

*Creating an Algorithm*

All the algorithms (EM, SEM, CEM and SemiSEM) are encoded in a S4 class and can be created using the utility function `clusterAlgo`. This function take as input three parameters:

- `algo`: name of the algorithm to define ("EM", "SEM", "CEM" or "SemiSEM"). Default value is "EM".
- `nbIteration`: maximal number of iteration to perform. Default value is 200.
- `epsilon`: threshold to use in order to stop the iterations (not used by the SEM and SemiSEM algorithms).

```
> clusterAlgo()

****************************************
*** MixAll ClusterAlgo:
* algorithm            =  EM
* number of iterations =  200
* epsilon              =  1e-07
****************************************

> clusterAlgo(algo="SemiSEM",nbIteration=100,epsilon=1e-08)

****************************************
*** MixAll ClusterAlgo:
* algorithm            =  SemiSEM
* number of iterations =  100
* epsilon              =  1e-08
****************************************
```

## 2.3. Initialization Methods

All the estimation algorithms need a first value of the parameter $\theta$. There is three kinds of initialization that can be performed: either by generating directly random parameters, or by using random classes labels/random fuzzy classes and estimating $\theta^0$. In order to prevent unlucky initialization, multiple initialization with a limited number of an algorithm are performed and the best initialization (in the likelihood sense) is conserved.

The initialization step is encoded in a S4 class and can be created using the utility function `clusterInit`. This function take as input four parameters:

- `method`: name of the initialization to perform (`"random"`, `"class"` or `"fuzzy"`). Default value is `"class"`
- `nbInit` number of initialization to do. Default value is 5.
- `algo` name of the algorithm to use during the limited estimation steps (see also 2.2). Default value is "EM".
- `nbIteration` maximal number of iteration to perform during the initialization algorithm. Default values is 20.
- `epsilon` threshold to use in order to stop the iterations. Default value: 0.01.

```
> clusterInit()

****************************************
*** MixAll ClusterInit:
* method               =  class
* number of init       =  5
* algorithm            =  EM
* number of iterations =  20
* epsilon              =  0.01
****************************************

> clusterInit(method="random", nbInit= 2, algo="CEM", nbIteration=10,epsilon=1e-04)

****************************************
*** MixAll ClusterInit:
* method               =  class
* number of init       =  2
* algorithm            =  CEM
* number of iterations =  10
* epsilon              =  1e-04
****************************************
```

## 2.4. Estimation Strategy

A strategy is a way to find a good estimate of the parameters of a mixture model and to avoid local maxima of the likelihood function. A strategy is an efficient three steps Search/Run/Select way for maximizing the likelihood:

1. Build a search method for generating `nbShortRun` initial positions. This is based on the initialization method we describe previously.
2. Run a short algorithm for each initial position.
3. Select the solution providing the best likelihood and launch a long run algorithm from this solution.

A strategy is encoded in a S4 class and can be created using the utility function `clusterStrategy()`. This function have no mandatory argument but the default strategy can be tuned. In table 1 the reader will find a summary of all the input parameters of the `clusterStrategy()` function.

| Input Parameter | Description |
|---|---|
| `nbTry` | Integer defining the number of tries. `nbTry` must be a positive integer. Default value is 1. |
| `nbInit` | Integer defining the number of initialization to do during the initialization step. Default is 5. |
| `initAlgo` | String with the estimation algorithm to use in the initialization step. Possible values are `"EM"`, `"SEM"`, `"CEM"`, `"SemiSEM"`. Default value is `"EM"`. |
| `nbInitIteration` | Integer defining the maximal number of iteration in `initAlgo` algorithm. `nbInitIteration` can be 0. Default value is 20. |
| `initEpsilon` | Real defining the epsilon value for the initial algorithm. `initEpsilon` is not used by the `"SEM"` and `"SemiSEM"` algorithms. Default value is 0.01. |
| `nbShortRun` | Integer defining the number of short run to perform (remember the strategy launch an initialization step before each short run, so you get nbShortRun*nbInit initialization). Default value is 5. |
| `shortRunAlgo` | String with the estimation algorithm to use in short run(s). Possible values are `"EM"`, `"SEM"`, `"CEM"`, `"SemiSEM"`. Default value is `"EM"`. |
| `nbShortIteration` | Integers defining the maximal number of iterations in a short run. Default value is 100. |
| `shortEpsilon` | Real defining the epsilon value in a short run. It is not used if `shortRunAlgo` is `"SEM"` or `"SemiSEM"`. Default value is 1e-04. |
| `longRunAlgo` | String with the estimation algorithm to use for the long run. Possible values are `"EM"`, `"SEM"`, `"CEM"` or `"SemiSEM"`. Default value is `"EM"`. |
| `nbLongIteration` | Integers defining the maximal number of iterations in the the long run. Default value is 1000. |
| `longEpsilon` | Real defining the epsilon value in the long run. It is not used if `shortRunAlgo` is `"SEM"` or `"SemiSEM"`. Default value is 1e-07. |

Table 1: List of all the input parameters of the `clusterStrategy()` function.

```
> clusterStrategy()

****************************************
*** Cluster Strategy:
* number of try         =  1
* number of short run   =  5
****************************************
*** Initialization :
* method =  class
* number of init       =  5
* algorithm            =  EM
* number of iterations =  20
* epsilon              =  0.01
****************************************
*** short Algorithm :
* algorithm            =  EM
* number of iterations =  100
* epsilon              =  1e-04
****************************************
*** long algorithm :
* algorithm            =  EM
* number of iterations =  1000
* epsilon              =  1e-07
****************************************
```

Users have to take care that there will be `nbInit` $\times$ `nbShortRun` starting points $\theta^0$ during the estimation process. The default generate randomly fifty times $\theta^0$.

The strategy class is very flexible and allow to tune the estimation process. There is two defined utility functions for the end-user:

- the `clusterFastStrategy` for impatient users,
- the `clusterSemiSEMStrategy` for user with missing values.

For impatient user, the `clusterFastStrategy` furnish results very quickly. The accuracy of the result is not guaranteed if the model is a bit difficult to estimate.

```
> clusterFastStrategy()

****************************************
*** Cluster Strategy:
* number of try         =  1
* number of short run   =  2
****************************************
*** Initialization :
* method =  class
* number of init       =  3
* algorithm            =  EM
* number of iterations =  5
* epsilon              =  0.01
****************************************
*** short Algorithm :
```

```
* algorithm           =  CEM
* number of iterations =  10
* epsilon             =  0.001
****************************************
*** long algorithm :
* algorithm           =  EM
* number of iterations =  100
* epsilon             =  1e-07
****************************************
```

The function `clusterSemiSEMStrategy` is highly recommended if there is missing values int the data set. The "SemiSEM" algorithm simulate the missing values and computes a Monte-Carlo estimator of the $\theta$ parameter during the iterations allowing to get unbiased estimators.

```
> clusterSemiSEMStrategy()

****************************************
*** Cluster Strategy:
* number of try         =  2
* number of short run   =  5
****************************************
*** Initialization :
* method =  class
* number of init      =  5
* algorithm           =  SemiSEM
* number of iterations =  20
* epsilon             =  0
****************************************
*** short Algorithm :
* algorithm           =  SemiSEM
* number of iterations =  50
* epsilon             =  0
****************************************
*** long algorithm :
* algorithm           =  SemiSEM
* number of iterations =  400
* epsilon             =  0
****************************************
```

## 3. Implemented Mixture Models

### 3.1. Multivariate (diagonal) Gaussian Mixture Models

A Gaussian density on $\mathbb{R}$ is a density of the form:

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\} \quad \sigma > 0. \tag{9}$$

A joint diagonal Gaussian density on $\mathbb{R}^d$ is a density of the form:

$$h(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{j=1}^{d} f(x^j; \mu^j, \sigma^j) \quad \sigma^j > 0. \tag{10}$$

The parameters $\boldsymbol{\mu} = (\mu^1, \ldots, \mu^d)$ are the position parameters and the parameters $\boldsymbol{\sigma} = (\sigma^1, \ldots, \sigma^d)$ are the standard-deviation parameters. Assumptions on the standard-deviation parameters among the variables and the components lead to define four families of mixture model.

Let us write a multidimensional Gaussian mixture model in the from `Gaussian_s*` with `s*`, the different ways to parameterize the standard-deviation parameters of a Gaussian mixture:

- `sjk` means that we have one standard-deviation parameter for each variable in each component,
- `sk` means that the standard-deviation parameters are the same for all the variables inside a component,
- `sj` means that the standard-deviation parameters are different for each variable but are equals between the components,
- and finally `s` means that the standard-deviation parameters are all equals.

The `gaussian_pk_sjk` model is the most general model and has a density function of the form

$$f(\mathbf{x}|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} g(x_i^j | \mu_k^j, \sigma_k^j). \tag{11}$$

On the other side, the `gaussian_p_s` model is the most parsimonious model and has a density function of the form

$$f(\mathbf{x}|\theta) = \sum_{k=1}^{K} \frac{1}{K} \prod_{j=1}^{d} g(x_i^j | \mu_k^j, \sigma). \tag{12}$$

It is possible to get a vector with all Gaussian mixture model names using the `clusterDiagGaussianNames` function.

```
> clusterDiagGaussianNames()

[1] "gaussian_pk_sjk" "gaussian_pk_sj"  "gaussian_pk_sk"  "gaussian_pk_s"
[5] "gaussian_p_sjk"  "gaussian_p_sj"   "gaussian_p_sk"   "gaussian_p_s"

> clusterDiagGaussianNames("all", "equal", "free")

[1] "gaussian_pk_sk" "gaussian_p_sk"

> clusterValidDiagGaussianNames(c("gaussian_pk_sjk","gaussian_p_ljk"))

[1] FALSE
```

## 3.2. Multivariate categorical Mixture Models

A Categorical probability distribution on a finite space $\mathcal{X} = \{1, \dots, L\}$ is a probability distribution of the form:

$$P(x = l) = p_l \quad p_l > 0,\, l \in \mathcal{X}, \tag{13}$$

with the constraint $p_1 + \dots + p_L = 1$.

A joint Categorical probability distribution on $\mathcal{X}^d$ is a probability distribution of the form:

$$P(\mathbf{x} = (x_1, \dots, x_d)) = \prod_{j=1}^{d} p^j_{x_j} \tag{14}$$

The parameters $\mathbf{p} = (p^1, \dots, p^d)$ are the probabilities of the possibles outcomes. Assumptions on the probabilities among the variables and the components lead to define two families of mixture model.

It is possible to get a vector with all Gaussian model names using the `clusterDiagGaussianNames` function.

It is possible to get a vector with all categorical mixture model names using the `clusterCategoricalNames` function.

```
> clusterCategoricalNames()

[1] "categorical_pk_pjk" "categorical_pk_pk"  "categorical_p_pjk"
[4] "categorical_p_pk"

> clusterCategoricalNames("all", "equal")

[1] "categorical_pk_pk" "categorical_p_pk"

> clusterValidCategoricalNames(c("categorical_pk_pjk","categorical_p_pk"))

[1] TRUE
```

## 3.3. Multivariate Poisson Mixture Models

A Poisson probability distribution is a probability over $\mathbb{N}$ of the form

$$p(k; \lambda) = \frac{\lambda^k}{k!} e^{-\lambda} \quad \lambda > 0. \tag{15}$$

A joint Poisson probability on $\mathbb{N}^d$ is a probability distribution of the form

$$h(\mathbf{x}; \boldsymbol{\lambda}) = \prod_{j=1}^{d} p(x^j; \lambda^j) \quad \lambda^j > 0. \tag{16}$$

The parameters $\boldsymbol{\lambda} = (\lambda^1, \dots, \lambda^d)$ are the mean parameters. Assumptions on the mean among the variables and the components lead to define three families of mixture model.

The `poisson_pk_ljk` is the most general Poisson model and has a probability distribution of the form

$$f(\mathbf{x}|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} h(x^j; \lambda_k^j). \tag{17}$$

The `poisson_p_lk` is the most parsimonious Poisson model and has a probability distribution of the form

$$f(\mathbf{x}|\theta) = \sum_{k=1}^{K} \frac{1}{K} \prod_{j=1}^{d} h(x^j; \lambda_k). \tag{18}$$

The `poisson_pk_ljlk` is an intermediary model for the number of parameters and has a density of the form

$$f(\mathbf{x}|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} h(x^j; \lambda_j \lambda_k). \tag{19}$$

It is possible to get a vector with all Poisson mixture model names using the `clusterPoissonNames` function.

```
> clusterPoissonNames()

[1] "poisson_pk_ljk"  "poisson_pk_lk"   "poisson_pk_ljlk" "poisson_p_ljk"
[5] "poisson_p_lk"    "poisson_p_ljlk"

> clusterPoissonNames("all","proportional")

[1] "poisson_pk_ljlk" "poisson_p_ljlk"

> clusterValidPoissonNames(c("poisson_pk_ljk","poisson_p_ljlk"))

[1] TRUE
```

## 3.4. Multivariate Gamma Mixture Models

A gamma density on $\mathbb{R}_+$ is a density of the form:

$$g(x; a, b) = \frac{(x)^{a-1} e^{-x/b}}{\Gamma(a)\,(b)^a} \quad a > 0, \quad b > 0. \tag{20}$$

A joint gamma density on ${\mathbb{R}^d}_+$ is a density of the form:

$$h(\mathbf{x}; \mathbf{a}, \mathbf{b}) = \prod_{j=1}^{d} g(x^j; a^j, b^j) \quad a^j > 0, \quad b^j > 0. \tag{21}$$

The parameters $\mathbf{a} = (a^1, \ldots, a^d)$ are the shape parameters and the parameters $\mathbf{b} = (b^1, \ldots, b^d)$ are the scale parameters. Assumptions on the scale and shape parameters among the variables and the components lead to define twelve families of mixture model. Let us write a multidimensional gamma mixture model in the form `gamma_a*_b*` with `a*` (resp. `b*`), the different ways to parameterize the shape (resp. scale) parameters of a gamma mixture:

| | ajk | ak | aj | a |
|---|---|---|---|---|
| bjk | gamma_ajk_bjk (2dK) | gamma_ak_bjk (dK + K) | gamma_aj_bjk (dK+d) | gamma_a_bjk (dK+1) |
| bk | gamma_ajk_bk (dK+K) | gamma_ak_bk (2K) | gamma_aj_bk (K+d) | gamma_a_bk (K+1) |
| bj | gamma_ajk_bj (dK+d) | gamma_ak_bj (K+d) | NA | NA |
| b | gamma_ajk_b (dK+1) | gamma_ak_b (K+1) | NA | NA |

Table 2: The twelve multidimensional gamma mixture models. In parenthesis the number of parameters of each model.

- `ajk` (resp. `bjk`) means that we have one shape (resp. scale) parameter for each variable and for each component,
- `ak` (resp. `bk`) means that the shape (resp. scale) parameters are the same for all the variables inside a component,
- `aj` (resp. `bj`) means that the shape (resp. scale) parameters are different for each variable but are equals between the components,
- and finally `a` (resp. `b`) means that the shape (resp. scale) parameters are the same for all the variables and all the components.

The models we can build in this way are summarized in the table 2, in parenthesis we give the number of parameters of each models.

The `gamma_ajk_bjk` model is the most general and have a density function of the form

$$f(\mathbf{x}_i|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} g(x_i^j | a_k^j, b_k^j). \tag{22}$$

All the other models can be derived from this model by dropping the indexes in $j$ and/or $k$ from the expression (22). For example the mixture model `gamma_aj_bk` has a density function of the form

$$f(\mathbf{x}_i|\theta) = \sum_{k=1}^{K} p_k \prod_{j=1}^{d} g(x_i^j | a^j, b^k). \tag{23}$$

It is possible to get a vector with all gamma mixture model names using the `clusterGammaNames` function.

```
> clusterGammaNames()

 [1] "gamma_p_ajk_bjk"  "gamma_p_ajk_bk"   "gamma_p_ajk_bj"   "gamma_p_ajk_b"
 [5] "gamma_p_ak_bjk"   "gamma_p_ak_bk"    "gamma_p_ak_bj"    "gamma_p_ak_b"
 [9] "gamma_p_aj_bjk"   "gamma_p_aj_bk"    "gamma_p_a_bjk"    "gamma_p_a_bk"
[13] "gamma_pk_ajk_bjk" "gamma_pk_ajk_bk"  "gamma_pk_ajk_bj"  "gamma_pk_ajk_b"
[17] "gamma_pk_ak_bjk"  "gamma_pk_ak_bk"   "gamma_pk_ak_bj"   "gamma_pk_ak_b"
[21] "gamma_pk_aj_bjk"  "gamma_pk_aj_bk"   "gamma_pk_a_bjk"   "gamma_pk_a_bk"
```

```
> clusterGammaNames("all", "equal","free","free","all")

[1] "gamma_p_ak_bjk"  "gamma_p_ak_bj"   "gamma_pk_ak_bjk" "gamma_pk_ak_bj"

> clusterValidGammaNames(c("gamma_pk_aj_bk","gamma_p_ajk_bjk"))

[1] TRUE
```

## 3.5. Gaussian Kernel Mixture Models

Gaussian Kernel Mixture models are a generalization of the Kernel k-means method Shawe-Taylor and Cristianini (2004).

Let us consider a pair of random variables $(X, Z)$ with values in a measurable space $(\mathcal{X} \times \{1 \ldots K\}, \mathcal{B}_X \otimes \mathcal{P}(\{1...K\}))$. The random variate $Z$ is a categorical random variate and Conditionally to $Z$, the distribution of $X$ is a probability measure $\nu_k$ on $\mathcal{X}$.

$$P(X \in A|Z = k) = \nu_k(A).$$

The marginal distribution of $X$ is a *mixing* distribution such that

$$P(X \in A) = \sum_{k=1}^{K} p_k \nu_k(A),\ A \in \mathcal{B}_X.$$

Let $\mathcal{H}$ denote a reproducing Kernel Hilbert Space (RKHS) and let $\phi$ be a feature map from $\mathcal{X}$ to $\mathcal{H}$. The reproducing property of $\mathcal{H}$ ensure us assure there exists a kernel $k$ positive definite on $\mathcal{X} \times \mathcal{X}$ such that

$$\langle \phi(x), \phi(y) \rangle = k(x, y),\ \forall (x, y) \in \mathcal{X}^2.$$

It is easily verified that the the image by $\phi$ of the probability distribution of $X$ is also a mixing distribution. Let us denote by $\mu$ this distribution and by $\mu_k$ the conditional probability distribution of the random variate $\phi(X)$ conditionally to $Z = k$.

In a kernel mixture model, we assume the following hypothesis about $\mu$:

**G**: *(Gaussianity) $\mu_k$ is well approximated by an isotropic finite Gaussian measure on $\mathcal{H}$ with mean $m_k$ and covariance matrix $\sigma_k I_d$.*

The log-likelihood to maximize in $\theta$ is thus

$$l(x_1, \ldots, x_n; \theta) = \prod_{i=1}^{n} \sum_{k=1}^{K} p_k \left( \frac{1}{\sqrt{2\pi}\sigma_k} \right)^d \exp\left\{ -\frac{\|\phi(x_i) - m_k\|^2}{2\sigma_k^2} \right\}. \tag{24}$$

with $\theta = \left( (p_k)_{k=1}^K, (m_k)_{k=1}^K, (\sigma_k)_{k=1}^K \right)$. This model is called a Kernel Mixture Model (KMM).

The dimension $d$ is an hyper-parameter fixed by the user. Assumptions about the variance lead to four models. The `kernelGaussian_sk` model is the most general and have a density function given in (24), the `kernelGaussian_s` model assume that all variances are equals.

It is possible to get a vector with all (Gaussian) kernel mixture model names using `clusterKernelNames` function.

```
> clusterKernelNames()

[1] "kernelGaussian_pk_sk" "kernelGaussian_pk_s"  "kernelGaussian_p_sk"
[4] "kernelGaussian_p_s"

> clusterValidKernelNames(c("kernelGaussian_pk_sk","kernelGaussian_pk_s"))

[1] TRUE
```

There is threee kernels availables: ”gaussian”, ”exponential” and ”polynomial”

- The Gaussian Kernel is a kernel of the form
$$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{h}\right)$$
where $h$ represents the bandwidth of the kernel (default value 1).
- The Exponential Kernel is a kernel of the form
$$k(x,y) = \exp\left(-\frac{\|x-y\|}{h}\right)$$
where $h$ represents the bandwidth of the kernel (default value 1).
- The Polynomial Kernel is a kernel of the form
$$k(x,y) = (< x - y > +c)^d$$
where $c$ represents the shift of the kernel (default value 0) and $d$ represents the degree (defaut value 1).

## 3.6. Heterogeneous Mixture Models

Heterogeneous mixture models are special models allowing to cluster heterogeneous data sets assuming conditional independency. More precisely, assume that the observation space is of the form $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \dots \mathcal{X}_L$. Then it is assumed that each $\mathbf{x}_i$ arises from a mixture probability distribution with density

$$f(\mathbf{x}_i = (\mathbf{x}_{1i}, \mathbf{x}_{2i}, \dots \mathbf{x}_{Li})|\theta) = \sum_{k=1}^{K} p_k \prod_{l=1}^{L} h^l(\mathbf{x}_{li}|\boldsymbol{\lambda}_{lk}, \boldsymbol{\alpha}_l). \tag{25}$$

The density functions (or probability distribution functions) $h^l(.|\boldsymbol{\lambda}_{lk}, \boldsymbol{\alpha}_l)$ can be any implemented model (Gaussian, Poisson,...).

# 4. Clustering with MixAll

Cluster analysis can be performed with the functions

1. `clusterDiagGaussian` for diagonal Gaussian mixture models,
2. `clusterCategorical` for Categorical mixture models,
3. `clusterPoisson` for Poisson mixture models,
4. `clusterGamma` for gamma mixture models,
5. `clusterKernel` for kernel mixture models,
6. `clusterHeterogeneous` for Heterogeneous mixture models.

These functions have a common set of parameters with default values given in the table 3.

| Input Parameter | Description |
|---|---|
| `nbCluster` | Numeric. A vector with the number of clusters to try. Default is 2. |
| `strategy` | A `Strategy` object containing the strategy to run. Call `clusterStrategy()` (see 2.4) method by default. |
| `criterion` | A string defining the model selection criterion to use. The best model is the one with the lowest criterion value. Possible values: `"AIC"`, `"BIC"`, `"ICL"`. Default is `"ICL"`. |
| `nbCore` | An integer defining the number of processor to use. Default is 1, 0 for all cores. |

Table 3: List of common parameters of the clustering functions.

## 4.1. Clustering with Multivariate (diagonal) Gaussian Mixture Models

Multivariate Gaussian data (without correlations) can be clustered using the `clusterDiagGaussian` function.

This function has one mandatory argument: a `matrix` or `data.frame` `x`. In Table 4 the reader will find a summary of all the specific input parameters of this function with its default value.

| Input Parameter | Description |
|---|---|
| `data` | Matrix or data frame |
| `modelNames` | A `vector` object defining the list of models to estimate. Call `clusterDiagGaussianNames()` by default (see 3.1). |

Table 4: List of all the specific parameters of the `clusterDiagGaussian` function.

We illustrate this function with the well known geyser data set (Azzalini and Bowman (1990),Härdle (1991)).

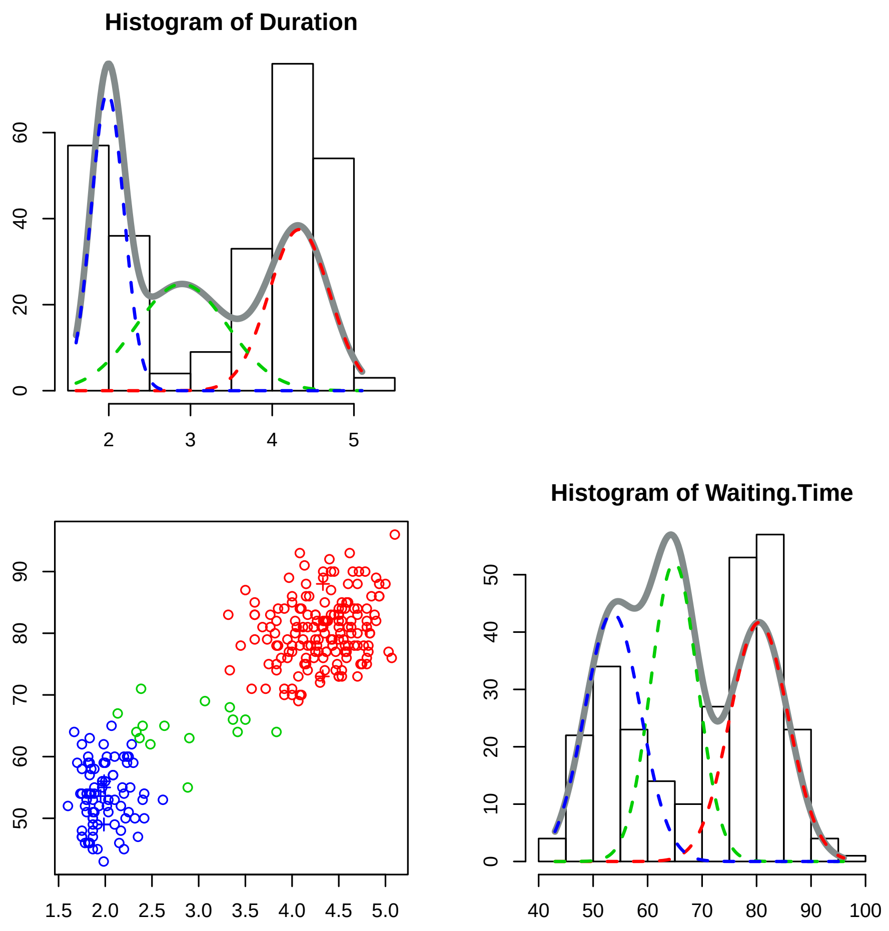
```
> data(geyser);
> x = as.matrix(geyser); n <- nrow(x); p <- ncol(x);
> # add missing values at random
> indexes  <- matrix(c(round(runif(10,1,n)), round(runif(10,1,p))), ncol=2);
> x[indexes] <- NA;
> model <- clusterDiagGaussian(data=x, nbCluster=3, strategy = clusterFastStrategy())
> summary(model)
```

```
**************************************************************
* nbSample       =  272
* nbCluster      =  3
* lnLikelihood   =  -1117.094
* nbFreeParameter=  14
* criterion      =  2339.021
* model name     = gaussian_pk_sjk
**************************************************************
```

```
> missingValues(model)
```

```
      row col     value
 [1,] 102   1  4.330235
 [2,] 133   1  1.987495
 [3,] 148   1  1.985046
 [4,] 209   1  1.985046
 [5,] 219   1  1.986394
 [6,] 253   1  4.328307
 [7,]   9   2 53.591306
 [8,]  28   2 80.463814
 [9,] 163   2 53.592256
[10,] 218   2 80.464020
```

```
> plot(model)
```



## 4.2. Clustering with Multivariate categorical Mixture Models

Categorical (nominal) data can be clustered using the `clusterCategorical` function.

This function has one mandatory argument: a data.frame or matrix `x`. The matrix `x` can contain characters (nominal values), these characters will be mapped as integer using the `factor` function.

In Table 5 the reader will find a summary of all the specific input parameters of this function with its default value.

| Input Parameter | Description |
|---|---|
| `data` | Matrix or data frame |
| `modelNames` | A `vector` defining the models to estimate. Call `clusterCatgoricalNames()` by default (see 3.2). |

Table 5: List of all the specific parameters of the `clusterCategorical` function.

We illustrate this function with the birds data set.

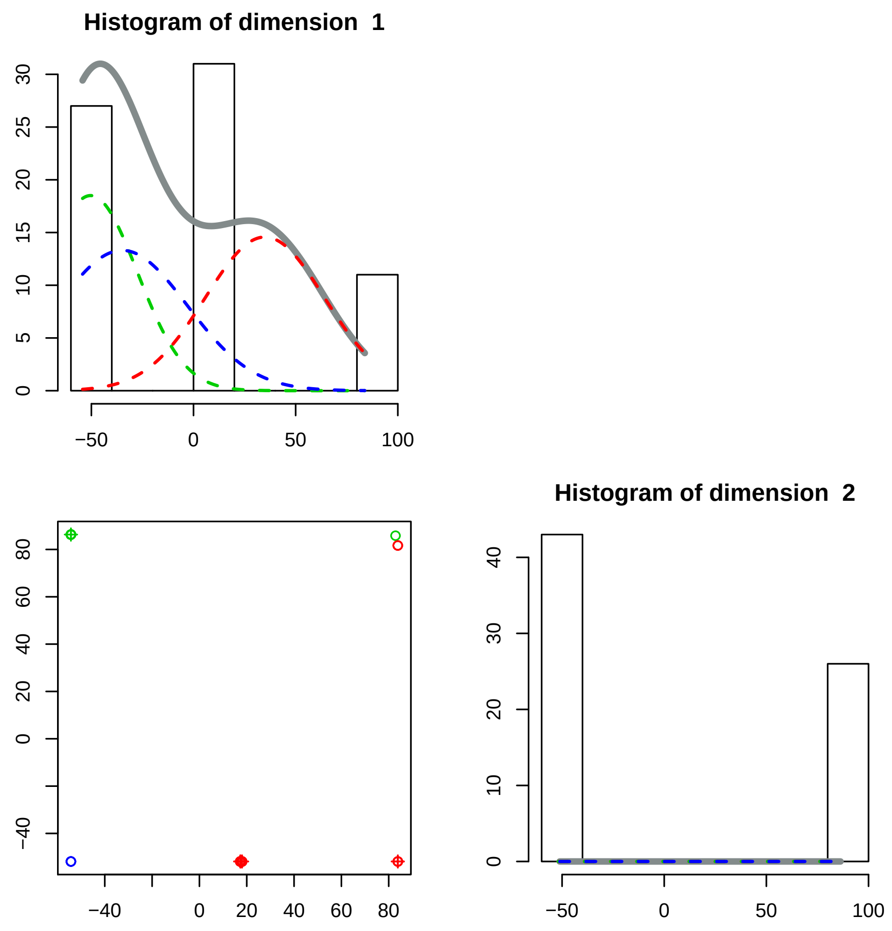
```
> data(birds)
> x = as.matrix(birds);  n <- nrow(x); p <- ncol(x);
> indexes  <- matrix(c(round(runif(10,1,n)), round(runif(10,1,p))), ncol=2);
> x[indexes] <- NA;
> model <- clusterCategorical(data=x, nbCluster=3, strategy = clusterFastStrategy())
> summary(model)
```

```
**************************************************************
* nbSample       =  69
* nbCluster      =  3
* lnLikelihood   =  -190.3708
* nbFreeParameter=  32
* criterion      =  531.736
* model name     = categorical_pk_pjk
* nbModalities   =  4
**************************************************************
```

```
> missingValues(model)
```

```
   row col value
13  13   1     1
31  31   2     3
36  36   2     3
45  45   2     3
17  17   3     1
39  39   3     2
52  52   3     2
4    4   4     4
39  39   4     4
54  54   4     4
```

```
> plot(model)
```



Categorical mixture models are plotted using the *logistic latent representation*.

## 4.3. Clustering with Multivariate gamma Mixture Models

Gamma data can be clustered using the `clusterGamma` function.

This function has one mandatory argument: a data.frame or matrix **x**.

In Table 6 the reader will find a summary of all the specific input parameters of this function with its default value.

| Input Parameter | Description |
|---|---|
| `data` | Matrix or data frame |
| `modelNames` | A vector defining the models to estimate. Call `clusterGammaNames()` by default (see 3.4). |

Table 6: List of all the specific parameters of the `clusterGamma` function.

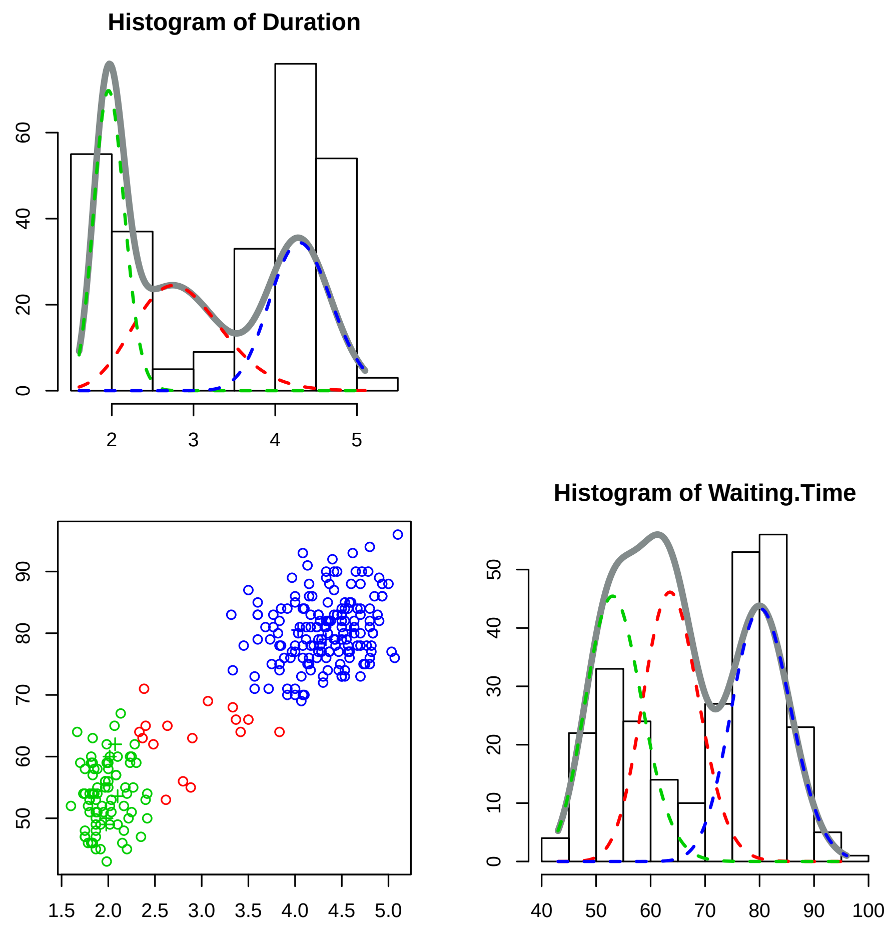
```
> data(geyser);
> x = as.matrix(geyser); n <- nrow(x); p <- ncol(x);
> indexes  <- matrix(c(round(runif(10,1,n)), round(runif(10,1,p))), ncol=2);
> x[indexes] <- NA;
> model <- clusterGamma(data=x, nbCluster=3, strategy = clusterFastStrategy())
> summary(model)
```

```
**************************************************************
* nbSample       =  272
* nbCluster      =  3
* lnLikelihood   =  -1122.695
* nbFreeParameter=  14
* criterion      =  2355.614
* model name     = gamma_pk_ajk_bjk
**************************************************************
```

```
> missingValues(model)
```

```
      row col     value
 [1,]  17   1  2.073234
 [2,]  91   1  2.014638
 [3,] 117   1  1.978440
 [4,] 209   1  1.978294
 [5,] 221   1  1.978440
 [6,] 239   1  4.329060
 [7,]  48   2 53.580309
 [8,]  71   2 80.524264
 [9,] 154   2 80.525713
[10,] 205   2 80.525713
```

```
> plot(model)
```



## 4.4. Clustering with Multivariate Poisson Models

Poisson data (count data) can be clustered using the `clusterPoisson` function.

This function has one mandatory argument: a data.frame or matrix **x**.

In Table 7 the reader will find a summary of all the specific input parameters of this function with its default value.

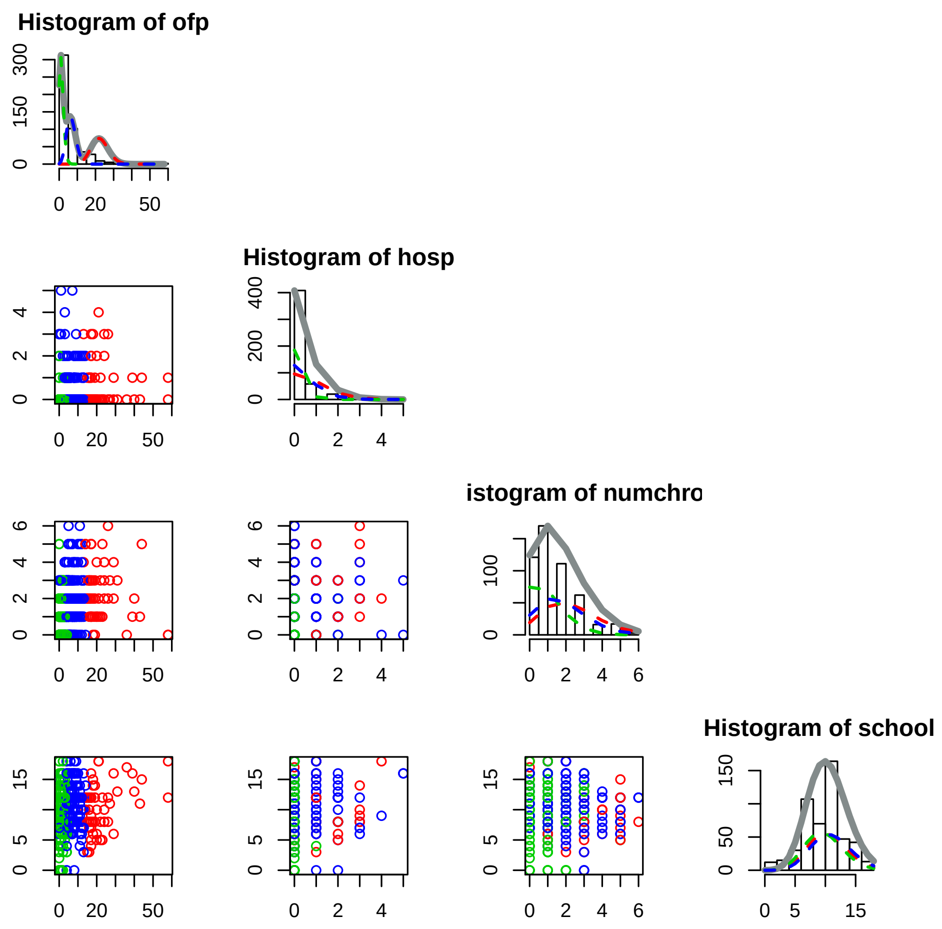
| Input Parameter | Description |
|---|---|
| `data` | Matrix or data frame |
| `modelNames` | A vector defining the models to estimate. Call `clusterPoissonNames()` by default (see 3.3). |

Table 7: List of all the specific parameters of the `clusterPoisson` function.

```
> data(DebTrivedi)
> dt <- DebTrivedi[1:500, c(1, 6,8, 15)]
> model <- clusterPoisson( data=dt, nbCluster=3, strategy = clusterFastStrategy())
> summary(model)

***************************************************************
```

```
* nbSample        =  500
* nbCluster       =  3
* lnLikelihood    =  -3986.894
* nbFreeParameter=  14
* criterion       =  8249.844
* model name      = poisson_pk_ljk
****************************************************************
```

```
> missingValues(model)

     row col value

> plot(model)
```



## 4.5. Clustering with Kernel Mixture Models

Data can be clustered using the `clusterKernel` function.

This function has one mandatory argument: a data.frame or matrix **x**.

In Table 8 the reader will find a summary of all the specific input parameters of this function with its default value.

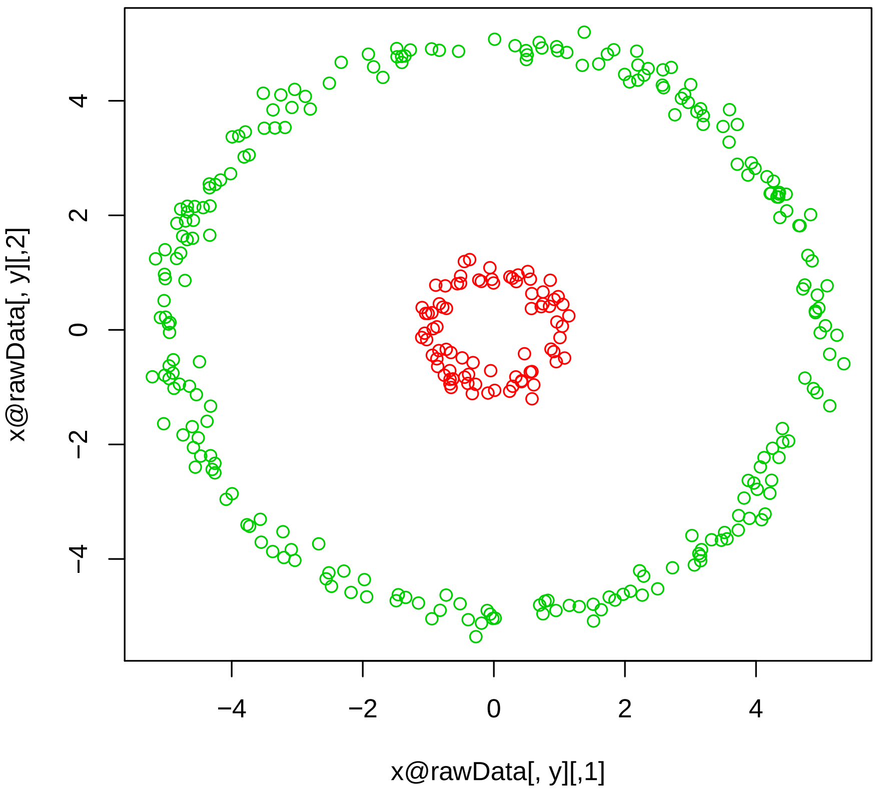
| Input Parameter | Description |
|---|---|
| `data` | Matrix or data frame |
| `modelNames` | A `vector` defining the models to estimate. Call `clusterKernelNames()` by default (see 3.5). |
| `kernelName` | A `string` defining the kernel to use. Use a ”gaussian” kernel by default (Possible values are ”gaussian”, ”polynomial” or ”exponential”). |
| `kernelParameters` | A vector with the kernel parameter value(s). Use 1 by default. |

Table 8: List of all the specific parameters of the `clusterPoisson` function.

```
> data(bullsEye)
> model <- clusterKernel( data=bullsEye[,1:2], nbCluster=2, modelNames = "kernelGaussian_p
> summary(model)
```

```
**************************************************************
* nbSample       =  320
* nbCluster      =  2
* lnLikelihood   =  -1211.76
* nbFreeParameter=  2
* criterion      =  2442.9
* model name     = kernelGaussian_pk_s
**************************************************************
```

```
> plot(model)
```



## 4.6. Clustering Heterogeneous data sets

Heterogeneous data sets can be clustered using the `clusterHeterogeneous` function. The original heterogeneous data set has to be splited in multiple homogeneous data sets and each one associated to a mixture model name.

In Table 9 the reader will find a summary of all the specific input parameters of this function with its default value.

| Input Parameter | Description |
|---|---|
| `data` | A `list` containing the homogeneous data sets (matrices and/or data.frames). All the data sets must have the same number of rows. |
| `modelNames` | A `vector` of character and of same length than data containing the model names to fit to each data set (see 3.6). |

Table 9: List of all the specific parameters of the `clusterHeterogeneous` function.

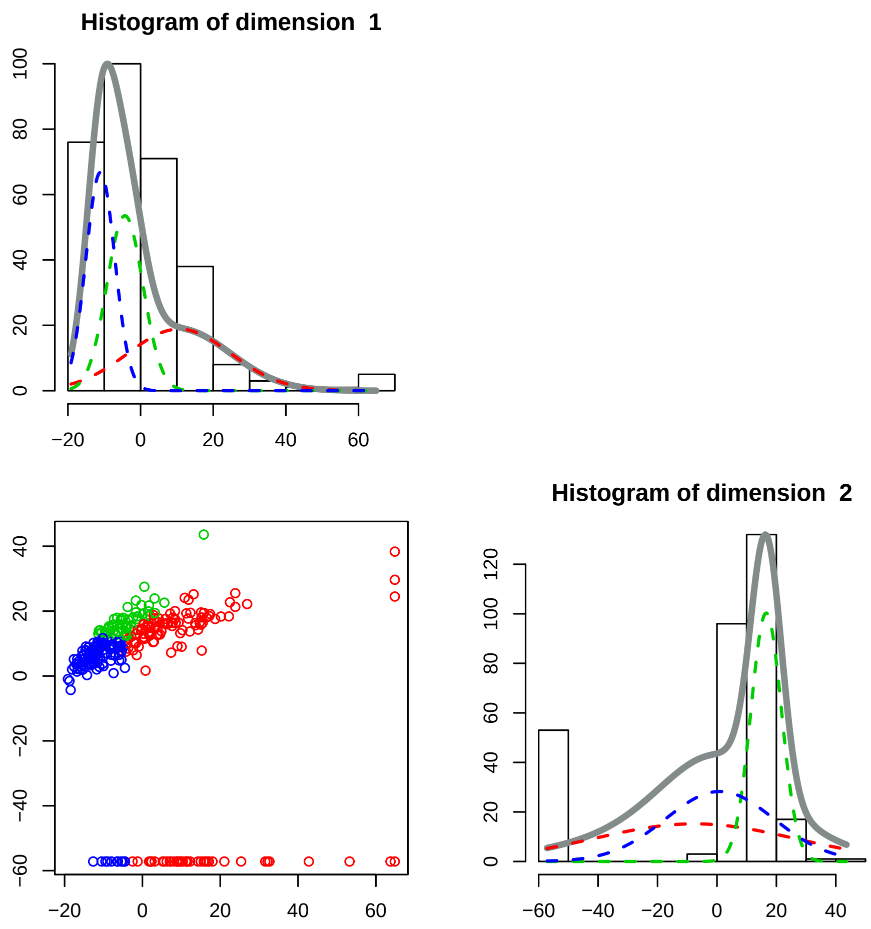
```
> data(HeartDisease.cat)
> data(HeartDisease.cont)
> ldata = list(HeartDisease.cat,HeartDisease.cont);
> lnames = c("categorical_pk_pjk","gaussian_pk_sjk")
> model <- clusterHeterogeneous(ldata, lnames, nbCluster=3, strategy = clusterFastStrategy
> summary(model)
```

```
**************************************************************
* nbSample       =  303
* nbCluster      =  3
* lnLikelihood   =  -7499.262
* nbFreeParameter=  77
* criterion      =  15560.36
**************************************************************
```

```
> missingValues(model)
```

```
[[1]]
     row col value
[1,] 167   7     1
[2,] 193   7     1
[3,] 288   7     1
[4,] 303   7     1
[5,]  88   8     1
[6,] 267   8     3

[[2]]
     row col value
```

```
> plot(model)
```



Heterogeneous mixture models are plotted using the *logistic latent representation*.

## References

Azzalini A, Bowman AW (1990). "A look at some data on the Old Faithful geyser." *Applied Statistics*, pp. 357–365.

Brent RP (1973). "Some Efficient Algorithms for Solving Systems of Nonlinear Equations." **10**(2), 327–344. ISSN 0036-1429 (print), 1095-7170 (electronic).

Dempster A, Laird N, Rubin D (1997). "Maximum Likelihood from Incomplete Data with the EM Algorithm (with discussion)." *Journal of the Royal Statistical Society, Series B*, **39**, 1.

Härdle W (1991). *Smoothing techniques: with implementation in S*. Springer Science & Business Media.

Iovleff S (2012). "The Statitiscal ToolKit." `http://www.stkpp.org/`.

Mclachlan G, Peel D (2000). *Finite Mixture Models*. Wiley Series in Probability and Statistics, 1 edition. Wiley-Interscience. ISBN 9780471006268. URL `http://www.worldcat.org/isbn/0471006262`.

R Development Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

Shawe-Taylor J, Cristianini N (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press. ISBN 0521813972. Other identifier: 9780521813976.

## A. M step computation for the Gaussian models

For all the M Step, the mean is updated using the following formula

$$\boldsymbol{\mu}_k = \frac{1}{t_{.k}} \sum_{i=1}^{n} t_{ik} \mathbf{x}_i,$$

with $t_{.k} = \sum_{i=1}^{n} t_{ik}$, for $k = 1, \ldots, K$.

### A.1. M Step of the gaussian sjk model

Using the equation (5) and dropping the constant, we obtain that we have to maximize in $\boldsymbol{\sigma} = (\sigma_k^j)^2$, for $j = 1, \ldots, d$ and $k = 1, \ldots, K$ the expression

$$l(\boldsymbol{\sigma}|\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{t}^m) = \sum_{i=1}^{n} \sum_{k=1}^{K} t_{ik}^m \sum_{j=1}^{d} \left[ -\frac{1}{(\sigma_k^j)^2} (x_i^j - \hat{\mu}_j^k)^2 - \log((\sigma_k^j)^2) \right]. \tag{26}$$

For this model, the variance is updated using the formula:

$$(\hat{\sigma}_k^j)^2 = \frac{1}{t_{.k}} \sum_{i=1}^{n} t_{ik} (x_i^j - \hat{\mu}_k^j)^2.$$

### A.2. M Step of the gaussian sk model

For this model, the variance is updated using the formula:

$$(\hat{\sigma}_k)^2 = \frac{1}{dt_{.k}} \sum_{j=1}^{d} \sum_{i=1}^{n} t_{ik} (x_i^j - \hat{\mu}_k^j)^2.$$

### A.3. M Step of the gaussian sj model

For this model, the variance is updated using the formula:

$$(\hat{\sigma}^j)^2 = t_{ik} (x_i^j - \mu_k^j)^2.$$

### A.4. M Step of the gaussian s model

For this model, the variance is updated using the formula:

$$\hat{\sigma}^2 = \frac{1}{nd} \sum_{i=1}^{n} \sum_{k=1}^{K} t_{ik} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2.$$

## B. M step computation for the Gamma models

In this section, given the array $\mathbf{t}$ of conditional probabilities, we will write $t_{.k} = \sum_{i=1}^n t_{ik}$, for $k = 1, \ldots, K$ and will denote

$$\bar{x}_k^j = \frac{1}{t_{.k}} \sum_{i=1}^n t_{ik} x_i^j,$$

the $k$-th pondered mean of the $j$-th observation, and by

$$(\overline{\log(x)})_k^j = \frac{1}{t_{.k}} \sum_{i=1}^n t_{ik} \log(x_i^j),$$

the $k$-th pondered log-mean of the $j$-th observation.

Replacing $h$ by its expression in the equation (5) and summing in $i$, the M-step for the twelve gamma mixture models defined in table (2) is equivalent to maximize the following expression

$$l(A,B) = \sum_{k=1}^K \sum_{j=1}^d t_{.k} \left( A (\overline{\log(x)})_k^j - \frac{\bar{x}_k^j}{B} - \log(\Gamma(A)) - A \log(B) \right), \tag{27}$$

with $A \in \{a, a^j, a_k, a_k^j\}$ and $B \in \{b, b^j, b_k, b_k^j\}$.

We now describe the various derivatives and for each models explicit the maximum likelihood equations to solve. Taking the derivative with respect to $B$:

- If $B = b_k^j$ then
$$\frac{dl}{db_k^j} = t_{.k} \left( \frac{\bar{x}_k^j}{b^2} - \frac{A}{b} \right) \text{ and thus } \hat{b}_k^j = \frac{\bar{x}_k^j}{A}$$
- If $B = b_k$ then
$$\frac{dl}{db_k} = t_{.k} \sum_{j=1}^d \left( \frac{\bar{x}_k^j}{b_k^2} - \frac{A}{b_k} \right) \text{ and thus } \hat{b}_k = \frac{\sum_{j=1}^d \bar{x}_k^j}{\sum_{j=1}^d A}$$
- If $B = b^j$ then
$$\frac{dl}{db^j} = \sum_{k=1}^K t_{.k} \left( \frac{\bar{x}_k^j}{(b^j)^2} - \frac{A}{b^j} \right) \text{ and thus } \hat{b}^j = \frac{\sum_{k=1}^K t_{.k} \bar{x}_k^j}{\sum_{k=1}^K t_{.k} A}$$
- If $B = b$ then
$$\frac{dl}{db} = \sum_{k=1}^K \sum_{j=1}^d t_{.k} \left( \frac{\bar{x}_k^j}{b^2} - \frac{A}{b} \right) \text{ and thus } \hat{b} = \frac{\sum_{k=1}^K \sum_{j=1}^d t_{.k} \bar{x}_k^j}{\sum_{k=1}^K \sum_{j=1}^d t_{.k} A}$$

Taking now the derivative with respect to $A$:

1. If $A = a_k^j$, then
$$\frac{dl}{da_k^j} = t_{.k} \left( (\overline{\log(x)})_k^j - \log(B) \right) - t_{.k} \Psi(a_k^j).$$
   and thus

- if $B = b_k^j$ (model `gamma_ajk_bjk`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k^j) & = & (\overline{\log(x)})_k^j - \log(\hat{b}_k^j) \\ \hat{b}_k^j & = & \frac{\bar{x}_k^j}{\hat{a}_k^j}, \end{array}\right. \tag{28}$$

- if $B = b_k$ (model `gamma_ajk_bk`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k^j) & = & (\overline{\log(x)})_k^j - \log(\hat{b}_k) \\ \hat{b}_k & = & \frac{\sum_{j=1}^d \bar{x}_k^j}{\sum_{j=1}^d a_k^j} \end{array}\right. \tag{29}$$

- if $B = b^j$ (model `gamma_ajk_bj`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k^j) & = & (\overline{\log(x)})_k^j - \log(\hat{b}^j) \\ \hat{b}^j & = & \frac{\sum_{k=1}^K t_{.k}\bar{x}_k^j}{\sum_{k=1}^K t_{.k}a_k^j} \end{array}\right. \tag{30}$$

- if $B = b$ (model `gamma_ajk_b`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k^j) & = & (\overline{\log(x)})_k^j - \log(\hat{b}) \\ \hat{b} & = & \frac{\sum_{j=1}^d \sum_{k=1}^K t_{.k}\bar{x}_k^j}{\sum_{j=1}^d \sum_{k=1}^K t_{.k}a_k^j} \end{array}\right. \tag{31}$$

2. If $A = a_k$, then

$$\frac{dl}{da_k} = t_{.k} \sum_{j=1}^d \Big( (\overline{\log(x)})_k^j - \log(B) \Big) - t_{.k} d\Psi(a_k).$$

and thus

- if $B = b_k^j$ (model `gamma_ak_bjk`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k) & = & \frac{1}{d}\sum_{j=1}^d \Big( (\overline{\log(x)})_k^j - \log(\hat{b}_k^j) \Big) \\ \hat{b}_k^j & = & \frac{\bar{x}_k^j}{\hat{a}_k}, \end{array}\right. \tag{32}$$

- if $B = b_k$ (model `gamma_ak_bk`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k) & = & \frac{1}{d}\sum_{j=1}^d \Big( (\overline{\log(x)})_k^j - \log(\hat{b}_k) \Big) \\ \hat{b}_k & = & \frac{\sum_{j=1}^d \bar{x}_k^j}{da_k} \end{array}\right. \tag{33}$$

- if $B = b^j$ (model `gamma_ak_bj`)

$$\left\{\begin{array}{lcl} \Psi(\hat{a}_k) & = & \frac{1}{d}\sum_{j=1}^d \Big( (\overline{\log(x)})_k^j - \log(\hat{b}^j) \Big) \\ \hat{b}^j & = & \frac{\sum_{k=1}^K t_{.k}\bar{x}_k^j}{\sum_{k=1}^K t_{.k}a_k} \end{array}\right. \tag{34}$$

- if $B = b$ (model `gamma_ak_b`)

$$\left\{ \begin{array}{lcl} \Psi(\hat{a}_k) & = & \frac{1}{d}\sum_{j=1}^{d}\left((\overline{\log(x)})_k^j - \log(\hat{b})\right) \\ \hat{b} & = & \frac{\sum_{j=1}^{d}\sum_{k=1}^{K} t_{.k}\bar{x}_k^j}{d\sum_{k=1}^{K} t_{.k}a_k} \end{array} \right. \tag{35}$$

3. If $A = a^j$, then

$$\frac{dl}{da^j} = \sum_{k=1}^{K} t_{.k}\left((\overline{\log(x)})_k^j - \log(B)\right) - n\Psi(a^j).$$

and thus

- if $B = b_k^j$ (model `gamma_aj_bjk`)

$$\left\{ \begin{array}{lcl} \Psi(\hat{a}^j) & = & \frac{1}{n}\sum_{k=1}^{K} t_{.k}\left((\overline{\log(x)})_k^j - \log(\hat{b}_k^j)\right) \\ \hat{b}_k^j & = & \frac{\bar{x}_k^j}{\hat{a}^j}, \end{array} \right. \tag{36}$$

- if $B = b_k$ (model `gamma_aj_bk`)

$$\left\{ \begin{array}{lcl} \Psi(\hat{a}^j) & = & \frac{1}{n}\sum_{k=1}^{K} t_{.k}\left((\overline{\log(x)})_k^j - \log(\hat{b}_k)\right) \\ \hat{b}_k & = & \frac{\sum_{j=1}^{d}\bar{x}_k^j}{\sum_{j=1}^{d}\hat{a}^j} \end{array} \right. \tag{37}$$

4. If $A = a$, then

$$\frac{dl}{da} = \sum_{k=1}^{K}\sum_{j=1}^{d} t_{.k}\left((\overline{\log(x)})_k^j - \log(B)\right) - nd\Psi(a).$$

and thus

- if $B = b_k^j$ (model `gamma_a_bjk`)

$$\left\{ \begin{array}{lcl} \Psi(\hat{a}) & = & \frac{1}{nd}\sum_{j=1}^{d}\sum_{k=1}^{K} t_{.k}\left((\overline{\log(x)})_k^j - \log(\hat{b}_k^j)\right) \\ \hat{b}_k^j & = & \frac{\bar{x}_k^j}{\hat{a}}, \end{array} \right. \tag{38}$$

- if $B = b_k$ (model `gamma_a_bk`)

$$\left\{ \begin{array}{lcl} \Psi(\hat{a}) & = & \frac{1}{nd}\sum_{j=1}^{d}\sum_{k=1}^{K} t_{.k}\left((\overline{\log(x)})_k^j - \log(\hat{b}_k)\right) \\ \hat{b}_k & = & \frac{\sum_{j=1}^{d}\bar{x}_k^j}{d\hat{a}} \end{array} \right. \tag{39}$$

In the next sections, we will describe for some models the way to estimate $A$ and $B$ when $A = a_k^j$.

## B.1. First algorithm for the M Step of the gamma models

Among the twelve models, we can find six models from whom it is possible to estimate in a single pass of the Brent's method the value of $A$ and then to estimate the value of $B$.

For example for the `gamma_ajk_bjk` model, using (28) gives $\hat{a}_k^j$ solution in $a$ of the following equation

$$(\overline{\log(x)})_k^j - \Psi(a) - \log(\bar{x}_k^j) + \log(a) = 0 \tag{40}$$

whom solution can be found using Brent's method Brent (1973).
Having found the estimator of the $a_k^j$, the estimator of $b_k^j$ can be computed.

## B.2. Second algorithm for the M Step of the gamma models

For the other models we have to iterate in order to find the ML estimators. For example for the `gamma_ajk_bj` model, the set of non-linear equations (30) can be solved using an iterative algorithm:

- **Initialization:** Compute an initial estimator of the $\mathbf{a}_k$, $k = 1, \ldots K$ and $\mathbf{b}$ using moment estimators.
- **Repeat until convergence :**
  - **a step:** For fixed $b^j$ solve for each $a_k^j$, the equation:

    $$\Psi(a) - (\overline{\log(x)})_k^j + \log(b^j) = 0.$$

  - **b step:** Update $b^j$ using equation (30).

This algorithm minimize alternatively the log-likelihood in $\mathbf{a}_k$, $k = 1, \ldots n$ and in $\mathbf{b}$ and converge in few iterations.

**Affiliation:**

Serge Iovleff
Univ. Lille 1, CNRS U.M.R. 8524, Inria Lille Nord Europe
59655 Villeneuve d'Ascq Cedex, France
E-mail: Serge.Iovleff@stkpp.org
URL: http://www.stkpp.org