Package 'MixtureFitting'

May 23, 2025

Version 0.6.1

Date 2025-05-20

Title Fitting of Univariate Mixture Distributions to Data using Various Approaches

Depends R (>= 2.0.1)

Description Methods for fitting mixture distributions to univariate data using expectation maximization, HWHM and other methods. Supports Gaussian, Cauchy, Student's t and von Mises mixtures. For more details see Merkys (2018) <https://www.lvb.lt/permalink/370LABT_NETWORK/1m6ui06/alma9910036312108451>.

License GPL-2

NeedsCompilation yes

Author Andrius Merkys [aut, cre]

Maintainer Andrius Merkys <andrius.merkys@gmail.com>

Repository CRAN

Date/Publication 2025-05-23 12:02:06 UTC

Contents

`abs_convergence` *Absolute Convergence Check.*

Description

Compare two values to tell whether an optimization process has converged.

Usage

```
abs_convergence( p_now, p_prev, epsilon = 1e-6 )
```

Arguments

`p_now`	function value of *i*-th iteration.
`p_prev`	function value of *i-1*-th iteration.
`epsilon`	convergence criterion

Value

TRUE if deemed to have converged, FALSE otherwise

Author(s)

Andrius Merkys

`bhattacharyya_dist` *Bhattacharyya distance for univariate Gaussian distributions.*

Description

Measures Bhattacharyya distance for two univariate Gaussian distributions.

Usage

```
bhattacharyya_dist( mu1, mu2, sigma1, sigma2 )
```

Arguments

`mu1`	mean of the first Gaussian distribution.
`mu2`	mean of the second Gaussian distribution.
`sigma1`	standard deviation of the first Gaussian distribution.
`sigma2`	standard deviation of the second Gaussian distribution.

Value

Bhattacharyya distance as double.

Author(s)

Andrius Merkys

`bic` *Bayesian Information Criterion (BIC)*

Description

Calculates Bayesian Information Criterion (BIC) for any type of mixture model. Log-likelihood function has to be provided.

Usage

```
bic( x, p, llf )
```

Arguments

`x`	data vector
`p`	vector of mixture model parameters
`llf`	function calculating log-likelihood, called as llf(x, p)

Value

Bayesian Information Criterion value.

Author(s)

Andrius Merkys

`cmm_fit_em` *Estimate Cauchy Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for Caucy mixture using Expectation Maximization algorithm.

Usage

```
cmm_fit_em( x, p, epsilon = c( 0.000001, 0.000001, 0.000001 ),
            iter.cauchy = 20, debug = FALSE, implementation = "C" )
```

Arguments

x	data vector
p	initialization vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, gamma1, gamma2, ..., gamma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and gamma*i* is the Cauchy scale of *i*-th component.
epsilon	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_gamma), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers and epsilon_gamma is threshold for component Cauchy scales.
iter.cauchy	number of iterations to fit a single Cauchy component.
debug	flag to turn the debug prints on/off.
implementation	flag to switch between C (default) and R implementations.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Ferenc Nahy. Parameter Estimation of the Cauchy Distribution in Information Theory Approach (2006). Journal of Universal Computer Science

---

cmm_fit_hwhm_spline_deriv

Estimate Cauchy Mixture Parameters Using Derivatives and Half-Width at Half-Maximum Method.

Description

Estimate Cauchy mixture parameters using derivatives and half-width at half-maximum (HWHM) method. The method smooths the histogram before attempting to locate the modes. Then it describes them using HWHM.

Usage

```
cmm_fit_hwhm_spline_deriv( x, y )
```

Arguments

x	data vector
y	response vector for *x*

Value

Parameter vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, gamma1, gamma2, ..., gamma$n$), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and gamma$i$ is the Cauchy scale of i-th component.

Author(s)

Andrius Merkys

`cmm_init_vector` *Estimate Cauchy Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Cauchy mixture fitting via Expectation Maximization. Proportions are set to equal, centers are equispaced through the whole domain of input sample, and scales are set to 1.

Usage

```
cmm_init_vector( x, m, implementation = "C" )
```

Arguments

`x`	data vector
`m`	number of mixture components
`implementation`	flag to switch between C (default) and R implementations.

Value

Parameter vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, gamma1, gamma2, ..., gamma$n$), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and gamma$i$ is the Cauchy scale of i-th component.

Author(s)

Andrius Merkys

`cmm_init_vector_kmeans` *Estimate Cauchy Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Cauchy mixture fitting using k-means. R implementation of k-means in kmeans() is used to find data point assignment to clusters. Then several iterations of Cauchy mixture fitting (per Nahy 2006) is used to derive mixture parameters.

Usage

```
cmm_init_vector_kmeans( x, m, iter.cauchy = 20 )
```

Arguments

`x`	data vector
`m`	number of mixture components
`iter.cauchy`	number of iterations to fit a single Cauchy component.

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, gamma1, gamma2, ..., gamma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and gamma*i* is the Cauchy scale of *i*-th component.

Author(s)

Andrius Merkys

References

Ferenc Nahy. Parameter Estimation of the Cauchy Distribution in Information Theory Approach (2006). Journal of Universal Computer Science

cmm_intersections *Intersections of Two Cauchy Distributions*

Description

Finds intersections of two Cauchy distributions by finding roots of a quadratic equation.

Usage

```
cmm_intersections( p )
```

Arguments

p	parameter vector of 6 parameters. Structure of p vector is p = c(A1, A2, mu1, mu2, gamma1, gamma2), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, gamma*i* is the Cauchy scale of *i*-th component.

Value

A vector of x values of intersections (zero, one or two). Returns NaN if both distributions are identical.

Author(s)

Andrius Merkys

dcgmm *Density of The Cauchy-Gaussian Distribution*

Description

Density function for the Cauchy-Gaussian distribution, according to Eqn. 2 of Swami (2000).

Usage

```
dcgmm( x, p )
```

Arguments

x	data vector
p	parameter vector of 5**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, e1, e2, ..., e*n*, mu1, mu2, ..., mu*n*, gamma1, gamma2, ..., gamma*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, e*i* is the proportion of Cauchy subcomponent of *i*-th component, mu*i* is the center of *i*-th component, gamma*i* is the Cauchy concentration of *i*-th component and sigma*i* is the Gaussian standard deviation of *i*-th component.

Value

A vector.

Author(s)

Andrius Merkys

References

Swami, A. Non-Gaussian mixture models for detection and estimation in heavy-tailed noise 2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100), 2000, 6, 3802-3805

---

dcmm *Density of The Cauchy Mixture Distribution*

Description

Density function for the Cauchy mixture distribution.

Usage

```
dcmm( x, p, implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, gamma1, gamma2, ..., gamma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, gamma*i* is the Cauchy scale of *i*-th component.
implementation	flag to switch between C (default) and R implementations.

Value

A vector.

Author(s)

Andrius Merkys

dgmm *The Gaussian Mixture Distribution*

Description

Density function for the Gaussian mixture distribution.

Usage

```
dgmm( x, p, normalise_proportions = FALSE, restrict_sigmas = FALSE,
      implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.
normalise_proportions	if TRUE, make component proportions sum up to 1 by dividing each one of them by their sum (R implementation only).
restrict_sigmas	if TRUE, skip components with scales less or equal to zero (R implementation only).
implementation	flag to switch between C (default) and R implementations.

Value

A vector.

Author(s)

Andrius Merkys

digamma_approx *Calculate Approximate Value of The Digamma Function.*

Description

Calculates approximate value of the digamma function using first eight non-zero members of asymptotic expression for digamma(x). Implemented according to Wikipedia.

Usage

```
digamma_approx( x )
```

Arguments

x	data vector

Value

Digamma function value.

Author(s)

Andrius Merkys

References

Users of Wikipedia. Digamma function. https://en.wikipedia.org/w/index.php?title=Digamma_function&oldid=708779689

---

ds *Density of The Student's t Model*

Description

Density function for the Student's t Model. Wrapper around R's dt(), supporting center and concentration parameters.

Usage

```
ds( x, c, s, ni )
```

Arguments

x	data vector
c	center
s	concentration
ni	degrees of freedom

Value

A vector.

Author(s)

Andrius Merkys

dsmm *Density of The Student's t Mixture Model*

Description

Density function for the Student's t Mixture Model.

Usage

```
dsmm( x, p )
```

Arguments

x	data vector
p	parameter vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.

Value

A vector.

Author(s)

Andrius Merkys

dvmm *Density of The von Mises Mixture Model.*

Description

Density function for the von Mises Mixture Model.

Usage

```
dvmm( x, p, implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.
implementation	flag to switch between C (default) and R implementations.

Value

A vector.

Author(s)

Andrius Merkys

`gmm_fit_em` *Estimate Gaussian Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for Gaussian mixture using Expectation Maximization algorithm.

Usage

```
gmm_fit_em( x, p, w = numeric(), epsilon = c( 0.000001, 0.000001, 0.000001 ),
            debug = FALSE, implementation = "C", ... )
```

Arguments

`x`	data vector
`p`	initialization vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and sigma*i* is the scale of *i*-th component.
`w`	weights of data points, must have the same length as the data vector; if not given or has different length, equal weights are assumed.
`epsilon`	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_sigma), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers and epsilon_sigma is threshold for component scales.
`debug`	flag to turn the debug prints on/off.
`implementation`	flag to switch between C (default) and R implementations.
`...`	additional arguments passed to gmm_fit_em_R() when R implementation is used.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

`gmm_fit_hwhm` *Estimate Gaussian Mixture Parameters Using Half-Width at Half-Maximum Method.*

Description

Estimate Gaussian mixture parameters using half-width at half-maximum (HWHM) method. Given a histogram, the method attempts to locate most prominent modes and describe them using HWHM.

Usage

```
gmm_fit_hwhm( x, y, n )
```

Arguments

x	data vector
y	response vector for *x*
n	number of mixture components

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

`gmm_fit_hwhm_spline_deriv` *Estimate Gaussian Mixture Parameters Using Derivatives and Half-Width at Half-Maximum Method.*

Description

Estimate Gaussian mixture parameters using derivatives and half-width at half-maximum (HWHM) method. The method smooths the histogram before attempting to locate the modes. Then it describes them using HWHM.

Usage

```
gmm_fit_hwhm_spline_deriv( x, y )
```

Arguments

x	data vector
y	response vector for *x*

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

---

gmm_fit_kmeans *Estimate Gaussian Mixture parameters from kmeans.*

Description

Estimates parameters for Gaussian mixture using kmeans.

Usage

```
gmm_fit_kmeans( x, n )
```

Arguments

x	data vector
n	number of mixture components

Value

Vector of 3**n* mixture parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

`gmm_init_vector` *Estimate Gaussian Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Gaussian mixture fitting via Expectation Maximization. Proportions and scales are set to equal, centers are equispaced through the whole domain of input sample.

Usage

```
gmm_init_vector( x, n, implementation = "C" )
```

Arguments

`x`	data vector
`n`	number of mixture components
`implementation`	flag to switch between C (default) and R implementations.

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

`gmm_init_vector_kmeans` *Estimate Gaussian Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Gaussian mixture fitting using k-means. R implementation of k-means in kmeans() is used to find data point assignment to clusters.

Usage

```
gmm_init_vector_kmeans( x, m )
```

Arguments

x	data vector
m	number of mixture components

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

---

gmm_init_vector_quantile

Estimate Gaussian Mixture parameters using Expectation Maximization.

Description

Estimate an initialization vector for Gaussian mixture fitting using (weighted) quantiles. Proportions and scales are set to equal, centers are placed at equispaced quantiles.

Usage

```
gmm_init_vector_quantile( x, m, w = numeric() )
```

Arguments

x	data vector
m	number of mixture components
w	weight vector

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

gmm_intersections *Intersections of Two Gaussian Distributions*

Description

Finds intersections of two Gaussian distributions by finding roots of a quadratic equation.

Usage

```
gmm_intersections( p )
```

Arguments

p	parameter vector of 6 parameters. Structure of p vector is p = c(A1, A2, mu1, mu2, sigma1, sigma2), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

A vector of x values of intersections (zero, one or two). Returns NaN if both distributions are identical.

Author(s)

Andrius Merkys

gmm_merge_components *Merge two Gaussian components into one.*

Description

Merges *i*th and *j*th components of Gaussian mixture model. Implemented in the same venue as in `mergeparameters` of `fpc`.

Usage

```
gmm_merge_components( x, p, i, j )
```

Arguments

x	data vector
p	vector of Gaussian mixture parameters. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where *n* is number of mixture components, A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component.
i	index of the first component to be merged. Component with this index will be replaced by a merged one in the output.
j	index of the second component to be merged. Component with this index will be removed in the output.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Hennig, C. Methods for merging Gaussian mixture components Advances in Data Analysis and Classification, Springer Nature, 2010, 4, 3-34

---

gmm_size_probability *The Gaussian Mixture Distribution*

Description

Calculates the posterior probability of a Gaussian mixture with *n* components. Internally, it attempts to maximize log-likelihood of data by calling optim() and returns the list as received from optim().

Usage

```
gmm_size_probability( x, n, method = "SANN" )
```

Arguments

x	data vector
n	number of mixture components
method	optimization method passed to optim()

Value

List representing the converged optim() run.

Author(s)

Andrius Merkys

`gmm_size_probability_nls` *The Gaussian Mixture Distribution*

Description

Calculates the posterior probability of a Gaussian mixture with *n* components. Internally, it bins the data vector and calls nls() to optimize the mixture fit. Returns the list of the same form as received from optim().

Usage

```
gmm_size_probability_nls( x, n, bins = 100, trace = FALSE )
```

Arguments

`x`	data vector
`n`	number of mixture components
`bins`	number of bins
`trace`	should debug trace be printed?

Value

List of the same form as received from optim().

Author(s)

Andrius Merkys

`gradient_descent` *Gradient Descent*

Description

Simple implementation of gradient descent method. Given a derivative function, it follows its decrease until convergence criterion is met.

Usage

```
gradient_descent( gradfn, start, gamma = 0.1, ..., epsilon = 0.01 )
```

Arguments

gradfn	derivative function
start	starting value
gamma	learning rate
...	additional arguments passed to derivative function
epsilon	convergence threshold for absolute squared difference

Value

log-likelihood

Author(s)

Andrius Merkys

---

kldiv *Kullback–Leibler Divergence of* i*th Student's t Mixture component.*

Description

Measures Kullback–Leibler divergence of *i*th Student's t Mixture component using Dirac's delta function. Implemented according to Chen et al. (2004).

Usage

```
kldiv( x, p, k )
```

Arguments

x	data vector
p	vector of Student's t mixture parameters. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where *n* is number of mixture components, A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
k	number of the component.

Value

Kullback–Leibler divergence as double.

Author(s)

Andrius Merkys

References

Chen, S.; Wang, H. & Luo, B. Greedy EM Algorithm for Robust T-Mixture Modeling Third International Conference on Image and Graphics (ICIG'04), Institute of Electrical & Electronics Engineers (IEEE), 2004, 548–551

`kmeans_circular` *K-Means Clustering for Points on Circle*

Description

Perform k-means clustering on angular data (in degrees).

Usage

```
kmeans_circular( x, centers, iter.max = 10 )
```

Arguments

`x`	data vector
`centers`	vector of initial centers (in degrees)
`iter.max`	maximum number of iterations

Value

Vector of the same length as *centers* defining cluster centers (in degrees).

Author(s)

Andrius Merkys

`llcmm` *Log-likelihood for Cauchy Mixture*

Description

Calculates log-likelihood for a given data vector using a Cauchy mixture distribution.

Usage

```
llcmm( x, p, implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, gamma1, gamma2, ..., gamma$n$), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and gamma$i$ is the Cauchy scale of i-th component.
implementation	flag to switch between C (default) and R implementations.

Value

log-likelihood

Author(s)

Andrius Merkys

---

llgmm *Log-likelihood for Gaussian Mixture*

Description

Calculates log-likelihood for a given data vector using a Gaussian mixture distribution.

Usage

```
llgmm( x, p, implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, sigma1, sigma2, ..., sigma$n$), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and sigma$i$ is the scale of i-th component.
implementation	flag to switch between C (default) and R implementations.

Value

log-likelihood

Author(s)

Andrius Merkys

llgmm_conservative *Log-likelihood for Gaussian Mixture*

Description

Calculates log-likelihood for a given data vector using a Gaussian mixture distribution. This is a straightforward implementation, different from llgmm() in that that it does not detect and shortcut edge cases.

Usage

```
llgmm_conservative( x, p )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and sigma*i* is the scale of *i*-th component.

Value

log-likelihood

Author(s)

Andrius Merkys

llgmm_opposite *Opposite Log-likelihood for Gaussian Mixture*

Description

Calculates opposite log-likelihood for a given data vector using a Gaussian mixture distribution.

Usage

```
llgmm_opposite( x, p )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and sigma*i* is the scale of *i*-th component.

Value

opposite log-likelihood (negated log-likelihood value)

Author(s)

Andrius Merkys

`llsmm` *Log-likelihood for Student's t Mixture*

Description

Calculates log-likelihood for a given data vector using a Student's t mixture distribution.

Usage

```
llsmm( x, p )
```

Arguments

`x`	data vector
`p`	parameter vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.

Value

log-likelihood

Author(s)

Andrius Merkys

llvmm *Log-likelihood for von Mises Mixture*

Description

Calculates log-likelihood for a given data vector using a von Mises mixture distribution.

Usage

```
llvmm( x, p, implementation = "C" )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.
implementation	flag to switch between C (default) and R implementations.

Value

log-likelihood

Author(s)

Andrius Merkys

llvmm_opposite *Opposite Log-likelihood for von Mises Mixture*

Description

Calculates opposite log-likelihood for a given data vector using a von Mises mixture distribution.

Usage

```
llvmm_opposite( x, p )
```

Arguments

x	data vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.

Value

opposite log-likelihood (negated log-likelihood value)

Author(s)

Andrius Merkys

---

mk_fit_images *Mixture Distribution Modeling*

Description

Draw a PNG histogram with a mixture density on top of it for each iteration of mixture optimization process.

Usage

```
mk_fit_images( h, l, prefix = "img_" )
```

Arguments

h	histogram object, as returned from hist()
l	list containing model vectors
prefix	prefix of file name to write

Author(s)

Andrius Merkys

---

plot_circular_hist *Mixture Distribution Modeling*

Description

Plot a circular histogram.

Usage

```
plot_circular_hist( x, breaks = 72, ball = 0.5, ... )
```

Arguments

x	data vector
breaks	number of breaks in histogram
ball	radius of the drawn circle
...	parameters passed to plot()

Author(s)

Andrius Merkys

`plot_density` *Mixture Distribution Modeling*

Description

Draw a PNG histogram with a mixture density on top of it.

Usage

```
plot_density( x, model, density_f, width, height,
              cuts = 400, main = "",
              filename = NULL,
              obs_good = c(), obs_bad = c(),
              scale_density = FALSE )
```

Arguments

`x`	data vector
`cuts`	number of breaks in histogram
`main`	main title of the plot
`model`	model passed to density_f()
`density_f`	probability density function
`filename`	name of the file to write
`width`	image width, passed to png()
`height`	image height, passed to png()
`obs_good`	vector of values to mark with rug() in green color
`obs_bad`	vector of values to mark with rug() in red color
`scale_density`	should probability density be scaled?

Author(s)

Andrius Merkys

`polyroot_NR` *Find one real polynomial root using Newton–Raphson method.*

Description

Finds one real polynomial root using Newton–Raphson method, implemented according to Wikipedia.

Usage

```
polyroot_NR( p, init = 0, epsilon = 1e-6, debug = FALSE, implementation = "C" )
```

Arguments

`p`	vector of polynomial coefficients.
`init`	initial value.
`epsilon`	tolerance threshold for convergence.
`debug`	flag to turn the debug prints on/off.
`implementation`	flag to switch between C (default) and R implementations.

Value

Real polynomial root.

Author(s)

Andrius Merkys

References

Users of Wikipedia. Newton's method. https://en.wikipedia.org/w/index.php?title=Newton%27s_method&oldid=710342140

`pssd` *Penalized Sum of Squared Differences Using Gaussian Mixture Distribution*

Description

Given two vectors of same length and a Gaussian mixture, calculate the penalized sum of squared differences (SSD) between the first vector and Gaussian mixture densities measured at points from second vector. Penalties are included for proportions and scales that are less than or equal to 0.

Usage

```
pssd( x, y, p )
```

Arguments

x	data vector
y	response vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

Penalized sum of squared differences.

Author(s)

Andrius Merkys

---

pssd_gradient *Penalized Sum of Squared Differences Using Gaussian Mixture Distribution*

Description

Gradient (derivative) function of pssd().

Usage

```
pssd_gradient( x, y, p )
```

Arguments

x	data vector
y	response vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

Gradient values measured at *x*.

Author(s)

Andrius Merkys

ratio_convergence *Ratio Convergence Check.*

Description

Compare two values to tell whether an optimization process has converged. The absolute difference between values of two iterations is divided by the value of previous iteration and compared to the epsilon value.

Usage

```
ratio_convergence( p_now, p_prev, epsilon = 1e-6 )
```

Arguments

p_now	function value of *i*-th iteration.
p_prev	function value of *i-1*-th iteration.
epsilon	convergence criterion

Value

TRUE if deemed to have converged, FALSE otherwise

Author(s)

Andrius Merkys

rcmm *Random Sample of The Cauchy Mixture Distribution*

Description

Generates a random sample of the Cauchy mixture distribution.

Usage

```
rcmm( n, p )
```

Arguments

n	sample size
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, gamma1, gamma2, ..., gamma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, gamma*i* is the Cauchy scale of *i*-th component.

Value

A vector.

Author(s)

Andrius Merkys

`rgmm` *Random Sample of the Gaussian Mixture Distribution*

Description

Generates a random sample of the Gaussian mixture distribution.

Usage

```
rgmm( n, p )
```

Arguments

`n`	data vector
`p`	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

A vector.

Author(s)

Andrius Merkys

rsimplex_start *Nelder-Mead's Simplex Method for Function Minimization.*

Description

Generate initial simplices for simplex().

Usage

```
rsimplex_start( seed, n, lower, upper )
```

Arguments

seed	seed for random number generator
n	number of simplices
lower	vector with lower bounds of each dimension
upper	vector with upper bounds of each dimension

Value

A list with *n* simplices.

Author(s)

Andrius Merkys

rvmm *Random Sample of the von Mises Mixture Model.*

Description

Generates a random sample of the von Mises Mixture Model.

Usage

```
rvmm( n, p )
```

Arguments

n	sample size
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.

Value

A vector.

Author(s)

Andrius Merkys

References

Best & Fisher. Efficient Simulation of the von Mises Distribution. Journal of the RSS, Series C, 1979, 28, 152-157.

`simplex` *Nelder-Mead's Simplex Method for Function Minimization.*

Description

Nelder-Mead's Simplex Method for Function Minimization.

Usage

```
simplex( fn, start, ..., epsilon = 0.000001, alpha = 1,
         gamma = 2, rho = 0.5, delta = 0.5, trace = FALSE )
```

Arguments

`fn`	minimized function, has to accept the argmin vector as first parameter
`start`	start vector
`...`	other parameters passed to the minimized function
`epsilon`	convergence criterion
`alpha`	reflection coefficient
`gamma`	expansion coefficient
`rho`	contraction coefficient
`delta`	shrink coefficient
`trace`	should debug trace be printed?

Value

Vector yielding the minimum value of the minimized function

Author(s)

Andrius Merkys

References

Nelder, J. A. & Mead, R. A Simplex Method For Function Minimization. The Computer Journal, 1965, 308-313.

Users of Wikipedia. Nelder-Mead method. https://en.wikipedia.org/w/index.php?title=Nelder%E2%80%93Mead_method&oldid=1287347131

`smm_fit_em` *Estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for Student's t mixture using Expectation Maximization algorithm. Calls smm_fit_em_APK10().

Usage

```
smm_fit_em( x, p, ... )
```

Arguments

`x`	data vector
`p`	initialization vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
`...`	additional arguments passed to smm_fit_em_GNL08().

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

`smm_fit_em_APK10` *Estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for univariate Student's t mixture using Expectation Maximization algorithm, according to Fig. 2 of Aeschliman et al. (2010).

Usage

```
smm_fit_em_APK10( x, p, epsilon = c( 1e-6, 1e-6, 1e-6, 1e-6 ),
                  collect.history = FALSE, debug = FALSE )
```

Arguments

`x`	data vector
`p`	initialization vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
`epsilon`	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_k, epsilon_ni), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers, epsilon_k is threshold for component concentrations and epsilon_ni is threshold for component degrees of freedom.
`collect.history`	flag to turn accumulation of estimation history on/off.
`debug`	flag to turn the debug prints on/off.

Value

A list.

Author(s)

Andrius Merkys

References

Aeschliman, C.; Park, J. & Kak, A. C. A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and Its Application to Computer Vision European Conference on Computer Vision 2010, 2010 https://engineering.purdue.edu/RVL/Publications/Aeschliman2010ANovel.pdf

`smm_fit_em_CWL04` *Greedily estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimates (greedily) parameters for univariate Student's t mixture using Expectation Maximization algorithm, implemented according to Chen et al. (2004). The algorithm relies upon smm_fit_em_GNL08() to estimate mixture parameters iteratively.

Usage

```
smm_fit_em_CWL04( x, p, collect.history = FALSE, debug = FALSE,
                  ... )
```

Arguments

`x`	data vector
`p`	initialization vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
`collect.history`	logical. If set to TRUE, a list of parameter values of all iterations is returned.
`debug`	flag to turn the debug prints on/off.
`...`	parameters passed to smm_fit_em_GNL08().

Value

A list.

Author(s)

Andrius Merkys

References

Chen, S.; Wang, H. & Luo, B. Greedy EM Algorithm for Robust T-Mixture Modeling Third International Conference on Image and Graphics (ICIG'04), Institute of Electrical & Electronics Engineers (IEEE), 2004, 548–551

`smm_fit_em_GNL08` *Estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for univariate Student's t mixture using Expectation Maximization algorithm, according to Eqns. 12–17 of Gerogiannis et al. (2009).

Usage

```
smm_fit_em_GNL08( x, p, epsilon = c( 1e-6, 1e-6, 1e-6, 1e-6 ),
                  collect.history = FALSE, debug = FALSE,
                  min.sigma = 1e-256, min.ni = 1e-256,
                  max.df = 1000, max.steps = Inf,
                  polyroot.solution = 'jenkins_taub',
                  convergence = abs_convergence,
                  unif.component = FALSE )
```

Arguments

`x`	data vector
`p`	initialization vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
`epsilon`	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_k, epsilon_ni), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers, epsilon_k is threshold for component concentrations and epsilon_ni is threshold for component degrees of freedom.
`collect.history`	logical. If set to TRUE, a list of parameter values of all iterations is returned.
`debug`	flag to turn the debug prints on/off.
`min.sigma`	minimum value of sigma
`min.ni`	minimum value of degrees of freedom
`max.df`	maximum value of degrees of freedom
`max.steps`	maximum number of steps, may be infinity
`polyroot.solution`	polyroot finding method used to approximate digamma function. Possible values are 'jenkins_taub' and 'newton_raphson'.
`convergence`	function to use for convergence checking. Must accept function values of the last two iterations and return TRUE or FALSE.
`unif.component`	should a uniform component for outliers be added, as suggested by Cousineau & Chartier (2010)?

Value

A list.

Author(s)

Andrius Merkys

References

Gerogiannis, D.; Nikou, C. & Likas, A. The mixtures of Student's t-distributions as a robust framework for rigid registration. Image and Vision Computing, Elsevier BV, 2009, 27, 1285–1294 https://www.cs.uoi.gr/~arly/papers/imavis09.pdf

Cousineau, D. & Chartier, S. Outliers detection and treatment: a review. International Journal of Psychological Research, 2010, 3, 58–67 https://revistas.usb.edu.co/index.php/IJPR/article/view/844

`smm_init_vector` *Estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Student's t mixture fitting via Expectation Maximization. Proportions are set to be equal, centers are equispaced through the whole domain of input sample, concentrations and degrees of freedom are set to 1.

Usage

```
smm_init_vector( x, n )
```

Arguments

x	data vector
n	number of mixture components

Value

Parameter vector of 4**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.

Author(s)

Andrius Merkys

`smm_init_vector_kmeans` *Estimate Student's t Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for Student's t mixture fitting via Expectation Maximization. R implementation of k-means in kmeans() is used to find data point assignment to clusters. s_fit_primitive() is then used to estimate component parameters for each cluster.

Usage

```
smm_init_vector_kmeans( x, m )
```

Arguments

`x`	data vector
`m`	number of mixture components

Value

Parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Author(s)

Andrius Merkys

`smm_split_component` *Split a component of Student's t-distribution in two.*

Description

Splits a component of Student's t-distribution mixture. Implemented according to Eqns. 30–36 of Chen et al. (2004).

Usage

```
smm_split_component( p, alpha = 0.5, beta = 0.5, u = 0.5 )
```

Arguments

p	vector of Student's t mixture parameters. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*, ni1, ni2, ..., ni*n*), where *n* is number of mixture components, A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component, k*i* is the concentration of *i*-th component and ni*i* is the degrees of freedom of *i*-th component.
alpha	split proportion for component proportions
beta	split proportion for component concentrations
u	split proportion for component centers

Value

Vector of parameters for resulting two-component mixture, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Chen, S.-B. & Luo, B. Robust t-mixture modelling with SMEM algorithm Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), Institute of Electrical & Electronics Engineers (IEEE), 2004, 6, 3689–3694

---

ssd *Sum of Squared Differences Using Gaussian Mixture Distribution*

Description

Given two vectors of same length and a Gaussian mixture, calculate the sum of squared differences (SSD) between the first vector and Gaussian mixture densities measured at points from second vector.

Usage

```
ssd( x, y, p )
```

Arguments

x	data vector
y	response vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

Sum of squared differences.

Author(s)

Andrius Merkys

`ssd_gradient` *Sum of Squared Differences Using Gaussian Mixture Distribution*

Description

Gradient (derivative) function of ssd().

Usage

```
ssd_gradient( x, y, p )
```

Arguments

x	data vector
y	response vector
p	parameter vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, sigma1, sigma2, ..., sigma*n*), where A*i* is the proportion of *i*-th component, mu*i* is the location of *i*-th component, sigma*i* is the scale of *i*-th component.

Value

Gradient values measured at *x*.

Author(s)

Andrius Merkys

s_fit_primitive — *Estimate Student's t distribution parameters using Batch Approximation Algorithm.*

Description

Estimates parameters for univariate Student's t distribution parameters using Batch Approximation Algorithm, according to Fig. 2 of Aeschliman et al. (2010).

Usage

```
s_fit_primitive( x )
```

Arguments

x	data vector

Value

Vector `c( mu, k, ni )`, where mu is the center, k is the concentration and ni is the degrees of freedom of the distribution.

Author(s)

Andrius Merkys

References

Aeschliman, C.; Park, J. & Kak, A. C. A Novel Parameter Estimation Algorithm for the Multivariate t-Distribution and Its Application to Computer Vision European Conference on Computer Vision 2010, 2010 https://engineering.purdue.edu/RVL/Publications/Aeschliman2010ANovel.pdf

vmm_fit_em — *Estimate von Mises Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for univariate von Mises mixture using Expectation Maximization algorithm.

Usage

```
vmm_fit_em( x, p, epsilon = c( 0.000001, 0.000001, 0.000001 ),
            debug = FALSE, implementation = "C" )
```

Arguments

x	data vector
p	initialization vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.
epsilon	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_k), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers and epsilon_k is threshold for component concentrations.
debug	flag to turn the debug prints on/off.
implementation	flag to switch between C (default) and R implementations.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Banerjee et al. Expectation Maximization for Clustering on Hyperspheres (2003), manuscript, accessible on: https://web.archive.org/web/20130120061240/http://www.lans.ece.utexas.edu/~abanerjee/papers/05/banerjee05a.pdf

---

vmm_fit_em_by_diff *Estimate von Mises Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for univariate von Mises mixture using Expectation Maximization algorithm. In this version stopping criterion is the difference between parameters in the subsequent iterations.

Usage

```
vmm_fit_em_by_diff( x, p, epsilon = c( 0.000001, 0.000001, 0.000001 ),
                    debug = FALSE, implementation = "C" )
```

Arguments

x	data vector
p	initialization vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, k1, k2, ..., kn), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and ki is the concentration of i-th component.
epsilon	tolerance threshold for convergence. Structure of epsilon is epsilon = c(epsilon_A, epsilon_mu, epsilon_k), where epsilon_A is threshold for component proportions, epsilon_mu is threshold for component centers and epsilon_k is threshold for component concentrations.
debug	flag to turn the debug prints on/off.
implementation	flag to switch between C (default) and R implementations.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Banerjee et al. Expectation Maximization for Clustering on Hyperspheres (2003), manuscript, accessible on: https://web.archive.org/web/20130120061240/http://www.lans.ece.utexas.edu/~abanerjee/papers/05/banerjee05a.pdf

---

vmm_fit_em_by_ll *Estimate von Mises Mixture parameters using Expectation Maximization.*

Description

Estimates parameters for univariate von Mises mixture using Expectation Maximization algorithm. In this version stopping criterion is the difference between log-likelihood estimates of subsequent iterations.

Usage

```
vmm_fit_em_by_ll( x, p, epsilon = .Machine$double.eps,
                  debug = FALSE, implementation = "C" )
```

Arguments

x	data vector
p	initialization vector of 3**n* parameters, where *n* is number of mixture components. Structure of p vector is p = c(A1, A2, ..., A*n*, mu1, mu2, ..., mu*n*, k1, k2, ..., k*n*), where A*i* is the proportion of *i*-th component, mu*i* is the center of *i*-th component and k*i* is the concentration of *i*-th component.
epsilon	tolerance threshold for convergence
debug	flag to turn the debug prints on/off.
implementation	flag to switch between C (default) and R implementations.

Value

Vector of mixture parameters, whose structure is the same as of input parameter's p.

Author(s)

Andrius Merkys

References

Banerjee et al. Expectation Maximization for Clustering on Hyperspheres (2003), manuscript, accessible on: https://web.archive.org/web/20130120061240/http://www.lans.ece.utexas.edu/~abanerjee/papers/05/banerjee05a.pdf

---

vmm_init_vector *Estimate von Mises Mixture parameters using Expectation Maximization.*

Description

Estimate an initialization vector for von Mises mixture fitting via Expectation Maximization. Proportions are set to equal, centers are equispaced through the whole domain of input sample, and concentrations are set to (m/(12*180))^2.

Usage

```
vmm_init_vector( m, implementation = "C" )
```

Arguments

m	number of mixture components
implementation	flag to switch between C (default) and R implementations.

Value

Parameter vector of 3*n parameters, where n is number of mixture components. Structure of p vector is p = c(A1, A2, ..., An, mu1, mu2, ..., mu$n$, k1, k2, ..., kn), where Ai is the proportion of i-th component, mu$i$ is the center of i-th component and ki is the concentration of i-th component.

Author(s)

Andrius Merkys

`wmedian` *Calculate Weighted Median.*

Description

Calculated weighted median.

Usage

```
wmedian( x, w, start = 1, end = length( x ) )
```

Arguments

`x`	sample vector
`w`	weights vector
`start`	start index (default: 1)
`end`	end index (default: last index in x)

Value

Median

Author(s)

Andrius Merkys

References

Users of Wikipedia. Weighted median. https://en.wikipedia.org/w/index.php?title=Weighted_median&oldid=690896947

Index