Package 'admix'

August 21, 2025

Title Package Admix for Admixture (aka Contamination) Models

Version 2.4.3

Description Implements techniques to estimate the unknown quantities related to two-component admixture models, where the two components can belong to any distribution (note that in the case of multinomial mixtures, the two components must belong to the same family). Estimation methods depend on the assumptions made on the unknown component density; see Bordes and Vandekerkhove (2010) <doi:10.3103/S1066530710010023>, Patra and Sen (2016) <doi:10.1111/rssb.12148>, and Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <doi:10.3150/23-BEJ1593>. In practice, one can estimate both the mixture weight and the unknown component density in a wide variety of frameworks. On top of that, hypothesis tests can be performed in one and two-sample contexts to test the unknown component density (see Milhaud, Pommeret, Salhi and Vandekerkhove (2022) <doi:10.1016/j.jspi.2021.05.010>, and Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <doi:10.3150/23-BEJ1593>). Finally, clustering of unknown mixture components is also feasible in a K-sample setting (see Milhaud, Pommeret, Salhi, Vandekerkhove (2024) <https://jmlr.org/papers/v25/23-0914.html>).

License GPL (>= 3)

URL https://github.com/XavierMilhaud/admix-Rpackage

BugReports https://github.com/XavierMilhaud/admix-Rpackage/issues

Encoding UTF-8

RoxygenNote 7.3.2

Imports base, cubature, EnvStats, fdrtool, graphics, Iso, MASS, orthopolynom, pracma, Rcpp, Rdpack, stats, utils

Suggests doParallel, doRNG, evd, flexsurv, foreach, gridExtra, knitr, lattice, logitnorm, plyr, reshape2, rmarkdown, rmutil, testthat (>= 3.0.0)

Depends R (>= 2.10)

LazyData true

LinkingTo Rcpp

RdMacros Rdpack

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation yes

Author Xavier Milhaud [aut, cre],
Pierre Vandekerkhove [ctb],
Denys Pommeret [ctb],
Yahia Salhi [ctb]

Maintainer Xavier Milhaud <xavier.milhaud.research@gmail.com>

Repository CRAN

Date/Publication 2025-08-21 07:50:08 UTC

Contents

`admix_cluster` *Cluster K populations following admixture models*

Description

Create clusters on the unknown components related to the K populations following admixture models. Based on the K-sample test using Inversion - Best Matching (IBM) approach, see 'Details' below for further information.

Usage

```
admix_cluster(
  samples,
  admixMod,
  conf_level = 0.95,
  tune_penalty = TRUE,
  tabul_dist = NULL,
  echo = TRUE,
  ...
)
```

Arguments

`samples`	A list of the K (K>1) samples to be studied, all following admixture distributions.
`admixMod`	A list of objects of class admix_model, containing useful information about distributions and parameters.
`conf_level`	(default to 0.95) The confidence level of the k-sample tests used in the clustering procedure.
`tune_penalty`	(default to TRUE) A boolean that allows to choose between a classical penalty term or an optimized penalty (embedding some tuning parameters, automatically optimized). Optimized penalty is particularly useful for low/mid-sized samples, or unbalanced sample sizes to detect alternatives to the null hypothesis (H0). It is recommended to use it.
`tabul_dist`	(default to NULL) Only useful for comparisons of detected clusters at different confidence levels. A list of the tabulated distributions of the stochastic integral used in the k-sample test, each element for each cluster previously detected.

echo	(default to TRUE) Display the remaining computation time.
...	Optional arguments to IBM_k_samples_test; namely 'n_sim_tab', 'parallel' and 'n_cpu'. These are crucial to speed-up the building of clusters.

Value

An object of class admix_cluster, containing 12 attributes: 1) the number of samples under study; 2) the sizes of samples; 3) the information about mixture components in each sample (distributions and parameters); 4) the number of detected clusters; 5) the list of p-values for each k-sample test at the origin of detected clusters; 6) the cluster affiliation for each sample; 7) the confidence level of statistical tests; 8) which samples in which cluster; 9) the size of clusters; 10) the estimated weights of the unknown component distributions inside each cluster (remind that estimated weights are consistent only if unknown components are tested to be identical, which is the case inside clusters); 11) the matrix of pairwise discrepancies across all samples; 12) the list of tabulated distributions used for statistical tests involved in building the clusters.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Contamination-source based K-sample clustering." *Journal of Machine Learning Research*, **25**(287), 1–32. https://jmlr.org/papers/v25/23-0914.html.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 1600, weight = 0.8,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 16, "scale" = 1/4),
                                        list("rate" = 1/3.5)))
mixt2 <- twoComp_mixt(n = 2000, weight = 0.7,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 14, "scale" = 1/2),
                                        list("rate" = 1/5)))
mixt3 <- twoComp_mixt(n = 2500, weight = 0.6,
                      comp.dist = list("gamma", "gamma"),
                      comp.param = list(list("shape" = 16, "scale" = 1/4),
                                        list("shape" = 12, "scale" = 1/2)))
mixt4 <- twoComp_mixt(n = 3800, weight = 0.5,
                      comp.dist = list("gamma", "exp"),
                      comp.param = list(list("shape" = 14, "scale" = 1/2),
                                        list("rate" = 1/7)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
data3 <- get_mixture_data(mixt3)
data4 <- get_mixture_data(mixt4)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
```

```
                          knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                          knownComp_param = mixt2$comp.param[[2]])
admixMod3 <- admix_model(knownComp_dist = mixt3$comp.dist[[2]],
                          knownComp_param = mixt3$comp.param[[2]])
admixMod4 <- admix_model(knownComp_dist = mixt4$comp.dist[[2]],
                          knownComp_param = mixt4$comp.param[[2]])
## Clustering procedure:
admix_cluster(samples = list(data1, data2, data3, data4),
              admixMod = list(admixMod1, admixMod2, admixMod3, admixMod4),
              conf_level = 0.95, tune_penalty = TRUE, n_sim_tab = 10)
```

`admix_estim` *Estimate the unknown weight in the admixture model*

Description

Estimate the unknown component weight (and location shift parameter in case of a symmetric unknown component density), using different estimation techniques. We remind that the i-th admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, where g_i is the known component density. The unknown quantities p_i and f_i then have to be estimated.

Usage

```
admix_estim(samples, admixMod, est_method = c("PS", "BVdk", "IBM"), ...)
```

Arguments

`samples`	A list of the K (K>0) samples to be studied, all following admixture distributions.
`admixMod`	A list of objects of class admix_model, containing useful information about distributions and parameters.
`est_method`	The estimation method to be applied. Can be one of 'BVdk' (Bordes and Vandekerkhove estimator), 'PS' (Patra and Sen estimator), or 'IBM' (Inversion Best-Matching approach) in the continuous case (continuous random variable). Only 'IBM' for discrete random variables. The same estimation method is performed on each sample if several samples are provided.
`...`	Optional arguments to estim_PS, estim_BVdk or estim_IBM depending on the choice made by the user for the estimation method.

Details

For further details on the different estimation techniques, see references below on i) Patra and Sen estimator ; ii) Bordes and Vandekerkhove estimator ; iii) Inversion Best-Matching approach. Important note: estimation by 'IBM' requires at least two samples at hand, and provides unbiased estimators only if the distributions of unknown components are equal (meaning that it requires to perform previously this test between the pairs of samples, see admix_test.

Value

An object of class admix_estim, containing at least 5 attributes: 1) the number of samples under study; 2) the information about the mixture components (distributions and parameters); 3) the sizes of the samples; 4) the chosen estimation technique (one of 'BVdk', 'PS' or 'IBM'); 5) the estimated mixing proportions (weights of the unknown component distributions in the mixture model). In case of 'BVdk' estimation, one additional attribute corresponding to the estimated location shift parameter is included.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Patra RK, Sen B (2016). "Estimation of a two-component mixture model with applications to multiple testing." *Journal of the Royal Statistical Society Series B*, **78**(4), 869-893. Bordes L, Delmas C, Vandekerkhove P (2006). "Semiparametric Estimation of a Two-Component Mixture Model Where One Component Is Known." *Scandinavian Journal of Statistics*, **33**(4), 733–752. ISSN 03036898, 14679469, http://www.jstor.org/stable/4616955. Bordes L, Vandekerkhove P (2010). "Semiparametric two-component mixture model with a known component: An asymptotically normal estimator." *Mathematical Methods of Statistics*, **19**(1), 22–41. doi:10.3103/S1066530710010023. Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Two-sample contamination model test." *Bernoulli*, **30**(1), 170–197. doi:10.3150/23BEJ1593.

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 300, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 250, weight = 0.85,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("rate" = 1)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
```

```
# Estimation by different methods:
admix_estim(samples=list(data1), admixMod=list(admixMod1), est_method = "BVdk")
admix_estim(samples=list(data1), admixMod=list(admixMod1), est_method = "PS")
admix_estim(samples=list(data1,data2), admixMod=list(admixMod1,admixMod2), est_method = "PS")
admix_estim(samples=list(data1,data2), admixMod=list(admixMod1,admixMod2), est_method = "IBM")
```

`admix_model` *Define the distribution/parameter(s) of the known component*

Description

Create an object of class 'admix_model', containing the information about the known component distribution in the admixture model. An admixture (aka contamination) model is a two-component mixture model with one known component. Both the second component distribution and the mixing weight are unknown.

Usage

```
admix_model(knownComp_dist, knownComp_param)
```

Arguments

`knownComp_dist`	(Character) The name of the distribution (specified as in R glossary) of the known component of the admixture model
`knownComp_param`	(Character) A vector of the names of the parameters (specified as in R glossary) involved in the chosen known distribution, with their values.

Value

An object of class admix_model, containing 2 attributes: 1) a list that gives the information about the distributions involved in the two-component mixture model (the unknown and the known ones); 2) a list that gives the information about the corresponding parameters of those distributions.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
admix_model(knownComp_dist = "norm", knownComp_param = list("mean"=0, "sd"=1))
admix_model(knownComp_dist = "exp", knownComp_param = list("rate"=2))
admix_model(knownComp_dist = "pois", knownComp_param = list("lambda"=5))
admix_model(knownComp_dist = "multinom", knownComp_param = list("size"=1, "prob"=c(0.2,0.8,0.1)))
```

admix_test *Equality test for the unknown components of admixture models*

Description

Perform hypothesis test between unknown components of a list of admixture models, where we remind that the i-th admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities p_i and f_i are thus estimated, leading to the test given by the following null and alternative hypothesis: H0: f_i = f_j for all i != j against H1 : there exists at least i != j such that f_i differs from f_j. The test can be performed using two methods, either the comparison of coefficients obtained through polynomial basis expansions of the component densities, or by the inner-convergence property obtained using the IBM approach. See 'Details' below for further information.

Usage

```
admix_test(
  samples,
  admixMod,
  test_method = c("poly", "icv"),
  conf_level = 0.95,
  ...
)
```

Arguments

samples	A list of the K (K > 0) samples to be studied, each one assumed to follow a mixture distribution.
admixMod	A list of objects of class admix_model, containing useful information about distributions and parameters of the contamination / admixture models under study.
test_method	The testing method to be applied. Can be either 'poly' (polynomial basis expansion) or 'icv' (inner convergence from IBM). The same testing method is performed between all samples. In the one-sample case, only 'poly' is available and the test is a gaussianity test. For further details, see section 'Details' below.
conf_level	The confidence level of the K-sample test.
...	Depending on the choice made by the user for the test method ('poly' or 'icv'), optional arguments to gaussianity_test or orthobasis_test (in case of 'poly'), and to IBM_k_samples_test in case of 'icv'.

Details

For further details on implemented hypothesis tests, see the references hereafter. .

Value

An object of class 'htest' containing the classical attributes of the latter class, as well as other attributes specific to the inherited object class. Usually, the test decision (reject the null hypothesis or not); the confidence level of the test (1-alpha, where alpha denotes the level of the test or equivalently the type-I error); the number of samples under study; the respective size of each sample; the information about known mixture components.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

References

Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2024). "Contamination-source based K-sample clustering." *Journal of Machine Learning Research*, **25**(287), 1–32. https://jmlr.org/papers/v25/23-0914.html. Milhaud X, Pommeret D, Salhi Y, Vandekerkhove P (2022). "Semiparametric two-sample admixture components comparison test: The symmetric case." *Journal of Statistical Planning and Inference*, **216**, 135-150. ISSN 0378-3758, doi:10.1016/j.jspi.2021.05.010. Pommeret D, Vandekerkhove P (2019). "Semiparametric density testing in the contamination model." *Electronic Journal of Statistics*, 4743–4793. doi:10.1214/19EJS1650.

Examples

```
####### Example with 2 samples
mixt1 <- twoComp_mixt(n = 380, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 350, weight = 0.85,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = -1, "sd" = 1)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
admix_test(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
        conf_level = 0.95, test_method = "poly", ask_poly_param = FALSE, support = "Real")
```

`allGalaxies` *Measurements of heliocentric velocities*

Description

An evolving data frame of velocities for 4 dSph galaxies (namely Carina, Sextans, Sculptor and Fornax), from SIMBAD astronomical database.

Usage

```
allGalaxies
```

Format

Currently contains 8,862 rows and 3 columns, with information on:

Target Target identification; Galaxy-ID number

HV Weighted mean Heliocentric rest frame velocity

Name The name of the galaxy

Source

https://vizier.u-strasbg.fr/viz-bin/VizieR-3?-source=J/AJ/137/3100/stars&-out.max=50&-out.form=HTML%20Table&-out.add=_r&-out.add=_RAJ,_DEJ&-out.add=_RA%2a-c.eq,_DE%2a-c.eq&-sort=_r&-oc.form=sexa

---

decontaminated_density

Probability density function of the unknown component

Description

Estimate the decontaminated density of the unknown component in the admixture model under study, after inversion of the admixture cumulative distribution function. Recall that an admixture model follows the cumulative distribution function (CDF) L, where L = p*F + (1-p)*G, with g a known CDF and p and f unknown quantities.

Usage

```
decontaminated_density(sample1, estim.p, admixMod)
```

Arguments

sample1	Sample under study.
estim.p	The estimated weight, related to the proportion of the unknown component distribution in the admixture model studied.
admixMod	An object of class 'admix_model', containing useful information about known distribution(s) and parameter(s).

Details

The decontaminated density is obtained by inverting the admixture density, given by l = p*f + (1-p)*g, to isolate the unknown component f after having estimated p.

Value

An object of class 'decontaminated_density', containing 3 attributes: 1) the data under study; 2) the type of support for the underlying distribution (either discrete or continuous, useful for plots); 3) the decontaminated density function.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 400, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = -2, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
data1 <- get_mixture_data(mixt1)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1), admixMod = list(admixMod1),
                   est_method = 'PS')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)

####### Discrete support:
mixt1 <- twoComp_mixt(n = 5000, weight = 0.6,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 4000, weight = 0.8,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1, data2),
                   admixMod = list(admixMod1, admixMod2), est_method = 'IBM')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)

####### Finite discrete support:
mixt1 <- twoComp_mixt(n = 12000, weight = 0.6,
```

```
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.6,0.3,0.1))))
mixt2 <- twoComp_mixt(n = 10000, weight = 0.8,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.2,0.6,0.2))))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1, data2),
                   admixMod = list(admixMod1, admixMod2), est_method = 'IBM')
## Determine the decontaminated version of the unknown density by inversion:
decontaminated_density(sample1 = data1, estim.p = est$estimated_mixing_weights[1],
                       admixMod = admixMod1)
```

`detect_support_type` *Detect the type of support of some random variables*

Description

Given one or two sets of observations (samples), the function provides with the most plausible type of support for the underlying random variables to be studied. If less than 3 percents of the observations have different values, we consider that the support is discrete. Otherwise, we consider it as a continuous support.

Usage

```
detect_support_type(sample1, sample2 = NULL)
```

Arguments

`sample1`	The first sample of observations under study.
`sample2`	The second sample of observations under study.

Value

The type of support, either discrete or continuous.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 1500, weight = 0.5,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
data1 <- get_mixture_data(mixt1)
mixt2 <- twoComp_mixt(n = 2000, weight = 0.7,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 5, "sd" = 2)))
data2 <- get_mixture_data(mixt2)
## Test the type of support:
detect_support_type(data1, data2)
```

---

get_cluster_members *Extractor for object of class 'admix_cluster'*

Description

Extract the clusters that were discovered among K samples, where belonging to one given cluster means having equal unknown component distributions.

Usage

```
get_cluster_members(x)
```

Arguments

x	An object of class 'admix_cluster'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

get_cluster_sizes *Extractor for object of class 'admix_cluster'*

Description

Provide the number of samples in each cluster.

Usage

```
get_cluster_sizes(x)
```

Arguments

x	An object of class 'admix_cluster'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

get_discrepancy_id *Extractor for object of class 'IBM_test'*

Description

Provide the matrix storing the identifiers of discrepancies using Inversion-Best Matching approach between all couples among the K samples under study.

Usage

```
get_discrepancy_id(x)
```

Arguments

x	An object of class 'IBM_test'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

get_discrepancy_matrix *Extractor for object of class 'IBM_test' or 'admix_cluster'*

Description

Provide the matrix storing discrepancies using Inversion-Best Matching approach between all couples among the K samples under study.

Usage

```
get_discrepancy_matrix(x)
```

Arguments

x	An object of class 'IBM_test' or 'admix_cluster'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_discrepancy_rank *Extractor for object of class 'IBM_test'*

Description

Provide the matrix storing the ranks of discrepancies using Inversion-Best Matching approach between all couples among the K samples under study.

Usage

```
get_discrepancy_rank(x)
```

Arguments

x	An object of class 'IBM_test'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_known_component *Extractor for objects of class 'admix_estim', 'htest' or 'admix_cluster'*

Description

Get the known component(s) of admixture model(s) considered for estimation, test, or clustering.

Usage

```
get_known_component(x)
```

Arguments

x	An object of class 'admix_estim', 'htest', or 'admix_cluster'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_mixing_weights *Extractor for objects of class 'admix_estim' or 'htest'*

Description

Get the estimated unknown mixing proportion(s) related to the weight(s) of the unknown component distribution(s) of the admixture model(s).

Usage

```
get_mixing_weights(x)
```

Arguments

x	An object of class 'admix_estim' or 'htest'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_mixture_data *Extractor for object of class 'twoComp_mixt'*

Description

Get the mixture data generated from method 'twoComp_mixt'.

Usage

```
get_mixture_data(x)
```

Arguments

x	An object of class 'twoComp_mixt'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_statistic_components *Extractor for object of class 'IBM_test'*

Description

Provide the terms (or discrepancies) that compose the test statistic for the k-sample test.

Usage

```
get_statistic_components(x)
```

Arguments

x	An object of class 'IBM_test'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_tabulated_dist *Extractor for object of class 'IBM_test'*

Description

Provide the tabulated distribution that allows to define the extreme quantile against which the test statistic is compared.

Usage

```
get_tabulated_dist(x)
```

Arguments

x	An object of class 'IBM_test'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

`is_equal_knownComp` *Equality of known components in two admixture models*

Description

Test if the known component distributions coming from two admixture models are identical.

Usage

```
is_equal_knownComp(admixMod1, admixMod2)
```

Arguments

`admixMod1`	An object of class 'admix_model' related to the first admixture model.
`admixMod2`	An object of class 'admix_model' related to the second admixture model.

Value

A boolean (TRUE if the known components are the same, otherwise FALSE).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
admixMod1 <- admix_model(knownComp_dist = "norm",
                         knownComp_param = list("mean"=0, "sd"=1))
admixMod2 <- admix_model(knownComp_dist = "norm",
                         knownComp_param = list("mean"=0, "sd"=1))
is_equal_knownComp(admixMod1, admixMod2)

admixMod1 <- admix_model(knownComp_dist = "multinom",
                         knownComp_param = list("size"=1, "prob"=c(0.2,0.5,0.3)))
admixMod2 <- admix_model(knownComp_dist = "multinom",
                         knownComp_param = list("size"=1, "prob"=c(0.2,0.4,0.4)))
is_equal_knownComp(admixMod1, admixMod2)
```

milkyWay *Dataset of heliocentric velocity for the Milky Way*

Description

Dataset of heliocentric velocity for the Milky Way

Usage

```
milkyWay
```

Format

A data frame with 170,601 rows and 1 column:

V1 Heliocentric velocity measurements of the Milky way

Source

https://www.aanda.org/articles/aa/full_html/2018/08/aa32905-18/aa32905-18.html

References

Walker MG, Mateo M, Olszewski EW, Gnedin OY, Wang X, Sen B, Woodroofe M (2007). "Velocity Dispersion Profiles of Seven Dwarf Spheroidal Galaxies." *The Astrophysical Journal*, **667**(1), L53–L56. ISSN 1538-4357, doi:10.1086/521998, http://dx.doi.org/10.1086/521998.

mortality_sample *Dataset of deaths statistics in 11 european countries*

Description

Dataset of deaths statistics in 11 european countries

Usage

```
mortality_sample
```

Format

Dataset providing the exposure-to-death (population size) and number of deaths for males in 11 european countries, between 1908 and 2020, with ages ranging from 30 years old to 85 years old. Exported from the Human Mortality Database (HMD). The two first lists relate to some subsample of the population size and number of deaths in those countries, with random sampling from the original dataset.

An evolving data frame of exposure-to-death and number of deaths in Belgium, Switzerland, Denmark, Spain, Finland, France, United Kingdom, Italia, The Netherlands, Norway and Sweden.

XP A list of eleven elements (one for each country) giving a subset of the exposure-to-death (or reduced population size), each element having 56 rows (ages 30-85) and 113 columns (period 1908-2020)

DX A list of eleven elements (one for each country) giving a subset of the number of deaths, each element having 56 rows (ages 30-85) and 113 columns (period 1908-2020)

names A list of eleven elements giving the names of the countries, in the same order as the elements in other lists

Source

https://www.mortality.org

`plot.admix_model` *Plot method for objects of class 'admix_model'*

Description

Plots the probability density function of the known component of the admixture model, where we recall that an admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities are therefore p_i and f_i.

Usage

```
## S3 method for class 'admix_model'
plot(x, ...)
```

Arguments

`x`	An object of class 'admix_model'.
`...`	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
plot(admix_model(knownComp_dist = "norm", knownComp_param = list("mean"=0, "sd"=1)))
plot(admix_model(knownComp_dist = "pois", knownComp_param = list("lambda"=1.5)))
```

`plot.decontaminated_density` *Plot method for object of class 'decontaminated_density'*

Description

Plot the decontaminated density of the unknown component from some admixture model, after inversion of the admixture cumulative distribution functions.

Usage

```
## S3 method for class 'decontaminated_density'
plot(x, x_val, add_plot = FALSE, ...)
```

Arguments

`x`	An object of class 'decontaminated_density' (see ?decontaminated_density).
`x_val`	(numeric) A vector of points at which to evaluate the probability mass/density function.
`add_plot`	(default to FALSE) A boolean specifying if one plots the decontaminated density over an existing plot. Used for visual comparison purpose.
`...`	Arguments to be passed to generic method 'plot', such as graphical parameters (see ?par).

Details

The decontaminated density is obtained by inverting the admixture density, given by l = p*f + (1-p)*g, to isolate the unknown component f after having estimated p and l.

Value

The plot of the decontaminated density.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
####### Continuous support:
## Simulate mixture data:
mixt1 <- twoComp_mixt(n = 400, weight = 0.4,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 0, "sd" = 1)))
mixt2 <- twoComp_mixt(n = 350, weight = 0.6,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean" = 3, "sd" = 0.5),
                                        list("mean" = 5, "sd" = 2)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2),
                   admixMod = list(admixMod1,admixMod2), est_method = 'PS')
prop <- get_mixing_weights(est)
## Determine the decontaminated version of the unknown density by inversion:
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1],
                               admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2],
                               admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from = 0, to = 6, length.out = 100), add_plot = FALSE)
plot(x = res2, x_val = seq(from=0, to=6, length.out=100), col="red", add_plot=TRUE)

####### Countable discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.7,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 3500, weight = 0.85,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2),
                   admixMod = list(admixMod1,admixMod2), est_method = "IBM")
prop <- get_mixing_weights(est)
## Determine the decontaminated version of the unknown density by inversion:
```

```
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1],
                               admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2],
                               admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from = 0, to = 15, by = 1), add_plot = FALSE)
plot(x = res2, x_val = seq(from = 0, to = 15, by = 1), add_plot = TRUE, col = "red")

####### Finite discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.7,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.6,0.3,0.1))))
mixt2 <- twoComp_mixt(n = 3500, weight = 0.85,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size" = 1, "prob" = c(0.3,0.4,0.3)),
                                        list("size" = 1, "prob" = c(0.2,0.6,0.2))))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
## Define the admixture models:
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
admixMod2 <- admix_model(knownComp_dist = mixt2$comp.dist[[2]],
                         knownComp_param = mixt2$comp.param[[2]])
## Estimation:
est <- admix_estim(samples = list(data1,data2),
                   admixMod = list(admixMod1,admixMod2), est_method = "IBM")
prop <- get_mixing_weights(est)
## Determine the decontaminated version of the unknown density by inversion:
res1 <- decontaminated_density(sample1 = data1, estim.p = prop[1],
                               admixMod = admixMod1)
res2 <- decontaminated_density(sample1 = data2, estim.p = prop[2],
                               admixMod = admixMod2)
## Use appropriate sequence of x values:
plot(x = res1, x_val = seq(from=1, to=3, by=1), add_plot = FALSE)
plot(x = res2, x_val = seq(from=1, to=3, by=1), add_plot = TRUE, col = "red")
```

`plot.twoComp_mixt` *Plots several mixture densities on the same graph*

Description

Plots the empirical densities of the samples with optional arguments to improve the visualization.

Usage

```
## S3 method for class 'twoComp_mixt'
plot(x, add_plot = FALSE, ...)
```

Arguments

x	Object of class 'twoComp_mixt' from which the density will be plotted.
add_plot	(default to FALSE) Option to plot another mixture distribution on the same graph.
...	further classical arguments and graphical parameters for methods plot and hist.

Value

a plot with the densities of the samples provided as inputs.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

print.admix_cluster *Print method for object of class 'admix_cluster'*

Description

Print the main results when clustering the unknown component distributions coming from various admixture samples, i.e. the obtained clusters.

Usage

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

Arguments

x	An object of class 'admix_cluster' (see ?admix_clustering).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.admix_estim *Print method for object of class 'admix_estim'*

Description

Print method for object of class 'admix_estim'

Usage

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

Arguments

x	An object of class 'admix_estim' (see ?admix_estim).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.admix_model *Print method for objects of class 'admix_model'*

Description

Print an object of class 'admix_mod'. An admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities are therefore p_i and f_i.

Usage

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

Arguments

x	An object of class 'admix_model'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.decontaminated_density *Print method for object of class 'decontaminated_density'*

Description

Print some overview of the decontaminated density function.

Usage

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

Arguments

x	An object of class 'decontaminated_density' (see ?decontaminated_density).
...	Arguments to be passed to generic method 'plot', such as graphical parameters (see par).

Value

The function related to the estimated decontaminated density.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

print.twoComp_mixt *Print method for objects 'twoComp_mixt'*

Description

Print an object of class 'twoComp_mixt'. A two-component mixture model has probability density function (pdf) l such that: l = p * f + (1-p) * g, where p is the mixing proportion, and f and g are the component distributions.

Usage

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

Arguments

x	An object of class 'twoComp_mixt'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

reject_nullHyp *Extractor for object of class 'htest'*

Description

Provide the decision of the statistical test: reject or do not reject the null hypothesis.

Usage

```
reject_nullHyp(x)
```

Arguments

x	An object of class 'htest'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

stmf_small *Dataset of Short-term Mortality Fluctuations (STMF) from HMD*

Description

Restricted to 6 countries: Belgium, France, Italy, Netherlands, Spain, Germany. Weekly death counts provide the most objective and comparable way of assessing the scale of short-term mortality elevations across countries and time. Extraction date from the Human Mortality Database (HMD): 09/21/2020.

Usage

```
stmf_small
```

Format

A data frame with 88146 rows and 19 variables:

CountryCode Mortality database country code

Year Year

Week Week number

Sex Gender ('m': male, 'f': female, 'b': both)

D0_14 Age range 0-14

D15_64 Age range 15-64

D65_74 Age range 65-74

D75_84 Age range 75-84

D85p Age range 85-+

DTotal Count of deaths for all ages combined

R0_14 Crude death rate for age range 0-14

R15_64 Crude death rate for age range 15-64

R65_74 Crude death rate for age range 65-74

R75_84 Crude death rate for age range 75-84

R85p Crude death rate for age range 85-+

RTotal Crude death rate for all ages combined

Split Indicates if data were split from aggregated age groups (0 if the original data has necessary detailed age scale). For example, if the original age scale was 0-4, 5-29, 30-65, 65+, then split will be equal to 1

SplitSex Indicates if the original data are available by sex (0) or data are interpolated (1)

Forecast Equals 1 for all years where forecasted population exposures were used to calculate weekly death rates

Source

https://www.mortality.org

`summary.admix_cluster` *Summary method for object of class 'admix_cluster'*

Description

Summarizes the results obtained when clustering the unknown component distributions coming from various admixture samples.

Usage

```
## S3 method for class 'admix_cluster'
summary(object, ...)
```

Arguments

`object`	An object of class 'admix_cluster' (see ?admix_clustering).
`...`	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.admix_estim *Summary method for object of class 'admix_estim'*

Description

Summarize the estimated weight(s) of the unknown component(s), and admixture model(s) under study. Recall that an admixture model follows the cumulative distribution function (CDF) L, where L = p*F + (1-p)*G, with G a known CDF, and p and F unknown quantities.

Usage

```
## S3 method for class 'admix_estim'
summary(object, ...)
```

Arguments

object	An object of class 'admix_estim' (see ?admix_estim).
...	further arguments passed to or from other methods.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.admix_model *Summary method for objects of class 'admix_model'*

Description

Summarizes the information related to the known component of the two-component mixture. An admixture model has probability density function (pdf) l_i such that: l_i = p_i * f_i + (1-p_i) * g_i, with g_i the known component density. The unknown quantities are therefore p_i and f_i.

Usage

```
## S3 method for class 'admix_model'
summary(object, ...)
```

Arguments

object	An object of class 'admix_model'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.decontaminated_density *Summary method for object of class 'decontaminated_density'*

Description

Summarizes information about the estimated decontaminated density function.

Usage

```
## S3 method for class 'decontaminated_density'
summary(object, ...)
```

Arguments

object	An object of class 'decontaminated_density' (see ?decontaminated_density).
...	Arguments to be passed to generic method 'summary'.

Value

Classical statistical indicators about the decontaminated density.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

summary.twoComp_mixt *Summary method for objects 'twoComp_mixt'*

Description

Provides statistical indicators of an object of class 'twoComp_mixt'. A two-component mixture model has probability density function (pdf) l such that: l = p * f + (1-p) * g, where p is the mixing proportion, and f and g are the component distributions.

Usage

```
## S3 method for class 'twoComp_mixt'
summary(object, ...)
```

Arguments

object	An object of class 'twoComp_mixt'.
...	A list of additional parameters belonging to the default method.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

`twoComp_mixt` *Simulation of a two-component mixture model*

Description

Simulate a two-component mixture model following the probability density function (pdf) l such that l = p*f + (1-p)*g, with f and g the mixture component distributions, and p the mixing weight.

Usage

```
twoComp_mixt(
  n = 1000,
  weight = 0.5,
  comp.dist = list("norm", "norm"),
  comp.param = list(list(mean = 0, sd = 1), list(mean = 2, sd = 1))
)
```

Arguments

`n`	Number of observations to be simulated.
`weight`	Weight of the first component distribution (distribution f) in the mixture.
`comp.dist`	A list of two elements corresponding to the component distributions (with available names listed in object 'Distribution.df' in package EnvStats) involved in the mixture model. These elements respectively refer to the two component distributions f and g. By convention, in the framework of admixture models where one of the two components is unknown, the first element of the list corresponds to the 'unknown' component distribution, whereas the second one refers to the known one.
`comp.param`	A list of two elements corresponding to the parameters of the component distributions, each element being a list itself. The names used in each list must correspond to the available parameters listed in object 'Distribution.df' in package EnvStats. These elements respectively refer to the parameters of f and g distributions of the mixture model. By convention, in the framework of admixture models where one of the two components is unknown, the first element of the list corresponds to the 'unknown' component parameters, whereas the second one refers to the known ones.

Value

An object of class twoComp_mixt, containing eight attributes: 1) the number of simulated observations, 2) the simulated mixture data, 3) the support of the distributions, 4) the name of the component distributions, 5) the name of the parameters of the component distributions and their values, 6) the mixing proportion, 7) the observations coming from the first component, 8) the observations coming from the second component.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Mixture of continuous random variables:
sim.X <- twoComp_mixt(n = 2000, weight = 0.5,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean"=3, "sd"=0.5),
                                        list("mean"=0, "sd"=1)))
print(sim.X)
sim.Y <- twoComp_mixt(n = 1200, weight = 0.7,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean"=-3, "sd"=0.5),
                                        list("rate"=1)))
plot(sim.X, xlim=c(-5,5), ylim=c(0,0.5))
plot(sim.Y, add_plot = TRUE, xlim=c(-5,5), ylim=c(0,0.5), col = "red")

## Mixture of discrete random variables:
sim.X <- twoComp_mixt(n = 2000, weight = 0.5,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size"=1, "prob"=c(0.3,0.4,0.3)),
                                        list("size"=1, "prob"=c(0.1,0.2,0.7))))
plot(sim.X)
```

---

which_rank *Extractor for object of class 'htest'*

Description

Provide the selected rank of the test statistic (connected to the expansion order of the densities in the orthonormal polynomial basis if method 'poly' was chosen; or to the number of terms, i.e. discrepancies between couples of samples, included in the test statistic).

Usage

```
which_rank(x)
```

Arguments

x	An object of class 'htest'.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Index