Package 'admix'

May 30, 2026

Title Package Admix for Admixture (aka Contamination) Models

Version 2.6.1

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 7.3.3

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

Suggests doParallel, doRNG, evd, flexsurv, foreach, gridExtra, knitr, lattice, logitnorm, plyr, reshape2, rmarkdown, rmutil, mockery, 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 2026-05-30 05:50:08 UTC

Contents

admix_cluster . . . 3
admix_estim . . . 5
admix_model . . . 7
admix_test . . . 8
allGalaxies . . . 10
decontaminated_density . . . 11
detect_support_type . . . 13
distribution_type . . . 14
get_cluster_members . . . 15
get_cluster_sizes . . . 16
get_discrepancy_matrix . . . 16
get_discrepancy_rank . . . 17
get_distribution_parameters . . . 18
get_known_component . . . 18
get_mixing_weights . . . 19
get_mixture_data . . . 20
get_statistic_components . . . 21
get_tabulated_dist . . . 21
is_equal_knownComp . . . 22
milkyWay . . . 23
mortality_sample . . . 24
plot.admix_model . . . 24
plot.decontaminated_density . . . 25
plot.twoComp_mixt . . . 27
print.admix_cluster . . . 29
print.admix_estim . . . 29
print.admix_model . . . 30
print.decontaminated_density . . . 30
print.twoComp_mixt . . . 31
reject_nullHyp . . . 31

stmf_small 32
summary.admix_cluster 33
summary.admix_estim 34
summary.admix_model 34
summary.decontaminated_density 35
summary.twoComp_mixt 35
twoComp_mixt 36
validate_distribution 37
which_rank 38

Index **39**

`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 named list of the K (K>1) samples to be studied, all following admixture distributions. If names are provided, they are used in the output to identify samples; otherwise default labels are used.
`admixMod`	A list of objects of class admix_model, with information about known distributions and known 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 14 attributes: 1) the number of samples under study; 2) the names of samples; 3) the sizes of samples; 4) the information about mixture components in each sample (distributions and parameters); 5) the number of detected clusters; 6) the list of p-values for each k-sample test at the origin of detected clusters; 7) the cluster affiliation for each sample; 8) the confidence level of statistical tests; 9) which samples in which cluster; 10) the size of clusters; 11) 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); 12) the matrix of pairwise discrepancies across all samples; 13) the list of tabulated distributions used for statistical tests involved in building the clusters; 14) the call.

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.

See Also

print.admix_cluster(), summary.admix_cluster(), get_known_component(), get_cluster_members(), get_cluster_sizes(), get_tabulated_dist() to access the tabulated distribution under the null hypothesis, which defines the quantile against which the test statistics is tested; get_discrepancy_matrix() for a pairwise measure of discrepancy between samples.

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 an admixture model*

Description

Estimate the unknown component weight (and possibly a location shift parameter in case of a symmetric unknown component density), using different estimation techniques. We recall that the i-th admixture model has probability density function ℓ_i such that:

$$\ell_i = p_i f_i + (1 - p_i) g_i,$$

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, with information about known distributions and known 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). When selecting 'BVdk' estimation method, the initialization parameters for the optimization process have been arbitrarily set. The mixing proportion is fixed to 0.5. For the localization parameter, it is based on taking the first moment in the model ($\ell(x) = pf(x-\mu) + (1-p)g(x)$), and isolating μ by an inversion formula.

Value

An object of class `estim_BVdk`, `estim_PS` or `estim_IBM` (that inherits from class admix_estim), with two attributes, 'class' and 'names'. The latter contains three elements, among which 'estim_objects' that lists for each sample under study all the information of the estimation procedure.

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.

See Also

`get_mixing_weights()` to access the estimated mixing weight(s), `get_known_component()` to access the known component(s), `print.admix_estim()` for a brief description of the results, and `summary.admix_estim()` for an overview of the estimation process. More precisely, 1) the number of samples under study; 2) the information about the known 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.

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)))
mixt3 <- twoComp_mixt(n = 500, weight = 0.5,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 2),
                                        list("lambda" = 7)))
mixt4 <- twoComp_mixt(n = 1500, weight = 0.2, comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size"=1, "prob" = c(0.8,0.1,0.1)),
                                        list("size"=1, "prob" = c(0.1,0.2,0.7))))
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]])
# Estimation by different methods:
admix_estim(samples = list(data1), admixMod = list(admixMod1), est_method = "BVdk")
admix_estim(samples = list(data1, data2, data3, data4),
          admixMod = list(admixMod1, admixMod2, admixMod3, admixMod4), 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	The name of the distribution (specified as in R glossary) of the known component of the admixture model.
knownComp_param	A list 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(0.1,0.8,0.1)))
```

---

admix_test *Hypothesis test for the unknown component(s) in admixture model(s)*

Description

Perform hypothesis test on the unknown component(s) of a list of admixture model(s), where we remind that the i-th admixture model has probability density function (pdf) ℓ_i such that:

$$\ell_i = p_i f_i + (1 - p_i) g_i,$$

with g_i the known component density, and where ℓ_i can be estimated consistently thanks to the observations. The test is made on the f_i's, 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, with information about known distributions and known parameters.
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, orthobasis_test (in case of 'poly'), or IBM_k_samples_test in case of 'icv'.

Details

For further details on implemented hypothesis tests, see the references hereafter. When choosing the 'icv' testing method, it is recommended to use parallel computing.

Value

An object of class `gaussianity_test`, `orthobasis_test`, or `IBM_test` (that inherits from class `htest`), containing attributes specific to the object class (in addition to classical attributes from `htest`). 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). "Semi-parametric 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.

See Also

gaussianity_test(), orthobasis_test(), IBM_k_samples_test(), get_known_component(), get_mixing_weights(), reject_nullHyp(), which_rank()

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

Estimates the decontaminated probability density function (pdf) of the unknown component in an admixture model, based on the inversion of the admixture density equation

$$\ell = pf + (1 - p)g,$$

where p and f are unknown, ℓ is observed and g is the known component.

Usage

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

Arguments

`sample1`	Numeric vector, sample under study.
`admixMod`	An object of class admix_model, with information about the known distribution and associated known parameter(s).
`estim.p`	Numeric. The estimated mixing proportion $\hat{p}$ of the unknown component.

Details

The decontaminated density f is computed as:

$$f(x) = (1/\hat{p})[\hat{\ell}(x) - (1 - \hat{p})g(x)]$$

where:

- $\hat{\ell}(x)$ is the empirical density of the sample,
- $g(x)$ is the known component's theoretical density,
- $\hat{p}$ is the estimated mixture weight.

For continuous data, $\hat{\ell}(x)$ is estimated using kernel density estimation. For discrete data, it is approximated from normalized frequencies.

Value

An object of class `decontaminated_density` containing:

data Original sample.

support Type of support ("Continuous" or "Discrete").

decontaminated_density_fun A function returning the estimated decontaminated density.

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:
x <- decontaminated_density(sample1 = data1, admixMod = admixMod1,
                            estim.p = get_mixing_weights(est))
print(x)
summary(x)

####### Discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.6,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 3000, weight = 0.8,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
mixt3 <- twoComp_mixt(n = 1500, weight = 0.5,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 7),
                                        list("lambda" = 1)))
data1 <- get_mixture_data(mixt1)
data2 <- get_mixture_data(mixt2)
data3 <- get_mixture_data(mixt3)
## 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]])
## Estimation:
est <- admix_estim(samples = list(data1,data2),
                   admixMod = list(admixMod1,admixMod2), est_method = 'IBM')
est2 <- admix_estim(samples = list(data3), admixMod = list(admixMod3), est_method = 'PS')
## Determine the decontaminated version of the unknown density by inversion:
x <- decontaminated_density(sample1 = data1, admixMod = admixMod1,
                            estim.p = get_mixing_weights(est)[1])
y <- decontaminated_density(sample1 = data2, admixMod = admixMod2,
                            estim.p = get_mixing_weights(est)[2])
z <- decontaminated_density(sample1 = data3, admixMod = admixMod3,
                            estim.p = get_mixing_weights(est2))
print(x)
summary(y)
print(z)
```

`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 there are too many duplicated 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)
```

---

distribution_type *Determine the type of distribution under consideration*

Description

Determine the type of distribution under consideration

Usage

```
distribution_type(dist)
```

Arguments

dist	A single string naming the distribution under consideration.

Value

The type of distribution under study (continuous, discrete, or multivariate).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
distribution_type("norm")
distribution_type("pois")
distribution_type("multinom")
distribution_type("weibull")
```

`get_cluster_members` *Extractor for members of clusters*

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`.

Value

The samples included in each detected cluster.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

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:
x <- 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)
get_cluster_members(x)
get_cluster_sizes(x)
```

`get_cluster_sizes` *Extractor for cluster sizes*

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_matrix` *Extractor for discrepancies b/w unknown components*

Description

Provide the matrix storing pairwise discrepancies b/w unknown components in admixture models, using Inversion-Best Matching approach.

Usage

```
get_discrepancy_matrix(x)
```

Arguments

x	An object of class IBM_test or admix_cluster.

Value

A matrix of pairwise discrepancies among the K (K>2) samples under study.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

get_discrepancy_rank *Extractor for pairwise discrepancy rankings*

Description

Provide the matrix storing the ranks of discrepancies using Inversion-Best Matching approach between all couples among the K (K>2) samples under study.

Usage

```
get_discrepancy_rank(x)
```

Arguments

x	An object of class IBM_test.

Value

A matrix of ranks, from the closest couple (rank 1) in terms of discrepancy measure to the most different one.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

get_distribution_parameters *Check the validity of the specified distributions*

Description

Check the validity of the specified distributions

Usage

```
get_distribution_parameters(dist)
```

Arguments

dist	A single string naming the distribution under consideration.

Value

A vector composed of the names of the parameters of the distribution.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
get_distribution_parameters("norm")
get_distribution_parameters("gamma")
get_distribution_parameters("weibull")
```

get_known_component *Extractor for known component(s) in admixture model(s)*

Description

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

Usage

```
get_known_component(x)
```

Arguments

x	An object of class admix_estim, gaussianity_test, orthobasis_test, IBM_test, or admix_cluster.

Details

This is a generic extractor, providing with the same information whatever the object class.

Value

A list providing information on the known component (distribution, parameters).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Simulate a two-component Gaussian mixture:
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)))
data1 <- get_mixture_data(mixt1)
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
## Estimate the unknown quantities:
x <- admix_estim(samples = list(data1), admixMod = list(admixMod1), est_method = "BVdk")
## Extract the information about the known component:
get_known_component(x)
```

---

get_mixing_weights *Extractor for estimated mixing weights*

Description

Extracts the estimated mixing weights from fitted objects of class `admix_estim`, `gaussianity_test` and `orthobasis_test`.

Usage

```
get_mixing_weights(x)
```

Arguments

`x`	An object of class `admix_estim`, `gaussianity_test` or `orthobasis_test`.

Details

This is a generic extractor. The exact behavior depends on the class of the input object:

- `admix_estim`: returns the estimated mixture proportions.
- `gaussianity_test`, `orthobasis_test`: returns weights derived from hypothesis testing results.

Value

A numeric vector of estimated mixing weight(s).

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
## Simulate a two-component Gaussian mixture:
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)))
data1 <- get_mixture_data(mixt1)
admixMod1 <- admix_model(knownComp_dist = mixt1$comp.dist[[2]],
                         knownComp_param = mixt1$comp.param[[2]])
## Estimate the unknown quantities:
x <- admix_estim(samples = list(data1), admixMod = list(admixMod1), est_method = "BVdk")
## Extract the information about the known component:
get_mixing_weights(x)
```

---

get_mixture_data *Extractor for simulated data from two-component mixture*

Description

Get the mixture data generated from method `twoComp_mixt()`.

Usage

```
get_mixture_data(x)
```

Arguments

`x`	An object of class `twoComp_mixt`.

Value

A numeric vector of the simulated data.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
sim.X <- twoComp_mixt(n = 20, weight = 0.5,
                      comp.dist = list("norm", "norm"),
                      comp.param = list(list("mean"=3, "sd"=0.5),
                                        list("mean"=0, "sd"=1)))
get_mixture_data(sim.X)
```

---

get_statistic_components *Extractor for components involved in test statistic*

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.

Value

The components finally included in the test statistic, i.e. the discrepancies of the couples that were aggregated in the built sequence of statistics.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

---

get_tabulated_dist *Extractor for tabulated distribution in the k-sample test*

Description

Provide (the list of) tabulated distribution(s) that allow to define the extreme quantile(s) against which the test statistic(s) is compared.

Usage

```
get_tabulated_dist(x)
```

Arguments

x	An object of class IBM_test or admix_cluster.

Value

A numeric vector containing the simulated values of the tabulated distribution, sorted in increasing order.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
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]])
x <- admix_test(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
                conf_level = 0.95, test_method = "icv", n_sim_tab = 10)
get_tabulated_dist(x)
```

---

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.

Usage

```
## S3 method for class 'admix_model'
plot(x, n = 1000, main = "Known component distribution", ...)
```

Arguments

x	An object of class admix_model.
n	The number of 'x' values to consider for plotting the pdf in the continuous case.
main	The title of the plot.
...	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 = NULL,
  add_plot = FALSE,
  offset = 0,
  bar_width = 0.3,
  main = "pdf of the unknown component",
  ...
)
```

Arguments

x	An object of class decontaminated_density (see ?decontaminated_density).
x_val	Values at which to evaluate the decontaminated density.
add_plot	Boolean, TRUE when a new plot is added to the existing one.
offset	Numeric. Position of the bars relative to the labels on the x-axis.
bar_width	Width of bars to be plotted.

main	The title of the plot (typically 'probability density function of the unknown component', or 'probability mass function of the unknown component').
...	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 if one sample is provided, or the comparison of decontaminated densities plotted on the same graph in the case of multiple samples.

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:
x <- decontaminated_density(sample1 = data1, admixMod = admixMod1,
                            estim.p = get_mixing_weights(est))
plot(x)

####### Discrete support:
mixt1 <- twoComp_mixt(n = 4000, weight = 0.6,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 2)))
mixt2 <- twoComp_mixt(n = 3000, weight = 0.8,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 3),
                                        list("lambda" = 4)))
mixt3 <- twoComp_mixt(n = 1500, weight = 0.5,
                      comp.dist = list("pois", "pois"),
                      comp.param = list(list("lambda" = 7),
                                        list("lambda" = 1)))
data1 <- get_mixture_data(mixt1)
```

```
data2 <- get_mixture_data(mixt2)
data3 <- get_mixture_data(mixt3)
## 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]])
## Estimation:
est <- admix_estim(samples = list(data1,data2),
                   admixMod = list(admixMod1,admixMod2), est_method = 'IBM')
est2 <- admix_estim(samples = list(data3), admixMod = list(admixMod3), est_method = 'PS')
## Determine the decontaminated version of the unknown density by inversion:
x <- decontaminated_density(sample1 = data1, admixMod = admixMod1,
                            estim.p = get_mixing_weights(est)[1])
y <- decontaminated_density(sample1 = data2, admixMod = admixMod2,
                            estim.p = get_mixing_weights(est)[2])
z <- decontaminated_density(sample1 = data3, admixMod = admixMod3,
                            estim.p = get_mixing_weights(est2))
plot(x, offset = -0.2, bar_width = 0.2, main = "pmf of the unknown component", col = "steelblue")
plot(y, add_plot = TRUE, offset = 0, bar_width = 0.2, col = "red")
plot(z, add_plot = TRUE, offset = 0.2, bar_width = 0.2, col = "orange")
legend("topright", legend = c("data1","data2","data3"), col = c("steelblue","red","orange"),
       lty = rep(1,3), lwd = rep(2,3), bty = "n")
```

`plot.twoComp_mixt` *Plot the empirical mixture pdf*

Description

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

Usage

```
## S3 method for class 'twoComp_mixt'
plot(
  x,
  add_plot = FALSE,
  offset = 0,
  bar_width = 0.2,
  main = "Mixture distribution (density or mass function)",
  ...
)
```

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.
offset	Numeric. Position of the bars relative to the labels on the x-axis.
bar_width	Width of bars to be plotted.
main	The title of the plot.
...	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

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)))
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")
legend("topright", legend = c("sim.X","sim.Y"), col = c("black","red"),
       lty = rep(1,2), bty = "n")

## 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))))
sim.Y <- twoComp_mixt(n = 1800, 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(0.6,0.2,0.2))))
sim.Z <- twoComp_mixt(n = 1800, weight = 0.3,
                      comp.dist = list("multinom", "multinom"),
                      comp.param = list(list("size"=1, "prob"=c(0.2,0.1,0.7)),
                                        list("size"=1, "prob"=c(1/3,1/3,1/3))))
plot(sim.X, offset = -0.05, bar_width = 0.05, col = "steelblue")
plot(sim.Y, add_plot = TRUE, offset = 0, bar_width = 0.05, col = "orange")
plot(sim.Z, add_plot = TRUE, offset = +0.05, bar_width = 0.05, col = "red")
legend("topleft", legend = c("sim.X","sim.Y","sim.Z"), col = c("steelblue","orange","red"),
```

```
        lty = rep(1,3), bty = "n")
```

`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 the information about the distribution of the known component of an admixture model, as well as the known parameters for this distribution.

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 the test decision*

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 gaussianity_test, orthobasis_test or IBM_test.

Value

A boolean giving the result of the test, TRUE if the null hypothesis is rejected, otherwise FALSE.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
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]])
x <- admix_test(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
          conf_level = 0.95, test_method = "poly", ask_poly_param = FALSE, support = "Real")
reject_nullHyp(x)
```

---

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.

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

$$\ell = pf + (1 - p)g,$$

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

See Also

`get_mixture_data()` to access the simulated mixture data.

Examples

```
## Mixture of continuous random variables:
sim.X <- twoComp_mixt(n = 1200, weight = 0.7,
                      comp.dist = list("norm", "exp"),
                      comp.param = list(list("mean"=-3, "sd"=0.5),
                                        list("rate"=1)))
print(sim.X)
data.X <- get_mixture_data(sim.X)
plot(density(data.X))

## Mixture of discrete random variables:
sim.Y <- twoComp_mixt(n = 1800, 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(0.6,0.2,0.2))))
print(sim.Y)
```

---

validate_distribution *Check the validity of the specified distribution and parameter(s)*

Description

Check the validity of the specified distribution and parameter(s)

Usage

```
validate_distribution(dist, params)
```

Arguments

`dist`	A single string naming the distribution under consideration.
`params`	A named list of parameters for the distribution under consideration.

Value

A vector composed of the names of the parameters of the distribution.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
validate_distribution(dist = "norm", params = c("mean" = 0, "sd" = 1))
```

---

which_rank *Extractor for the selected rank in the test statistic*

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 with method 'icv').

Usage

```
which_rank(x)
```

Arguments

x	An object of class `gaussianity_test`, `orthobasis_test` or `IBM_test`.

Value

An integer corresponding to the selected rank in the test statistics, i.e. how many terms were kept in the test statistic.

Author(s)

Xavier Milhaud xavier.milhaud.research@gmail.com

Examples

```
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]])
x <- admix_test(samples = list(data1,data2), admixMod = list(admixMod1,admixMod2),
           conf_level = 0.95, test_method = "poly", ask_poly_param = FALSE, support = "Real")
which_rank(x)
```

Index

∗ **datasets**
allGalaxies, 10
milkyWay, 23
mortality_sample, 24
stmf_small, 32

admix_cluster, 3, *4*
admix_estim, 5, *6*
admix_model, *3*, *5*, 7, *8*, *9*, *11*
admix_test, *6*, 8
allGalaxies, 10

decontaminated_density, 11
detect_support_type, 13
distribution_type, 14

estim_BVdk, *6*
estim_IBM, *6*
estim_PS, *6*

gaussianity_test, *9*
gaussianity_test(), *10*
get_cluster_members, 15
get_cluster_members(), *4*
get_cluster_sizes, 16
get_cluster_sizes(), *4*
get_discrepancy_matrix, 16
get_discrepancy_matrix(), *4*
get_discrepancy_rank, 17
get_distribution_parameters, 18
get_known_component, 18
get_known_component(), *4*, *6*, *10*
get_mixing_weights, 19
get_mixing_weights(), *6*, *10*
get_mixture_data, 20
get_mixture_data(), *37*
get_statistic_components, 21
get_tabulated_dist, 21
get_tabulated_dist(), *4*

IBM_k_samples_test, *4*, *9*
IBM_k_samples_test(), *10*
is_equal_knownComp, 22

milkyWay, 23
mortality_sample, 24

orthobasis_test, *9*
orthobasis_test(), *10*

plot.admix_model, 24
plot.decontaminated_density, 25
plot.twoComp_mixt, 27
print.admix_cluster, 29
print.admix_cluster(), *4*
print.admix_estim, 29
print.admix_estim(), *6*
print.admix_model, 30
print.decontaminated_density, 30
print.twoComp_mixt, 31

reject_nullHyp, 31
reject_nullHyp(), *10*

stmf_small, 32
summary.admix_cluster, 33
summary.admix_cluster(), *4*
summary.admix_estim, 34
summary.admix_estim(), *6*
summary.admix_model, 34
summary.decontaminated_density, 35
summary.twoComp_mixt, 35

twoComp_mixt, *36*, 36

validate_distribution, 37

which_rank, 38
which_rank(), *10*