Package: fdacluster (via r-universe)

August 22, 2024

Title Joint Clustering and Alignment of Functional Data

Version 0.3.0.9000

Description Implementations of the k-means, hierarchical agglomerative and DBSCAN clustering methods for functional data which allows for jointly aligning and clustering curves. It supports functional data defined on one-dimensional domains but possibly evaluating in multivariate codomains. It supports functional data defined in arrays but also via the 'fd' and 'funData' classes for functional data defined in the 'fda' and 'funData' packages respectively. It currently supports shift, dilation and affine warping functions for functional data defined on the real line and uses the SRSF framework to handle boundary-preserving warping for functional data defined on a specific interval. Main reference for the k-means algorithm: Sangalli L.M., Secchi P., Vantini S., Vitelli V. (2010) ``k-mean alignment for curve clustering'' <doi:10.1016/j.csda.2009.12.008>. Main reference for the SRSF framework: Tucker, J. D., Wu, W., & Srivastava, A. (2013) ``Generative models for functional data using phase and amplitude separation'' <doi:10.1016/j.csda.2012.12.001>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

LazyDataCompression xz

LinkingTo Rcpp, RcppArmadillo, nloptr

RoxygenNote 7.3.1

Roxygen list(markdown = TRUE)

Suggests fda, funData, future, knitr, rmarkdown, testthat (>= 3.0.0), withr

Imports cli, cluster, dbscan, dplyr, fdasrvf, forcats, furrr, ggplot2, lpSolve, nloptr, progressr, purrr, Rcpp, rlang, tibble, tidyr

Depends R (>= 3.5.0)

URL https://astamm.github.io/fdacluster/index.html, https://github.com/astamm/fdacluster

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://astamm.r-universe.dev

RemoteUrl https://github.com/astamm/fdacluster

RemoteRef HEAD

RemoteSha 0cecb99115e86f29ecfee404a3321303ead42f4d

Contents

`autoplot.caps` *Visualizes the result of a clustering strategy stored in a* `caps` *object with ggplot2*

Description

This function creates a visualization of the result of the k-mean alignment algorithm and invisibly returns the corresponding ggplot2::ggplot object which enable further customization of the plot. The user can choose to visualize either the amplitude information data in which case original and aligned curves are shown or the phase information data in which case the estimated warping functions are shown.

Usage

```
## S3 method for class 'caps'
autoplot(object, type = c("amplitude", "phase"), ...)
```

Arguments

object	An object of class caps.
type	A string specifying the type of information to display. Choices are "amplitude" for plotting the original and aligned curves which represent amplitude information data or "phase" for plotting the corresponding warping functions which represent phase information data. Defaults to "amplitude".
...	Not used.

Value

A ggplot2::ggplot object invisibly.

Examples

```
ggplot2::autoplot(sim30_caps, type = "amplitude")
ggplot2::autoplot(sim30_caps, type = "phase")
```

---

autoplot.mcaps *Visualizes results of multiple clustering strategies using ggplot2*

Description

This is an S3 method implementation of the ggplot2::autoplot() generic for objects of class mcaps to visualize the performances of multiple caps objects applied on the same data sets either in terms of WSS or in terms of silhouette values.

Usage

```
## S3 method for class 'mcaps'
autoplot(
  object,
  validation_criterion = c("wss", "silhouette"),
  what = c("mean", "distribution"),
  ...
)
```

Arguments

object	An object of class mcaps.
validation_criterion	A string specifying the validation criterion to be used for the comparison. Choices are "wss" or "silhouette". Defaults to "wss".
what	A string specifying the kind of information to display about the validation criterion. Choices are "mean" (which plots the mean values) or "distribution" (which plots the boxplots). Defaults to "mean".
...	Other arguments passed to specific methods.

Value

An object of class ggplot2::ggplot.

Examples

```
p <- ggplot2::autoplot(sim30_mcaps)
```

---

caps *Class for clustering with amplitude and phase separation*

Description

The k-means algorithm with joint amplitude and phase separation produces a number of outputs. This class is meant to encapsulate them into a single object for providing dedicated S3 methods for e.g. plotting, summarizing, etc. The name of the class stems from **C**lustering with **A**mplitude and **P**hase **S**eparation.

Usage

```
as_caps(x)

is_caps(x)
```

Arguments

x	A list coercible into an object of class caps.

Details

An object of class caps is a list with the following components:

- original_curves: A numeric matrix of shape $N \times L \times M$ storing a sample with the N L-dimensional original curves observed on grids of size M.
- original_grids: A numeric matrix of size $N \times M$ storing the grids of size M on which original curves are evaluated;
- aligned_grids: A numeric matrix of size $N \times M$ storing the grids of size M on which original curves must be evaluated to be aligned;
- center_curves: A numeric matrix of shape $K \times L \times M$ storing the K centers which are L-dimensional curves observed on a grid of size M;
- center_grids: A numeric matrix of size $K \times M$ storing the grids of size M on which center curves are evaluated;
- warpings: A numeric matrix of shape $N \times M$ storing the estimated warping functions for each of the N curves evaluated on the within-cluster common grids of size M;
- n_clusters: An integer value storing the number of clusters;

- memberships: An integer vector of length N storing the cluster ID which each curve belongs to;
- distances_to_center: A numeric vector of length N storing the distance of each curve to the center of its cluster;
- silhouettes: A numeric vector of length N storing the silhouette values of each observation;
- amplitude_variation: A numeric value storing the fraction of total variation explained by amplitude variability.
- total_variation: A numeric value storing the amount of total variation.
- n_iterations: An integer value storing the number of iterations performed until convergence;
- call_name: A string storing the name of the function that was used to produce the k-means alignment results;
- call_args: A list containing the exact arguments that were passed to the function call_name that produced this output.

Value

The function as_caps() returns an object of class caps. The function is_caps() returns a boolean which evaluates to TRUE is the input object is of class caps.

Examples

```
as_caps(sim30_caps)
is_caps(sim30_caps)
```

---

compare_caps *Generates results of multiple clustering strategies*

Description

This function searches for clusters in the input data set using different strategies and generates an object of class mcaps which stores multiple objects of class caps. This is a helper function to facilitate comparison of clustering methods and choice of an *optimal* one.

Usage

```
compare_caps(
  x,
  y,
  n_clusters = 1:5,
  is_domain_interval = FALSE,
  transformation = c("identity", "srsf"),
  metric = c("l2", "normalized_l2", "pearson"),
 clustering_method = c("kmeans", "hclust-complete", "hclust-average", "hclust-single",
    "dbscan"),
```

```
  warping_class = c("none", "shift", "dilation", "affine", "bpd"),
  centroid_type = c("mean", "medoid", "median", "lowess", "poly"),
  cluster_on_phase = FALSE
)
```

Arguments

x	A numeric vector of length M or a numeric matrix of shape $N \times M$ or an object of class `funData::funData`. If a numeric vector or matrix, it specifies the grid(s) of size M on which each of the N curves have been observed. If an object of class `funData::funData`, it contains the whole functional data set and the y argument is not used.
y	Either a numeric matrix of shape $N \times M$ or a numeric array of shape $N \times L \times M$ or an object of class `fda::fd`. If a numeric matrix or array, it specifies the N-sample of L-dimensional curves observed on grids of size M. If an object of class `fda::fd`, it contains all the necessary information about the functional data set to be able to evaluate it on user-defined grids.
n_clusters	An integer vector specifying a set of clustering partitions to create. Defaults to `1:5`.
is_domain_interval	A boolean specifying whether the sample of curves is defined on a fixed interval. Defaults to `FALSE`.
transformation	A string specifying the transformation to apply to the original sample of curves. Choices are no transformation (`transformation = "identity"`) or square-root slope function `transformation = "srsf"`. Defaults to `"identity"`.
metric	A string specifying the metric used to compare curves. Choices are `"l2"`, `"normalized_l2"` or `"pearson"`. If `transformation == "srsf"`, the metric **must be** `"l2"` because the SRSF transform maps absolutely continuous functions to square-integrable functions. If `transformation == "identity"` and `warping_class` is either `dilation` or `affine`, the metric cab be either `"normalized_l2"` or `"pearson"`. The L2 distance is indeed **not** dilation-invariant or affine-invariant. The metric can also be `"l2"` if `warping_class == "shift"`. Defaults to `"l2"`.
clustering_method	A character vector specifying one or more clustering methods to be fit. Choices are `"kmeans"`, `"hclust-complete"`, `"hclust-average"`, `"hclust-single"` or `"dbscan"`. Defaults to all of them.
warping_class	A character vector specifying one or more classes of warping functions to use for curve alignment. Choices are `"affine"`, `"dilation"`, `"none"`, `"shift"` or `"srsf"`. Defaults to all of them.
centroid_type	A character vector specifying one or more ways to compute centroids. Choices are `"mean"`, `"medoid"`, `"median"`, `"lowess"` or `"poly"`. Defaults to all of them.
cluster_on_phase	A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to `FALSE` which implies amplitude variation.

Value

An object of class mcaps which is a tibble::tibble storing the objects of class caps in correspondence of each combination of possible choices from the input arguments.

Examples

```
#----------------------------------
# Compare k-means results with k = 1, 2, 3, 4, 5 using mean centroid and
# various warping classes.
## Not run:
sim30_mcaps <- compare_caps(
  x = simulated30_sub$x,
  y = simulated30_sub$y,
  warping_class = c("none", "shift", "dilation", "affine"),
  clustering_method = "kmeans",
  centroid_type = "mean"
)

## End(Not run)

#----------------------------------
# Then visualize the results
# Either with ggplot2 via ggplot2::autoplot(sim30_mcaps)
# or using graphics::plot()
# You can visualize the WSS values:
plot(sim30_mcaps, validation_criterion = "wss", what = "mean")
plot(sim30_mcaps, validation_criterion = "wss", what = "distribution")
# Or the average silhouette values:
plot(sim30_mcaps, validation_criterion = "silhouette", what = "mean")
plot(sim30_mcaps, validation_criterion = "silhouette", what = "distribution")
```

---

diagnostic_plot *Diagnostic plot for the result of a clustering strategy stored in a* caps *object*

Description

This function plots the values of the distance to center and silhouette for each observation. Observations are ordered within cluster by decreasing value of silhouette.

Usage

```
diagnostic_plot(x)
```

Arguments

x	An object of class caps.

Value

An object of class ggplot2::ggplot.

Examples

```
diagnostic_plot(sim30_caps)
```

`fdadbscan` *Performs density-based clustering for functional data with amplitude and phase separation*

Description

This function extends `DBSCAN` to functional data. It includes the possibility to separate amplitude and phase information.

Usage

```
fdadbscan(
  x,
  y,
  is_domain_interval = FALSE,
  transformation = c("identity", "srsf"),
  warping_class = c("none", "shift", "dilation", "affine", "bpd"),
  centroid_type = "mean",
  metric = c("l2", "normalized_l2", "pearson"),
  cluster_on_phase = FALSE,
  use_verbose = FALSE,
  warping_options = c(0.15, 0.15),
  maximum_number_of_iterations = 100L,
  number_of_threads = 1L,
  parallel_method = 0L,
  distance_relative_tolerance = 0.001,
  use_fence = FALSE,
  check_total_dissimilarity = TRUE,
  compute_overall_center = FALSE
)
```

Arguments

`x`	A numeric vector of length M or a numeric matrix of shape $N \times M$ or an object of class `funData::funData`. If a numeric vector or matrix, it specifies the grid(s) of size M on which each of the N curves have been observed. If an object of class `funData::funData`, it contains the whole functional data set and the `y` argument is not used.

y — Either a numeric matrix of shape $N \times M$ or a numeric array of shape $N \times L \times M$ or an object of class `fda::fd`. If a numeric matrix or array, it specifies the N-sample of L-dimensional curves observed on grids of size M. If an object of class `fda::fd`, it contains all the necessary information about the functional data set to be able to evaluate it on user-defined grids.

is_domain_interval — A boolean specifying whether the sample of curves is defined on a fixed interval. Defaults to `FALSE`.

transformation — A string specifying the transformation to apply to the original sample of curves. Choices are no transformation (`transformation = "identity"`) or square-root slope function `transformation = "srsf"`. Defaults to `"identity"`.

warping_class — A string specifying the class of warping functions. Choices are no warping (`warping_class = "none"`), shift `y = x + b` (`warping_class = "shift"`), dilation `y = ax` (`warping_class = "dilation"`), affine `y = ax + b` (`warping_class = "affine"`) or boundary-preserving diffeomorphism (`warping_class = "bpd"`). Defaults to `"none"`.

centroid_type — A string specifying the type of centroid to compute. Choices are `"mean"`, `"median"` `"medoid"`, `"lowess"` or `"poly"`. Defaults to `"mean"`. If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in `"lowess20"`. This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in `"poly3"`. This number will be used as the degree of the polynomial model. The default value is `4L`.

metric — A string specifying the metric used to compare curves. Choices are `"l2"`, `"normalized_l2"` or `"pearson"`. If `transformation == "srsf"`, the metric **must be** `"l2"` because the SRSF transform maps absolutely continuous functions to square-integrable functions. If `transformation == "identity"` and `warping_class` is either `dilation` or `affine`, the metric cab be either `"normalized_l2"` or `"pearson"`. The L2 distance is indeed **not** dilation-invariant or affine-invariant. The metric can also be `"l2"` if `warping_class == "shift"`. Defaults to `"l2"`.

cluster_on_phase — A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to `FALSE` which implies amplitude variation.

use_verbose — A boolean specifying whether the algorithm should output details of the steps to the console. Defaults to `FALSE`.

warping_options — A numeric vector supplied as a helper to the chosen `warping_class` to decide on warping parameter bounds. This is used only when `warping_class != "srsf"`.

maximum_number_of_iterations — An integer specifying the maximum number of iterations before the algorithm stops if no other convergence criterion was met. Defaults to `100L`.

number_of_threads — An integer value specifying the number of threads used for parallelization. Defaults to `1L`. This is used only when `warping_class != "srsf"`.

parallel_method
: An integer value specifying the type of desired parallelization for template computation, If `0L`, templates are computed in parallel. If `1L`, parallelization occurs within a single template computation (only for the medoid method as of now). Defaults to `0L`. This is used only when `warping_class != "srsf"`.

distance_relative_tolerance
: A numeric value specifying a relative tolerance on the distance update between two iterations. If all observations have not sufficiently improved in that sense, the algorithm stops. Defaults to `1e-3`. This is used only when `warping_class != "srsf"`.

use_fence
: A boolean specifying whether the fence algorithm should be used to robustify the algorithm against outliers. Defaults to `FALSE`. This is used only when `warping_class != "srsf"`.

check_total_dissimilarity
: A boolean specifying whether an additional stopping criterion based on improvement of the total dissimilarity should be used. Defaults to `TRUE`. This is used only when `warping_class != "srsf"`.

compute_overall_center
: A boolean specifying whether the overall center should be also computed. Defaults to `FALSE`. This is used only when `warping_class != "srsf"`.

Value

An object of class caps.

Examples

```
#----------------------------------
# Extracts 15 out of the 30 simulated curves in `simulated30_sub` data set
idx <- c(1:5, 11:15)
x <- simulated30_sub$x[idx, ]
y <- simulated30_sub$y[idx, , ]

#----------------------------------
# Runs an HAC with affine alignment, searching for 2 clusters
out <- fdadbscan(
  x = x,
  y = y,
  warping_class = "affine",
  metric = "normalized_l2"
)

#----------------------------------
# Then visualize the results
# Either with ggplot2 via ggplot2::autoplot(out)
# or using graphics::plot()
# You can visualize the original and aligned curves with:
plot(out, type = "amplitude")
# Or the estimated warping functions with:
plot(out, type = "phase")
```

`fdadist` *Computes the distance matrix for functional data with amplitude and phase separation*

Description

This function computes the matrix of pairwise distances between curves a functional data sample. This can be achieved with or without phase and amplitude separation, which can be done using a variety of warping classes.

Usage

```
fdadist(
  x,
  y = NULL,
  is_domain_interval = FALSE,
  transformation = c("identity", "srsf"),
  warping_class = c("none", "shift", "dilation", "affine", "bpd"),
  metric = c("l2", "normalized_l2", "pearson"),
  cluster_on_phase = FALSE,
  labels = NULL
)
```

Arguments

`x`	A numeric vector of length M or a numeric matrix of shape $N \times M$ or an object of class `funData::funData`. If a numeric vector or matrix, it specifies the grid(s) of size M on which each of the N curves have been observed. If an object of class `funData::funData`, it contains the whole functional data set and the `y` argument is not used.
`y`	Either a numeric matrix of shape $N \times M$ or a numeric array of shape $N \times L \times M$ or an object of class `fda::fd`. If a numeric matrix or array, it specifies the N-sample of L-dimensional curves observed on grids of size M. If an object of class `fda::fd`, it contains all the necessary information about the functional data set to be able to evaluate it on user-defined grids.
`is_domain_interval`	A boolean specifying whether the sample of curves is defined on a fixed interval. Defaults to `FALSE`.
`transformation`	A string specifying the transformation to apply to the original sample of curves. Choices are no transformation (`transformation = "identity"`) or square-root slope function `transformation = "srsf"`. Defaults to `"identity"`.
`warping_class`	A string specifying the class of warping functions. Choices are no warping (`warping_class = "none"`), shift `y = x + b` (`warping_class = "shift"`), dilation `y = ax` (`warping_class = "dilation"`), affine `y = ax + b` (`warping_class = "affine"`) or boundary-preserving diffeomorphism (`warping_class = "bpd"`). Defaults to `"none"`.

`metric` A string specifying the metric used to compare curves. Choices are `"l2"`, `"normalized_l2"` or `"pearson"`. If `transformation == "srsf"`, the metric **must be** `"l2"` because the SRSF transform maps absolutely continuous functions to square-integrable functions. If `transformation == "identity"` and `warping_class` is either `dilation` or `affine`, the metric cab be either `"normalized_l2"` or `"pearson"`. The L2 distance is indeed **not** dilation-invariant or affine-invariant. The metric can also be `"l2"` if `warping_class == "shift"`. Defaults to `"l2"`.

`cluster_on_phase` A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to `FALSE` which implies amplitude variation.

`labels` A character vector specifying curve labels. Defaults to `NULL` which uses sequential numbers as labels.

Value

A stats::dist object storing the distance matrix between the input curves using the metric specified through the argument `metric` and the warping class specified by the argument `warping_class`.

Examples

```
idx <- c(1:5, 11:15, 21:25)
D <- fdadist(simulated30_sub$x[idx, ], simulated30_sub$y[idx, , ])
```

---

fdahclust *Performs hierarchical clustering for functional data with amplitude and phase separation*

Description

This function extends hierarchical agglomerative clustering to functional data. It includes the possibility to separate amplitude and phase information.

Usage

```
fdahclust(
  x,
  y = NULL,
  n_clusters = 1L,
  is_domain_interval = FALSE,
  transformation = c("identity", "srsf"),
  warping_class = c("none", "shift", "dilation", "affine", "bpd"),
  centroid_type = "mean",
  metric = c("l2", "normalized_l2", "pearson"),
  cluster_on_phase = FALSE,
  linkage_criterion = c("complete", "average", "single", "ward.D2"),
  use_verbose = FALSE,
  warping_options = c(0.15, 0.15),
```

```
  maximum_number_of_iterations = 100L,
  number_of_threads = 1L,
  parallel_method = 0L,
  distance_relative_tolerance = 0.001,
  use_fence = FALSE,
  check_total_dissimilarity = TRUE,
  compute_overall_center = FALSE
)
```

Arguments

x	A numeric vector of length M or a numeric matrix of shape $N \times M$ or an object of class `funData::funData`. If a numeric vector or matrix, it specifies the grid(s) of size M on which each of the N curves have been observed. If an object of class `funData::funData`, it contains the whole functional data set and the y argument is not used.
y	Either a numeric matrix of shape $N \times M$ or a numeric array of shape $N \times L \times M$ or an object of class `fda::fd`. If a numeric matrix or array, it specifies the N-sample of L-dimensional curves observed on grids of size M. If an object of class `fda::fd`, it contains all the necessary information about the functional data set to be able to evaluate it on user-defined grids.
n_clusters	An integer value specifying the number of clusters. Defaults to `1L`.
is_domain_interval	A boolean specifying whether the sample of curves is defined on a fixed interval. Defaults to `FALSE`.
transformation	A string specifying the transformation to apply to the original sample of curves. Choices are no transformation (`transformation = "identity"`) or square-root slope function `transformation = "srsf"`. Defaults to `"identity"`.
warping_class	A string specifying the class of warping functions. Choices are no warping (`warping_class = "none"`), shift `y = x + b` (`warping_class = "shift"`), dilation `y = ax` (`warping_class = "dilation"`), affine `y = ax + b` (`warping_class = "affine"`) or boundary-preserving diffeomorphism (`warping_class = "bpd"`). Defaults to `"none"`.
centroid_type	A string specifying the type of centroid to compute. Choices are `"mean"`, `"median"` `"medoid"`, `"lowess"` or `"poly"`. Defaults to `"mean"`. If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in `"lowess20"`. This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in `"poly3"`. This number will be used as the degree of the polynomial model. The default value is `4L`.
metric	A string specifying the metric used to compare curves. Choices are `"l2"`, `"normalized_l2"` or `"pearson"`. If `transformation == "srsf"`, the metric **must be** `"l2"` because the SRSF transform maps absolutely continuous functions to square-integrable functions. If `transformation == "identity"` and `warping_class` is either `dilation` or `affine`, the metric cab be either `"normalized_l2"`

or "pearson". The L2 distance is indeed **not** dilation-invariant or affine-invariant. The metric can also be "l2" if warping_class == "shift". Defaults to "l2".

cluster_on_phase
: A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to FALSE which implies amplitude variation.

linkage_criterion
: A string specifying which linkage criterion should be used to compute distances between sets of curves. Choices are "complete" for complete linkage, "average" for average linkage and "single" for single linkage. See stats::hclust() for more details. Defaults to "complete".

use_verbose
: A boolean specifying whether the algorithm should output details of the steps to the console. Defaults to FALSE.

warping_options
: A numeric vector supplied as a helper to the chosen warping_class to decide on warping parameter bounds. This is used only when warping_class != "srsf".

maximum_number_of_iterations
: An integer specifying the maximum number of iterations before the algorithm stops if no other convergence criterion was met. Defaults to 100L.

number_of_threads
: An integer value specifying the number of threads used for parallelization. Defaults to 1L. This is used only when warping_class != "srsf".

parallel_method
: An integer value specifying the type of desired parallelization for template computation, If 0L, templates are computed in parallel. If 1L, parallelization occurs within a single template computation (only for the medoid method as of now). Defaults to 0L. This is used only when warping_class != "srsf".

distance_relative_tolerance
: A numeric value specifying a relative tolerance on the distance update between two iterations. If all observations have not sufficiently improved in that sense, the algorithm stops. Defaults to 1e-3. This is used only when warping_class != "srsf".

use_fence
: A boolean specifying whether the fence algorithm should be used to robustify the algorithm against outliers. Defaults to FALSE. This is used only when warping_class != "srsf".

check_total_dissimilarity
: A boolean specifying whether an additional stopping criterion based on improvement of the total dissimilarity should be used. Defaults to TRUE. This is used only when warping_class != "srsf".

compute_overall_center
: A boolean specifying whether the overall center should be also computed. Defaults to FALSE. This is used only when warping_class != "srsf".

Details

The number of clusters is required as input because, with functional data, once hierarchical clustering is performed, curves within clusters need to be aligned to their corresponding centroid.

Value

An object of class caps.

Examples

```
#----------------------------------
# Extracts 15 out of the 30 simulated curves in `simulated30_sub` data set
idx <- c(1:5, 11:15, 21:25)
x <- simulated30_sub$x[idx, ]
y <- simulated30_sub$y[idx, , ]

#----------------------------------
# Runs an HAC with affine alignment, searching for 2 clusters
out <- fdahclust(
  x = x,
  y = y,
  n_clusters = 2,
  warping_class = "affine",
  metric = "normalized_l2"
)

#----------------------------------
# Then visualize the results
# Either with ggplot2 via ggplot2::autoplot(out)
# or using graphics::plot()
# You can visualize the original and aligned curves with:
plot(out, type = "amplitude")
# Or the estimated warping functions with:
plot(out, type = "phase")
```

---

fdakmeans *Performs k-means clustering for functional data with amplitude and phase separation*

Description

This function provides implementations of the k-means clustering algorithm for functional data, with possible joint amplitude and phase separation. A number of warping class are implemented to achieve this separation.

Usage

```
fdakmeans(
  x,
  y = NULL,
  n_clusters = 1L,
  seeds = NULL,
  seeding_strategy = c("kmeans++", "exhaustive-kmeans++", "exhaustive", "hclust"),
  is_domain_interval = FALSE,
```

```
  transformation = c("identity", "srsf"),
  warping_class = c("none", "shift", "dilation", "affine", "bpd"),
  centroid_type = "mean",
  metric = c("l2", "normalized_l2", "pearson"),
  cluster_on_phase = FALSE,
  use_verbose = FALSE,
  warping_options = c(0.15, 0.15),
  maximum_number_of_iterations = 100L,
  number_of_threads = 1L,
  parallel_method = 0L,
  distance_relative_tolerance = 0.001,
  use_fence = FALSE,
  check_total_dissimilarity = TRUE,
  compute_overall_center = FALSE,
  add_silhouettes = TRUE
)
```

Arguments

x	A numeric vector of length M or a numeric matrix of shape $N \times M$ or an object of class funData::funData. If a numeric vector or matrix, it specifies the grid(s) of size M on which each of the N curves have been observed. If an object of class funData::funData, it contains the whole functional data set and the y argument is not used.
y	Either a numeric matrix of shape $N \times M$ or a numeric array of shape $N \times L \times M$ or an object of class fda::fd. If a numeric matrix or array, it specifies the N-sample of L-dimensional curves observed on grids of size M. If an object of class fda::fd, it contains all the necessary information about the functional data set to be able to evaluate it on user-defined grids.
n_clusters	An integer value specifying the number of clusters. Defaults to 1L.
seeds	An integer value or vector specifying the indices of the initial centroids. If an integer vector, it is interpreted as the indices of the intial centroids and should therefore be of length n_clusters. If an integer value, it is interpreted as the index of the first initial centroid and subsequent centroids are chosen according to the k-means++ strategy. It can be NULL in which case the argument seeding_strategy is used to automatically provide suitable indices. Defaults to NULL.
seeding_strategy	A character string specifying the strategy for choosing the initial centroids in case the argument seeds is set to NULL. Choices are "kmeans++", "exhaustive-kmeans++" which performs an exhaustive search over the choice of the first centroid, "exhaustive" which tries on all combinations of initial centroids or "hclust" which first performs hierarchical clustering using Ward's linkage criterion to identify initial centroids. Defaults to "kmeans++", which is the fastest strategy.
is_domain_interval	A boolean specifying whether the sample of curves is defined on a fixed interval. Defaults to FALSE.

`transformation` A string specifying the transformation to apply to the original sample of curves. Choices are no transformation (`transformation = "identity"`) or square-root slope function `transformation = "srsf"`. Defaults to `"identity"`.

`warping_class` A string specifying the class of warping functions. Choices are no warping (`warping_class = "none"`), shift `y = x + b` (`warping_class = "shift"`), dilation `y = ax` (`warping_class = "dilation"`), affine `y = ax + b` (`warping_class = "affine"`) or boundary-preserving diffeomorphism (`warping_class = "bpd"`). Defaults to `"none"`.

`centroid_type` A string specifying the type of centroid to compute. Choices are `"mean"`, `"median"` `"medoid"`, `"lowess"` or `"poly"`. Defaults to `"mean"`. If LOWESS appproximation is chosen, the user can append an integer between 0 and 100 as in `"lowess20"`. This number will be used as the smoother span. This gives the proportion of points in the plot which influence the smooth at each value. Larger values give more smoothness. The default value is 10%. If polynomial approximation is chosen, the user can append an positive integer as in `"poly3"`. This number will be used as the degree of the polynomial model. The default value is `4L`.

`metric` A string specifying the metric used to compare curves. Choices are `"l2"`, `"normalized_l2"` or `"pearson"`. If `transformation == "srsf"`, the metric **must be** `"l2"` because the SRSF transform maps absolutely continuous functions to square-integrable functions. If `transformation == "identity"` and `warping_class` is either `dilation` or `affine`, the metric cab be either `"normalized_l2"` or `"pearson"`. The L2 distance is indeed **not** dilation-invariant or affine-invariant. The metric can also be `"l2"` if `warping_class == "shift"`. Defaults to `"l2"`.

`cluster_on_phase`
A boolean specifying whether clustering should be based on phase variation or amplitude variation. Defaults to `FALSE` which implies amplitude variation.

`use_verbose` A boolean specifying whether the algorithm should output details of the steps to the console. Defaults to `FALSE`.

`warping_options`
A numeric vector supplied as a helper to the chosen `warping_class` to decide on warping parameter bounds. This is used only when `warping_class != "srsf"`.

`maximum_number_of_iterations`
An integer specifying the maximum number of iterations before the algorithm stops if no other convergence criterion was met. Defaults to `100L`.

`number_of_threads`
An integer value specifying the number of threads used for parallelization. Defaults to `1L`. This is used only when `warping_class != "srsf"`.

`parallel_method`
An integer value specifying the type of desired parallelization for template computation, If `0L`, templates are computed in parallel. If `1L`, parallelization occurs within a single template computation (only for the medoid method as of now). Defaults to `0L`. This is used only when `warping_class != "srsf"`.

`distance_relative_tolerance`
A numeric value specifying a relative tolerance on the distance update between two iterations. If all observations have not sufficiently improved in that sense,

the algorithm stops. Defaults to `1e-3`. This is used only when `warping_class != "srsf"`.

`use_fence` A boolean specifying whether the fence algorithm should be used to robustify the algorithm against outliers. Defaults to `FALSE`. This is used only when `warping_class != "srsf"`.

`check_total_dissimilarity`
A boolean specifying whether an additional stopping criterion based on improvement of the total dissimilarity should be used. Defaults to `TRUE`. This is used only when `warping_class != "srsf"`.

`compute_overall_center`
A boolean specifying whether the overall center should be also computed. Defaults to `FALSE`. This is used only when `warping_class != "srsf"`.

`add_silhouettes`
A boolean specifying whether silhouette values should be computed for each observation for internal validation of the clustering structure. Defaults to `TRUE`.

Value

An object of class caps.

Examples

```
#----------------------------------
# Extracts 15 out of the 30 simulated curves in `simulated30_sub` data set
idx <- c(1:5, 11:15, 21:25)
x <- simulated30_sub$x[idx, ]
y <- simulated30_sub$y[idx, , ]

#----------------------------------
# Runs a k-means clustering with affine alignment, searching for 2 clusters
out <- fdakmeans(
  x = x,
  y = y,
  n_clusters = 2,
  warping_class = "affine",
  metric = "normalized_l2"
)

#----------------------------------
# Then visualize the results
# Either with ggplot2 via ggplot2::autoplot(out)
# or using graphics::plot()
# You can visualize the original and aligned curves with:
plot(out, type = "amplitude")
# Or the estimated warping functions with:
plot(out, type = "phase")
```

plot.caps *Plots the result of a clustering strategy stored in a* caps *object*

Description

This function creates a visualization of the result of the k-mean alignment algorithm **without** returning the plot data as an object. The user can choose to visualize either the amplitude information data in which case original and aligned curves are shown or the phase information data in which case the estimated warping functions are shown.

Usage

```
## S3 method for class 'caps'
plot(x, type = c("amplitude", "phase"), ...)
```

Arguments

x	An object of class caps.
type	A string specifying the type of information to display. Choices are "amplitude" for plotting the original and aligned curves which represent amplitude information data or "phase" for plotting the corresponding warping functions which represent phase information data. Defaults to "amplitude".
...	Not used.

Examples

```
plot(sim30_caps, type = "amplitude")
plot(sim30_caps, type = "phase")
```

plot.mcaps *Plots results of multiple clustering strategies*

Description

This is an S3 method implementation of the graphics::plot() generic for objects of class mcaps to visualize the performances of multiple caps objects applied on the same data sets either in terms of WSS or in terms of silhouette values.

Usage

```
## S3 method for class 'mcaps'
plot(
  x,
  validation_criterion = c("wss", "silhouette"),
  what = c("mean", "distribution"),
  ...
)
```

Arguments

x	An object of class mcaps.
validation_criterion	A string specifying the validation criterion to be used for the comparison. Choices are "wss" or "silhouette". Defaults to "wss".
what	A string specifying the kind of information to display about the validation criterion. Choices are "mean" (which plots the mean values) or "distribution" (which plots the boxplots). Defaults to "mean".
...	Other arguments passed to specific methods.

Examples

```
plot(sim30_mcaps)
```

---

sim30_caps *A* caps *object from simulated data for examples*

Description

An object of class caps storing the result of the fdakmeans() function applied on the data set simulated30 using the affine warping class and the Pearson metric and searching for 2 clusters.

Usage

```
sim30_caps
```

Format

An object of class caps.

---

sim30_mcaps *An* mcaps *object from simulated data for examples*

Description

An object of class mcaps storing the result of the compare_caps() function applied on the data set simulated30_sub for comparing the clustering structures found by the fdakmeans() function with mean centroid type used with various classes of warping functions and varying number of clusters.

Usage

```
sim30_mcaps
```

Format

An object of class `mcaps` which is effectively a tibble::tibble with 5 columns and as many rows as there are clustering strategies to compare. The 5 column-variables are:

- `n_clusters`: The number of clusters;
- `clustering_method`: The clustering method;
- `warping_class`: The class of warping functions used for curve alignment;
- `centroid_type`: The type of centroid used to compute a cluster representative;
- `caps_obj`: The result of the corresponding clustering strategy as objects of class `caps`.

`simulated30` *Simulated data for examples*

Description

A data set containing 30 simulated uni-dimensional curves.

Usage

```
simulated30
```

Format

A list with abscissas x and values y:

x Matrix 30x200;

y Array 30x1x200.

`simulated30_sub` *Simulated data for examples*

Description

A data set containing 30 simulated uni-dimensional curves.

Usage

```
simulated30_sub
```

Format

A list with abscissas x and values y:

x Matrix 30x30;

y Array 30x1x30.

`simulated90` *Simulated data from the CSDA paper*

Description

A data set containing 90 simulated uni-dimensional curves.

Usage

```
simulated90
```

Format

A list with abscissas x and values y:

x Vector of size 100;

y Matrix if size 90x100.

Index