Package 'funMoDisco'

November 14, 2024

Type Package

Title Motif Discovery in Functional Data

Version 1.0.0

Date 2024-10-05

Author Authors@R: c(
person(given = c(``Marzia", ``Angela"), family = ``Cremona", role = ``aut"),
person(given = ``Francesca", family = ``Chiaromonte", role = ``aut"),
person(given = c(``Jacopo", ``Di"), family = ``Iorio", role = c(``aut", ``cre"),
email = ``jacopo.di.iorio@emory.edu"),
person(given = ``Niccolo", family = ``Feresini", role = ``aut"),
person(given = ``Riccardo", family = ``Lazzarini", role = ``aut"))

Maintainer Jacopo Di Iorio <jacopo.di.iorio@emory.edu>

Description Tools for discovering motifs in functional data, efficiently implementing two complementary methodologies: ProbKMA and FunBIalign.
ProbKMA (Cremona and Chiaromonte, 2023) is a probabilistic K-means algorithm that leverages local alignment and fuzzy clustering to identify recurring patterns (candidate functional motifs) across and within curves, allowing different portions of the same curve to belong to different clusters. It includes a family of distances and a normalization to discover various motif types and learns motif lengths in a data-driven manner. It can also be used for local clustering of misaligned data.
FunBIalign (Di Iorio, Cremona, and Chiaromonte, 2023) applies hierarchical agglomerative clustering with a functional generalization of the Mean Squared Residue Score to identify motifs of a specified length in curves. This deterministic method includes a small set of user-tunable parameters.
Both algorithms are suitable for single curves or sets of curves. The package also includes a flexible function to simulate functional data with embedded motifs, allowing users to generate benchmark datasets for validating and comparing motif discovery methods.

License GPL (>= 2)

Suggests knitr,
rmarkdown,
tinytest,
kableExtra,
R.rsp

Encoding UTF-8

RoxygenNote 7.3.2

Imports Rcpp (>= 1.0.12),dendextend,fastcluster,
fda,ggtext,methods,purrr,scales,shinyWidgets,shinybusy,shinyjs,
zoo,utils,class,combinat,data.table,stringr

Depends R (>= 2.10),dplyr,parallel,ggplot2,shiny,progress

LinkingTo Rcpp, RcppArmadillo

SystemRequirements C++20

LazyData true

VignetteBuilder R.rsp

BuildVignettes true

Language en-US

R topics documented:

`funMoDisco-package` *funMoDisco: Motif Discovery in Functional Data*

Description

Tools for discovering motifs in functional data, efficiently implementing two complementary methodologies: ProbKMA and FunBIalign. ProbKMA (Cremona and Chiaromonte, 2023) is a probabilistic K-means algorithm that leverages local alignment and fuzzy clustering to identify recurring patterns (candidate functional motifs) across and within curves, allowing different portions of the same curve to belong to different clusters. It includes a family of distances and a normalization to discover various motif types and learns motif lengths in a data-driven manner. It can also be used for local clustering of misaligned data. FunBIalign (Di Iorio, Cremona, and Chiaromonte, 2023) applies hierarchical agglomerative clustering with a functional generalization of the Mean Squared Residue Score to identify motifs of a specified length in curves. This deterministic method includes a small set of user-tunable parameters. Both algorithms are suitable for single curves or sets of curves. The package also includes a flexible function to simulate functional data with embedded motifs, allowing users to generate benchmark datasets for validating and comparing motif discovery methods.

`.check_fits` *Check Fits*

Description

This function checks if motifs fit within a specified curve length and whether there are overlaps between motifs. It returns TRUE if all conditions are met and FALSE otherwise.

Usage

```
.check_fits(df, min_dist_motifs, len)
```

Arguments

`df`	A data frame containing motif information with columns 'start' and 'end' indicating the positions of motifs.
`min_dist_motifs`	A numeric value specifying the minimum distance required between motifs.
`len`	A numeric value representing the total length of the curve.

Details

Check Fits for Motifs

Value

A logical value indicating whether the motifs fit within the curve and do not overlap.

`.diss_d0_d1_L2` *Dissimilarity Index for Multidimensional Curves*

Description

Computes a Sobolev-type dissimilarity index for multidimensional curves based on a weighted combination of L2 norms of the function and its derivative. The distance is normalized on a common support, allowing for a comparison between curves considering both their levels and variations.

Usage

```
.diss_d0_d1_L2(y, v, w, alpha, transform_y = FALSE, transform_v = FALSE)
```

Arguments

`y`	A list of two matrices: • `y[[1]]`: Values of the curve at points in the domain (y(x)). • `y[[2]]`: Values of the derivative of the curve at the same points (y'(x)). Each matrix should have `d` columns, where `d` is the dimensionality of the curves.
`v`	A list of two matrices: • `v[[1]]`: Values of the reference curve at points in the domain (v(x)). • `v[[2]]`: Values of the derivative of the reference curve at the same points (v'(x)). Each matrix should have `d` columns, where `d` is the dimensionality of the curves.
`w`	A numeric vector of weights for the dissimilarity index in different dimensions. Each weight should be greater than 0.
`alpha`	A numeric value (between 0 and 1) that specifies the weight coefficient between the L2 norm of the function (d0.L2) and the L2 norm of the derivative (d1.L2): • `alpha` = `0`: Only the levels (d0.L2) are considered. • `alpha` = `1`: Only the derivative information (d1.L2) is considered. • Values between 0 and 1 provide a mixture of both.
`transform_y`	A logical value indicating whether to normalize `y` to the range [0, 1] before applying the distance computation. Default is `FALSE`.
`transform_v`	A logical value indicating whether to normalize `v` to the range [0, 1] before applying the distance computation. Default is `FALSE`.

Details

The dissimilarity index is calculated based on the following Sobolev-type distance:

$$D = (1 - \alpha) \cdot d0.L2 + \alpha \cdot d1.L2$$

where:

- `d0.L2`: L2 distance considering only the levels of the curves.
- `d1.L2`: L2 distance considering only the derivatives of the curves.

The function normalizes the inputs based on the specified flags to ensure that all features are comparable.

Value

A numeric value representing the dissimilarity index between the curves defined by y and v.

Examples

```
## Not run:
# Define two curves and their derivatives
y_curve <- list(matrix(rnorm(100), ncol = 2), matrix(rnorm(100), ncol = 2))
v_curve <- list(matrix(rnorm(100), ncol = 2), matrix(rnorm(100), ncol = 2))
weights <- c(1, 1)

# Calculate dissimilarity index without normalization
diss_index <- .diss_d0_d1_L2(y = y_curve, v = v_curve, w = weights, alpha = 0.5)

# Output the dissimilarity index
print(diss_index)

## End(Not run)
```

`.domain` *Domain Length Calculation for Curves*

Description

Computes the length of the domain of a curve by checking the presence of non-NA values in the specified matrix. The function returns a logical vector indicating which rows contain at least one non-NA element, allowing for an evaluation of the effective length of the domain.

Usage

```
.domain(v, use0)
```

Arguments

`v`	A list containing two elements: • `v[[1]]`: A matrix representing the values of the curve (v(x)). • `v[[2]]`: A matrix representing the values of the derivative of the curve (v'(x)). Each matrix should have `d` columns, where `d` is the dimensionality of the curve.
`use0`	A logical value indicating which element of `v` to consider: • If `use0` = `TRUE`, the function considers `v[[1]]` (the curve values). • If `use0` = `FALSE`, the function considers `v[[2]]` (the derivative values).

Details

The function evaluates each row of the specified matrix (either `v[[1]]` or `v[[2]]`) to determine if there is at least one non-NA entry. This is particularly useful for assessing the effective domain of curves where some values may be missing.

Value

A logical vector where each element is `TRUE` if at least one element in the corresponding row of the selected matrix (either `v[[1]]` or `v[[2]]`) is not NA, and `FALSE` otherwise.

Examples

```
## Not run:
# Define a list containing two matrices
v_curve <- list(matrix(c(1, 2, NA, 4, NA, 6), ncol = 2), matrix(c(NA, NA, 3, 4, 5, 6), ncol = 2))

# Calculate the domain length considering curve values
domain_length_v0 <- .domain(v = v_curve, use0 = TRUE)

# Calculate the domain length considering derivative values
domain_length_v1 <- .domain(v = v_curve, use0 = FALSE)

# Output the results
print(domain_length_v0)  # Logical vector for v0
print(domain_length_v1)  # Logical vector for v1

## End(Not run)
```

`.find_min_diss` *Find Minimum Dissimilarity*

Description

Finds the shift warping that minimizes dissimilarity between multidimensional curves. The function operates on curves represented by two sets of data, each containing multiple dimensions.

Usage

```
.find_min_diss(
  y,
  v,
  alpha,
  w,
  c_k,
  d,
  use0,
  use1,
  transform_y = FALSE,
  transform_v = FALSE
)
```

Arguments

`y`	A list containing two matrices: - `y0`: The first set of curve values \(y(x)\). - `y1`: The second set of curve values \(y'(x)\). Each matrix should have \(d\) columns corresponding to the dimensions.

v	A list containing two matrices: - v0: The first set of curve values \(v(x)\). - v1: The second set of curve values \(v'(x)\). Each matrix should have \(d\) columns corresponding to the dimensions.
alpha	A numeric weight coefficient that balances the contributions of the L2 norms of the two curve sets.
w	A numeric vector of weights for the dissimilarity index across different dimensions. All weights must be positive (\(w > 0\)).
c_k	An integer specifying the minimum length of the intersection of the supports of the shifted \(y\) and \(v\).
d	An integer indicating the dimensionality of the curves.
use0	A logical value indicating whether to use the first component of the curves (i.e., \(y0\) and \(v0\)).
use1	A logical value indicating whether to use the second component of the curves (i.e., \(y1\) and \(v1\)).
transform_y	A logical value indicating whether to normalize \(y\) to the range \([0,1]\) before calculating the distance.
transform_v	A logical value indicating whether to normalize \(v\) to the range \([0,1]\) before calculating the distance.

Details

This function computes the shift warping between the provided multidimensional curves by examining various shifts and calculating the corresponding dissimilarity. The user can control which components of the curves to include in the calculation and whether to normalize the data.

The function returns both the optimal shift and the minimal dissimilarity, which can be used to assess the similarity between the two sets of curves under the specified constraints.

Value

A numeric vector containing: - The optimal shift that minimizes the dissimilarity. - The minimum dissimilarity value found.

Author(s)

Marzia Angela Cremona & Francesca Chiaromonte

Examples

```
## Not run:
# Example usage
y <- list(y0 = matrix(runif(100), ncol = 2), y1 = matrix(runif(100), ncol = 2))
v <- list(v0 = matrix(runif(100), ncol = 2), v1 = matrix(runif(100), ncol = 2))
result <- .find_min_diss(y = y, v = v, alpha = 0.5, w = c(1, 1),
                         c_k = 5, d = 2, use0 = TRUE, use1 = TRUE)
print(result)

## End(Not run)
```

`.find_occurrences` *Find Occurrences of a Motif*

Description

Finds occurrences of a specified motif in a set of curves, where the dissimilarity is lower than a specified threshold \(R \). The function compares the motif against curves represented in multiple dimensions and returns the details of matching occurrences.

Usage

```
.find_occurrences(v, Y, R, alpha, w, c_k, use0, use1, transformed = FALSE)
```

Arguments

`v`	A list containing two matrices: - `v0`: The first set of motif values. - `v1`: The second set of motif values. Each matrix should have \(d \) columns corresponding to the dimensions.
`Y`	A list of \(N \) lists, each containing two matrices: - `Y0`: The first set of curve values. - `Y1`: The second set of curve values. Each matrix should have \(d \) columns corresponding to the dimensions.
`R`	A numeric value representing the maximum allowed dissimilarity.
`alpha`	A numeric value that serves as a weight coefficient between the L2 norms of the two sets of motifs when using the dissimilarity function `diss_d0_d1_L2`.
`w`	A numeric vector of weights for the dissimilarity index across different dimensions. All weights must be positive (\(w > 0 \)).
`c_k`	An integer specifying the minimum length of the intersection of the supports of the shifted motif and the curves.
`use0`	A logical value indicating whether to use the first component of the curves (i.e., \(Y0 \) and \(v0 \)).
`use1`	A logical value indicating whether to use the second component of the curves (i.e., \(Y1 \) and \(v1 \)).
`transformed`	A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting ‘transformed = TRUE‘ scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.

Details

The function systematically checks each curve in the provided list against the specified motif to identify positions where the dissimilarity does not exceed the defined threshold \(R \). It handles multidimensional curves and can selectively consider different components of the motifs and curves.

Value

A matrix with three columns: - `curve`: The ID of the curve where the motif was found. - `shift`: The optimal shift at which the motif occurs. - `diss`: The dissimilarity value associated with the match. If no occurrences are found, an empty matrix is returned.

Author(s)

Marzia Angela Cremona & Francesca Chiaromonte

Examples

```
## Not run:
# Example usage
v <- list(v0 = matrix(runif(100), ncol = 2), v1 = matrix(runif(100), ncol = 2))
Y <- list(list(Y0 = matrix(runif(200), ncol = 2), Y1 = matrix(runif(200), ncol = 2)))
result <- .find_occurrences(v = v, Y = Y, R = 0.5, alpha = 0.5, w = c(1, 1),
                             c_k = 5, use0 = TRUE, use1 = TRUE)
print(result)

## End(Not run)
```

`.generate_coefficients` *Generate Coefficients*

Description

This function generates a vector of coefficients for a motif based on the specified distribution. It can sample coefficients from a numeric vector, a uniform distribution, or a beta distribution.

Usage

```
.generate_coefficients(
  motif_i,
  distrib,
  dist_knots,
  norder,
  coeff_min,
  coeff_max
)
```

Arguments

`motif_i`	A list representing the motif structure that includes the length (‘len‘).
`distrib`	A string or numeric vector indicating the distribution from which to generate coefficients. Accepted values are "unif", "beta", or a numeric vector.
`dist_knots`	A numeric value indicating the distance between knots.
`norder`	An integer specifying the order of the B-spline.
`coeff_min`	A numeric value indicating the minimum coefficient value for uniform or beta distributions.

`coeff_max`	A numeric value indicating the maximum coefficient value for uniform or beta distributions.

Details

Generate Coefficients for Motif

- For uniform distribution, coefficients are generated within the specified range defined by ‘coeff_min‘ and ‘coeff_max‘. - For beta distribution, coefficients are scaled to the desired range using the specified minimum and maximum values.

Value

A modified motif structure containing the generated coefficients.

`.mapply_custom` *Custom mapply Function for Parallel Processing*

Description

A wrapper function that conditionally applies either the classical ‘mapply‘ function or ‘parallel::clusterMap‘ based on whether a cluster object is provided. If the number of clusters is not null, it applies ‘clusterMap‘ for parallel execution; otherwise, it defaults to the classical ‘mapply‘.

Usage

```
.mapply_custom(
  cl,
  FUN,
  ...,
  MoreArgs = NULL,
  SIMPLIFY = TRUE,
  USE.NAMES = TRUE
)
```

Arguments

`cl`	A cluster object created by ‘makeCluster‘. If ‘NULL‘, the classical ‘mapply‘ function is used.
`FUN`	A function to apply to the arguments.
`...`	Arguments to be passed to ‘FUN‘. These should be vectors of equal length.
`MoreArgs`	A list of additional arguments to be passed to ‘FUN‘.
`SIMPLIFY`	A logical value indicating whether to simplify the result if possible. Default is ‘TRUE‘.
`USE.NAMES`	A logical value indicating whether to use names from the first argument. Default is ‘TRUE‘.

Details

This function is useful for switching between parallel and non-parallel execution based on the availability of a cluster, allowing for more flexible and efficient code.

Value

A vector or list containing the results of applying the function ‘FUN‘ to the provided arguments. The output type depends on the value of ‘SIMPLIFY‘.

Author(s)

Marzia Angela Cremona & Francesca Chiaromonte

Examples

```
## Not run:
# Example usage
library(parallel)
cl <- makeCluster(4) # Create a cluster with 4 workers

# Define a simple function to square numbers
square_func <- function(x) {
  return(x^2)
}

# Using .mapply_custom with a cluster
result_parallel <- .mapply_custom(cl, square_func, 1:5, MoreArgs = list())
print(result_parallel) # Prints squared values

# Using .mapply_custom without a cluster
result_serial <- .mapply_custom(NULL, square_func, 1:5, MoreArgs = list())
print(result_serial) # Prints squared values

# Stop the cluster after use
stopCluster(cl)

## End(Not run)
```

`.resample` *Resample Vector*

Description

This function resamples a vector by randomly selecting indices. The number of samples can be specified.

Usage

```
.resample(x, ...)
```

Arguments

x	A vector to be resampled.
...	Additional arguments passed to the sampling function.

Details

Resample Vector

Value

A resampled vector.

`.select_domain` *.select_domain*

Description

This function selects the portion of a motif that is free from NA values based on a specified domain. It allows for the selection of two matrices, ‘v0‘ and ‘v1‘, depending on the boolean flags provided.

Usage

```
.select_domain(v, v_dom, use0, use1)
```

Arguments

`v`	A list containing two elements: • v[[1]]: A matrix representing \(v(x) \) with \(d \) columns. • v[[2]]: A matrix representing \(v'(x) \) with \(d \) columns.
`v_dom`	A boolean vector indicating the domain for ‘v0‘ and ‘v1‘.
`use0`	A boolean value indicating whether to select ‘v0‘. If TRUE, ‘v0‘ is selected based on ‘v_dom‘.
`use1`	A boolean value indicating whether to select ‘v1‘. If TRUE, ‘v1‘ is selected based on ‘v_dom‘.

Details

Select Domain of a Motif

Value

A list containing the selected portions of ‘v0‘ and ‘v1‘ (if applicable), with dimensions adjusted accordingly.

Author(s)

Marzia Angela Cremona & Francesca Chiaromonte

`.transform_list` *Transform List Structure*

Description

This function modifies a list of structures by checking for the presence of a "with_noise" field. If absent, it creates a new sub-list containing relevant fields.

Usage

```
.transform_list(lst, noise_str)
```

Arguments

`lst`	A list containing elements to be transformed.
`noise_str`	A string indicating the noise structure to apply.

Details

Transform List Structure

Value

The transformed list with updated structures.

`.transform_to_matrix` *Transform to Matrix*

Description

This function transforms a numeric vector into a matrix. If the input is a single number, it creates a 1x1 matrix. If the input is a vector, it creates a 1xN matrix. If the input is already a matrix, it returns it unchanged.

Usage

```
.transform_to_matrix(input)
```

Arguments

`input`	A numeric vector or matrix to be transformed.

Details

Transform Input to Matrix

- If the input is neither a numeric vector nor a matrix, an error is raised.

Value

A matrix representation of the input.

`add_error_to_motif` *Add Additive Error to Motif*

Description

This function adds an additive noise to a given motif based on specified parameters. The noise is generated as a percentage of the standard deviation of the motif.

Usage

```
add_error_to_motif(or_y, noise_str, start_point, end_point, k)
```

Arguments

`or_y`	A numeric vector representing the original motif values.
`noise_str`	A string or numeric vector indicating the structure of the noise to be added.
`start_point`	A numeric vector indicating the starting points of motifs.
`end_point`	A numeric vector indicating the ending points of motifs.
`k`	An integer representing the current iteration or motif index.

Details

Add Error to Motif

Value

A numeric vector of the motif values with added noise.

`add_motif` *Add Motif to Base Curve*

Description

This function adds specified motifs to a base curve, adjusting coefficients and applying noise as needed. It also computes the signal-to-noise ratio (SNR) for the motifs.

Usage

```
add_motif(
  base_curve,
  mot_pattern,
  mot_len,
  dist_knots,
  mot_order,
  mot_weights,
  noise_str,
  only_der,
  coeff_min_shift,
  coeff_max_shift
)
```

Arguments

`base_curve`	A list representing the base curve structure with coefficients and basis.
`mot_pattern`	A data frame containing information about the motif patterns to add.
`mot_len`	A matrix specifying the lengths of the motifs.
`dist_knots`	A numeric value indicating the distance between knots for the motifs.
`mot_order`	An integer specifying the order of the B-spline for the motifs.
`mot_weights`	A list of numeric vectors containing weights for each motif.
`noise_str`	A list of structures defining the noise to be added to motifs.
`only_der`	A logical value indicating whether to add only derivatives.
`coeff_min_shift`	A numeric value for the minimum shift to apply to coefficients.
`coeff_max_shift`	A numeric value for the maximum shift to apply to coefficients.

Details

Add Motif to Base Curve

Value

A list containing the updated base curve, background information, motifs with and without noise, and SNR data.

`cluster_candidate_motifs` *Cluster Candidate Motifs*

Description

This function clusters candidate motifs based on their distances and computes group-specific radii for motif clusters. It utilizes K-nearest neighbors (KNN) for determining a global radius and evaluates overlaps among motifs. The function supports parallel computation for efficiency.

Usage

```
cluster_candidate_motifs(
  filter_candidate_motifs_results,
  motif_overlap = 0.6,
  k_knn = 3,
  votes_knn_Rall = 0.5,
  votes_knn_Rm = 0.5,
  worker_number = NULL
)
```

Arguments

`filter_candidate_motifs_results`	A list containing results from filtering candidate motifs, including various components like ‘Y0‘, ‘Y1‘, ‘V0_clean‘, ‘V1_clean‘, ‘D_clean‘, and more, which are essential for the clustering process.
`motif_overlap`	A numeric value representing the minimum proportion of overlap required between motifs to be considered similar (default is 0.6).
`k_knn`	An integer specifying the number of nearest neighbors to consider when determining the global radius (default is 3).
`votes_knn_Rall`	A numeric value indicating the threshold for KNN voting when determining the global radius (default is 0.5).
`votes_knn_Rm`	A numeric value indicating the threshold for KNN voting when determining group-specific radii (default is 0.5).
`worker_number`	An optional integer specifying the number of parallel workers to use. If NULL, it defaults to the number of available cores minus one.

Details

This function performs the following steps: 1. Sets up parallel jobs based on the specified ‘worker_number‘. 2. Prepares input data based on the type of distance measure used. 3. Computes distances between motifs. 4. Determines a global radius (‘R_all‘) using KNN classification. 5. Clusters motifs and determines group-specific radii (‘R_m‘) for each cluster.

Value

A list containing: - ‘VV_D‘: Matrix of distances between motifs. - ‘VV_S‘: Matrix of shifts between motifs. - ‘k_knn‘: The value of K used in KNN. - ‘votes_knn_Rall‘: Voting threshold for the global radius. - ‘R_all‘: The global radius determined from the clustering process. - ‘hclust_res‘: Result of hierarchical clustering (if applicable). - ‘votes_knn_Rm‘: Voting threshold for group-specific radius. - ‘R_m‘: Vector of group-specific radii for each cluster. - All components from the input ‘filter_candidate_motifs_results‘.

Examples

```
## Not run:
# Example usage
results <- list(Y0 = ..., Y1 = ..., V0_clean = ..., V1_clean = ...,
                D_clean = ..., diss = "d0_d1_L2", alpha = 0.5)
clustered_results <- cluster_candidate_motifs(results, motif_overlap = 0.6,
                                              k_knn = 3, votes_knn_Rall = 0.5,
                                              votes_knn_Rm = 0.5, worker_number = 2)
print(clustered_results)

## End(Not run)
```

cluster_candidate_motifs_plot *cluster_candidate_motifs_plot*

Description

This function generates plots to visualize the results of the ‘cluster_candidate_motifs‘ function. It provides insights into the distances between motifs and curves, group-specific radii, and displays aligned motifs for each cluster.

Usage

```
cluster_candidate_motifs_plot(
  cluster_candidate_motifs_results,
  ylab = "",
  R_all = cluster_candidate_motifs_results$R_all,
  R_m = NULL,
  ask = TRUE
)
```

Arguments

`cluster_candidate_motifs_results`	A list containing the output of the ‘cluster_candidate_motifs‘ function, which includes clustering results and distances between motifs and curves.
`ylab`	A character string specifying the label for the y-axis in the plots. Default is an empty string.
`R_all`	A numeric value representing the global radius. This is used to cut the dendrogram, requiring groups to be more than twice ‘R_all‘ apart. The default is set to the ‘R_all‘ component of the input results.
`R_m`	A numeric vector with group-specific radii. The length of this vector must match the number of clusters obtained by cutting the dendrogram at height ‘2 * R_all‘. If set to NULL, the radius ‘R_m‘ is determined within each group based on the distances between motifs of the same group and all curves.
`ask`	A logical value indicating whether the user should be prompted before a new plot is drawn. Default is TRUE.

Details

This function produces: - Histograms of distances between all motifs and all curves, highlighting the global radius. - A dendrogram of motifs cut at ‘2 * R_all‘, which allows visualization of the clustering structure. - Histograms of distances for motifs within specific clusters, comparing motifs from curves with and without the motifs. - Plots of aligned motifs for each cluster, showing their positions and distributions. - A scatter plot displaying the approximate average distance against the approximate frequency for motifs in each cluster.

Author(s)

Marzia Angela Cremona & Francesca Chiaromonte

compare_nodes *Compare Nodes*

Description

This function compares two nodes to determine if all elements of the first node (‘node_1‘) are present within the accolites of the second node (‘node_2‘). Accolites are defined as a set of elements retrieved from the ‘get_accolites‘ function, which operates on ‘node_1‘ within a specified window of data. The comparison is successful if all elements in ‘node_1‘ are found in the accolites of ‘node_2‘.

Usage

```
compare_nodes(node_1, node_2)
```

Arguments

`node_1`	A vector representing the first node whose elements will be compared against the accolites of ‘node_2‘.
`node_2`	A vector representing the second node, from which accolites will be derived for comparison.

Value

A logical value. The function returns ‘TRUE‘ if all elements of ‘node_1‘ are found in the accolites of ‘node_2‘. It returns ‘FALSE‘ otherwise.

Examples

```
## Not run:
  # Example usage
  node_1 <- c("A", "B", "C")
  node_2 <- c("A", "B", "C", "D", "E")
  result <- compare_nodes(node_1, node_2)
  print(result)  # Returns TRUE if all elements of node_1 are in node_2's accolades

## End(Not run)
```

discoverMotifs *Functional Motif Discovery*

Description

The ‘discoverMotifs‘ function facilitates the discovery of recurring patterns, or motifs, within functional data by employing two sophisticated algorithms: `ProbKMA` (Probabilistic K-means with Local Alignment) and `funBIalign`. These algorithms are designed to identify and cluster functional motifs across multiple curves, leveraging advanced clustering and alignment techniques to handle complex data structures.

`ProbKMA` integrates probabilistic clustering with local alignment strategies, enabling the detection of motifs that exhibit variability in both shape and position across different curves. This method is particularly adept at handling noisy data and motifs that may appear at varying scales or locations within the curves.

On the other hand, `funBIalign` utilizes hierarchical clustering based on mean squared residue scores to uncover motifs. This approach effectively captures the additive nature of functional motifs, considering both portion-specific adjustments and time-varying components to accurately identify recurring patterns.

By providing a flexible interface that accommodates different clustering paradigms, 'discoverMotifs' empowers users to perform robust motif discovery tailored to their specific data characteristics and analytical requirements. Whether opting for the probabilistic and alignment-focused `ProbKMA` or the hierarchical and residue-based `funBIalign`, users can leverage these methods to extract meaningful and interpretable motifs from their functional datasets.

Usage

```
discoverMotifs(
  Y0,
  method,
  stopCriterion,
  name,
  plot,
  probKMA_options = list(),
 funBIalign_options = list(portion_len = NULL, min_card = NULL, cut_off = NULL),
  worker_number = NULL
)
```

Arguments

`Y0`	A list containing N vectors (for univariate curves) or N matrices (for multivariate curves) representing the functional data.
`method`	A character string specifying the motif discovery algorithm to use. Acceptable values are "ProbKMA" for Probabilistic K-means with Local Alignment and "funBIalign" for Functional Bi-directional Alignment.
`stopCriterion`	A character string indicating the convergence criterion for the selected algorithm.
`name`	A character string specifying the name of the output directory where results will be saved.
`plot`	A logical value indicating whether to generate and save plots of the discovered motifs and clustering results.
`probKMA_options`	A list of options specific to the ProbKMA algorithm.
`funBIalign_options`	A list of options specific to the funBIalign algorithm.
`worker_number`	An integer specifying the number of CPU cores to utilize for parallel computations.

Details

The 'discoverMotifs' function dynamically switches between two advanced motif discovery algorithms based on the user's specification. Each algorithm employs distinct strategies to identify and cluster motifs within functional data, offering flexibility and adaptability to various analytical scenarios.

Value

A list containing the discovered motifs and their corresponding statistics, tailored to the selected method:

`motifs` A list of identified motifs, each containing the motif's representative curve, membership probabilities, and alignment information.

`statistics` Detailed statistics for each motif, including measures such as silhouette scores, variance explained, and other relevant metrics that quantify the quality and significance of the discovered motifs.

`parameters` The final parameters and configurations used during the motif discovery process, providing transparency and facilitating reproducibility of the results.

`plots` If `plot = TRUE`, this component contains the generated plots visualizing the motifs and their distribution across the functional data.

Theoretical Background for ProbKMA

`ProbKMA` is inspired by methodologies prevalent in bioinformatics, particularly those involving local alignment techniques extended from high-similarity seeds. This algorithm combines fuzzy clustering approaches with local alignment strategies to effectively minimize a generalized least squares functional. The minimization process can incorporate both the levels and derivatives of the curves through a Sobolev-based distance metric, enhancing the algorithm's sensitivity to both shape and rate changes in the data.

Throughout its iterative process, `ProbKMA` refines motif centers, membership probabilities, and alignment shifts, making it highly effective for capturing complex motif structures and motifs distributed across multiple curves. This ensures that the discovered motifs are both representative and robust against variations and noise within the functional data.

Theoretical Background for funBIalign

`funBIalign` models functional motifs as an additive combination of motif means, portion-specific adjustments, and time-varying components. The algorithm constructs a hierarchical dendrogram utilizing the generalized mean squared residue score (fMSR) to identify candidate motifs across curves.

A critical aspect of `funBIalign` is its post-processing step, which filters out redundant motifs and refines the final selection to ensure that only the most significant and representative motifs are retained. This hierarchical approach allows for a nuanced identification of motifs, capturing both broad and subtle patterns within the data.

Common Parameters

The following parameters are common to both `ProbKMA` and `funBIalign` algorithms:

`Y0` A list containing `N` vectors (for univariate curves) or `N` matrices (for multivariate curves) representing the functional data. Each curve is evaluated on a uniform grid, ensuring consistency across the dataset.

`method` A character string specifying the motif discovery algorithm to use. Acceptable values are `"ProbKMA"` for Probabilistic K-means with Local Alignment and `"funBIalign"` for Functional Bi-directional Alignment.

`stopCriterion` A character string indicating the convergence criterion for the selected algorithm. For `ProbKMA`, options include `"max"`, `"mean"`, or `"quantile"` based on the Bhattacharyya distance between memberships in successive iterations. For `funBIalign`, options are `"fMRS"` (functional Mean Squared Residue) or `"Variance"` to guide the ranking of motifs.

`name` A character string specifying the name of the output directory where results will be saved. This facilitates organized storage and easy retrieval of analysis results.

`plot` A logical value indicating whether to generate and save plots of the discovered motifs and clustering results. When set to `TRUE`, visualizations are produced to aid in the qualitative assessment of the motif discovery process.

`worker_number` An integer specifying the number of CPU cores to utilize for parallel computations. By default, the function uses the total number of available cores minus one, optimizing computational efficiency without overloading the system.

ProbKMA Options

The following parameters are specific to the `ProbKMA` algorithm:

`K` An integer or vector specifying the number of motifs to be discovered. It can be a single integer for uniform motif discovery or a vector for specifying different numbers of motifs.

`c` An integer or vector indicating the minimum motif lengths. This ensures that each discovered motif meets a specified minimum length requirement, maintaining the integrity of motif structures.

`c_max` An integer or vector specifying the maximum motif lengths, allowing control over the upper bounds of motif sizes to prevent excessively long motifs.

`diss` A character string defining the dissimilarity measure to use. Possible values include `"d0_L2"`, `"d1_L2"`, and `"d0_d1_L2"`, which determine how the algorithm quantifies differences between motifs based on level and derivative information.

`alpha` A numeric value between 0 and 1 that serves as a weight parameter between `d0_L2` and `d1_L2` when using `d0_d1_L2`. An `alpha` of 0 emphasizes `d0_L2`, while an `alpha` of 1 emphasizes `d1_L2`, allowing for balanced consideration of both metrics.

`w` A numeric vector specifying the weight for the dissimilarity index across different dimensions. All values must be positive, enabling the algorithm to prioritize certain dimensions over others based on their relative importance.

`m` A numeric value greater than 1 that acts as the weighting exponent in the least-squares functional method. This parameter influences the sensitivity of the algorithm to differences in motif alignment and membership probabilities.

`iter_max` An integer specifying the maximum number of iterations allowed for the algorithm to converge. This prevents excessive computation time by limiting the number of optimization steps.

`quantile` A numeric value representing the quantile probability used when `stopCriterion` is set to `"quantile"`. This determines the threshold for convergence based on the distribution of Bhattacharyya distances.

`tol` A numeric value specifying the tolerance level for convergence. The algorithm stops iterating if the change in the stop criterion falls below this threshold, ensuring precise and stable convergence.

`iter4elong` An integer indicating the number of iterations after which motif elongation is performed. If set to a value greater than `iter_max`, no elongation is performed. Motif elongation allows the algorithm to extend motifs to better fit the data.

`tol4elong` A numeric value defining the tolerance on the Bhattacharyya distance for motif elongation. This parameter controls how much the objective function can increase during elongation, ensuring that motif extensions do not degrade the overall fit.

`max_elong` A numeric value representing the maximum elongation allowed in a single iteration, expressed as a percentage of the motif length. This prevents excessive extension of motifs in any single step.

`trials_elong` An integer specifying the number of elongation trials (equispaced) on each side of the motif in a single iteration. Multiple trials enhance the robustness of motif elongation by exploring various extension possibilities.

`deltaJK_elong` A numeric value indicating the maximum relative increase in the objective function permitted during motif elongation. This ensures that elongation steps contribute positively to the motif fitting process.

`max_gap` A numeric value defining the maximum gap allowed in each alignment as a percentage of the motif length. This parameter controls the allowable discontinuity between aligned motifs, maintaining coherence in motif placement.

`iter4clean` An integer specifying the number of iterations after which motif cleaning is performed. If set to a value greater than `iter_max`, no cleaning is performed. Motif cleaning removes redundant or poorly fitting motifs to refine the final motif set.

`tol4clean` A numeric value representing the tolerance on the Bhattacharyya distance for motif cleaning. This parameter determines the threshold for identifying and removing redundant motifs during the cleaning process.

`quantile4clean` A numeric value specifying the dissimilarity quantile used for motif cleaning. This quantile determines which motifs are considered sufficiently dissimilar to be retained in the final set.

`return_options` A logical value indicating whether to return the options passed to the `ProbKMA` method. When set to `TRUE`, users receive detailed information about the algorithm's configuration, facilitating transparency and reproducibility.

`Y1` A list of derivative curves used if the dissimilarity measure `"d0_d1_L2"` is selected. These derivatives enhance the algorithm's ability to capture both shape and rate changes in the functional data.

`P0` An initial membership matrix (N x K), where N is the number of curves and K is the number of clusters. If set to `NULL`, a random matrix is generated, initiating the probabilistic clustering process.

`S0` An initial shift warping matrix (N x K). If set to `NULL`, a random matrix is generated to initialize the alignment process, allowing motifs to adapt to variations in the data.

`n_subcurves` An integer specifying the number of splitting subcurves used when the number of curves is equal to one. This parameter allows the algorithm to handle single-curve datasets by dividing them into manageable segments for motif discovery.

`sil_threshold` A numeric value representing the threshold to filter candidate motifs based on their silhouette scores. This ensures that only motifs with sufficient clustering quality are retained in the final results.

`set_seed` A logical value indicating whether to set a random seed for reproducibility. When set to `TRUE`, the function initializes the random number generator to ensure consistent results across multiple runs.

`seed` An integer specifying the random seed used for initialization when `set_seed` is `TRUE`. This parameter guarantees reproducibility of the clustering and alignment processes.

`exe_print` A logical value determining whether to print execution details for each iteration. When set to `TRUE`, users receive real-time feedback on the algorithm's progress, aiding in monitoring and debugging.

`V_init` A list of motif sets provided as specific initializations for clustering rather than using random initializations. The 'V_init' parameter allows users to provide a set of motifs as starting points for the algorithm, instead of relying on random initialization. If 'n_init' is specified as greater than the number of motifs given in 'V_init', the remaining initializations will be randomly generated. For example, if 'n_init = 10' but only 5 motif sets are given in 'V_init', the algorithm will use these 5 initializations and generate an additional 5 randomly.

`transformed` A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting 'transformed = TRUE' scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.

`n_init_motif` The number of initial motif sets from 'V_init' to be used directly as starting points in clustering. If 'n_init_motif' is set to a value larger than the number of motifs provided in 'V_init', additional initializations will be generated randomly to meet the specified number. For example, if 'n_init = 10' and 'n_init_motif = 5' with only 3 motif sets in 'V_init', the algorithm will use these 3 sets and generate 7 additional random initializations.

funBIalign Options

The following parameters are specific to the `funBIalign` algorithm:

`portion_len` An integer specifying the length of curve portions to align. This parameter controls the granularity of alignment, allowing the algorithm to focus on specific segments of the curves for motif discovery.

`min_card` An integer representing the minimum cardinality of motifs, i.e., the minimum number of motif occurrences required for a motif to be considered valid. This ensures that only motifs with sufficient representation across the dataset are retained.

`cut_off` A double that specifies the number of top-ranked motifs to keep based on the ranking criteria, facilitating focused visualization of the most significant motifs. In particular, all motifs that rank below the cut_off are retained.

See Also

ProbKMA: Probabilistic K-means with Local Alignment
funBIalign: Hierarchical Clustering with Mean Squared Residue Scores.

Examples

```
## Not run:
# Example 1: Discover motifs using ProbKMA

# Define dissimilarity measure and weight parameter
diss <- 'd0_d1_L2'
alpha <- 0.5

# Define number of motifs and their minimum lengths
K <- c(2, 3)
c <- c(61, 51)
n_init <- 10

# Load simulated data
data("simulated200")

# Perform motif discovery using ProbKMA
results <- funMoDisco::discoverMotifs(
  Y0 = simulated200$Y0,
  method = "ProbKMA",
  stopCriterion = "max",
  name = './results_ProbKMA_VectorData/',
  plot = TRUE,
  probKMA_options = list(
    Y1 = simulated200$Y1,
    K = K,
    c = c,
    n_init = n_init,
    diss = diss,
    alpha = alpha
  ),
  worker_number = NULL
)

# Modify silhouette threshold and re-run post-processing
results <- funMoDisco::discoverMotifs(
  Y0 = simulated200$Y0,
  method = "ProbKMA",
  stopCriterion = "max",
  name = './results_ProbKMA_VectorData/',
  plot = TRUE,
  probKMA_options = list(
    Y1 = simulated200$Y1,
    K = K,
    c = c,
    n_init = n_init,
    diss = diss,
    alpha = alpha,
    sil_threshold = 0.5
  ),
  worker_number = NULL
)

# Example 2: Discover motifs using funBIalign
results_funbialign <- funMoDisco::discoverMotifs(
  Y0 = simulated200$Y0,
  method = "funBIalign",
  stopCriterion = 'Variance',
  name = './results_FunBialign/',
  plot = TRUE,
  funBIalign_options = list(
    portion_len = 60,
    min_card = 3,
    cut_off = 1.0
  )
)

## End(Not run)
```

`filter_candidate_motifs`

Filter Candidate Motifs

Description

Filters the candidate motifs based on a specified threshold for the average silhouette index and a threshold for the size of the curves in the motif. This function is useful for refining the set of candidate motifs by removing those that do not meet the defined criteria.

Usage

```
filter_candidate_motifs(
  find_candidate_motifs_results,
  sil_threshold = 0.5,
  size_threshold = 2,
  K = find_candidate_motifs_results$K,
  c = find_candidate_motifs_results$c
)
```

Arguments

`find_candidate_motifs_results`	Output from the `find_candidate_motif` function, which contains the results of the motif discovery process.
`sil_threshold`	A numeric value representing the threshold for the average silhouette index. Values should be between -1 and 1.
`size_threshold`	An integer representing the threshold for the size of the motif, defined as the number of curves in the cluster. Should be at least 1.
`K`	A vector containing the numbers of motifs that must be considered. This should be a subset of `find_candidate_motifs_results$K`.
`c`	A vector with the minimum motif lengths that must be considered. This should be a subset of `find_candidate_motifs_results$c`.

Details

This function checks the provided thresholds for silhouette indices and motif sizes, ensuring they are valid before filtering the motifs. If any parameters are invalid, default values from the `find_candidate_motifs_results` are used.

The function then loads the necessary data files for each combination of motif numbers and sizes, filtering them using the silhouette criteria. The resulting motifs are sorted by their length in descending order, and the filtered results are returned in a structured list.

Value

A list containing the filtered candidate motifs and some ProbKMA options:

`V0_clean`	A vector of candidate motifs after filtering.
`V1_clean`	A vector of derived candidate motifs after filtering.
`D_clean`	A matrix of distances of candidate motifs from the curves after filtering.
`P_clean`	A matrix of probabilities of membership of the candidate motifs after filtering.
`Y0`	The original input data for the first curve.
`Y1`	The original input data for the derivative of the curve.
`diss`	The dissimilarity matrix used in motif discovery.
`alpha`	The weight parameter for the Sobolev distance.
`w`	Weights for the dissimilarity index across different dimensions.
`max_gap`	Maximum allowable gap between curves.

Examples

```
## Not run:
# Example usage
candidate_results <- find_candidate_motif(data)
filtered_motifs <- filter_candidate_motifs(candidate_results,
                                           sil_threshold = 0.6,
                                           size_threshold = 3)

# Inspect the cleaned candidate motifs
print(filtered_motifs$V0_clean)

## End(Not run)
```

`find_recommended_path` *Find Recommended Path in a Tree Structure*

Description

This function identifies a recommended path in a tree structure based on user-defined minimum criteria. It filters nodes based on the number of leaves and calculates scores for potential recommendations.

Usage

```
find_recommended_path(minidend, window_data, min_card)
```

Arguments

`minidend`	A tree structure representing the hierarchy of nodes, from which leaves will be derived for analysis.
`window_data`	A matrix or data frame containing data associated with each node.
`min_card`	An integer specifying the minimum number of leaves that must be present in a node for it to be considered for recommendation.

Details

The function follows these steps: 1. Extracts the leaves and heights of each node in the tree. 2. Filters nodes based on the minimum number of leaves specified by `min_card`. 3. Identifies nodes that are parents of the filtered nodes, marking them for deletion. 4. For each seed node, calculates crossing points with other seed nodes. 5. Calculates adjusted scores for nodes in the seed path using the C++ function `fMSR_adj`. 6. Recommends nodes based on calculated scores and returns a summary of results.

Value

A list containing: - `seed_path_info`: A data frame summarizing the valid seed nodes, their number of leaves, heights, and recommended nodes. - `seed_path_list`: A list containing the names of elements for each node in the seed path. - `score_path_list`: A list of scores adjusted for each node in the seed path. - `recommended_node_labels`: A list of labels for the recommended nodes. - `recommended_node_scores`: A numeric vector containing scores for the recommended nodes.

Examples

```
## Not run:
# Example usage
minidend <- create_tree_structure(...)  # Replace with actual tree creation function
window_data <- data.frame(...)  # Replace with actual data
result <- find_recommended_path(minidend, window_data, min_card = 3)
print(result)

## End(Not run)
```

`generateCurves` *Generate Functional Curves with Embedded Motifs*

Description

The ‘generateCurves‘ function is designed to create synthetic functional data by embedding predefined motifs into background curves. This is particularly useful for testing and benchmarking motif discovery and clustering algorithms in functional data analysis. By allowing the incorporation of various noise types and controlled motif placements, the function provides a flexible framework for simulating realistic scenarios where motifs may or may not be noisy.

The function supports two types of noise addition:

- **Pointwise Noise**: Adds noise directly to the data points of the curves, simulating random fluctuations or measurement errors.
- **Coefficient Noise**: Perturbs the coefficients of the basis functions used to represent the curves, allowing for smoother variations and controlled distortions.

Additionally, ‘generateCurves‘ allows for the specification of vertical shifts to motifs, enabling the simulation of motifs appearing at different baseline levels within the curves. The function ensures that all generated subcurves meet the minimum motif length requirement, maintaining the integrity of the embedded motifs.

This function is integral to the ‘funMoDisco‘ package’s motif simulation capabilities, providing users with the ability to create complex functional datasets tailored to their specific research or testing needs.

Usage

```
generateCurves(
  object,
  noise_type = NULL,
  noise_str = NULL,
  seed_background = 777,
  seed_motif = 43213,
  only_der = TRUE,
  coeff_min_shift = -10,
  coeff_max_shift = 10
)
```

Arguments

- `object` An S4 object of class ‘motifSimulation‘ that has been previously constructed using the ‘motifSimulationBuilder‘ function. This object encapsulates all necessary parameters and configurations for curve and motif generation (mandatory).
- `noise_type` A character string specifying the type of noise to add to the curves. Acceptable values are ‘’pointwise’‘ for adding noise directly to data points or ‘’coeff’‘ for perturbing the coefficients of the basis functions (mandatory).
- `noise_str` A list detailing the structure and magnitude of the noise to be added for each motif.
  - If ‘noise_type‘ is ‘’pointwise’‘, ‘noise_str‘ should contain vectors or matrices indicating the noise level for each motif.
  - If ‘noise_type‘ is ‘’coeff’‘, ‘noise_str‘ should include individual values or vectors representing the noise to be applied to the coefficients.

  This parameter allows fine-grained control over the noise characteristics applied to each motif (mandatory).
- `seed_background` An integer value setting the seed for the random number generator used in background curve generation. This ensures reproducibility of the background curves. Default is ‘777‘.
- `seed_motif` An integer value setting the seed for the random number generator used in motif generation. This ensures reproducibility of the motif embedding process. Default is ‘43213‘.
- `only_der` A logical value indicating whether to apply only derivative-based modifications to the motifs (‘TRUE‘) or to add a vertical shift in addition to derivative modifications (‘FALSE‘). Setting to ‘FALSE‘ introduces vertical shifts to each motif instance, allowing motifs to appear at different baseline levels within the curves. Default is ‘TRUE‘.
- `coeff_min_shift` A numeric value specifying the minimum vertical shift to be applied to motifs when ‘only_der‘ is set to ‘FALSE‘. This parameter controls the lower bound of the vertical displacement of motifs. Default is ‘-10‘.
- `coeff_max_shift` A numeric value specifying the maximum vertical shift to be applied to motifs when ‘only_der‘ is set to ‘FALSE‘. This parameter controls the upper bound of the vertical displacement of motifs. Default is ‘10‘.

Value

A list containing the following components:

- **basis**: The basis functions used to represent the curves.
- **background**: A list containing the coefficients and the background curves without any motifs.
- **no_noise**: A list containing the coefficients and the background curves with embedded motifs but without added noise.
- **with_noise**: A list containing the noise structure and the curves with embedded motifs and added noise.
- **SNR**: A list of Signal-to-Noise Ratio (SNR) metrics calculated for each motif within each curve, useful for assessing the quality of motif embedding.

Examples

```
## Not run:
# Example 0: Special case with no motifs
mot_len <- 100
mot_details <- NULL  # or list()
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details)
curves <- funMoDisco::generateCurves(builder)

# Example 1: Set the motif position and add pointwise noise
# Define motif positions and their respective curves
motif_str <- rbind.data.frame(
  c(1, 1, 20),
  c(2, 1, 2),
  c(2, 7, 1),
  c(2,17,1)
)
names(motif_str) <- c("motif_id", "curve", "start_break_pos")

# Define motif details
mot1 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = motif_str %>% filter(motif_id == 1)
)
mot2 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = motif_str %>% filter(motif_id == 2)
)
mot_details <- list(mot1, mot2)

# Define noise structure for pointwise noise
noise_str <- list(
  rbind(rep(2, 100), rep(c(rep(0.1, 50), rep(2, 50)), 1)),
  rbind(rep(0.0, 100), rep(0.5, 100))
)

# Build the simulation object
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details, distribution = 'beta')

# Generate curves with pointwise noise
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str = noise_str)

# Example 2: Set the motif position and add coefficient noise
# Define noise structure for coefficient noise
noise_str <- list(c(0.1, 1.0, 5.0), c(0.0, 0.0, 0.0))

# Generate curves with coefficient noise without vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)

# Example 3: Random motif positions and add pointwise noise
mot1 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = 5
)
```

```
mot2 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = 6
)
mot_details <- list(mot1, mot2)

# Define noise structure for pointwise noise
noise_str <- list(
  rbind(rep(2, 100)),
  rbind(rep(0.5, 100))
)

# Build the simulation object
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details, distribution = 'beta')

# Generate curves with pointwise noise and vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str, only_der = FALSE)

# Example 4: Random motif positions and add coefficient noise
# Define noise structure for coefficient noise
noise_str <- list(c(0.1, 5.0, 10.0), c(0.1, 5.0, 10.0))

# Generate curves with coefficient noise and vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)

## End(Not run)
```

---

generateCurves,motifSimulation-method
Generate Functional Curves with Embedded Motifs

Description

The ‘generateCurves‘ function is designed to create synthetic functional data by embedding predefined motifs into background curves. This is particularly useful for testing and benchmarking motif discovery and clustering algorithms in functional data analysis. By allowing the incorporation of various noise types and controlled motif placements, the function provides a flexible framework for simulating realistic scenarios where motifs may or may not be noisy.

The function supports two types of noise addition:

- **Pointwise Noise**: Adds noise directly to the data points of the curves, simulating random fluctuations or measurement errors.
- **Coefficient Noise**: Perturbs the coefficients of the basis functions used to represent the curves, allowing for smoother variations and controlled distortions.

Additionally, ‘generateCurves‘ allows for the specification of vertical shifts to motifs, enabling the simulation of motifs appearing at different baseline levels within the curves. The function ensures that all generated subcurves meet the minimum motif length requirement, maintaining the integrity of the embedded motifs.

This function is integral to the ‘funMoDisco‘ package’s motif simulation capabilities, providing users with the ability to create complex functional datasets tailored to their specific research or testing needs.

Usage

```
## S4 method for signature 'motifSimulation'
generateCurves(
  object,
  noise_type = NULL,
  noise_str = NULL,
  seed_background = 777,
  seed_motif = 43213,
  only_der = TRUE,
  coeff_min_shift = -10,
  coeff_max_shift = 10
)
```

Arguments

`object` An S4 object of class ‘motifSimulation‘ that has been previously constructed using the ‘motifSimulationBuilder‘ function. This object encapsulates all necessary parameters and configurations for curve and motif generation (mandatory).

`noise_type` A character string specifying the type of noise to add to the curves. Acceptable values are ‘’pointwise’‘ for adding noise directly to data points or ‘’coeff’‘ for perturbing the coefficients of the basis functions (mandatory).

`noise_str` A list detailing the structure and magnitude of the noise to be added for each motif.

- If ‘noise_type‘ is ‘’pointwise’‘, ‘noise_str‘ should contain vectors or matrices indicating the noise level for each motif.
- If ‘noise_type‘ is ‘’coeff’‘, ‘noise_str‘ should include individual values or vectors representing the noise to be applied to the coefficients.

This parameter allows fine-grained control over the noise characteristics applied to each motif (mandatory).

`seed_background` An integer value setting the seed for the random number generator used in background curve generation. This ensures reproducibility of the background curves. Default is ‘777‘.

`seed_motif` An integer value setting the seed for the random number generator used in motif generation. This ensures reproducibility of the motif embedding process. Default is ‘43213‘.

`only_der` A logical value indicating whether to apply only derivative-based modifications to the motifs (‘TRUE‘) or to add a vertical shift in addition to derivative modifications (‘FALSE‘). Setting to ‘FALSE‘ introduces vertical shifts to each motif instance, allowing motifs to appear at different baseline levels within the curves. Default is ‘TRUE‘.

`coeff_min_shift` A numeric value specifying the minimum vertical shift to be applied to motifs when ‘only_der‘ is set to ‘FALSE‘. This parameter controls the lower bound of the vertical displacement of motifs. Default is ‘-10‘.

`coeff_max_shift` A numeric value specifying the maximum vertical shift to be applied to motifs when ‘only_der‘ is set to ‘FALSE‘. This parameter controls the upper bound of the vertical displacement of motifs. Default is ‘10‘.

Value

A list containing the following components:

- **basis**: The basis functions used to represent the curves.
- **background**: A list containing the coefficients and the background curves without any motifs.
- **no_noise**: A list containing the coefficients and the background curves with embedded motifs but without added noise.
- **with_noise**: A list containing the noise structure and the curves with embedded motifs and added noise.
- **SNR**: A list of Signal-to-Noise Ratio (SNR) metrics calculated for each motif within each curve, useful for assessing the quality of motif embedding.

Examples

```
## Not run:
# Example 0: Special case with no motifs
mot_len <- 100
mot_details <- NULL  # or list()
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details)
curves <- funMoDisco::generateCurves(builder)

# Example 1: Set the motif position and add pointwise noise
# Define motif positions and their respective curves
motif_str <- rbind.data.frame(
  c(1, 1, 20),
  c(2, 1, 2),
  c(2, 7, 1),
  c(2,17,1)
)
names(motif_str) <- c("motif_id", "curve", "start_break_pos")

# Define motif details
mot1 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = motif_str %>% filter(motif_id == 1)
)
mot2 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = motif_str %>% filter(motif_id == 2)
)
mot_details <- list(mot1, mot2)

# Define noise structure for pointwise noise
noise_str <- list(
  rbind(rep(2, 100), rep(c(rep(0.1, 50), rep(2, 50)), 1)),
  rbind(rep(0.0, 100), rep(0.5, 100))
)

# Build the simulation object
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details, distribution = 'beta')

# Generate curves with pointwise noise
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str = noise_str)
```

```
# Example 2: Set the motif position and add coefficient noise
# Define noise structure for coefficient noise
noise_str <- list(c(0.1, 1.0, 5.0), c(0.0, 0.0, 0.0))

# Generate curves with coefficient noise without vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)

# Example 3: Random motif positions and add pointwise noise
mot1 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = 5
)
mot2 <- list(
  "len" = mot_len,
  "coeffs" = NULL,
  "occurrences" = 6
)
mot_details <- list(mot1, mot2)

# Define noise structure for pointwise noise
noise_str <- list(
  rbind(rep(2, 100)),
  rbind(rep(0.5, 100))
)

# Build the simulation object
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details, distribution = 'beta')

# Generate curves with pointwise noise and vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str, only_der = FALSE)

# Example 4: Random motif positions and add coefficient noise
# Define noise structure for coefficient noise
noise_str <- list(c(0.1, 5.0, 10.0), c(0.1, 5.0, 10.0))

# Generate curves with coefficient noise and vertical shifts
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)

## End(Not run)
```

---

generate_background_curve
Generate Background Curve

Description

This function generates a background curve using B-splines with specified parameters, including knots, order, and weights. Optionally, additive noise can be added to the curve.

Usage

```
generate_background_curve(len, dist_knots, norder, weights, add_noise)
```

Arguments

`len`	An integer indicating the length of the curve to generate.
`dist_knots`	A numeric value indicating the distance between knots.
`norder`	An integer specifying the order of the B-spline.
`weights`	A numeric vector containing the coefficients for the B-spline.
`add_noise`	A logical value indicating whether to add Gaussian noise to the curve.

Details

Generate Background Curve

Value

A list containing the generated basis, coefficients, and curve values with or without noise.

`generate_curve_vector` *Generate Curve Vector from FD Object*

Description

This function generates a curve vector from a functional data (fd) object, with a specified step size and optional derivative.

Usage

```
generate_curve_vector(fd_curve, step_by = 1, Lfdobj = 0)
```

Arguments

`fd_curve`	A functional data object from which to generate the curve.
`step_by`	A numeric value indicating the step size for generating the curve.
`Lfdobj`	A numeric value indicating the order of the derivative to compute. Default is 0 (no derivative).

Details

Generate Curve Vector

Value

A numeric vector representing the generated curve.

get_accolites *Get Accolites for a Given Leaf Label*

Description

This function retrieves the accolites (adjacent elements) of a specified leaf label from a dataset. Accolites are defined as the elements that overlap with the specified portion length around the leaf. The function can filter accolites based on their origin curve in cases where multiple curves are present.

Usage

```
get_accolites(leaf_label, window_data, portion_len, multiple)
```

Arguments

`leaf_label`	A character string representing the label of the leaf for which to find accolites.
`window_data`	A data frame or matrix where the row names correspond to the labels of elements, containing the data from which accolites will be extracted.
`portion_len`	An integer specifying the total length of the portion used to define the accolites. The overlap is computed as half of this length.
`multiple`	A logical indicating whether to check for multiple curves. If TRUE, the function filters out accolites that do not originate from the same curve as the specified leaf label.

Details

The function works as follows: 1. It calculates the index of the specified leaf label within the provided 'window_data'. 2. Determines the range of indices representing the accolites by calculating the overlap based on 'portion_len'. 3. Retrieves the corresponding leaf labels from the 'window_data'. 4. If the 'multiple' argument is TRUE, it checks if the accolites come from the same curve as the leaf label, removing those that do not.

Value

A character vector containing the labels of the identified accolites. If no accolites are found, the function returns an empty vector.

Examples

```
## Not run:
# Example usage
window_data <- data.frame(matrix(ncol = 3, nrow = 6))
rownames(window_data) <- c("1_1", "1_2", "1_3", "2_1", "2_2", "2_3")
result <- get_accolites("1_2", window_data, portion_len = 4, multiple = TRUE)
print(result)

## End(Not run)
```

get_minidend — *Generate Minimum Dendrogram from Hierarchical Clustering*

Description

This function generates a minimum dendrogram by performing hierarchical clustering on an input distance matrix. The function identifies an optimal height cut based on the largest gap in cluster heights and returns the resulting dendrogram, allowing for further analysis of clustering structures.

Usage

```
get_minidend(adj_fMSR)
```

Arguments

`adj_fMSR`	A numeric matrix or a distance object representing the dissimilarity matrix computed from functional data. This matrix is used to perform hierarchical clustering.

Details

The function performs the following steps: 1. Uses ‘fastcluster::hclust‘ to perform hierarchical clustering on the provided dissimilarity matrix using the "complete" method. 2. Extracts the heights of the clusters and identifies the largest gap in heights to determine the optimal cut position for the dendrogram. 3. Cuts the dendrogram at the identified height and returns the lower part of the cut.

Value

A dendrogram object representing the clustered data. The dendrogram is cut at the identified height, resulting in a tree structure that can be used for further analysis or visualization.

Examples

```
## Not run:
# Example usage
# Create a sample adjacency matrix
sample_matrix <- matrix(runif(100), nrow = 10)
adj_fMSR <- dist(sample_matrix) # Convert to a distance object
minidend <- get_minidend(adj_fMSR)
plot(minidend) # Visualize the resulting dendrogram

## End(Not run)
```

`get_parents` *Get Parent Nodes from a Given Node*

Description

This function identifies the parent nodes of a given node in a hierarchical structure. It checks which nodes in the provided list contain the specified node, thereby determining its parent nodes.

Usage

```
get_parents(node, node_list)
```

Arguments

`node`	A vector representing a single node whose parents are to be found. It is expected to contain the elements that may be present in the parent nodes.
`node_list`	A list of vectors, where each vector represents a node in the hierarchical structure. The function will search through these nodes to find parents of the specified node.

Details

The function works by applying a logical check across the 'node_list'. It returns the indices of all nodes that contain all elements of the specified node vector. If no parent nodes are found, the function will return an empty integer vector.

Value

A numeric vector containing the indices of the parent nodes in the 'node_list' that include the specified node.

Examples

```
## Not run:
# Example usage
node_list <- list(c(1, 2), c(2, 3), c(1, 3, 4), c(4, 5))
node <- c(2, 3)
parents <- get_parents(node, node_list)
print(parents) # Should return the index of the parent node(s) that contain 2 and 3

## End(Not run)
```

get_path_complete *Get Complete Paths from a Dendrogram*

Description

This function computes recommended paths from a dendrogram structure using parallel processing. It utilizes the ‘find_recommended_path‘ function to identify optimal paths based on a minimum cardinality constraint, distributing the computation across multiple worker nodes.

Usage

```
get_path_complete(minidend, window_data, min_card, worker_number)
```

Arguments

`minidend`	A dendrogram structure from which to derive paths. This should be created from a hierarchical clustering result.
`window_data`	A data frame or matrix containing the data associated with the nodes in the dendrogram. This data is used for path recommendations.
`min_card`	An integer specifying the minimum number of leaves (or nodes) that must be present in a path for it to be considered valid.
`worker_number`	An integer representing the number of worker nodes to be used for parallel processing.

Details

The function creates a cluster of worker nodes, loads necessary libraries, and exports required variables and functions to each worker. It then applies the ‘find_recommended_path‘ function in parallel to the leaves of the provided dendrogram, gathering results into a single list. Finally, the cluster is stopped, and the results are returned.

Value

A list where each element contains the recommended paths for the corresponding node in the dendrogram. Each path includes information about the nodes and their associated scores.

Examples

```
## Not run:
# Example usage
library(fastcluster)
library(dplyr)

# Assuming `adj_fMSR` is defined and `window_data` is prepared
minidend <- get_minidend(adj_fMSR) # Get the dendrogram
paths <- get_path_complete(minidend, window_data, min_card = 5, worker_number = 4)
print(paths) # Output the computed paths

## End(Not run)
```

`initialChecks` *Initial Checks for ProbKMA*

Description

This function performs various input checks on the parameters provided by the user to ensure they are valid for running the ProbKMA algorithm. It verifies the structure and content of the input data, including curves, derivatives, initial membership probabilities, shift matrices, and various parameters.

Usage

```
initialChecks(Y0, Y1, P0, S0, params, diss, V_init)
```

Arguments

`Y0`	A list of matrices or vectors representing the curves.
`Y1`	A list of matrices or vectors representing the derivatives of the curves.
`P0`	A numeric matrix representing initial membership probabilities. Rows correspond to curves, and columns correspond to clusters.
`S0`	A numeric matrix representing the initial shift matrix.
`params`	A list containing various parameters for ProbKMA, including: • `standardize` Logical indicating whether to standardize the curves. • `K` Number of motifs. • `c` Minimum motif length. • `c_max` Maximum motif length. • `iter_max` Maximum number of iterations. • `quantile` Quantile value for stopping criterion. • `alpha` A numeric value related to the dissimilarity measure. • `w` Weights used in the algorithm. • `stopCriterion` Stopping criterion for the algorithm. • `m` Weighting exponent. • `tol` Tolerance level for stopping criteria. • `iter4elong` Maximum iterations for elongation. • `tol4elong` Tolerance for elongation. • `max_elong` Maximum elongation allowed. • `trials_elong` Number of trials for elongation. • `deltaJK_elong` Threshold for elongation. • `max_gap` Maximum gap allowed. • `iter4clean` Number of iterations for cleaning. • `tol4clean` Tolerance for cleaning. • `quantile4clean` Quantile for cleaning. • `return_options` Options for returning results. • `seed` Seed for random number generation. • `exe_print` Boolean to control printing of execution messages. • `set_seed` Boolean to control whether to set a random seed.

- `transformed` A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting ‘transformed = TRUE‘ scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.
- `n_threads` Number of threads for parallel processing.

`diss`	A character string indicating the type of dissimilarity measure to be used. Possible values are: `'d0_L2'`, `'d1_L2'`, `'d0_d1_L2'`.
`V_init`	A list containing initial values for the clusters. If provided, it must match the expected structure based on `K`.

Value

A list containing:

`FuncData`	A list of processed curves and derivatives after performing the checks.
`Parameters`	A list of validated parameters ready for use in initializing the ProbKMA object.

`motifSimulation-class` *motifSimulationS4Class*

Description

The ‘motifSimulation‘ class is an S4 class designed for simulating functional data with embedded motifs. This class is essential for modeling various aspects of the data, such as the number of curves, motif details, curve characteristics, and knot-based spline definitions. It allows users to generate realistic synthetic functional data to test and benchmark motif discovery algorithms like ProbKMA and funBIalign.

Slots

`N` A numeric value representing the number of curves to be simulated. This parameter controls the number of functional curves generated in the simulation.

`mot_details` A list containing details of the motifs to be embedded within the curves. Each motif detail specifies attributes such as motif shape, location, and frequency of occurrence across the simulated curves.

`motifs_in_curves` A list specifying which curves contain motifs and the positions where they are embedded. This slot allows precise control over the placement of motifs in the generated curves, enabling flexible motif-to-curve assignments.

`distribution` A character string or numeric value representing the distribution of the weights for the motifs. This slot defines the distribution used to generate the weight coefficients for the motifs, influencing their amplitude in the simulated data. Accepted values include any distribution supported in R or a custom vector provided by the user.

`dist_knots` A numeric value indicating the distance between knots in the spline representation of the curves. This parameter is crucial for defining the smoothness of the generated curves and the precision of the spline-based motif embedding process.

`len` A numeric value representing the length of the generated curves. This parameter determines the number of time points or the granularity of the data, allowing for high-resolution simulations.

`norder` A numeric value specifying the order of the B-spline used in the functional data representation. Higher-order splines provide smoother representations of the curves, while lower-order splines offer more flexible curve shapes.

`coeff_min` A numeric value specifying the minimum coefficient value for the spline-based curve generation. This controls the lower bound of the weight coefficients applied to the basis functions, influencing the range of curve amplitudes.

`coeff_max` A numeric value specifying the maximum coefficient value for the spline-based curve generation. This controls the upper bound of the weight coefficients applied to the basis functions, setting the maximum amplitude for the curves.

`min_dist_motifs` A numeric value indicating the minimum distance between motifs in the simulated curves. This ensures that motifs are not placed too closely, preserving their distinctiveness and reducing overlap during the simulation.

`motifSimulationApp` *motifSimulationApp: A Shiny-Based GUI for Motif Simulation*

Description

The ‘motifSimulationApp‘ is a Shiny-based graphical user interface (GUI) designed to simplify the execution of the ‘motifSimulation‘ functions. The app allows users to interact with all motif simulation features in an intuitive and user-friendly manner, offering a seamless experience for generating and analyzing synthetic functional data with embedded motifs. It consistently provides summary plots to enhance data visualization and facilitate analysis.

Usage

```
motifSimulationApp(noise_str, mot_details)
```

Arguments

`noise_str` A list corresponding to the number of motifs, specifying the noise structure to be applied to each motif. Users can choose between two noise types:

- `'pointwise'` Allows the specification of noise as a list of vectors or matrices. Each element of the list defines the amount of noise applied pointwise to the corresponding motif.
- `'coeff'` Enables the specification of noise in terms of coefficients, where a list of individual values or vectors can be provided to define the noise level for each motif.

`mot_details` A list outlining the specifications for the motifs to be embedded within the functional curves. Each motif is characterized by its:

- `length` The length of the motif (number of points or time steps).
- `coefficients` An optional set of coefficients that can be provided to define the motif’s shape.
- `occurrences` The number of occurrences of each motif within the curves. These can be specified by exact positions within the curves or by providing a total count of occurrences, in which case the algorithm will randomly assign the positions.

Details

The app provides an accessible platform for users to experiment with different motif definitions, noise structures, and curve characteristics. By interacting with the app, users can simulate functional data, visualize the results in real-time, and adjust the parameters accordingly to fine-tune the simulation process.

Examples

```
## Not run:
# Launch the motifSimulationApp with specified noise structure and motif details
funMoDisco::motifSimulationApp(noise_str, mot_details)

## End(Not run)
```

---

motifSimulationBuilder

Create motifSimulation Object

Description

It represents the constructor of the S4 class 'motifSimulation'..

Usage

```
motifSimulationBuilder(
  N,
  len,
  mot_details,
  norder = 3,
  coeff_min = -15,
  coeff_max = 15,
  dist_knots = 10,
  min_dist_motifs = NULL,
  distribution = "unif"
)
```

Arguments

`N`	An integer specifying the number of background curves to be generated (mandatory).
`len`	An integer specifying the length of the background curves (mandatory).
`mot_details`	A list outlining the definitions of the motifs to be included. Each motif is characterized by its length, a set of coefficients that may be optionally specified, and the number of occurrences. These occurrences can be indicated either by specific positions within the curves or by a total count. In the latter case, the algorithm will randomly position the motifs throughout the curves (mandatory).
`norder`	An integer specifying the order of the B-splines (default = 3).
`coeff_min`	Additive coefficients to be incorporated into the generation of coefficients for the background curves (default = -15).

`coeff_max`	Additive coefficients to be incorporated into the generation of coefficients for the background curves (default = 15).
`dist_knots`	An integer specifying the distance between two consecutive knots (default = 10).
`min_dist_motifs`	An integer specifying the minimum distance between two consecutive motifs embedded in the same curve (default = 'norder' * 'dist_knots').
`distribution`	A character string specifying the distribution from which the coefficients of the background curves are generated. You can choose between a uniform distribution or a beta distribution. Alternatively, you can pass a vector representing the empirical distribution from which you wish to sample (default = "unif").

Value

An object of class motifSimulation

Examples

```
## Not run:
mot_len <- 100
motif_str <- rbind.data.frame(c(1, 1, 20),
                              c(2, 1, 2),
                              c(1, 3, 1),
                              c(1, 2, 1),
                              c(1, 2, 15),
                              c(1, 4, 1),
                              c(2, 5, 1),
                              c(2, 7, 1),
                              c(2,17,1))

names(motif_str) <- c("motif_id", "curve","start_break_pos")

mot1 <- list("len" = mot_len, #length
             "coeffs" = NULL, # weights for the motif
             "occurrences" = motif_str %>% filter(motif_id == 1))

mot2 <- list("len" = mot_len,
             "coeffs" = NULL,
             "occurrences" = motif_str %>% filter(motif_id == 2))

mot_details <- list(mot1,mot2)

# MATRIX ERROR
noise_str <- list(rbind(rep(2, 100)),
                  rbind(rep(0.0, 100)))

builder <- funMoDisco::motifSimulationBuilder(N = 20,len = 300,mot_details,
                                              distribution = 'beta')

## End(Not run)
```

motifs_search *Motif Search in Curves*

Description

The ‘motifs_search‘ function identifies and ranks motifs within a set of curves based on their frequencies and dissimilarity measures. It processes candidate motifs clustered from hierarchical clustering results, selects optimal motifs within each cluster, and determines their occurrences in the original curves. The function supports parallel processing to enhance computational efficiency and offers flexibility in handling different dissimilarity metrics and motif selection criteria.

Usage

```
motifs_search(
  cluster_candidate_motifs_results,
  R_all = cluster_candidate_motifs_results$R_all,
  R_m = NULL,
  different_R_m_finding = FALSE,
  R_m_finding = NULL,
  use_real_occurrences = FALSE,
  length_diff = Inf,
  worker_number = NULL
)
```

Arguments

`cluster_candidate_motifs_results`
: A list containing the output from the ‘cluster_candidate_motifs‘ function. This list must include elements such as:

  Y0 A list of matrices representing the original curves.

  Y1 A list of matrices representing the derivatives of the curves (if applicable).

  V0_clean A list of candidate motifs derived from ‘Y0‘.

  V1_clean A list of candidate motifs derived from ‘Y1‘ (if applicable).

  D_clean A matrix of dissimilarity measures between motifs and curves.

  P_clean A matrix indicating positive matches (e.g., presence of motifs in curves).

  hclust_res A hierarchical clustering object obtained from ‘hclust‘.

  R_all A numeric value representing the global radius used for dendrogram cutting.

  w A numeric vector of weights for the dissimilarity index across different dimensions.

  transformed A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting ‘transformed = TRUE‘ scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.

  max_gap A numeric value defining the maximum allowed gap in distances for cluster separation.

k_knn An integer specifying the number of neighbors for K-Nearest Neighbors classification.

votes_knn_Rm A numeric value defining the probability threshold for KNN-based radius determination.

c A numeric vector specifying the minimum number of overlapping elements required for motif validation.

`R_all` A numeric value representing the global radius used to cut the dendrogram, ensuring that clusters are at least twice this radius apart. This parameter defines the grouping of motifs into clusters.

`R_m` A numeric vector containing group-specific radii used to identify motif occurrences within each cluster. The length of this vector must match the number of clusters obtained by cutting the dendrogram at a height of ‘2 * R_all‘. If ‘NULL‘, the function automatically determines ‘R_m‘ for each group based on the distances between motifs within the same cluster and all curves.

`different_R_m_finding` A logical value indicating whether to use a different radius (‘R_m_finding‘) for finding motif occurrences compared to the initial radius (‘R_m‘). If ‘TRUE‘, ‘R_m_finding‘ is used; otherwise, ‘R_m‘ is employed. This allows for separate tuning of motif occurrence detection.

`R_m_finding` A numeric vector containing group-specific radii used specifically for finding motif occurrences when ‘different_R_m_finding‘ is set to ‘TRUE‘. The length of this vector must match the number of clusters obtained by cutting the dendrogram at a height of ‘2 * R_all‘. If ‘NULL‘, ‘R_m_finding‘ is determined automatically for each group based on distances between motifs within the same cluster and all curves.

`use_real_occurrences` A logical value indicating whether to compute real occurrences of candidate motifs within the curves. If ‘TRUE‘, the function calculates actual frequencies and mean dissimilarities for motif selection, providing more accurate results at the cost of increased computation time. If ‘FALSE‘, it uses approximate frequencies and mean dissimilarities for faster execution. Defaults to ‘FALSE‘.

`length_diff` A numeric value specifying the minimum percentage difference in length required among motifs within the same group to retain multiple motifs. This parameter ensures diversity in motif selection by preventing motifs of similar lengths from being selected simultaneously. It is defined as a percentage relative to the length of the most frequent motif. Defaults to ‘Inf‘, meaning no additional motifs are selected based on length differences.

`worker_number` An integer indicating the number of CPU cores to utilize for parallel processing. By default, the function uses one less than the total number of available cores (‘detectCores() - 1‘). Setting ‘worker_number = 1‘ forces the function to run sequentially without parallelization. If ‘NULL‘, the function automatically determines the optimal number of workers based on the system’s available cores.

Details

The ‘motifs_search‘ function operates through the following steps:

1. **Parallelization Setup**: Determines the number of worker cores to use based on ‘worker_number‘. If ‘worker_number > 1‘, it initializes a cluster for parallel processing.
2. **Input Preparation**: Depending on the dissimilarity metric (‘d0_L2‘, ‘d1_L2‘, or ‘d0_d1_L2‘), it prepares the data structures ‘Y‘ and ‘V‘ for processing.

3. **Dendrogram Cutting**: Cuts the hierarchical clustering dendrogram at a height of '2 * R_all' to define clusters of motifs.
4. **Radius Determination**: If 'R_m' or 'R_m_finding' is not provided, the function calculates these radii for each cluster based on motif distances and K-Nearest Neighbors (KNN) classification.
5. **Candidate Motif Selection**: Depending on 'use_real_occurrences', the function either computes real occurrences and uses actual frequencies and mean dissimilarities to select motifs, or it uses approximate measures for faster processing.
6. **Motif Filtering**: Within each cluster, motifs are ranked based on their frequency and mean dissimilarity. Additional motifs can be selected if their lengths differ sufficiently from the most frequent motif, as defined by 'length_diff'.
7. **Output Compilation**: The selected motifs and their associated properties are compiled into a comprehensive list for further analysis or visualization.

Value

A list containing:

V0 A list of selected motifs derived from 'Y0'.

V1 A list of selected motifs derived from 'Y1' (if applicable).

V_length A numeric vector representing the real lengths of the selected motifs.

V_occurrences A list detailing the occurrences of each selected motif within the curves.

V_frequencies A numeric vector indicating the real frequencies of each selected motif.

V_mean_diss A numeric vector representing the average dissimilarity of each selected motif.

Y0 A list of matrices corresponding to the original curves, as provided in 'cluster_candidate_motifs_results'.

Y1 A list of matrices corresponding to the derivatives of the curves (if applicable), as provided in 'cluster_candidate_motifs_results'.

R_motifs A numeric vector containing the radii associated with each selected motif.

Examples

```
## Not run:
# Load necessary libraries
library(parallel)
library(dplyr)
library(data.table)
library(class)

# Assuming `cluster_results` is the output from `cluster_candidate_motifs`
# and `window_data` is your dataset

# Perform motif search with default parameters
search_results <- motifs_search(cluster_candidate_motifs_results = cluster_results)

# Perform motif search with custom frequency threshold and top_n
search_results <- motifs_search(
  cluster_candidate_motifs_results = cluster_results,
  R_all = 0.5,
  R_m = NULL,
  different_R_m_finding = FALSE,
  R_m_finding = NULL,
  use_real_occurrences = TRUE,
  length_diff = 0.2,
  worker_number = 4
)

# Accessing the results
selected_motifs <- search_results$V0
motif_frequencies <- search_results$V_frequencies

## End(Not run)
```

`motifs_search_plot` *Plot Motif Search Results*

Description

The ‘motifs_search_plot‘ function visualizes the results obtained from the ‘motifs_search‘ function. It generates plots for detected motifs across multiple dimensions, displaying both the motifs and their corresponding derivative curves (if available). Users can filter motifs based on frequency thresholds and choose to display either all motifs or the top ‘n‘ motifs. Additionally, the function provides an option to plot all underlying curves with colored motifs highlighted.

Usage

```
motifs_search_plot(
  motifs_search_results,
  ylab = "",
  freq_threshold = 5,
  top_n = "all",
  plot_curves = TRUE,
  transformed = FALSE
)
```

Arguments

`motifs_search_results`	A list containing the output from the ‘motifs_search‘ function. This includes elements such as ‘V0‘, ‘V1‘, ‘V_frequencies‘, ‘Y0‘, ‘Y1‘, ‘V_length‘, ‘V_occurrences‘, ‘V_mean_diss‘, and ‘R_motifs‘, which store information about the detected motifs and their properties.
`ylab`	A character string specifying the label for the y-axis in the plots. This label will be appended with the dimension number to create individual titles for each plot. Default is an empty string (‘”‘).
`freq_threshold`	An integer indicating the minimum frequency a motif must have to be included in the plots. Only motifs with a frequency equal to or greater than ‘freq_threshold‘ will be visualized. Default value is ‘5‘.
`top_n`	Determines how many motifs to plot based on their frequency. If set to ‘’all’‘, all motifs meeting the ‘freq_threshold‘ will be plotted. If an integer is provided, only the top ‘top_n‘ motifs with the highest frequencies will be displayed. Default is ‘’all’‘.

`plot_curves`	A logical value indicating whether to plot all underlying curves with colored motifs highlighted. If 'TRUE', the function overlays motifs on the curves for better visualization. Default is 'TRUE'.
`transformed`	A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting 'transformed = TRUE' scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.

Details

The 'motifs_search_plot' function performs the following steps:

1. Validates input parameters, ensuring that the frequency threshold and 'top_n' are appropriate.
2. Selects motifs that meet the frequency threshold and, if specified, limits the number of motifs to the top 'n'.
3. For each dimension, it plots the motif centers ('V0') and their derivatives ('V1', if available).
4. If 'plot_curves' is 'TRUE', it overlays the motifs on the original curves, highlighting them with distinct colors.
5. Adds legends to the plots for clear identification of each motif.

Value

The function does not return a value but generates plots visualizing the motifs and their occurrences across different dimensions. It creates separate plots for each dimension and includes legends for easy identification of motifs.

Examples

```
## Not run:
# Assuming `results` is the output from `motifs_search`
# and it contains motifs across 2 dimensions

# Basic usage plotting all motifs with a frequency threshold of 5
motifs_search_plot(results, ylab = "Signal Intensity", freq_threshold = 5)

# Plotting the top 10 motifs with a higher frequency threshold
motifs_search_plot(results, ylab = "Signal Intensity", freq_threshold = 10, top_n = 10)

# Plotting without overlaying the underlying curves
motifs_search_plot(results, ylab = "Signal Intensity", plot_curves = FALSE)

# Using a custom y-axis label
motifs_search_plot(results, ylab = "Amplitude", freq_threshold = 3, top_n = 5)

## End(Not run)
```

`padding` *Pad a Matrix to a Specified Number of Rows*

Description

This function pads a matrix with ‘NA‘ values to ensure that the matrix reaches a specified number of rows (‘maxLen‘). If the matrix has fewer rows than the desired length, additional rows of ‘NA‘ are appended to the matrix. Otherwise, the matrix is returned unchanged.

Usage

```
padding(dataMatrix, maxLen)
```

Arguments

`dataMatrix`	A matrix to be padded. Each row represents a data point.
`maxLen`	An integer specifying the desired number of rows in the matrix.

Details

The function compares the number of rows in ‘dataMatrix‘ with ‘maxLen‘. If the matrix has fewer rows, it pads the matrix with ‘NA‘ values until it reaches the specified number of rows. No changes are made if ‘dataMatrix‘ has rows equal to or greater than ‘maxLen‘.

Value

A matrix padded with ‘NA‘ values up to the specified number of rows. If ‘dataMatrix‘ already has the desired number of rows or more, the original matrix is returned.

Examples

```
## Not run:
# Create a matrix with 3 rows and 2 columns
mat <- matrix(1:6, nrow = 3, ncol = 2)

# Pad the matrix to 5 rows
padded_mat <- padding(mat, maxLen = 5)

# Check the result
print(padded_mat)

## End(Not run)
```

plot_motifs *Plot Embedded Motifs in Functional Curves*

Description

The ‘plot_motifs‘ function visualizes the results generated by the ‘generateCurves‘ function. It provides comprehensive plots of functional curves with embedded motifs, allowing users to inspect the placement and characteristics of each motif within the curves. This visualization is crucial for validating the correctness of motif embedding and for gaining insights into the distribution and variability of motifs across different curves.

The function supports saving the generated plots to a specified directory, facilitating the creation of reports or the sharing of visual results. By plotting motifs in distinct colors or styles, ‘plot_motifs‘ ensures that overlapping motifs and their respective curves remain distinguishable, enhancing the clarity and interpretability of the visualizations.

This plotting utility is an essential tool within the ‘funMoDisco‘ package, aiding users in the exploratory analysis of simulated functional data and the assessment of motif detection algorithms.

Usage

```
plot_motifs(object, curves, name, path = getwd())
```

Arguments

`object`	An S4 object of class ‘motifSimulation‘ that has been previously constructed using the ‘motifSimulationBuilder‘ function. This object contains all necessary parameters and configurations used during curve and motif generation (mandatory).
`curves`	The output list from the ‘generateCurves‘ function, containing the generated functional curves with embedded motifs. This parameter provides the data to be visualized (mandatory).
`name`	Name of the output file.
`path`	A character string specifying the directory path where the generated plots will be saved. The function will save the plots in this directory, allowing for organized storage and easy access to visual results (mandatory).

Value

The function does not return any value but generates and saves plots of the functional curves with embedded motifs in the specified directory. Each plot visually represents the motifs within the curves, aiding in the qualitative assessment of motif embedding.

Examples

```
## Not run:
# Example: Plotting motifs in generated curves
# Assume 'builder' has been created and 'curves' have been generated using generateCurves
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details)
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str = noise_str)

# Specify the directory to save plots
```

```
plots_name <- "plots_1"

# Generate and save the plots
funMoDisco::plot_motifs(builder, curves, plots_name)

## End(Not run)
```

`plot_motifs,motifSimulation-method`
Plot Embedded Motifs in Functional Curves

Description

The ‘plot_motifs‘ function visualizes the results generated by the ‘generateCurves‘ function. It provides comprehensive plots of functional curves with embedded motifs, allowing users to inspect the placement and characteristics of each motif within the curves. This visualization is crucial for validating the correctness of motif embedding and for gaining insights into the distribution and variability of motifs across different curves.

The function supports saving the generated plots to a specified directory, facilitating the creation of reports or the sharing of visual results. By plotting motifs in distinct colors or styles, ‘plot_motifs‘ ensures that overlapping motifs and their respective curves remain distinguishable, enhancing the clarity and interpretability of the visualizations.

This plotting utility is an essential tool within the ‘funMoDisco‘ package, aiding users in the exploratory analysis of simulated functional data and the assessment of motif detection algorithms.

Usage

```
## S4 method for signature 'motifSimulation'
plot_motifs(object, curves, name, path = getwd())
```

Arguments

`object`	An S4 object of class ‘motifSimulation‘ that has been previously constructed using the ‘motifSimulationBuilder‘ function. This object contains all necessary parameters and configurations used during curve and motif generation (mandatory).
`curves`	The output list from the ‘generateCurves‘ function, containing the generated functional curves with embedded motifs. This parameter provides the data to be visualized (mandatory).
`name`	Name of the output file.
`path`	A character string specifying the directory path where the generated plots will be saved. The function will save the plots in this directory, allowing for organized storage and easy access to visual results (mandatory).

Value

The function does not return any value but generates and saves plots of the functional curves with embedded motifs in the specified directory. Each plot visually represents the motifs within the curves, aiding in the qualitative assessment of motif embedding.

Examples

```
## Not run:
# Example: Plotting motifs in generated curves
# Assume 'builder' has been created and 'curves' have been generated using generateCurves
builder <- funMoDisco::motifSimulationBuilder(N = 20, len = 300, mot_details)
curves <- funMoDisco::generateCurves(builder, noise_type = 'pointwise', noise_str = noise_str)

# Specify the directory to save plots
plots_name <- "plots_1"

# Generate and save the plots
funMoDisco::plot_motifs(builder, curves, plots_name)

## End(Not run)
```

ProbKMA *ProbKMA Class*

Description

The ‘ProbKMA‘ class is an R wrapper for the C++ implementation of the Probabilistic K-means Algorithm (ProbKMA) with local alignment. This class facilitates local clustering of functional data and functional motif discovery, as proposed in the paper ‘Probabilistic K-means with local alignment for clustering and motif discovery in functional data‘, authored by Marzia A. Cremona and Francesca Chiaromonte.

Value

A ‘ProbKMA‘ object from the C++ ProbKMA class.

Constructor

Create a ‘ProbKMA‘ object using the following constructor:

```
prok <- new(ProbKMA, data$Y, data$V, params, data$P0, data$S0, "H1")
```

Parameters

Y A list containing functional data and possibly derivatives.

params An instance of the Parameters class, containing algorithm settings.

P0 A matrix representing the initial membership probabilities.

S0 A matrix representing the initial shift warping parameters.

diss A character string specifying the dissimilarity measure. Possible choices are:

- ‘’d0_L2’‘
- ‘’d1_L2’‘
- ‘’d0_d1_L2’‘

Usage

You can access and modify the ‘ProbKMA‘ object with the following methods:

Getters: **prok$get_parameters()** Returns a list of parameters.
prok$get_motifs() Returns a list containing the motifs found.

Setters: **prok$set_P0(P)** Sets the membership matrix.
prok$set_S0(S) Sets the shift warping matrix.
prok$set_parameters(param) Sets parameters field by passing a list of parameters.

Initialize Motifs: **prok$reinit_motifs(c, d)** Reinitializes (empty) K motifs with dimension c_k x d.

Run ProbKMA algorithm: **prok$probKMA_run()** Runs the algorithm.

Author(s)

Niccolò Feresini and Riccardo Lazzarini

Examples

```
## Not run:
# Example usage
# Seed for random initialization of P0 and S0
seed <- 1

# Set type of distance
diss <- 'd0_d1_L2'  # options: 'd0_L2', 'd1_L2', 'd0_d1_L2'

# Null matrices for random initialization
P0 <- matrix()
S0 <- matrix()

# Define parameters for the algorithm
params <- list(standardize = TRUE, K = 2, c = 61, c_max = 71,
               iter_max = 1000, quantile = 0.25,
               stopCriterion = 'max', tol = 1e-8,
               iter4elong = 1, tol4elong = 1e-3, max_elong = 0.5,
               trials_elong = 201, deltaJK_elong = 0.05, max_gap = 0,
               iter4clean = 50, tol4clean = 1e-4,
               quantile4clean = 1/2, return_options = TRUE,
               m = 2, w = 1, alpha = 0.5, seed = seed, exe_print = TRUE,
               set_seed = TRUE)

# Check input data
a <- initialChecks(simulated200$Y0, simulated200$Y1, P0, S0, params, diss, seed)

# Get data and parameters
params <- a$Parameters
data <- a$FuncData

# Create an object of the class ProbKMA
prok <- new(ProbKMA, data$Y, data$V, params, data$P0, data$S0, "H1")

# Run ProbKMA algorithm
output <- prok$probKMA_run()
```

```
## End(Not run)
```

`probKMA_plot` *Plot the Results of probKMA*

Description

The ‘probKMA_plot‘ function visualizes the results obtained from the ‘probKMA‘ analysis. It generates a series of plots including motif memberships across different curves, the progression of the objective function over iterations, and the Bhattacharyya distance between memberships. Depending on the parameters, it can plot both original and cleaned motifs across multiple dimensions, providing insights into the embedding and characteristics of identified motifs.

Usage

```
probKMA_plot(
  probKMA_results,
  plot,
  ylab = "",
  sil_avg = NULL,
  cleaned = FALSE,
  transformed = FALSE
)
```

Arguments

`probKMA_results`
A list containing the output from the ‘probKMA‘ function. This list should include elements such as:

Y0 A list of matrices representing the original curves.

Y1 A list of matrices representing the derivatives of the curves (if applicable).

V0 A list of motifs.

V1 A list of derived motifs (if applicable).

P A matrix indicating motif memberships across curves.

S A matrix indicating motif start positions in curves.

S_clean A matrix indicating cleaned motif start positions (if applicable).

P_clean A matrix indicating cleaned motif memberships (if applicable).

V0_clean A list of cleaned motifs (if applicable).

V1_clean A list of derived cleaned motifs (if applicable).

iter An integer indicating the number of iterations performed.

J_iter A numeric vector recording the objective function value at each iteration.

BC_dist_iter A numeric vector recording the Bhattacharyya distance between memberships at each iteration.

`plot`
A logical flag indicating whether to produce the plots. If ‘TRUE‘, the function generates all relevant plots. If ‘FALSE‘, no plots are produced.

`ylab`
A character vector of length ‘d‘, providing labels for the y-axis in each dimension. Defaults to an empty string (‘”‘) for all dimensions.

`sil_avg`	A numeric vector containing the average silhouette scores for each embedded motif. This parameter is used to annotate the plots with silhouette information. Defaults to 'NULL', meaning no silhouette scores are displayed.
`cleaned`	A logical value indicating whether to plot only the cleaned motifs ('TRUE') or all motifs ('FALSE'). When set to 'TRUE', the function highlights motifs that have been cleaned based on predefined criteria. Defaults to 'FALSE'.
`transformed`	A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting 'transformed = TRUE' scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.

Details

The 'probKMA_plot' function performs the following operations:

1. **Motif Visualization**:
   - Plots the original curves ('Y0') with embedded motifs ('V0'). If derivatives ('Y1' and 'V1') are available, additional plots are generated for them.
   - When 'cleaned = TRUE', the function highlights only the cleaned motifs ('V0_clean' and 'V1_clean'), providing a clearer view of significant motifs.
   - Utilizes color coding and legends to differentiate between different motifs and their instances.
2. **Memberships**:
   - Generates bar plots showing the membership scores ('P' or 'P_clean') of each motif across all curves.
   - Provides a visual representation of how strongly each motif is associated with different curves.
3. **Objective Function and Bhattacharyya Distance**:
   - Plots the objective function ('J_iter') over the iterations to demonstrate the optimization process.
   - Plots the Bhattacharyya distance ('BC_dist_iter') to measure the similarity between motif memberships across iterations.

The function is designed to handle multiple dimensions ('d') and can accommodate both original and derivative data if provided. It also supports the visualization of cleaned motifs, which are motifs that have been refined based on specific criteria to ensure quality and relevance.

Value

The function generates a series of plots visualizing:

- **Motifs with Matched Curves**: Displays the original curves with embedded motifs overlaid. If 'cleaned = TRUE', only cleaned motifs are highlighted.
- **Memberships**: Shows bar plots representing the membership scores of each motif across all curves.
- **Objective Function**: Plots the progression of the objective function ('J_iter') over iterations to illustrate convergence.

- **Bhattacharyya Distance**: Plots the Bhattacharyya distance (‘BC_dist_iter‘) between memberships over iterations to assess similarity.

No value is returned; the function is used solely for its side effects of generating visualizations.

Examples

```
## Not run:
# Load necessary libraries
library(funMoDisco)

# Assume `results` is the output from the `probKMA` function
# and it contains all necessary components.

# Basic plot with default parameters
probKMA_plot(
  probKMA_results = results,
  plot = TRUE,
  ylab = c("Amplitude", "Frequency"),
  cleaned = FALSE
)

# Plot only cleaned motifs with silhouette averages
probKMA_plot(
  probKMA_results = results,
  plot = TRUE,
  ylab = c("Amplitude", "Frequency"),
  sil_avg = c(0.75, 0.80),
  cleaned = TRUE
)

# Disable plotting
probKMA_plot(
  probKMA_results = results,
  plot = FALSE
)

## End(Not run)
```

probKMA_silhouette_filter

Filter Motifs from probKMA Results Based on Silhouette and Size Thresholds

Description

This function filters the motifs identified by the ‘probKMA‘ algorithm based on a threshold for the average silhouette index and a minimum size criterion. Motifs that meet or exceed both the silhouette index threshold and the minimum number of curves (size) are retained. The function returns a cleaned version of the input data, including the filtered motifs, their derivatives, and associated matrices.

Usage

```
probKMA_silhouette_filter(
  probKMA_results,
  silhouette,
  sil_threshold = 0.5,
  size_threshold = 2
)
```

Arguments

`probKMA_results`	A list representing the output of the ‘probKMA‘ function with ‘return_options = TRUE‘. This output includes the motifs, dissimilarity matrices, and membership matrices required for filtering.
`silhouette`	A list output from the ‘probKMA_silhouette‘ function, which provides silhouette scores for each motif.
`sil_threshold`	A numeric value representing the threshold for the average silhouette index. Motifs with a silhouette index greater than or equal to this value will be retained. Default is 0.5.
`size_threshold`	An integer representing the minimum number of curves (size) in a motif cluster. Motifs with a cluster size greater than or equal to this value will be retained. Default is 2.

Value

A list containing the filtered results:

`V0_clean`	Filtered motifs.
`V1_clean`	Derivatives of the filtered motifs.
`D`	Dissimilarity matrix.
`D_clean`	Filtered dissimilarity matrix after cleaning motifs.
`P`	Membership matrix.
`P_clean`	Filtered membership matrix after cleaning motifs.
`c`	Filtered minimum motif lengths.
`K`	Vector containing the number of motifs repeated by the filtered motifs.

`probKMA_silhouette_plot`

Plot Silhouette Index from probKMA Results

Description

This function generates a bar plot displaying the adapted silhouette index based on the results from the ‘probKMA‘ algorithm. It visually represents the quality of the motifs identified by illustrating the average silhouette width for each motif. Additionally, it provides relevant information about the number of curves associated with each motif.

Usage

```
probKMA_silhouette_plot(silhouette_results, K, plot = TRUE)
```

Arguments

`silhouette_results`	A list containing the results from the silhouette analysis, including: - Silhouette indices for each motif. - Motif identifiers. - Curves associated with each motif. - Average silhouette widths. - Number of curves in each motif.
`K`	An integer representing the number of motifs identified by the 'probKMA' algorithm.
`plot`	A logical value indicating whether to generate the plot. Default is 'TRUE'.

Value

A list containing the following elements:

`silhouette`	A vector of silhouette indices for the motifs.
`motifs`	A vector of motif identifiers.
`curves`	A vector containing all curves associated with the motifs.
`silhouette_average`	A vector of average silhouette widths for each motif.

`probKMA_wrap` *Wrapper for the Probabilistic K-means Algorithm (ProbKMA)*

Description

This function serves as a wrapper for the Probabilistic K-means Algorithm (ProbKMA) to cluster functional data. It handles preprocessing, parameter setup, and execution of the core algorithm, returning the results along with silhouette analysis to assess the clustering quality.

Usage

```
probKMA_wrap(
  Y0 = NULL,
  Y1 = NULL,
  P0 = matrix(),
  S0 = matrix(),
  standardize = FALSE,
  c_max = Inf,
  iter_max = 1000,
  iter4elong = 10,
  trials_elong = 10,
  return_options = TRUE,
  alpha = 0,
  max_gap = 0.2,
  quantile = 0.25,
  stopCriterion = "max",
  tol = 1e-08,
  tol4elong = 0.001,
  max_elong = 0.5,
  deltaJK_elong = 0.05,
  iter4clean = 50,
  tol4clean = 1e-04,
  m = 2,
  w = 1,
  seed = 1,
  K = 2,
  c = 40,
  quantile4clean = 1/K,
  exe_print = FALSE,
  set_seed = FALSE,
  diss = "d0_2",
  transformed = FALSE,
  V_init = NULL,
  align = TRUE,
  n_threads = 1
)
```

Arguments

`Y0`	A matrix of functional data for the first set of observations.
`Y1`	A matrix of functional data for the second set of observations.
`P0`	A matrix representing the initial membership probabilities.
`S0`	A matrix representing the initial shift parameters.
`standardize`	A logical value indicating whether to standardize the data. Default is 'FALSE'.
`c_max`	Maximum number of motifs to extract. Default is 'Inf'.
`iter_max`	Maximum number of iterations for the algorithm. Default is 1000.
`iter4elong`	Number of iterations for elongation. Default is 10.
`trials_elong`	Number of trials for elongation. Default is 10.
`return_options`	A logical value indicating whether to return additional options. Default is 'TRUE'.
`alpha`	A numeric value representing the weighting parameter. Default is 0.
`max_gap`	Maximum allowable gap between motifs. Default is 0.2.
`quantile`	Quantile to be used for cleaning. Default is 0.25.
`stopCriterion`	Stopping criterion for the algorithm, can be 'max' or other specified values. Default is 'max'.
`tol`	Tolerance for convergence. Default is 1e-8.
`tol4elong`	Tolerance for elongation iterations. Default is 1e-3.
`max_elong`	Maximum elongation allowed. Default is 0.5.
`deltaJK_elong`	Increment for the elongation. Default is 0.05.
`iter4clean`	Number of iterations for the cleaning process. Default is 50.
`tol4clean`	Tolerance for the cleaning process. Default is 1e-4.
`m`	Parameter controlling the clustering behavior. Default is 2.
`w`	Weighting parameter for the dissimilarity measure. Default is 1.
`seed`	Random seed for reproducibility. Default is 1.

`K`	Number of motifs to extract. Default is 2.
`c`	Minimum motif length. Default is 40.
`quantile4clean`	Quantile used for the cleaning process. Default is 1/K.
`exe_print`	A logical value indicating whether to print execution details. Default is ‘FALSE‘.
`set_seed`	A logical value indicating whether to set the random seed. Default is ‘FALSE‘.
`diss`	Dissimilarity measure to be used. Default is ’d0_2’.
`transformed`	A logical value indicating whether to normalize the curve segments to the interval [0,1] before applying the dissimilarity measure. Setting ‘transformed = TRUE‘ scales each curve segment between 0 and 1, which allows for the identification of motifs with consistent shapes but different amplitudes. This normalization is useful for cases where motif occurrences may vary in amplitude but have similar shapes, enabling better pattern recognition across diverse data scales.
`V_init`	Initial values for the motifs. Default is ‘NULL‘.
`align`	A logical value indicating whether to align the curves. Default is ‘TRUE‘.
`n_threads`	Number of threads to use for parallel processing. Default is 1.

Value

A list containing:

`probKMA_results`	A list of results from the ProbKMA algorithm, including processed functional data and model parameters.
`silhouette_results`	Results from silhouette analysis, indicating the quality of the clustering.

`recommend_node` *Recommend Node from a Numeric Vector*

Description

This function analyzes a numeric vector and recommends an index based on the characteristics of the vector. If the vector is strictly increasing, it suggests stopping before the maximum growth rate. If the vector is not strictly increasing, it recommends the index of the minimum value.

Usage

```
recommend_node(node)
```

Arguments

`node`	A numeric vector representing scores or metrics. Must contain at least two elements.

Details

Recommend Node Function

- If the input vector has fewer than two elements, the function defaults to returning an index of 1. - The function first checks if the input is a numeric vector. If not, an error is thrown. - The function calculates the differences between successive elements and identifies whether the vector is strictly increasing. - In the case of a strictly increasing vector, it returns the index of the maximum growth rate. - For non-increasing vectors, it returns the index of the minimum value.

Value

An integer index indicating the recommended node based on the analysis of the input vector.

Examples

```
## Not run:
# Example usage of recommend_node function
recommend_node(c(1, 2, 3, 4, 5))  # Returns 5 (index of max growth)
recommend_node(c(5, 4, 3, 2, 1))  # Returns 5 (index of min value)
recommend_node(c(1, 2, 2, 3, 1))  # Returns 5 (index of min value)

## End(Not run)
```

`simulated200` *Simulated database for local clustering*

Description

18 curves of length 200 (corresponding to 201 evaluation points), each containing exactly one motif of length 60 (corresponding to 61 evaluation points), with noise level sigma=0.1 Curves 1-6, 14-16 contain one occurrence of motif 1 Curves 7-13, 17-18 contain one occurrence of motif 2 The data have been generated using a B-spline basis of order 3 and knots at distance 10, following the simulation procedure presented in Cremona and Chiaromonte (Comparison with non-sparse and sparse functional clustering simulation section).

Usage

```
data(simulated200)
```

`to_motifDiscovery` *to_motifDiscovery*

Description

Transforms the result of generateCurves into a format suitable for discoverMotifs

Usage

```
to_motifDiscovery(curves)
```

Arguments

`curves`	A list coming from the generateCurves function.

Value

A list containing all curves formatted to be suitable for input into the discoverMotifs function.

Examples

```
## Not run:
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)
formatted_curves <- to_motifDiscovery(curves)

## End(Not run)
```

`to_motifDiscovery,list-method`
to_motifDiscovery

Description

Transforms the result of generateCurves into a format suitable for discoverMotifs

Usage

```
## S4 method for signature 'list'
to_motifDiscovery(curves)
```

Arguments

`curves`	A list coming from the generateCurves function.

Value

A list containing all curves formatted to be suitable for input into the discoverMotifs function.

Examples

```
## Not run:
curves <- funMoDisco::generateCurves(builder, noise_type = 'coeff', noise_str, only_der = FALSE)
formatted_curves <- to_motifDiscovery(curves)

## End(Not run)
```