# Package 'mappeR'

June 9, 2025

**Type** Package

**Title** Construct and Visualize TDA Mapper Graphs

**Description** Topological data analysis (TDA) is a method of data analysis that uses techniques from topology to analyze high-dimensional data. Here we implement Mapper, an algorithm from this area developed by Singh, Mémoli and Carlsson (2007) which generalizes the concept of a Reeb graph <https://en.wikipedia.org/wiki/Reeb_graph>.

**License** MIT + file LICENSE

**URL** https://github.com/Uiowa-Applied-Topology/mappeR

**BugReports** https://github.com/Uiowa-Applied-Topology/mappeR/issues

**Version** 2.1.0

**Encoding** UTF-8

**Imports** fastcluster, stats, utils

**Suggests** testthat (>= 3.0.0)

**Config/testthat/edition** 3

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** George Clare Kennedy [aut, cre]

**Maintainer** George Clare Kennedy <george-clarekennedy@uiowa.edu>

**Repository** CRAN

**Date/Publication** 2025-06-09 09:30:02 UTC

## Contents

---

`assemble_mapper_object`

*Construct mapper graph from data*

---

## Description

Construct mapper graph from data

## Usage

```
assemble_mapper_object(binclust_data, dists, binning = TRUE)
```

## Arguments

| | |
|---|---|
| `binclust_data` | A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids |
| `dists` | A distance matrix for the data that has been binned and clustered. |
| `binning` | Whether the output dataframe should sort vertices into "bins" or not. Should be true if using clustering, leave false otherwise |

### Value

A list of two dataframes, one with node data containing bin membership, datapoints per cluster, and mean distance to the medoid, and one with edge data containing sources, targets, and weights representing overlap strength.

---

## check_in_interval *Get a tester function for an interval.*

---

### Description

Get a tester function for an interval.

### Usage

```
check_in_interval(endpoints)
```

### Arguments

| | |
|---|---|
| endpoints | A vector of interval endpoints, namely a left and a right. Must be in order. |

### Value

A function that eats a data point and outputs TRUE if the datapoint is in the interval and FALSE if not.

---

## compute_tightness *Compute dispersion of a single cluster*

---

### Description

Compute dispersion of a single cluster

### Usage

```
compute_tightness(dists, cluster)
```

### Arguments

| | |
|---|---|
| dists | A distance matrix for points in the cluster. |
| cluster | A list containing named vectors, whose names are data point names and whose values are cluster labels |

## Details

This method finds the medoid of the input data set and returns the average distance to the medoid, i.e.,

$$\tau(C) = \frac{1}{(|C|-1)} \sum_i \text{dist}(x_i, x_j)$$

where

$$x_j = \arg \min_{x_j \in C} \sum_{x_i \in C, i \neq j} \text{dist}(x_i, x_j)$$

A smaller value indicates a tighter cluster based on this metric.

## Value

A real number in $[0, 1]$ representing the mean distance to the medoid of the cluster.

---

`convert_to_clusters` *"Clustering" for ballmapper just means treating each bin as its own cluster.*

---

## Description

"Clustering" for ballmapper just means treating each bin as its own cluster.

## Usage

```
convert_to_clusters(bins)
```

## Arguments

| | |
|---|---|
| `bins` | A list of bins, each containing names of data from some data frame. |

## Value

A named vector whose names are data point names and whose values are cluster labels

`create_1D_mapper_object` *Run 1D mapper*

## Description

Run mapper using a one-dimensional filter, a cover of intervals, and a clustering algorithm.

## Usage

```
create_1D_mapper_object(
  data,
  dists,
  filtered_data,
  cover,
  clusterer = global_hierarchical_clusterer("single", dists)
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame. |
| `dists` | A distance matrix for the data frame. |
| `filtered_data` | The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. |
| `cover` | A 2D array of interval left and right endpoints; rows should be intervals and columns left and right endpoints (in that order). |
| `clusterer` | A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix. |

## Value

A list of two data frames, one with node data containing bin membership, data points per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

## Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

num_bins = 10
percent_overlap = 25

cover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

create_1D_mapper_object(data, dist(data), projx, cover)
```

`create_balls` *Make a cover of balls*

## Description

Make a cover of balls

## Usage

```
create_balls(data, dists, eps)
```

## Arguments

| | |
|---|---|
| `data` | A data frame. |
| `dists` | A distance matrix for the data frame. |
| `eps` | A positive real number. |

## Value

A list of vectors of data point names, one list element per ball. The output is such that every data point is contained in a ball of radius $\varepsilon$, and no ball center is contained in more than one ball. The centers are datapoints themselves.

## Examples

```
num_points = 5000

P.data = data.frame(
  x = sapply(1:num_points, function(x)
    sin(x) * 10) + rnorm(num_points, 0, 0.1),
  y = sapply(1:num_points, function(x)
    cos(x) ^ 2 * sin(x) * 10) + rnorm(num_points, 0, 0.1),
  z = sapply(1:num_points, function(x)
    10 * sin(x) ^ 2 * cos(x)) + rnorm(num_points, 0, 0.1)
)

P.dist = dist(P.data)
balls = create_balls(data = P.data, dists = P.dist, eps = .25)
```

## create_ball_mapper_object

*Run mapper using a trivial filter, a cover of balls, and no clustering algorithm.*

### Description

Run mapper using an $\varepsilon$-net cover (greedily generated) and the 2D inclusion function as a filter.

### Usage

```
create_ball_mapper_object(data, dists, eps)
```

### Arguments

| | |
|---|---|
| data | A data frame. |
| dists | A distance matrix for the data frame. |
| eps | A positive real number for your desired ball radius. |

### Value

A list of two data frames, one with node data containing ball size, data points per ball, ball tightness, and one with edge data containing sources, targets, and weights representing overlap strength.

### Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
eps = .5

create_ball_mapper_object(data, dist(data), eps)
```

## create_bins

*Create bins of data*

### Description

Create bins of data

### Usage

```
create_bins(data, filtered_data, cover_element_tests)
```

### Arguments

| | |
|---|---|
| `data` | A data frame. |
| `filtered_data` | The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. |
| `cover_element_tests` | A list of membership test functions for a set of cover elements. In other words, each element of `cover_element_tests` is a function that returns `TRUE` or `FALSE` when given a filter value. |

### Value

A list of level sets, each containing a vector of the names of the data inside it.

---

## `create_clusterball_mapper_object` *Run clusterball mapper*

---

### Description

Run ball mapper, but additionally cluster within the balls. Can use two different distance matrices to accomplish this.

### Usage

```
create_clusterball_mapper_object(
  data,
  dist1,
  dist2,
  eps,
  clusterer = local_hierarchical_clusterer("single")
)
```

### Arguments

| | |
|---|---|
| `data` | A data frame. |
| `dist1` | A distance matrix for the data frame; this will be used to ball the data. |
| `dist2` | Another distance matrix for the data frame; this will be used to cluster the data after balling. |
| `eps` | A positive real number for your desired ball radius. |
| `clusterer` | A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix. |

### Value

A list of two dataframes, one with node data containing bin membership, datapoints per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

## Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
data.dists = dist(data)
eps = 1

create_clusterball_mapper_object(data, data.dists, data.dists, eps)
```

---

`create_mapper_object` *Create a mapper object*

---

## Description

Run the mapper algorithm on a data frame.

## Usage

```
create_mapper_object(
  data,
  dists,
  filtered_data,
  cover_element_tests,
  clusterer = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame. |
| `dists` | A distance matrix for the data frame. |
| `filtered_data` | The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. |
| `cover_element_tests` | A list of membership test functions for a set of cover elements. In other words, each element of `cover_element_tests` is a function that returns `TRUE` or `FALSE` when given a filter value. |
| `clusterer` | A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix. Defaults to `NULL`, meaning no all data in each bin will be lumped into a single cluster. |

## Value

A list of two dataframes, one with node data and one with edge data. The node data includes:

- `id`: vertex ID
- `cluster_size`: number of datapoints in cluster
- `mean_dist_to_medoid`: mean distance to medoid of cluster
- `data`: names of datapoints in cluster

- patch: level set ID

The edge data includes:

- source: vertex ID of edge source
- target: vertex ID of edge target
- weight: Jaccard index of edge; intersection divided by union
- overlap_data: names of datapoints in overlap
- overlap_size: number of datapoints overlap

## Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

num_bins = 10
percent_overlap = 25
xcover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

check_in_interval <- function(endpoints) {
 return(function(x) (endpoints[1] - x <= 0) & (endpoints[2] - x >= 0))
}

# each of the "cover" elements will really be a function that checks if a data point lives in it
xcovercheck = apply(xcover, 1, check_in_interval)

# build the mapper object
xmapper = create_mapper_object(
  data = data,
  dists = dist(data),
  filtered_data = projx,
  cover_element_tests = xcovercheck
)
```

---

# create_single_bin *Create a bin of data*

---

## Description

Create a bin of data

## Usage

```
create_single_bin(data, filtered_data, cover_element_test)
```

### Arguments

| | |
|---|---|
| `data` | A data frame. |
| `filtered_data` | The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. |
| `cover_element_test` | A membership test function for a cover element. It should return `TRUE` or `FALSE` when given a filtered data point. |

### Value

A vector of names of points from the data frame, representing a level set.

---

## `create_width_balanced_cover` *Generate an overlapping cover of an interval*

---

### Description

This is a function that generates a cover of an interval $[a, b]$ with some number of (possibly) overlapping, evenly spaced, identical width subintervals.

### Usage

```
create_width_balanced_cover(min_val, max_val, num_bins, percent_overlap)
```

### Arguments

| | |
|---|---|
| `min_val` | The left endpoint $a$. A real number. |
| `max_val` | The right endpoint $b$. A real number. |
| `num_bins` | The number of cover intervals with which to cover the interval. A positive integer. |
| `percent_overlap` | How much overlap desired between the cover intervals (the percent of the intersection of each interval with its immediate neighbor relative to its length, e.g., $[0, 2]$ and $[1, 3]$ would have $50\%$ overlap). A real number between 0 and 100, inclusive. |

### Value

A 2D numeric array.

- left_ends - The left endpoints of the cover intervals.
- right_ends - The right endpoints of the cover intervals.

### Examples

```
create_width_balanced_cover(min_val=0, max_val=100, num_bins=10, percent_overlap=15)
create_width_balanced_cover(-11.5, 10.33, 100, 2)
```

## eccentricity_filter *Compute eccentricity of data points*

### Description

Compute eccentricity of data points

### Usage

```
eccentricity_filter(dists)
```

### Arguments

| | |
|---|---|
| dists | A distance matrix associated to a data frame. |

### Value

A vector of centrality measures, calculated per data point as the sum of its distances to every other data point, divided by the number of points.

### Examples

```
num_points = 100

P.data = data.frame(
  x = sapply(1:num_points, function(x)
    sin(x) * 10) + rnorm(num_points, 0, 0.1),
  y = sapply(1:num_points, function(x)
    cos(x) ^ 2 * sin(x) * 10) + rnorm(num_points, 0, 0.1)
)

P.dist = dist(P.data)
eccentricity = eccentricity_filter(P.dist)
```

## get_agglomerative_dendrogram *Perform agglomerative clustering on a single distance matrix.*

### Description

Perform agglomerative clustering on a single distance matrix.

### Usage

```
get_agglomerative_dendrogram(dist, method)
```

### Arguments

| | |
|---|---|
| dist | A distance matrix. |
| method | A string to pass to hclust to determine clustering method. |

### Value

A dendrogram generated by fastcluster.

---

## get_bin_vector *Recover bins*

---

### Description

Recover bins

### Usage

```
get_bin_vector(binclust_data)
```

### Arguments

| | |
|---|---|
| binclust_data | A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids. |

### Value

A vector of integers equal in length to the number of clusters, whose members identify which bin that cluster belongs to.

---

## get_clustered_data *Get data within a cluster*

---

### Description

Get data within a cluster

### Usage

```
get_clustered_data(binclust_data)
```

### Arguments

| | |
|---|---|
| binclust_data | A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids |

### Value

A list of strings, each a comma separated list of the toString values of the data point names.

## get_clusters *Perform the clustering step in mapper*

### Description

This function processes the binned data and global distance matrix to return freshly clustered data.

### Usage

```
get_clusters(bins, dists, clusterer)
```

### Arguments

| | |
|---|---|
| bins | A list containing "bins" of vectors of names of data points. |
| dists | A distance matrix containing pairwise distances between named data points. |
| clusterer | A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix. |

### Value

The output of `clusterer` applied to a list of distance matrices, which should be a list containing named vectors (one per bin), whose names are data point names and whose values are cluster labels.

## get_cluster_sizes *Compute cluster sizes*

### Description

Compute cluster sizes

### Usage

```
get_cluster_sizes(binclust_data)
```

### Arguments

| | |
|---|---|
| binclust_data | A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids. |

### Value

A vector of integers representing the lengths of the clusters in the input data.

---

`get_cluster_tightness_vector`

*Compute dispersion measures of a list of clusters*

---

## Description

Compute dispersion measures of a list of clusters

## Usage

```
get_cluster_tightness_vector(dists, binclust_data)
```

## Arguments

| | |
|---|---|
| `dists` | A distance matrix for the data points inside all the input clusters |
| `binclust_data` | A list of named vectors whose names are those of data points and whose values are cluster ids |

## Value

A vector of real numbers in $(0, \infty)$ containing mean distances to the medoids of each cluster in `dists`.

---

`get_edgelist_from_overlaps`

*Obtain edge list from cluster intersections*

---

## Description

Obtain edge list from cluster intersections

## Usage

```
get_edgelist_from_overlaps(overlaps, num_vertices)
```

## Arguments

| | |
|---|---|
| `overlaps` | A named list of edges, whose elements contain the names of clusters in the overlap represented by that edge; output of `get_overlaps()`. |
| `num_vertices` | The number of vertices in the graph. |

## Value

A 2D array representing the edge list of a graph.

## get_edge_weights *Calculate edge weights*

### Description

Calculate edge weights

### Usage

```
get_edge_weights(overlap_lengths, cluster_sizes, edges)
```

### Arguments

| | |
|---|---|
| overlap_lengths | A named vector of cluster overlap lengths, obtained by calling `length()` on the output from `[get_overlaps()]`. |
| cluster_sizes | A vector of cluster sizes. |
| edges | A 2D array of source and target nodes, representing an edge list. Should be ordered consistently with the `overlap_lengths` parameter. |

### Details

This value is calculated per edge by dividing the number of data points in the union of the two clusters by the number of data points in the intersection. Formally,

$$w(\{c_i, c_j\}) = \frac{|c_i \cap c_j|}{|c_i \cup c_j|} = \frac{|c_i \cap c_j|}{|c_i| + |c_j| - |c_i \cap c_j|}$$

### Value

A vector of real numbers representing the Jaccard index of each overlap.

## get_hierarchical_clusters *Perform hierarchical clustering and process dendrograms.*

### Description

Perform hierarchical clustering and process dendrograms.

### Usage

```
get_hierarchical_clusters(dist_mats, method, cut_height = -1)
```

### Arguments

| | |
|---|---|
| dist_mats | A list of distance matrices to be used for clustering. |
| method | A string to pass to hclust to tell it what kind of clustering to do. |
| cut_height | A global cut height. If not specified or negative, dendrograms will be cut individually. |

### Value

A list containing named vectors (one per dendrogram), whose names are data point names and whose values are cluster labels.

---

## get_longevity_cut_height *Find the the longest-lived hierarchy of a dendrogram.*

---

### Description

Find the the longest-lived hierarchy of a dendrogram.

### Usage

```
get_longevity_cut_height(dend, max_height = max(cophenetic(dend)))
```

### Arguments

| | |
|---|---|
| dend | A dendrogram. |
| max_height | The maximum height of the dendrogram; if this is not provided the last merge height of the input dendrogram will be used, which will make cutting to one cluster impossible! |

### Value

The point just above the merge height with the longest time to the next merge point.

## get_overlaps *Get cluster overlaps*

### Description

Get cluster overlaps

### Usage

```
get_overlaps(binclust_data)
```

### Arguments

| | |
|---|---|
| binclust_data | A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids. |

### Value

A named list of edges, whose elements contain the names of clusters in the overlap represented by that edge.

## get_raw_clusters *Ship data off to the clustering goblins*

### Description

This function tells the computer to look away for a second, so the goblins come and cluster your data while it's not watching.

### Usage

```
get_raw_clusters(dist_mats, clusterer)
```

### Arguments

| | |
|---|---|
| dist_mats | A list of distance matrices of each bin that is to be clustered. |
| clusterer | A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix in a list. |

### Value

The output of `clusterer(dist_mats)`, which needs to be a list containing named vectors (one per bin), whose names are data point names and whose values are cluster labels (within each bin)

`global_hierarchical_clusterer` *Create a dude to perform hierarchical clustering in a global context using the hclust package.*

## Description

Create a dude to perform hierarchical clustering in a global context using the hclust package.

## Usage

```
global_hierarchical_clusterer(method, dists)
```

## Arguments

| | |
|---|---|
| `method` | A string to pass to hclust to tell it what kind of clustering to do. |
| `dists` | The global distance matrix on which to run clustering to determine a global cutting height. |

## Details

This clusterer determines cutting heights for dendrograms by cutting them all according to the best cutting height when the data is clustered together. "Best" here means cutting the dendrogram just above the merge point with the longest unbroken gap until the next merge points.

## Value

A function that inputs a list of distance matrices and returns a list containing one vector per matrix, whose element names are data point names and whose values are cluster labels (relative to each matrix).

## Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

dists = dist(data)

num_bins = 10
percent_overlap = 25

cover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

create_1D_mapper_object(data, dists, projx, cover, global_hierarchical_clusterer("mcquitty", dists))
```

`is_in_ball` *Get a tester function for a ball.*

## Description

Get a tester function for a ball.

## Usage

```
is_in_ball(ball)
```

## Arguments

| | |
|---|---|
| `ball` | A list of data points. |

## Value

A function that eats a data point and returns TRUE or FALSE depending if the point is in the ball or not.

`local_hierarchical_clusterer` *Create a dude to perform hierarchical clustering in a local context using the hclust package.*

## Description

Create a dude to perform hierarchical clustering in a local context using the hclust package.

## Usage

```
local_hierarchical_clusterer(method)
```

## Arguments

| | |
|---|---|
| `method` | A string to pass to hclust to tell it what kind of clustering to do. |

## Details

This clusterer determines cutting heights for dendrograms by cutting them individually, just above the merge point with the longest unbroken gap until the next merge point.

## Value

A function that inputs a list of distance matrices and returns a list containing one vector per matrix, whose element names are data point names and whose values are cluster labels (within each patch).

## Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

dists = dist(data)

num_bins = 10
percent_overlap = 25

cover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

create_1D_mapper_object(data, dists, projx, cover, local_hierarchical_clusterer("mcquitty"))
```

---

# next_triangular *Find which triangular number you're on*

---

## Description

Find which triangular number you're on

## Usage

```
next_triangular(x)
```

## Arguments

| | |
|---|---|
| x | A positive integer. |

## Value

The index of the next greatest or equal triangular number to $x$.

---

# process_dendrograms *Cut many dendrograms at specified cut heights.*

---

## Description

Cut many dendrograms at specified cut heights.

## Usage

```
process_dendrograms(dends, cut_heights)
```

## Arguments

| | |
|---|---|
| dends | A list of dendrograms to be cut. |
| cut_heights | A list of cut heights which corresponds to each dendrogram in dends. |

## Value

A list of named vectors (one per dendrogram) whose names are data point names and whose values are cluster labels.

---

`subset_dists` *Subset a distance matrix*

---

## Description

Subset a distance matrix

## Usage

```
subset_dists(patch, dists)
```

## Arguments

| | |
|---|---|
| `patch` | A list of names of data points. |
| `dists` | A distance matrix for data points in the patch, possibly including extra points. |

# Index