Package 'SimplicialComplex'

August 24, 2026

Type Package

Title Topological Data Analysis: Simplicial Complex

Version 0.1.2

Maintainer ChiChien Wang <kennywang2003@gmail.com>

Description Provides an implementation of simplicial complexes for Topological Data Analysis (TDA). The package includes functions to compute faces, boundary operators, Betti numbers, Euler characteristic, and to construct simplicial complexes, including Vietoris-Rips, Cech, Alpha, Delaunay, Witness, flood, and (via a Freudenthal triangulation) cubical complexes for grid and image data. It also implements persistent homology, from building filtrations (via a single build_filtration() entry point covering all of the above) to computing persistence diagrams, persistence landscapes, and Wasserstein/bottleneck distances between diagrams, with the aim of helping readers understand the core concepts of computational topology.
Methods are based on standard references in persistent homology such as Zomorodian and Carlsson (2005) <doi:10.1007/s00454-004-1146-y>, Chazal and Michel (2021) <doi:10.3389/frai.2021.667963>, and Otter, Porter, Tillmann, Grindrod and Harrington (2017) <doi:10.1140/epjds/s13688-017-0109-5>.

Imports Matrix, gtools, igraph, ggplot2, geometry, RANN, clue

License MIT + file LICENSE

URL https://github.com/TDA-R/SimplicialComplex

BugReports https://github.com/TDA-R/SimplicialComplex/issues

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), torch

Config/testthat/edition 3

NeedsCompilation no

Author ChiChien Wang [aut, cre, trl]

Repository CRAN

Date/Publication 2026-08-24 18:40:50 UTC

Contents

`AlphaComplex` *Construct an Alpha Complex*

Description

Construct an Alpha Complex

Usage

```
AlphaComplex(points, epsilon = Inf)
```

Arguments

points	A numeric matrix or data.frame with one point per row (columns are coordinates); must be in "general position" (see circumsphere).
epsilon	The alpha-complex scale. A simplex is included once its alpha value is at most epsilon. Defaults to Inf, which gives the full Delaunay complex (see DelaunayComplex).

Value

A list of class "alpha_complex" with elements:

simplices A list of integer vectors, every included simplex (all dimensions, not just the maximal ones - membership in the alpha complex is not simply generated by cliques).

filtration A numeric vector, the alpha value of each simplex in simplices, in the same order.

Pass the result to as_filtration for the same list(simplex=, t=) format used by build_filtration and every persistence function, or just call build_filtration(points, method = "Alpha", eps_max = epsilon).

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
alpha_complex <- AlphaComplex(points, epsilon = 1)
```

---

as_filtration *Convert a flood_complex object to a filtration list*

Description

Convert a flood_complex object to a filtration list

Usage

```
as_filtration(fc)
```

Arguments

fc	A "flood_complex" object.

Value

A filtration list compatible with boundary_info.

bottleneck_distance *Bottleneck distance between two persistence diagrams*

Description

The bottleneck distance between the points of two persistence diagrams in a given homological dimension, allowing points to be matched to the diagonal: the infimum, over all matchings, of the largest single point-to-point distance (Cohen-Steiner, Edelsbrunner and Harer (2007), "Stability of Persistence Diagrams") - the $p \to \infty$ limit of `wasserstein_distance`. Essential (death = `Inf`) classes are kept rather than dropped - see `wasserstein_distance`'s Details for why and how. Computed exactly by binary search over the (finitely many) candidate distance values for the smallest one admitting a perfect matching, checked with a standard augmenting-path bipartite matcher, on the same augmented assignment problem `wasserstein_distance` uses (see `augmented_cost_matrix`).

Usage

```
bottleneck_distance(df1, df2, dimension = 0, ground = c("Linf", "L2"))
```

Arguments

`df1`, `df2`	Persistence diagram data frames with columns `dim`, `birth`, `death`.
`dimension`	Homological dimension to compare.
`ground`	Ground metric on the birth-death plane: `"L2"` or `"Linf"` (Chebyshev, the convention used by the reference above; default).

Value

A single non-negative number, the bottleneck distance.

Examples

```
df1 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 2))
df2 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 3))
bottleneck_distance(df1, df2, dimension = 0)
```

boundary *Compute the boundary operator for a simplicial complex*

Description

Compute the boundary operator for a simplicial complex

Usage

```
boundary(simplices, bound_dim)
```

Arguments

simplices	A list of simplices (each a numeric vector).
bound_dim	The dimension k of the boundary operator ∂_k.

Details

$$\partial_k \sigma = \sum_i (-1)^i [v_0 v_1 \dots \hat{v}_i \dots v_k]$$

Value

A sparse matrix representing ∂_k.

Examples

```
simplices <- list(c(1, 2), c(3, 4), c(2, 1, 3), c(4, 2))
boundary(simplices, 0)
```

---

boundary_info *Get the boundary matrix and its reduction information in matrix form*

Description

Get the boundary matrix and its reduction information in matrix form

Usage

```
boundary_info(filist, max_dimension = NULL)
```

Arguments

filist	Filtration list, each element includes simplex and time.
max_dimension	Optional maximum homology dimension to eventually report via extract_persistence_pairs. When set, filist is first restricted to max_dimension + 1 (one extra dimension, kept only so dimension-max_dimension classes are correctly killed - see restrict_filtration's Details) before the boundary matrix is built and reduced. Leave NULL (default) to fall back to filist's own "max_dimension" attribute (set automatically when it came from build_filtration with max_dimension supplied there), or to use filist uncapped if that attribute is also absent.

Value

A list containing the boundary matrix, the last boundary row, the pivot owner for persistence extraction, and filist - the exact (possibly max_dimension-restricted) filtration list the other three elements were computed from, re-tagged with the same "max_dimension" attribute so extract_persistence_pairs can auto-detect it too. Always pass THIS filist back into extract_persistence_pairs, not your original one - see its Details for why.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
res <- boundary_info(filtration)
```

`build_cubical_filtration` *Build a cubical (grid) filtration from an image*

Description

Build a cubical (grid) filtration from an image

Usage

```
build_cubical_filtration(image, superlevel = FALSE)
```

Arguments

`image`	A numeric matrix (grid of pixel/voxel values, e.g. a grayscale image with values in `[0, 255]`).
`superlevel`	If `TRUE`, filters by decreasing value instead (equivalent to negating `image` first) - useful for tracking bright structures shrinking rather than dark structures growing.

Value

A filtration list: one `list(simplex = <integer vector of pixel ids>, t = <numeric>)` per simplex, sorted by (t, dimension, lexicographic order) - the same format as `build_filtration`.

Examples

```
# a ring (value 1) around a hole (value 5) around a background (value 9):
# the hole is born once the ring closes and dies once its center fills in
image <- matrix(c(
  9, 9, 9, 9, 9,
  9, 1, 1, 1, 9,
  9, 1, 5, 1, 9,
  9, 1, 1, 1, 9,
  9, 9, 9, 9, 9
), nrow = 5, byrow = TRUE)
filtration <- build_cubical_filtration(image)
pairs <- persistence_pairs(filtration)
pairs[pairs$dim == 1, ] # one H1 bar: birth = 1 (ring closes), death = 5
plot_persistence(pairs)
```

`build_filtration` *Build a filtration from a point cloud, for any of five complex types*

Description

Build a filtration from a point cloud, for any of five complex types

Usage

```
build_filtration(
  points,
  method = c("VR", "Delaunay", "Alpha", "Cech", "Witness"),
  eps_max = NULL,
  landmarks = NULL,
  nu = 1,
  max_dimension = NULL
)
```

Arguments

`points`	A numeric matrix or data.frame, one point per row.
`method`	One of `"VR"`, `"Delaunay"`, `"Alpha"`, `"Cech"`, `"Witness"`.
`eps_max`	Maximum scale. Required for `"VR"`, `"Cech"` and `"Witness"`; for `"Alpha"` it caps the alpha value (defaults to `Inf`, i.e. no cap); ignored for `"Delaunay"` (which has no scale parameter - see `DelaunayComplex`).
`landmarks`	For `method = "Witness"` only: either an integer (number of landmarks to select via `generate_landmarks`) or an integer vector of landmark row indices into `points`. Required for `"Witness"`.
`nu`	For `method = "Witness"` only: the landmark-distance parameter passed to `WitnessComplex`. Defaults to `1`.
`max_dimension`	Optional maximum homology dimension you intend to compute persistence for.

Value

A filtration list, as described above. When `max_dimension` is set, the list carries it as a `"max_dimension"` attribute (see above); otherwise the attribute is absent, exactly as before this parameter existed.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
vr_filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
alpha_filtration <- build_filtration(points, method = "Alpha")
delaunay_filtration <- build_filtration(points, method = "Delaunay")
cech_filtration <- build_filtration(points, method = "Cech", eps_max = 0.8)
# cap at H1: persistence_pairs()/etc. need no max_dimension of their own
vr_capped <- build_filtration(points, method = "VR", eps_max = 1.2,
                              max_dimension = 1)
pairs <- persistence_pairs(vr_capped)
```

build_flood_filtration *Flood filtration in SimplicialComplex format*

Description

Wraps flood_complex and returns the filtration in the same format as build_filtration: a list of list(simplex = <integer vector>, t = <numeric>), sorted by (time, dimension, lexicographic order). The result plugs directly into boundary_info() / extract_persistence_pairs() / plot_persistence(), as well as into the faster flood_persistence.

Usage

```
build_flood_filtration(points, landmarks, ...)
```

Arguments

points	A numeric matrix (N x d) point cloud.
landmarks	Number of FPS landmarks, or an explicit landmark matrix.
...	Passed on to flood_complex.

Value

A filtration list compatible with boundary_info.

Examples

```
## Not run:
pts <- matrix(rnorm(2000), ncol = 2)
filtration <- build_flood_filtration(pts, landmarks = 30)
pairs <- flood_persistence(filtration)

## End(Not run)
```

CechComplex *Construct a Cech Complex*

Description

Construct a Cech Complex

Usage

```
CechComplex(points, epsilon)
```

Arguments

points	A numeric matrix or data.frame with one point per row (columns are coordinates).
epsilon	A non-negative numeric radius, used only to bound how many candidate simplices are generated (see Details) - not a promise that every returned simplex individually satisfies the Cech criterion at this scale.

Value

A list with:

network An igraph object: the 1-skeleton of the candidate graph (edges where distance $\leq 2\epsilon$).

simplices A list of integer vectors, each the vertex indices of a maximal clique of network.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
cech_complex <- CechComplex(points, epsilon = 0.8)
```

---

compare_complexes *Compare simplicial complex constructions on one point cloud*

Description

Builds the same point cloud's filtration under several build_filtration methods and summarizes, side by side, how expensive each one was to build and what persistent homology it found - useful for seeing directly how Cech's dimension blow-up, Alpha's Delaunay-bounded dimension, VR's cheap-but-approximate cliques, and Witness's landmark subsampling trade off against each other on the same data (see Otter et al. (2017), Section 5.2, for the underlying trade-offs).

Usage

```
compare_complexes(
  points,
  methods = c("VR", "Delaunay", "Alpha", "Cech", "Witness"),
  eps_max = NULL,
  landmarks = NULL,
  nu = 1
)
```

Arguments

points	A numeric matrix or data.frame, one point per row.
methods	Character vector of methods to compare, any subset of "VR", "Delaunay", "Alpha", "Cech", "Witness". Defaults to all five.

eps_max	Maximum scale, passed to build_filtration. Required whenever methods includes "VR", "Cech" or "Witness"; optional for "Alpha" (defaults to Inf); ignored for "Delaunay".
landmarks	Passed to build_filtration for "Witness". If NULL and "Witness" is included, defaults to min(30, nrow(points)) landmarks via generate_landmarks, with a message.
nu	Passed to build_filtration for "Witness".

Value

A list with:

summary A data frame, one row per method, with the number of vertices and simplices, the highest simplex dimension reached, build and persistence-computation time in seconds, and the number of finite/essential persistence pairs found.

diagrams A named list of persistence diagram data frames (one per method, from persistence_pairs), for further comparison (e.g. with wasserstein_distance/ bottleneck_distance, or plot_persistence).

filtrations A named list of the raw filtration lists.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1, 0.5, 0.5), ncol = 2, byrow = TRUE)
cmp <- compare_complexes(points, methods = c("VR", "Alpha", "Delaunay"), eps_max = 1.2)
cmp$summary
```

---

complex_distance *Wasserstein/bottleneck distance between two point clouds, via one chosen complex*

Description

Picks a single build_filtration method, builds the persistence diagram of each point cloud with it, and compares the two diagrams with either wasserstein_distance or bottleneck_distance. Useful for asking "how different are these two datasets topologically", holding the complex construction fixed so the comparison is apples-to-apples.

Usage

```
complex_distance(
  points1,
  points2,
  method = c("VR", "Delaunay", "Alpha", "Cech", "Witness"),
  distance = c("wasserstein", "bottleneck"),
  dimension = 0,
  p = 2,
  ground = NULL,
  eps_max = NULL,
```

```
  landmarks = NULL,
  nu = 1
)
```

Arguments

points1, points2	Numeric matrices or data.frames, one point per row.
method	One of "VR", "Delaunay", "Alpha", "Cech", "Witness" - passed to build_filtration.
distance	One of "wasserstein" or "bottleneck".
dimension	Homological dimension to compare.
p	Wasserstein order, used only when distance = "wasserstein".
ground	Ground metric passed to the chosen distance function ("L2" or "Linf"). Defaults to each distance's own default ("L2" for Wasserstein, "Linf" for bottleneck) if left NULL.
eps_max	Maximum scale, passed to build_filtration. Required for "VR", "Cech" and "Witness"; optional for "Alpha" (defaults to Inf); ignored for "Delaunay".
landmarks	Passed to build_filtration for "Witness"; if NULL, defaults independently for each point cloud to min(30, nrow(points)) landmarks, with a message.
nu	Passed to build_filtration for "Witness".

Value

A list with:

distance The computed distance (a single number).

method, distance_type, dimension Echoed back for reference.

diagram1, diagram2 The two persistence diagrams (data frames) the distance was computed from.

Examples

```
set.seed(1)
cloud_a <- matrix(rnorm(24), ncol = 2)
cloud_b <- matrix(rnorm(24), ncol = 2) + 0.1
complex_distance(cloud_a, cloud_b, method = "VR", distance = "bottleneck",
                 dimension = 0, eps_max = 0.6)
```

DelaunayComplex *Construct a Delaunay Complex*

Description

Construct a Delaunay Complex

Usage

```
DelaunayComplex(points)
```

Arguments

points	A numeric matrix or data.frame with one point per row (columns are coordinates); must be in "general position".

Value

A list of class `c("delaunay_complex", "alpha_complex")`; see `AlphaComplex` for the element description.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
delaunay_complex <- DelaunayComplex(points)
```

---

diagram_matching *Optimal point matching between two persistence diagrams*

Description

Computes the same optimal matching that `wasserstein_distance`/`bottleneck_distance` reduce to internally, and returns it decoded into a readable table instead of just the distance - what `plot_matching` draws. Every point of both diagrams appears in exactly one row: matched to a point of the other diagram (`type = "real-real"`), or matched to the diagonal, i.e. effectively unmatched (`type = "x-diagonal"` for a point of `df1`, `"y-diagonal"` for a point of `df2`).

Usage

```
diagram_matching(
  df1,
  df2,
  dimension,
  distance = c("wasserstein", "bottleneck"),
  p = 2,
  ground = NULL
)
```

Arguments

df1, df2	Persistence diagram data frames with columns `dim`, `birth`, `death`.
dimension	Homological dimension to compare.
distance	Which distance's optimal matching to compute, `"wasserstein"` (default) or `"bottleneck"`.

p	Wasserstein order, used only when distance = "wasserstein".
ground	Ground metric on the birth-death plane, "L2" or "Linf". Defaults to each distance's own default ("L2" for Wasserstein, "Linf" for bottleneck) when left NULL.

Value

A list with distance (the matching's cost, equal to what wasserstein_distance/bottleneck_distance would return), matches (a data frame with columns x_birth, x_death, x_essential, y_birth, y_death, y_essential, type; a NA pair on one side means that row's point matched the diagonal), and X, Y, essential_X, essential_Y (the two diagrams' points after the same essential-pair capping wasserstein_distance uses - see its Details - and which of them were essential before capping).

Examples

```
df1 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 2))
df2 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 3))
diagram_matching(df1, df2, dimension = 0, distance = "wasserstein", p = 2)
```

---

euler_characteristic *Compute the Euler characteristic χ of a simplicial complex*

Description

Compute the Euler characteristic χ of a simplicial complex

Usage

```
euler_characteristic(simplices, tol)
```

Arguments

simplices	A list of simplices (each a numeric vector).
tol	Optional numerical tolerance to pass to rankMatrix().

Details

The Euler characteristic is computed as:

$$\chi = \sum_{k=0}^{k_{\max}} (-1)^k \beta_k$$

where β_k is the kth Betti number, and $k_{\max}$ is the highest dimension of any simplex in the complex.

Interpretation of values:

- $\chi = 2$: Sphere-like surfaces
- $\chi = 1$: Disk-like spaces
- $\chi = 0$: Torus-like or circle-like spaces
- $\chi < 0$: Surfaces with multiple handles or genus

Value

An integer representing the Euler characteristic χ.

See Also

betti_number

Examples

```
simplices <- list(c(1, 2), c(3, 4), c(2, 1, 3), c(4, 2))
euler_characteristic(simplices, tol=0.1)
```

---

extract_persistence_pairs

This function extracts the persistence from combining the boundary matrix and its filtration

---

Description

This function extracts the persistence from combining the boundary matrix and its filtration

Usage

```
extract_persistence_pairs(filist, last_1, pivot_owner, max_dimension = NULL)
```

Arguments

filist	Filtration list, each element includes simplex and time. When boundary_info() was called with max_dimension set, this MUST be res$filist (the restricted list it returned), not your original filtration - last_1/pivot_owner are indexed positionally against whatever filist boundary_info() actually used, so passing a different-length filist here would silently pair the wrong simplices. See Details.
last_1	The last 1 row index for each column in boundary matrix (after reduction).
pivot_owner	The column index owning the pivot row.
max_dimension	Optional maximum homology dimension to report. Set this to the SAME value passed to boundary_info() (which kept one extra dimension internally for correct killers - see restrict_filtration's Details); only that extra dimension is dropped here. Leave NULL (default) to fall back to filist's own "max_dimension" attribute (present when filist is res$filist from a boundary_info() call that used max_dimension, directly or via its own fallback to build_filtration's attribute), or to report every dimension present in filist if that attribute is also absent.

Details

`boundary_info()` and `extract_persistence_pairs()` are two halves of one computation - `last_1`/`pivot_owner` only mean anything relative to the exact `filist` `boundary_info()` used internally. Whenever `max_dimension` is involved, always call as:

```
res <- boundary_info(filtration, max_dimension = k)
pairs <- extract_persistence_pairs(res$filist, res$last_1, res$pivot_owner,
                                   max_dimension = k)
```

A length mismatch between `filist` and `last_1`/`pivot_owner` is refused with an error rather than silently producing wrong pairs - see `persistence_pairs` for a one-call alternative that cannot run into this.

Value

A data frame with columns: dimension, birth, and death.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
res <- boundary_info(filtration)
pairs <- extract_persistence_pairs(filtration, res$last_1, res$pivot_owner)
```

`faces` *Generate all unique faces of a given dimension from simplices*

Description

Generate all unique faces of a given dimension from simplices

Usage

```
faces(simplices, target_dim)
```

Arguments

`simplices`	A list of simplices (each a numeric vector).
`target_dim`	The target dimension k for the faces (e.g., `0` for vertices, `1` for edges, etc.).

Details

The function generates all possible subsets (combinations) of each simplex, removes duplicates, and filters them to only include those of length `target_dim + 1`.

For example, a 2-simplex `c(1, 2, 3)` has three 1-dimensional faces (edges): `c(1,2)`, `c(1,3)`, and `c(2,3)`, and three 0-dimensional faces (vertices): `1`, `2`, and `3`.

Value

A list of faces (each a numeric vector) of dimension `target_dim`.

Examples

```
simplices <- list(c(1, 2), c(3, 4), c(2, 1, 3), c(4, 2))
faces(simplices, target_dim=0)
```

`flood_complex` *Construct a Flood complex*

Description

Builds the Flood complex of a point cloud: a Delaunay complex on a set of landmarks whose simplices are filtered by their covering radius with respect to the full point cloud ("flood time").

Usage

```
flood_complex(
  points,
  landmarks,
  max_dimension = NULL,
  points_per_edge = 30,
  backend = c("auto", "cpu", "torch"),
  batch_points = 2^22,
  delaunay = NULL
)
```

Arguments

`points`	A numeric matrix (N x d) of witness points.
`landmarks`	Either an integer (number of FPS landmarks) or a numeric matrix (N_l x d) of explicit landmark coordinates.
`max_dimension`	Top dimension of the simplices. Defaults to d.
`points_per_edge`	Grid resolution per simplex edge (accuracy vs. speed trade-off). Defaults to 30, as in flooder.
`backend`	One of `"auto"`, `"cpu"`, `"torch"`. `"cpu"` uses a kd-tree (RANN). `"torch"` uses the torch R package and runs on CUDA when available. `"auto"` picks torch only if a CUDA device is present, else the kd-tree.
`batch_points`	Maximum number of grid points processed per batch (bounds memory). Defaults to 2^22.
`delaunay`	Optional precomputed Delaunay triangulation of the landmarks: an m x (d+1) integer matrix of 1-based landmark indices. If NULL (default), computed via `geometry::delaunayn`.

Value

A list of class "flood_complex" with elements simplices (list of integer vectors, landmark indices), filtration (numeric vector of flood times), landmarks (matrix), landmark_indices (or NULL).

Examples

```
## Not run:
pts <- matrix(rnorm(3000), ncol = 2)
fc <- flood_complex(pts, landmarks = 40)

## End(Not run)
```

---

flood_persistence *Persistence pairs via sparse boundary reduction*

Description

Computes persistence pairs from a filtration list with the standard column-reduction algorithm, but on sparse columns (integer index vectors over GF(2)) instead of a dense matrix. Produces the same output format as extract_persistence_pairs(filist, res$last_1, res$pivot_owner) while scaling to the much larger complexes produced by build_flood_filtration.

Usage

```
flood_persistence(filist, max_dimension = NULL)
```

Arguments

filist	A filtration list (from build_flood_filtration or build_filtration).
max_dimension	Optional maximum homology dimension to report. When set, one extra dimension is kept internally so dimension-max_dimension classes still get correct death times from their true killers, and only that extra dimension is dropped from the output - see restrict_filtration's Details, and persistence_pairs which applies the same correction. Leave NULL (default) to report every dimension present.

Value

A data frame with columns dim, birth, death.

generate_landmarks *Farthest-Point Sampling of landmarks*

Description

Selects `n_lms` landmarks from a point cloud via (exact) Farthest-Point Sampling. Equivalent to `flooder::generate_landmarks` (which uses an approximate bucket-FPS; the exact version below gives the same qualitative coverage).

Usage

```
generate_landmarks(points, n_lms, start_idx = 1)
```

Arguments

`points`	A numeric matrix (N x d) point cloud.
`n_lms`	Number of landmarks to sample (<= N).
`start_idx`	Index of the starting point. Defaults to 1 (flooder defaults to index 0, i.e. the same first point).

Value

A list with `landmarks` (n_lms x d matrix) and `indices` (row indices into `points`).

Examples

```
## Not run:
pts <- matrix(rnorm(2000), ncol = 2)
lms <- generate_landmarks(pts, 50)

## End(Not run)
```

persistence_landscape *Compute the persistence landscape of a persistence diagram*

Description

Converts a persistence diagram data frame (the direct output of `extract_persistence_pairs()`, `persistence_pairs()`, or `flood_persistence()`) into its persistence landscape: a collection of continuous, piecewise-linear functions $\lambda(k, \cdot)$, obtained by overlaying the "tent" function of every birth-death pair and, at each time `t`, taking the kth largest tent value (Bubenik, 2015; Chazal and Michel, 2021, Section 5.4).

Usage

```
persistence_landscape(
  df,
  dimension = 0,
  k_max = NULL,
  resolution = 500,
  t_range = NULL
)
```

Arguments

df	A persistence diagram data frame with columns dim, birth, death (e.g. the output of extract_persistence_pairs()).
dimension	Homological dimension to extract the landscape for.
k_max	Number of landscape levels $\lambda(1,\cdot),\ldots,\lambda(k_{max},\cdot)$ to return. Defaults to all levels supported by the diagram (the number of finite birth-death pairs in dimension); levels beyond that are identically zero.
resolution	Number of grid points used to discretize each landscape function.
t_range	Optional c(t_min, t_max) grid range. Defaults to c(min(birth), max(death)) over the selected pairs.

Value

A data frame with columns t, k, value: the discretized landscape functions, one row per (grid point, level) pair.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
res <- boundary_info(filtration)
pairs <- extract_persistence_pairs(filtration, res$last_1, res$pivot_owner)
landscape <- persistence_landscape(pairs, dimension = 0)
```

---

persistence_pairs *Persistence pairs via sparse boundary reduction (for large filtrations)*

Description

Same standard column-reduction algorithm as boundary_info() + extract_persistence_pairs(), but each column is stored as a sorted integer vector over GF(2) instead of a row of a dense n x n matrix. Memory drops from O(n^2) to O(total number of non-zeros), which is what makes filtrations with thousands to hundreds of thousands of simplices (e.g. Flood or large Vietoris-Rips complexes) feasible. Output is identical to the dense pipeline.

Usage

```
persistence_pairs(filist, max_dimension = NULL)
```

Arguments

`filist`	Filtration list, each element includes simplex and time.
`max_dimension`	Optional maximum homology dimension to report (`0` = H0 only, `1` = H0 and H1, etc.). When set, one extra dimension is kept internally so dimension-`max_dimension` classes still get their correct death time from their true (max_dimension+1) killers, and only that extra dimension is dropped from the returned pairs - see `restrict_filtration`'s Details for why a plain truncation would be wrong. Leave `NULL` (default) to fall back to `filist`'s own `"max_dimension"` attribute (set automatically when `filist` came from `build_filtration` with `max_dimension` supplied there), or to report every dimension present in `filist` if that attribute is also absent.

Value

A data frame with columns: dim, birth, and death.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
pairs <- persistence_pairs(filtration)
pairs_h0_only <- persistence_pairs(filtration, max_dimension = 0)
# or bake the cap into the filtration itself, and drop the argument here:
capped <- build_filtration(points, method = "VR", eps_max = 1.2,
                           max_dimension = 0)
pairs_h0_only2 <- persistence_pairs(capped)
```

`plot_landscape` *Plot a Persistence Landscape*

Description

Plot a Persistence Landscape

Usage

```
plot_landscape(landscape_df)
```

Arguments

`landscape_df`	Data frame from `persistence_landscape()`, with columns `t`, `k`, `value`.

Value

A ggplot2 object with one line per landscape level $\lambda(k, \cdot)$.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
res <- boundary_info(filtration)
pairs <- extract_persistence_pairs(filtration, res$last_1, res$pivot_owner)
landscape <- persistence_landscape(pairs, dimension = 0)
plot_landscape(landscape)
```

---

plot_matching *Plot the optimal matching between two persistence diagrams*

Description

Visualizes the point correspondence that realizes the Wasserstein or bottleneck distance between two persistence diagrams in a given homological dimension: both diagrams' points, overlaid on the same axes, joined by dashed lines to their matched partner - either a point of the other diagram, or (for a point left unmatched) its own projection onto the diagonal `birth = death`.

Usage

```
plot_matching(
  df1,
  df2,
  dimension,
  distance = c("wasserstein", "bottleneck"),
  p = 2,
  ground = NULL,
  labels = c("Diagram 1", "Diagram 2")
)
```

Arguments

`df1`, `df2`	Persistence diagram data frames with columns `dim`, `birth`, `death`.
`dimension`	Homological dimension to compare.
`distance`	Which distance's optimal matching to visualize, `"wasserstein"` (default) or `"bottleneck"`.
`p`	Wasserstein order, used only when `distance = "wasserstein"`.
`ground`	Ground metric, `"L2"` or `"Linf"`. Defaults to each distance's own default (`"L2"` for Wasserstein, `"Linf"` for bottleneck) when left `NULL` - see wasserstein_distance/bottleneck_dista
`labels`	Legend labels identifying `df1` and `df2`, e.g. `c("Clean", "Noisy")`.

Details

Essential (death = Inf) points are capped exactly the way wasserstein_distance does (see its Details) so they take part in the matching instead of being dropped, and are drawn as triangles at their capped height - marked "essential" in the legend - so they stay visually distinguishable from genuine finite points that happen to reach that height. Matches to the diagonal are always drawn to the point's perpendicular projection ((birth+death)/2, (birth+death)/2) regardless of ground, since that is the standard, readable way to depict an unmatched point whichever ground metric produced the matching.

Value

A ggplot2 object; the title reports the resulting distance.

Examples

```
df1 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 2))
df2 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 3))
plot_matching(df1, df2, dimension = 0, distance = "wasserstein", p = 2)
```

---

plot_persistence *Plot Persistence Diagram*

Description

Plot Persistence Diagram

Usage

```
plot_persistence(df)
```

Arguments

df	Dataframe from plot_persistence.

Value

A ggplot2 object representing the persistence diagram.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
res <- boundary_info(filtration)
pairs <- extract_persistence_pairs(filtration, res$last_1, res$pivot_owner)
plot_persistence(pairs)
```

restrict_filtration *Restrict a filtration list to simplices up to a given dimension*

Description

Drops every simplex of dimension greater than `max_dimension` from a filtration list; everything else (order, ties, `t` values) is left untouched.

Usage

```
restrict_filtration(filist, max_dimension)
```

Arguments

`filist`	A filtration list, as produced by `build_filtration`/`build_flood_filtration`.
`max_dimension`	Maximum simplex dimension to keep (`0` = vertices only, `1` = vertices + edges, etc.).

Value

A filtration list containing only the entries with dimension <= `max_dimension`, in the same relative order.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
filtration <- build_filtration(points, method = "VR", eps_max = 1.2)
restrict_filtration(filtration, max_dimension = 1) # vertices + edges only
```

VietorisRipsComplex *Construct a Vietoris–Rips Complex (1-skeleton + maximal simplices)*

Description

Construct a Vietoris–Rips Complex (1-skeleton + maximal simplices)

Usage

```
VietorisRipsComplex(points, epsilon)
```

Arguments

`points`	A numeric matrix or data.frame with one point per row (columns are coordinates).
`epsilon`	A positive numeric threshold; connect points with distance < ϵ.

Details

The Vietoris–Rips complex at scale ϵ includes a simplex for every finite set of points with pairwise distances $< \epsilon$. This function constructs the 1-skeleton (edges only) and then uses maximal cliques in that graph as the maximal simplices.

Value

A list with:

network An `igraph` object representing the 1-skeleton.

simplices A list of integer vectors, each the vertex indices of a maximal simplex.

Examples

```
points <- matrix(c(0, 1, 1, 0, 0, 0, 1, 1), ncol = 2)
epsilon <- 1.5
vr_complex <- VietorisRipsComplex(points, epsilon)
```

---

wasserstein_distance *Wasserstein distance between two persistence diagrams*

Description

The p-Wasserstein distance between the points of two persistence diagrams in a given homological dimension, allowing points to be matched to the diagonal (Cohen-Steiner, Edelsbrunner, Harer and Mileyko (2010), "Lipschitz Functions Have L_p-Stable Persistence"). Essential (death = `Inf`) classes are kept rather than dropped - see Details. Computed exactly via the Hungarian algorithm (`clue::solve_LSAP`) on the augmented assignment problem described in `augmented_cost_matrix`.

Usage

```
wasserstein_distance(df1, df2, dimension = 0, p = 2, ground = c("L2", "Linf"))
```

Arguments

`df1`, `df2`	Persistence diagram data frames with columns `dim`, `birth`, `death` (e.g. the output of `extract_persistence_pairs()` or `persistence_pairs()`).
`dimension`	Homological dimension to compare.
`p`	Wasserstein order ($p \geq 1$). Defaults to `2`.
`ground`	Ground metric on the birth-death plane used to measure the distance between two points (or a point and the diagonal): `"L2"` (Euclidean, default) or `"Linf"` (Chebyshev).

Value

A single non-negative number, the p-Wasserstein distance.

Examples

```
df1 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 2))
df2 <- data.frame(dim = c(0, 0), birth = c(0, 0), death = c(1, 3))
wasserstein_distance(df1, df2, dimension = 0, p = 2)
```

WitnessComplex *Construct a Witness Complex*

Description

Construct a Witness Complex

Usage

```
WitnessComplex(points, landmarks, epsilon, nu = 1)
```

Arguments

points	A numeric matrix or data.frame of witness points S, one point per row.
landmarks	Either an integer (the number of landmarks to select from points via generate_landmarks, i.e. Farthest-Point Sampling) or an integer vector of row indices into points to use directly as landmarks L.
epsilon	A non-negative numeric scale.
nu	Landmark-distance parameter; defaults to 1.

Value

A list with:

network An igraph object: the 1-skeleton on the landmark vertex set.

simplices A list of integer vectors (indices into landmarks / landmark_indices), each the vertices of a maximal simplex.

landmarks The landmark coordinate matrix.

landmark_indices Row indices into points, or NULL if landmarks was given as coordinates rather than indices.

edge_birth A symmetric matrix giving, for every pair of landmarks, the exact scale at which their edge is witnessed (used by build_filtration to time every simplex exactly, rather than only checking membership at the single scale epsilon).

Examples

```
points <- matrix(rnorm(200), ncol = 2)
witness_complex <- WitnessComplex(points, landmarks = 15, epsilon = 0.5)
```

Index