

Package ‘SimplicialComplex’

September 30, 2026

Type Package

Title Topological Data Analysis: Simplicial Complex

Version 0.2.1

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](https://doi.org/10.1007/s00454-004-1146-y)>, Chazal and Michel (2021) <[doi:10.3389/frai.2021.667963](https://doi.org/10.3389/frai.2021.667963)>, and Otter, Porter, Tillmann, Grindrod and Harrington (2017) <[doi:10.1140/epjds/s13688-017-0109-5](https://doi.org/10.1140/epjds/s13688-017-0109-5)>.

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

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, rgl

Config/testthat/edition 3

NeedsCompilation no

Author ChiChien Wang [aut, cre, trl]

Repository CRAN

Date/Publication 2026-09-30 16:50:02 UTC

Contents

AlphaComplex	3
as_filtration	4
bottleneck_distance	4
boundary	5
boundary_info	6
build_cubical_filtration	6
build_filtration	7
build_flood_filtration	8
CechComplex	9
collect_g	10
compare_complexes	10
complex_distance	11
crocker	13
DelaunayComplex	14
diagram_matching	15
euler_characteristic	16
extract_persistence_pairs	17
faces	18
flood_complex	19
flood_persistence	20
gauss_jordan_eliminate	20
generate_landmarks	21
graph_laplacian	21
hodge_laplacian	22
homology	22
im	23
ker	24
lower_star_filtration	25
morse_recon	25
partial_pers_dmvf	26
persistence_landscape	27
persistence_pairs	28
persistent_laplacian	29
plot_crocker	29
plot_landscape	30
plot_matching	31
plot_morse_landscape	32
plot_morse_recon	33
plot_morse_vpath	34
plot_persistence	35
restrict_filtration	35
VietorisRipsComplex	36

<i>AlphaComplex</i>	3
---------------------	---

wasserstein_distance	37
WitnessComplex	37
zigzag_persistence	38

Index	40
--------------	-----------

AlphaComplex	<i>Construct an Alpha Complex</i>
--------------	-----------------------------------

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	<i>Convert a flood_complex object to a filtration list</i>
---------------	--

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	<i>Bottleneck distance between two persistence diagrams</i>
---------------------	---

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 $p \rightarrow \infty$ 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	<i>Get the boundary matrix and its reduction information in matrix form</i>
---------------	---

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	<i>Build a filtration from a point cloud, for any of five complex types</i>
------------------	---

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

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

Value

A filtration list compatible with `boundary_info`.

CechComplex	<i>Construct a Cech Complex</i>
-------------	---------------------------------

Description

Construct a Cech Complex

Usage

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

Arguments

<code>points</code>	A numeric matrix or data.frame with one point per row (columns are coordinates).
<code>epsilon</code>	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)
```

collect_g	<i>Collect 1-unstable manifolds into a reconstructed graph</i>
-----------	--

Description

Implements Algorithm 21 (CollectG): for every critical edge $e = (u, v)$ (i.e. every edge not used as a DMVF matching/tree edge - by construction these all have persistence greater than `dmvf$delta`, so unlike the book's pseudocode no extra filter is needed here), unions e with the unique tree paths from u and from v up to their respective roots.

Usage

```
collect_g(dmvf)
```

Arguments

`dmvf` A result of `partial_pers_dmvf` (or the equivalent list built inside `morse_recon`).

Value

A `data.frame(u, v)` of the (deduplicated, undirected) edges of the reconstructed graph.

compare_complexes	<i>Compare simplicial complex constructions on one point cloud</i>
-------------------	--

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	<i>Wasserstein/bottleneck distance between two point clouds, via one chosen complex</i>
------------------	---

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)
```

crocker

*Compute a CROCKER matrix for a time-varying point cloud***Description**

CROCKER = "Contour Realization Of Computed k -dimensional hole Evolution in the Rips complex". Treats the k -th Betti number as a function of two parameters at once.

Usage

```
crocker(
  point_clouds,
  dim,
  method = "VR",
  eps_max = NULL,
  n_eps = 50,
  max_dimension = NULL,
  n_cores = 1
)
```

Arguments

point_clouds	A list of length n : point_clouds[[i]] is the point cloud observed at time step i .
dim	The homology dimension k to track (b_k); e.g. 0 for connected components, 1 for loops.
method	Complex type passed through to build_filtration . Defaults to "VR" (Vietoris-Rips), which is what the CROCKER plot was originally defined for, it only needs pairwise distances, not an ambient embedding, so it applies to any time-varying metric space. Other complex can be substituted, but the result is then a generalised Betti-surface rather than a "CROCKER plot" in the strict.
eps_max	Maximum scale (epsilon) value. If NULL (default), it is set to the largest pairwise distance seen in any single frame.
n_eps	Number of proximity values sampled uniformly from 0 to eps_max (the original CROCKER paper uses 50).
max_dimension	Optional cap forwarded to build_filtration() / persistence_pairs() . Defaults to dim.
n_cores	Number of cores to use via <code>parallel::mclapply()</code> for the per-frame computation (see Details for why this parallelises cleanly). Defaults to 1 (sequential). Falls back to sequential on Windows, where <code>mclapply()</code> (fork-based) is unavailable.

Value

An object of class "crocker": a list with

matrix An $n_{\text{eps}} \times n$ numeric matrix; entry $[j, i]$ is b_k of frame i at $\text{eps_grid}[j]$.

eps_grid The sampled scale values (length n_{eps}).

time Frame indices, `seq_along(point_clouds)`.

dim The homology dimension tracked.

long A tidy data frame with columns `t`, `epsilon`, `betti`, ready for [plot_crocker](#).

DelaunayComplex	<i>Construct a Delaunay Complex</i>
-----------------	-------------------------------------

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	<i>Optimal point matching between two persistence diagrams</i>
------------------	--

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 k th 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

[beti_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 <code>boundary_info()</code> was called with <code>max_dimension</code> set, this MUST be <code>res\$filist</code> (the restricted list it returned), not your original filtration - <code>last_1/pivot_owner</code> are indexed positionally against whatever <code>filist</code> <code>boundary_info()</code> actually used, so passing a different-length <code>filist</code> 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 <code>boundary_info()</code> (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 <code>filist</code> 's own "max_dimension" attribute (present when <code>filist</code> is <code>res\$filist</code> from a <code>boundary_info()</code> call that used <code>max_dimension</code> , directly or via its own fallback to build_filtration 's attribute), or to report every dimension present in <code>filist</code> 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	<i>Generate all unique faces of a given dimension from simplices</i>
-------	--

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	<i>Construct a Flood complex</i>
---------------	----------------------------------

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 <code>geometry::delaunayn</code> .

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).

flood_persistence	<i>Persistence pairs via sparse boundary reduction</i>
-------------------	--

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

<code>filist</code>	A filtration list (from <code>build_flood_filtration</code> or <code>build_filtration</code>).
<code>max_dimension</code>	Optional maximum homology dimension to report. When set, one extra dimension is kept internally so dimension- <code>max_dimension</code> 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.

Value

A data frame with columns `dim`, `birth`, `death`.

gauss_jordan_eliminate	<i>Row-reduce (partial pivoting) a matrix</i>
------------------------	---

Description

Row-reduce (partial pivoting) a matrix

Usage

```
gauss_jordan_eliminate(M, tol = 1e-08)
```

Arguments

<code>M</code>	A numeric matrix.
<code>tol</code>	Pivot values smaller than this (in absolute value) are treated as zero. Defaults to <code>1e-8</code> .

Details

Shared elimination step behind `ker` and `im` (and, internally, the zigzag module's own linear solves): partial-pivoting Gauss-Jordan elimination, stopping early once every row has a pivot.

Value

A list with `R` (the row-reduced matrix) and `pivots` (the column indices where a pivot was found, in row order).

generate_landmarks	<i>Farthest-Point Sampling of landmarks</i>
--------------------	---

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

<code>points</code>	A numeric matrix ($N \times d$) point cloud.
<code>n_lms</code>	Number of landmarks to sample ($\leq N$).
<code>start_idx</code>	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).

graph_laplacian	<i>Basic graph Laplacian $L = D - A$</i>
-----------------	---

Description

The classical graph-theory Laplacian - degree matrix minus adjacency matrix - built directly from a complex's 1-skeleton (vertices + edges).

Usage

```
graph_laplacian(simplices)
```

Arguments

simplices A list of simplices; only the 0-simplices (vertices) and 1-simplices (edges) are used.

Value

A list with the Laplacian L, the degree matrix D, the adjacency matrix A, and the vertex basis (row/column order).

hodge_laplacian	<i>Ordinary Hodge Laplacian</i>
-----------------	---------------------------------

Description

Computes $L_k(K) = B_k^T B_k + B_{k+1} B_{k+1}^T$.

Usage

```
hodge_laplacian(K_simplices, k)
```

Arguments

K_simplices A single complex (list of maximal simplices).
k The dimension.

Value

A list with the full Laplacian, its down/up pieces, and the k-simplex basis (row/column order) everything is expressed in.

homology	<i>Compute a homology basis from a cycle basis and a boundary basis</i>
----------	---

Description

Compute a homology basis from a cycle basis and a boundary basis

Usage

```
homology(Z, B = matrix(numeric(0), nrow = nrow(Z), ncol = 0), tol = NULL)
```

Arguments

Z	A matrix whose columns form a basis of the cycle space $Z_k = \ker(\partial_k)$ (typically the output of <code>ker</code>).
B	A matrix whose columns form a basis of the boundary space $B_k = \text{im}(\partial_{k+1})$ (typically the output of <code>im</code>), living in the same ambient space as Z (i.e. <code>nrow(B) == nrow(Z)</code>). Defaults to an empty basis ($B_k = \emptyset$).
tol	Numerical tolerance passed to <code>betti_number</code> 's rank helper (<code>safe_rank</code>). Defaults to NULL, i.e. <code>Matrix::rankMatrix()</code> 's own default tolerance.

Details

$H_k = \ker(\partial_k)/\text{im}(\partial_{k+1})$: two cycles represent the same homology class exactly when they differ by a boundary. This walks the columns of Z in order, greedily keeping any column that increases the rank of the span accumulated so far (starting from B's span) - i.e. any cycle that is not already a linear combination of B and the cycles kept before it. The kept columns are one representative chain per homology class.

`betti_number` computes $\dim H_k$ directly from ranks (rank-nullity), without ever materializing Z, B, or a homology basis - that is cheaper when only the count is needed. Use `homology()` when the actual representative cycles matter (e.g. to visualize or track a specific hole), not to recompute a Betti number.

As with `ker` and `im`, the specific representative chosen for each class depends on the order cycles in Z are tested against the growing boundary span, and is not unique.

Value

A matrix with `nrow(Z)` rows, one column per representative of a basis of the quotient $H_k = Z_k/B_k$. If $H_k = 0$, the result has $\emptyset$ columns; `ncol()` of the result is the Betti number $\beta_k = \dim H_k$.

im	<i>Compute a basis for the image (column space) of a matrix</i>
----	---

Description

Compute a basis for the image (column space) of a matrix

Usage

```
im(M, tol = 1e-08)
```

Arguments

M	A numeric matrix (or an object coercible to one, e.g. a sparse <code>Matrix</code>).
tol	Pivoting tolerance passed to the Gauss-Jordan elimination used internally. Defaults to <code>1e-8</code> .

Details

$\text{im}(M) = \{Mx : x \in \mathbb{R}^{\text{ncol}(M)}\}$, the column space of M . The basis returned is a subset of M 's own columns, specifically, the pivot columns found by Gauss-Jordan elimination rather than synthetic linear combinations, so each basis vector is directly interpretable as one of the original columns of M (e.g. the boundary of one specific simplex).

Value

A matrix with $\text{nrow}(M)$ rows, one column per basis vector of $\text{im}(M)$. If the image is trivial ($\{0\}$), the result has 0 columns.

ker	<i>Compute a basis for the kernel (null space) of a matrix</i>
-----	--

Description

Compute a basis for the kernel (null space) of a matrix

Usage

```
ker(M, tol = 1e-08)
```

Arguments

M	A numeric matrix (or an object coercible to one, e.g. a sparse <code>Matrix</code>). Rows are the codomain, columns the domain: M is read as a linear map $M : \mathbb{R}^{\text{ncol}(M)} \rightarrow \mathbb{R}^{\text{nrow}(M)}$.
tol	Pivoting tolerance passed to the Gauss-Jordan elimination used internally (see im , which shares the same elimination step). Defaults to $1e-8$.

Details

$\text{ker}(M) = \{x \in \mathbb{R}^{\text{ncol}(M)} : Mx = 0\}$. The basis is obtained from Gauss-Jordan elimination of M : one basis vector per free (non-pivot) column, in the usual parametric-solution construction. If M has 0 rows (the zero map), every standard basis vector of the domain is in the kernel, so the identity matrix is returned.

As with any basis, the specific vectors returned are not unique, they depend on the pivoting order of the elimination, only the number of columns ($\dim \text{ker}(M)$) is an invariant of M .

Value

A matrix with $\text{ncol}(M)$ rows, one column per basis vector of $\text{ker}(M)$. If the kernel is trivial ($\{0\}$), the result has 0 columns.

`lower_star_filtration` Generalizes `build_cubical_filtration` to any triangulation: given the maximal simplices of a simplicial complex and a function defined at its vertices, builds the simplex-wise lower-star filtration $\mathcal{F}_f$.

Description

Generalizes `build_cubical_filtration` to any triangulation: given the maximal simplices of a simplicial complex and a function defined at its vertices, builds the simplex-wise lower-star filtration $\mathcal{F}_f$.

Usage

```
lower_star_filtration(top_simplices, f)
```

Arguments

`top_simplices` A list of maximal simplices (each an integer vector of 1-based vertex ids).
`f` A plain numeric vector giving the function value at each vertex; `f[v]` is the value at vertex `v`, so vertex ids must be integers in `1:length(f)`.

Value

A filtration list: one `list(simplex = integer vector of vertex ids, t = numeric)` per simplex (every face of every maximal simplex), sorted by (t, dimension, lexicographic order).

Examples

```
# two triangles sharing an edge, function increasing away from vertex 1
triangles <- list(c(1, 2, 3), c(2, 3, 4))
f <- c(0, 1, 1, 2)
filtration <- lower_star_filtration(triangles, f)
length(filtration) # 4 vertices + 5 edges + 2 triangles
```

`morse_recon`

Reconstruct a hidden graph from a density field

Description

Implements Algorithm 20 (MorseRecon): given a triangulated domain and a density function ρ that concentrates around a hidden geometric graph G , computes the "mountain ridges" of $f = -\rho$ - the 1-unstable manifolds of the discrete gradient field after cancelling vertex-edge persistence pairs with persistence at most δ - as an approximation $\hat{G}$ of G .

Usage

```
morse_recon(top_simplices, rho, delta = 0, vertex_coords = NULL)
```

Arguments

top_simplices	List of maximal simplices (e.g. triangles) of the ambient 2-complex, as accepted by lower_star_filtration .
rho	A plain numeric vector, the density value at each vertex.
delta	Persistence threshold used to cancel low-persistence vertex-edge pairs (noise); larger values denoise more aggressively.
vertex_coords	Optional n x 2 matrix of vertex coordinates, used only by plot_morse_recon for drawing.

Value

An object of class "morse_recon": a list with

filtration	the full lower-star filtration of the 2-complex
dmvf	the vertex-edge DMVF: pers, tree_edges, critical_edges, parent, roots, delta (same shape as a partial_pers_dmvf result, but with H0-creator edges' persistence refined against triangles, see Details)
graph_edges	data.frame(u, v) - the edges of $\hat{G}$
rho, delta, coords	the inputs, kept for plotting/inspection

partial_pers_dmvf	<i>Persistence-guided discrete Morse vector field on a graph (1-complex)</i>
-------------------	--

Description

Implements Algorithm 19 (SimplePersDMVF) together with the threshold- δ simplification: vertex-edge persistence pairs with persistence at most delta are cancelled, leaving a discrete Morse vector field where every non-critical vertex v is matched with the tree edge connecting it to its component's root, and every root is a critical vertex.

Usage

```
partial_pers_dmvf(filist1, delta = 0)
```

Arguments

filist1	A filtration list restricted to vertices and edges only (e.g. via restrict_filtration (filist, 1)), as produced by lower_star_filtration or build_filtration .
delta	Persistence threshold; vertex-edge pairs with persistence $\leq$ delta are cancelled (become matched V-field arrows). Default 0 cancels only exactly-zero-persistence pairs; use a larger value to denoise more aggressively, or Inf to cancel every finite pair (maximal simplification, Theorem 10.5).

Value

A list with components:

<code>pers</code>	data.frame(u, v, t, persistence, type) for every edge, type is "destroyer" (merges two components) or "creator" (closes a cycle; persistence Inf)
<code>tree_edges</code>	the subset of pers used as matching/DMVF edges
<code>critical_edges</code>	the rest of pers, i.e. $K^1 \setminus T$ – exactly the input <code>collect_g</code> needs
<code>parent</code>	named integer vector, <code>parent[["v"]]</code> is the vertex on v 's tree edge towards its component root (NA for roots)
<code>roots</code>	integer vector of critical (root) vertices
<code>delta</code>	the threshold used

`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 k th 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

<code>df</code>	A persistence diagram data frame with columns <code>dim</code> , <code>birth</code> , <code>death</code> (e.g. the output of <code>extract_persistence_pairs()</code>).
<code>dimension</code>	Homological dimension to extract the landscape for.
<code>k_max</code>	Number of landscape levels $\lambda(1, \cdot), \dots, \lambda(k_{max}, \cdot)$ 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.
<code>resolution</code>	Number of grid points used to discretize each landscape function.
<code>t_range</code>	Optional <code>c(t_min, t_max)</code> grid range. Defaults to <code>c(min(birth), max(death))</code> 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	<i>Persistence pairs via sparse boundary reduction (for large filtrations)</i>
-------------------	--

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 $n \times n$ matrix. Memory drops from $O(n^2)$ to $O(\text{total number of non-zeros})$, 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 ($0 = H_0$ only, $1 = H_0$ and H_1 , etc.). When set, one extra dimension is kept internally so dimension- <code>max_dimension</code> classes still get their correct death time from their true (<code>max_dimension+1</code>) 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 <code>filist</code> 's own "max_dimension" attribute (set automatically when <code>filist</code> came from build_filtration with <code>max_dimension</code> supplied there), or to report every dimension present in <code>filist</code> 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)

```

persistent_laplacian *Persistent (combinatorial) Laplacian*

Description

Computes the persistent Laplacian $\Delta_q^{X,Y}$.

Usage

```
persistent_laplacian(X_simplices, Y_simplices, q)
```

Arguments

X_simplices, Y_simplices	Lists of (maximal) simplices, same format used everywhere else in the package. X must be a subcomplex of Y.
q	The dimension.

Value

A list with the full Laplacian, its upper/down pieces, and the q-simplex basis (in row/column order) everything is expressed in.

plot_crocker *Plot a CROCKER matrix as a filled contour plot*

Description

Plot a CROCKER matrix as a filled contour plot

Usage

```
plot_crocker(cr)
```

Arguments

`cr` An object returned by [crocker](#), or a data frame with columns `t`, `epsilon`, `betti` (e.g. `cr$long`).

Value

A ggplot2 object: a tile plot of the Betti number over time and scale.

plot_landscape	<i>Plot a Persistence Landscape</i>
----------------	-------------------------------------

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	<i>Plot the optimal matching between two persistence diagrams</i>
---------------	---

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_distance .
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 $((\text{birth} + \text{death})/2, (\text{birth} + \text{death})/2)$ 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_morse_landscape` *Plot the density landscape and critical structure in interactive 3D*

Description

A 3D companion to [plot_morse_recon](#): renders the input density ρ as an actual terrain surface over the triangulated domain (using the triangles stored in `mr$filtration`), and overlays the discrete Morse critical structure on it with the **rgl** package, producing a mouse-rotatable 3D scene rather than a flat projection:

- the root (critical) vertices of `mr$dmvf`, the surviving density peaks after persistence simplification - as labelled spheres ("M1", "M2", ...);
- the endpoints of `mr$dmvf$critical_edges`, the surviving "saddle-like" edges of Section 10.4.1 as smaller spheres;
- `mr$graph_edges` itself, i.e. the collected 1-unstable manifolds from [collect_g](#), literally the vertex-to- saddle connections along the reconstructed ridge lines as raised line segments on the surface.

Usage

```
plot_morse_landscape(
  mr,
  z_scale = NULL,
  show_critical_edges = TRUE,
  label_roots = TRUE,
  surface_col = c("#f7fbff", "#6baed6", "#08306b"),
  root_col = "red",
  saddle_col = "blue",
  edge_col = "black",
  point_radius = NULL,
  edge_lwd = 3,
  window_size = c(1400, 1000),
  title_cex = 1.4
)
```

Arguments

<code>mr</code>	A "morse_recon" object returned by morse_recon (must have been run with <code>vertex_coords</code>).
<code>z_scale</code>	Vertical exaggeration applied to <code>mr\$rho</code> before plotting. Default NULL auto-scales so the density relief spans about 35% of the horizontal extent of <code>mr\$coords</code> .

show_critical_edges	Logical; if TRUE (default) also mark the endpoints of <code>mr\$dmvf\$critical_edges</code> (the saddle-like edges).
label_roots	Logical; if TRUE (default) label root vertices "M1", "M2", ... in the scene.
surface_col, root_col, saddle_col, edge_col	Colour for the density surface, the root markers, the saddle markers, and the graph_edges ridge lines, respectively.
point_radius, edge_lwd	Marker radius, auto-scaled from the domain size when NULL) and line width for graph_edges.
window_size	Length-2 c(width, height) in pixels for the new rgl device (only used when <code>new_window = TRUE</code>).
title_cex	Character expansion for the <code>rgl::title3d()</code> title.

Value

A interactive 3D **rgl** scene.

Display

On macOS, opening a native **rgl** window requires XQuartz; Windows and Linux do not need it. To avoid a native window altogether, render the scene as a WebGL widget in the RStudio Viewer or a browser:

```
options(rgl.useNULL = TRUE)
plot_morse_landscape(mr)
rgl::rglwidget()
```

plot_morse_recon	<i>Plot the graph reconstructed by morse_recon</i>
------------------	--

Description

Draws the reconstructed graph $\hat{G}$ (the union of 1-unstable manifolds of the surviving high-persistence critical edges) over the input vertices, optionally coloured by the input density ρ .

Usage

```
plot_morse_recon(mr, show_density = TRUE, point_size = 0.6, edge_size = 0.9)
```

Arguments

mr	A "morse_recon" object returned by morse_recon .
show_density	Logical; if TRUE (default) colour vertices by <code>mr\$rho</code> using a continuous viridis scale.
point_size, edge_size	Point/line sizes passed to <code>ggplot2</code> .

Value

A ggplot2 object.

plot_morse_vpath	<i>Plot a local patch of the triangulation with its discrete gradient field</i>
------------------	---

Description

Figure zooms into a small neighbourhood of the mesh (as most DMT papers illustrate the vector field, since drawing it over the whole domain is unreadable) and draws the actual computed structure on top of the local triangulation:

- every triangulation edge with both endpoints in the window, in light grey, for context;
- the surviving critical edges of `mr$dmvf$critical_edges` that fall inside the window, as thick red segments. These are the "saddles" that did not get matched to a vertex;
- for every non-critical vertex v in the window whose DMVF match `mr$dmvf$parent[[v]]` is also in the window, a black arrow from v to the midpoint of its matched edge $(v, \text{parent}(v))$, the vertex-edge V-path arrows themselves.

Usage

```
plot_morse_vpath(
  mr,
  center,
  radius,
  vertex_size = 1.2,
  arrow_size = 0.12,
  critical_size = 1.3
)
```

Arguments

<code>mr</code>	A "morse_recon" object returned by morse_recon .
<code>center</code>	Length-2 numeric, an (x, y) point in the same units as <code>mr\$coords</code> to centre the window on.
<code>radius</code>	Euclidean radius (same units as <code>mr\$coords</code>) of the window around center.
<code>vertex_size, arrow_size, critical_size</code>	Point size, arrow-head size, and line width for the critical edges, respectively.

Value

A ggplot2 object.

plot_persistence	<i>Plot Persistence Diagram</i>
------------------	---------------------------------

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	<i>Restrict a filtration list to simplices up to a given dimension</i>
---------------------	--

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 $\leq \text{max_dimension}$, 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	<i>Construct a Vietoris–Rips Complex (1-skeleton + maximal simplices)</i>
---------------------	---

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 $< \epsilon$.

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	<i>Wasserstein distance between two persistence diagrams</i>
----------------------	--

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

<code>df1, df2</code>	Persistence diagram data frames with columns <code>dim</code> , <code>birth</code> , <code>death</code> (e.g. the output of <code>extract_persistence_pairs()</code> or <code>persistence_pairs()</code>).
<code>dimension</code>	Homological dimension to compare.
<code>p</code>	Wasserstein order ($p \geq 1$). Defaults to 2.
<code>ground</code>	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	<i>Construct a Witness Complex</i>
----------------	------------------------------------

Description

Construct a Witness Complex

Usage

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

Arguments

<code>points</code>	A numeric matrix or data.frame of witness points S , one point per row.
<code>landmarks</code>	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 .
<code>epsilon</code>	A non-negative numeric scale.
<code>nu</code>	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)
```

<code>zigzag_persistence</code>	<i>Compute the persistence barcode of a zigzag filtration of simplicial complexes</i>
---------------------------------	---

Description

Internally, each $K_i \rightarrow K_{i+1}$ step is expanded into a sequence of single simplex insertions (faces before cofaces) or single simplex deletions (cofaces before faces), exactly as [boundary_info/persistence_pairs](#) assume for ordinary filtrations. At every elementary insertion or deletion, the representative-cycle basis of each affected homology dimension is updated directly via linear algebra (reusing [boundary](#) and [faces](#) for every boundary-matrix computation), an insertion either creates a new cycle (birth in dimension q) or turns an existing cycle into a boundary (death in dimension $q - 1$); a deletion either destroys an existing cycle (death in dimension q) or frees a previously-trivial cycle from being a boundary (birth in dimension $q - 1$), where q is the dimension of the simplex being inserted/deleted.

Usage

```
zigzag_persistence(complexes, max_dimension = NULL)
```

Arguments

complexes	A list of length $n + 1$: <code>complexes[[i+1]]</code> is K_i , a list of simplices exactly like the simplices argument of boundary/faces (each simplex a numeric vector; only the maximal simplices need to be listed, faces are inferred).
max_dimension	Optional integer cap: dimensions above this are not tracked (saves work for large complexes where only e.g. H_0/H_1 are of interest). NULL (default) tracks every dimension present.

Value

A data frame with columns `dim`, `birth`, `death` (integer indices into $0, \dots, n$, i.e. into complexes), one row per bar. Every bar uses a **closed interval**: `death` is the last index at which the class is still present, and a class still alive at K_n is reported with `death = n`, since a zigzag filtration, unlike an ordinary one, has no canonical "infinity" to extend to. This differs by one from [persistence_pairs](#)'s convention, where `death` is the index of the killing simplex and the interval is half-open (`birth <= i < death`); on a purely-growing filtration the two agree after `death_here = death_there - 1`.

Index

AlphaComplex, [3](#), [14](#)
as_filtration, [3](#), [4](#)
augmented_cost_matrix, [4](#), [37](#)

betti_number, [16](#), [23](#)
bottleneck_distance, [4](#), [11](#), [15](#), [31](#)
boundary, [5](#), [38](#), [39](#)
boundary_info, [6](#), [38](#)
build_cubical_filtration, [6](#), [25](#)
build_filtration, [3](#), [6](#), [7](#), [7](#), [8](#), [10–13](#), [17](#),
[26](#), [28](#), [35](#), [38](#)
build_flood_filtration, [8](#), [20](#), [35](#)

CechComplex, [9](#)
circumsphere, [3](#)
collect_g, [10](#), [27](#), [32](#)
compare_complexes, [10](#)
complex_distance, [11](#)
crocker, [13](#), [30](#)

DelaunayComplex, [3](#), [8](#), [14](#)
diagram_matching, [15](#)

euler_characteristic, [16](#)
extract_persistence_pairs, [6](#), [17](#)

faces, [18](#), [38](#), [39](#)
flood_complex, [8](#), [9](#), [19](#)
flood_persistence, [8](#), [20](#)

gauss_jordan_eliminate, [20](#)
generate_landmarks, [8](#), [11](#), [21](#), [38](#)
graph_laplacian, [21](#)

hodge_laplacian, [22](#)
homology, [22](#)

im, [21](#), [23](#), [23](#), [24](#)

ker, [21](#), [23](#), [24](#)

lower_star_filtration, [25](#), [26](#)

morse_recon, [10](#), [25](#), [32–34](#)

partial_pers_dmvf, [10](#), [26](#), [26](#)
persistence_landscape, [27](#)
persistence_pairs, [11](#), [17](#), [20](#), [28](#), [38](#), [39](#)
persistent_laplacian, [29](#)
plot_crocker, [14](#), [29](#)
plot_landscape, [30](#)
plot_matching, [15](#), [31](#)
plot_morse_landscape, [32](#)
plot_morse_recon, [26](#), [32](#), [33](#)
plot_morse_vpath, [34](#)
plot_persistence, [11](#), [35](#)

restrict_filtration, [6](#), [17](#), [20](#), [28](#), [35](#)

VietorisRipsComplex, [36](#)

wasserstein_distance, [4](#), [11](#), [15](#), [31](#), [37](#)
WitnessComplex, [8](#), [37](#)

zigzag_persistence, [38](#)