Package 'topologyR'

April 5, 2026

Type Package

Title Topological Connectivity Analysis for Numeric Data

Version 0.2.0

Description Topological data analysis methods based on graph-theoretic approaches for discovering topological structures in data. Constructs topological spaces from graphs following Nada et al. (2018) <doi:10.1002/mma.4726>, with visibility graph construction for time series following Lacasa et al. (2008) <doi:10.1073/pnas.0709247105>. Supports directed visibility graphs for bitopological analysis of temporal irreversibility (Kelly, 1963), and Alexandrov topology construction from reachability preorders.

License MIT + file LICENSE

URL https://github.com/IsadoreNabi/topologyR

BugReports https://github.com/IsadoreNabi/topologyR/issues

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.0.0)

Imports ggplot2, Rcpp, stats, utils

LinkingTo Rcpp

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation yes

Author José Mauricio Gómez Julián [aut, cre]

Maintainer José Mauricio Gómez Julián <isadore.nabi@pm.me>

Repository CRAN

Date/Publication 2026-04-05 06:50:02 UTC

Contents

`analyze_topology_factors` *Analyze topology characteristics for different IQR factors*

Description

Analyzes how different IQR factors affect topology characteristics. Helps determine the optimal factor by showing how the factor choice impacts base size and set sizes.

Note: This function uses threshold-based neighborhoods (metric approximation), not the graph-theoretic approach of Nada et al. (2018). For the theoretically faithful approach, use visibility graphs with `generate_topology`.

Usage

```
analyze_topology_factors(data, factors = NULL, plot = TRUE)
```

Arguments

`data`	Numeric vector containing the data to analyze.
`factors`	Numeric vector of factors to test (default: `c(1, 2, 4, 8, 16)`).
`plot`	Logical, whether to return a plot object (default: `TRUE`).

Value

A `data.frame` with columns:

factor Numeric. The IQR factor used.

threshold Numeric. The calculated threshold (IQR/factor).

base_size Integer. Number of sets in the base.

max_set_size Integer. Size of the largest set.

min_set_size Integer. Size of the smallest set.

If `plot = TRUE`, the data.frame carries an attribute `"plot"` containing a `ggplot` object.

Examples

```
data <- rnorm(50)
results <- analyze_topology_factors(data)
print(results)
```

---

bitopology_invariants *Compute bitopological invariants from forward and backward topologies*

Description

Given the forward topology τ^+ and backward topology τ^- (as returned by `generate_topology`), computes bitopological invariants that quantify temporal irreversibility and structural asymmetry.

Usage

```
bitopology_invariants(
  forward,
  backward,
  n_elements,
  undirected = NULL,
  alexandrov = NULL
)
```

Arguments

`forward`	The forward Nada topology τ^+ (output of `generate_topology` called with forward adjacency).
`backward`	The backward Nada topology τ^- (output of `generate_topology` called with backward adjacency).
`n_elements`	Integer. Total number of elements in the ground set V.
`undirected`	The undirected Nada topology (optional, for comparison).
`alexandrov`	The Alexandrov topology (optional, for resolution comparison).

Details

Pairwise connectedness (Kelly, 1963): the bitopological space (X, τ^+, τ^-) is pairwise disconnected if there exists a proper non-empty subset $A \subset X$ that is simultaneously τ^+-open and τ^--closed (its complement is τ^--open). This is checked by iterating over all τ^+ open sets and testing whether their complement is τ^--open.

Irreversibility prediction (Lacasa & Toral, 2010): for a time series with asymmetric dynamics (e.g., gradual expansions and abrupt recessions), the forward topology τ^+ should be more connected than τ^-, because forward visibility is less obstructed during gradual rises than backward visibility after abrupt drops.

Resolution gain: the difference $|\mathcal{B}_{\mathrm{Nada}}| - |\mathcal{B}_A|$ quantifies how much additional structure the Nada intersection/union closure captures beyond the pure Alexandrov order structure. A large gain means the Nada pipeline is extracting genuinely new topological information.

Value

A `list` with:

forward_components Integer. Number of connected components in τ^+.

backward_components Integer. Number of connected components in τ^-.

forward_connected Logical. Whether τ^+ is connected.

backward_connected Logical. Whether τ^- is connected.

forward_base_size Integer. Number of base elements in τ^+.

backward_base_size Integer. Number of base elements in τ^-.

irreversibility_components Numeric in the range 0 to 1. Normalized asymmetry of component counts: $|C^+ - C^-| / \max(C^+, C^-)$. Zero means equal component counts (necessary for reversibility).

irreversibility_base Numeric in the range 0 to 1. Normalized asymmetry of base sizes.

asymmetry_direction Integer. $C^- - C^+$. Positive means the forward topology is more connected (fewer components) than the backward topology, consistent with gradual expansions (forward visibility) and abrupt contractions (backward obstruction).

pairwise A list with pairwise connectedness results. Contains `pairwise_connected` (logical or `NA`), `clopen_forward` (integer vector, the forward-open / backward- closed set if disconnected), and `clopen_backward` (its complement). Requires both topologies to be fully enumerated (`max_open_sets > 0` and `complete = TRUE`).

resolution A list comparing the Nada and Alexandrov topologies (if `alexandrov` is provided). Contains base size comparisons and relative resolution gains.

undirected_info A list with undirected topology summary (if `undirected` is provided).

References

Kelly, J. C. (1963). Bitopological spaces. *Proceedings of the London Mathematical Society*, 3(1), 71-89.

See Also

`generate_bitopology` for the complete pipeline.

Examples

```
series <- c(3, 1, 4, 1, 5, 9, 2, 6)
g <- horizontal_visibility_graph(series, directed = TRUE)
tf <- generate_topology(g$out_adjacency, g$n, max_open_sets = 0L)
tb <- generate_topology(g$in_adjacency, g$n, max_open_sets = 0L)
inv <- bitopology_invariants(tf, tb, g$n)
inv$asymmetry_direction
```

---

calculate_thresholds *Calculate multiple threshold methods for topology analysis*

Description

Computes five different threshold methods for defining neighborhoods in threshold-based topology construction.

Note: Threshold-based methods produce metric topologies, not the graph-induced topologies of Nada et al. (2018). See `generate_topology` for the theoretically faithful approach.

Usage

```
calculate_thresholds(data)
```

Arguments

`data`	Numeric vector to calculate thresholds for.

Value

A named `list` with five threshold values:

mean_diff Mean of absolute differences between adjacent sorted values.

median_diff Median of absolute differences between adjacent sorted values.

sd Standard deviation of the data.

iqr IQR divided by 4.

dbscan k-th nearest neighbor distance (k = ceiling(log(n))).

Examples

```
calculate_thresholds(rnorm(100))
```

calculate_topology *Calculate topology base size for a given threshold*

Description

Calculate topology base size for a given threshold

Usage

```
calculate_topology(data, threshold)
```

Arguments

data	Numeric vector to analyze.
threshold	Numeric value for the threshold parameter. Must be non-negative.

Value

integer scalar. The number of sets in the topological base.

Examples

```
calculate_topology(rnorm(30), threshold = 0.5)
```

complete_topology *Create a complete-graph topology from data*

Description

Constructs a topology from the complete graph on the data points. In the complete graph, every vertex is a neighbor of every other vertex, so each neighborhood is $N(v) = V \setminus \{v\}$. The resulting topology is generated using the fixed-point closure algorithm.

Note: The complete graph produces trivial neighborhoods (all vertices except self). For meaningful topological structure, use visibility graphs with generate_topology.

Usage

```
complete_topology(data, verify_axioms = FALSE)
```

Arguments

data	Numeric vector containing the data points. Length must be between 2 and 64.
verify_axioms	Logical. Whether to verify topology axioms (default: FALSE).

Value

A `list` with the same structure as `generate_topology`.

Examples

```
result <- complete_topology(c(1, 2, 3, 4, 5))
result$n_open_sets
result$connected
```

`generate_alexandrov_topology` *Generate the Alexandrov topology from a directed visibility graph*

Description

Computes the Alexandrov topology induced by a directed acyclic graph (DAG) via bitset-based reachability propagation. The open sets are the upsets of the reachability relation: vertex v's minimal open set is the set of all vertices reachable from v via directed paths (including v itself).

The Alexandrov topology is a **lower bound** on the resolution of the Nada et al. (2018) topology: $\tau_A \subseteq \tau^+_{\text{Nada}}$. The difference in base sizes quantifies how much additional structure the intersection/union closure of Nada et al. captures beyond the pure order structure.

Usage

```
generate_alexandrov_topology(
  out_adjacency,
  n_elements,
  max_open_sets = 0L,
  check_connected = TRUE,
  verify_axioms = FALSE
)
```

Arguments

`out_adjacency`	A list of integer vectors, where element `i` contains the 1-based indices of all **forward** (later-in-time) neighbors of node `i`. Must represent a DAG where all edges go from lower to higher index. Typically obtained from `horizontal_visibility_graph` or `natural_visibility_graph` with `directed = TRUE`.
`n_elements`	Integer. Total number of elements in the ground set V.
`max_open_sets`	Integer. Maximum number of open sets to enumerate (default: 0, meaning skip enumeration). The Alexandrov topology can have up to 2^n open sets, so enumeration is only feasible for small n. Connectivity is always computed from the base without enumeration.
`check_connected`	Logical. Whether to compute exact topological connectivity via the specialization preorder (default: `TRUE`).
`verify_axioms`	Logical. Whether to verify topology axioms after enumeration (default: `FALSE`). Only meaningful when the topology is fully enumerated.

Details

The algorithm computes reachability in $O(n \cdot m/64)$ time by processing vertices in reverse topological order (= reverse time order for visibility graphs) and propagating reachability sets as multi-word bitsets with OR operations. This reuses the same bitset infrastructure as the Nada engine with compile-time template dispatch for $n \leq 192$.

The Alexandrov topology on a finite set equipped with a preorder is canonical: there is a bijection between preorders and Alexandrov topologies (Alexandrov, 1937). For directed visibility graphs, the reachability preorder captures the temporal ordering structure of the time series.

Value

A `list` with the same structure as `generate_topology`:

subbase List of integer vectors. The upsets (= base for Alexandrov).

base List of integer vectors. Same as subbase (upsets are already closed under intersection).

base_complete Always `TRUE` (no iteration needed).

connected Logical. Exact topological connectivity.

components List of integer vectors. Exact connected components.

topology List of integer vectors, or `NULL`. Open sets if enumerated.

n_open_sets Integer, or `NA`. Number of open sets if enumerated.

complete Logical. Whether enumeration finished.

References

Alexandrov, P. (1937). Diskrete Raume. *Matematicheskii Sbornik*, 2(44), 501-519.

See Also

`generate_topology` for the Nada et al. topology, `generate_bitopology` for the complete bitopological analysis.

Examples

```
series <- c(3, 1, 4, 1, 5)
g <- horizontal_visibility_graph(series, directed = TRUE)
alex <- generate_alexandrov_topology(g$out_adjacency, g$n)
alex$connected
length(alex$base)
```

`generate_bitopology` *Generate a bitopological analysis from a time series*

Description

Performs a complete bitopological analysis of a time series by constructing directed visibility graphs and computing four topological spaces:

1. **Undirected Nada topology** τ: from the symmetric (undirected) visibility graph (standard Nada et al. 2018).
2. **Forward Nada topology** τ^+: from closed forward neighborhoods $N^+[v] = \{v\} \cup \{w : v \to w\}$.
3. **Backward Nada topology** τ^-: from closed backward neighborhoods $N^-[v] = \{v\} \cup \{w : w \to v\}$.
4. **Alexandrov topology** τ_A: from reachability upsets of the directed graph.

The pair (X, τ^+, τ^-) forms a **bitopological space** (Kelly, 1963). The divergence between τ^+ and τ^- quantifies temporal irreversibility: in a reversible process, $\tau^+ \cong \tau^-$; in an irreversible process (e.g., asymmetric business cycles), they diverge.

Usage

```
generate_bitopology(
  series,
  graph_type = c("hvg", "nvg"),
  max_open_sets = 0L,
  max_base_sets = 100000L,
  alexandrov = TRUE
)
```

Arguments

`series`	Numeric vector representing the time series.
`graph_type`	Character. Either `"hvg"` (Horizontal Visibility Graph) or `"nvg"` (Natural Visibility Graph). Default: `"hvg"`.
`max_open_sets`	Integer. Maximum open sets to enumerate per topology (default: 0, skip enumeration). Required for pairwise connectedness check.
`max_base_sets`	Integer. Maximum base elements during intersection closure for the Nada topologies (default: 100,000).
`alexandrov`	Logical. Whether to also compute the Alexandrov topology for resolution comparison (default: `TRUE`).

Details

The existing undirected engine is called three times (undirected, forward, backward) and the Alexandrov engine once. No new C++ code is needed for the forward and backward topologies: passing directed adjacency lists to the existing `generate_topology` produces the correct closed neighborhoods $N^+[v]$ or $N^-[v]$ automatically (the engine always adds the self-loop $v \in N[v]$).

Value

A `list` with:

graph The directed visibility graph (output of `horizontal_visibility_graph` or `natural_visibility_graph` with `directed = TRUE`).

undirected The undirected Nada topology (output of `generate_topology`).

forward The forward Nada topology τ^+.

backward The backward Nada topology τ^-.

alexandrov The Alexandrov topology τ_A (if requested).

invariants Bitopological invariants (output of `bitopology_invariants`).

References

Kelly, J. C. (1963). Bitopological spaces. *Proceedings of the London Mathematical Society*, 3(1), 71-89.

Lacasa, L., & Toral, R. (2010). Description of stochastic and chaotic series using visibility graphs. *Physical Review E*, 82(3), 036120.

See Also

`bitopology_invariants` for computing invariants from pre-computed topologies.

Examples

```
series <- c(3, 1, 4, 1, 5, 9, 2, 6)
bt <- generate_bitopology(series, graph_type = "hvg")
bt$invariants$forward_components
bt$invariants$backward_components
bt$invariants$irreversibility_components
```

`generate_topology` *Generate a topology from a graph's neighborhood structure*

Description

Given a graph (represented by its adjacency list of neighborhoods), generates the induced topology following Nada et al. (2018). The procedure is:

1. The neighborhoods $N(v)$ form the **subbase**.
2. The base is the closure of the subbase under finite intersections (wavefront propagation in C++).
3. Topological connectivity is determined **exactly** via the specialization preorder on the base (Alexandrov theorem for finite spaces), without requiring topology enumeration.
4. Optionally, the full topology (closure under arbitrary unions) is enumerated, subject to a `max_open_sets` limit.

The connectivity result is **exact**: the connected components returned are the true topological connected components, not approximations.

Usage

```
generate_topology(
  adjacency,
  n_elements,
  max_open_sets = 1000000L,
  max_base_sets = 100000L,
  check_connected = TRUE,
  verify_axioms = FALSE
)
```

Arguments

adjacency	A list of integer vectors, where element i contains the 1-based indices of all neighbors of node i. Typically obtained from horizontal_visibility_graph or natural_visibility_graph.
n_elements	Integer. Total number of elements in the ground set V.
max_open_sets	Integer. Maximum number of open sets to enumerate (default: 1,000,000). If the topology exceeds this limit, enumeration stops and complete = FALSE. Connectivity is unaffected (computed from the base, not the topology). Set to 0 to skip enumeration entirely.
max_base_sets	Integer. Maximum number of base elements during intersection closure (default: 100,000). If exceeded, closure stops early and base_complete = FALSE. Connectivity results remain valid but may be approximate if the base is incomplete.
check_connected	Logical. Whether to compute exact topological connectivity via the specialization preorder (default: TRUE).
verify_axioms	Logical. Whether to verify topology axioms after enumeration (default: FALSE). Only meaningful when the topology is fully enumerated (complete = TRUE).

Value

A list with:

subbase List of integer vectors. The neighborhoods used as subbase.

base List of integer vectors. The base (closure under intersections).

base_complete Logical. Whether the base closure finished within the max_base_sets limit.

connected Logical. Whether the topological space is connected. This is an **exact** result based on the specialization preorder. NA if check_connected = FALSE.

components List of integer vectors. The exact topological connected components. Each component is a vector of 1-based element indices.

topology List of integer vectors, or NULL. The open sets, if enumeration was requested and feasible.

n_open_sets Integer, or NA. Number of open sets if enumerated.

complete Logical. Whether the topology enumeration finished within the max_open_sets limit.

iterations_base Integer. Wavefront iterations for base closure.

iterations_topo Integer. Wavefront iterations for union closure.

axioms_ok Logical (only if `verify_axioms = TRUE` and `complete = TRUE`).

axiom_failure Character (only if axiom verification fails).

References

Nada, S., El Atik, A. E. F., & Atef, M. (2018). New types of topological structures via graphs. *Mathematical Methods in the Applied Sciences*, 41(15), 5801-5810. doi:10.1002/mma.4726

Examples

```
# From a small visibility graph
series <- c(3, 1, 4, 1, 5)
g <- horizontal_visibility_graph(series)
topo <- generate_topology(g$adjacency, g$n)
topo$connected
length(topo$components)
```

`horizontal_visibility_graph`

Construct the Horizontal Visibility Graph of a time series

Description

Constructs the Horizontal Visibility Graph (HVG) from a time series. Two observations (t_a, y_a) and (t_b, y_b) with $t_a < t_b$ are connected if and only if every intermediate observation (t_c, y_c) satisfies $y_c < \min(y_a, y_b)$.

The HVG is parameter-free and captures structural properties of the series dynamics. It is computed in $O(n)$ time using a stack-based algorithm.

Usage

```
horizontal_visibility_graph(series, directed = FALSE)
```

Arguments

`series`	Numeric vector representing the time series. The position index is used as the time coordinate.
`directed`	Logical. If `TRUE`, also returns directed adjacency lists (`out_adjacency` and `in_adjacency`) for constructing forward and backward Nada topologies or Alexandrov topologies. Edges are directed from earlier to later time (default: `FALSE`).

Details

The algorithm uses a monotone stack with $O(n)$ time and space complexity. Each element is pushed and popped at most once.

The adjacency list returned as `adjacency` provides the neighborhood structure $N(v)$ for each vertex, which serves as the subbase for inducing a topology following Nada et al. (2018).

When `directed = TRUE`, the function additionally returns directed adjacency lists. By construction, all edges satisfy `from < to` (earlier time to later time), so the directed graph is a DAG with the time index as topological order.

Value

A `list` with:

edges A `data.frame` with columns `from` and `to` (integer, 1-based indices) representing undirected edges.

n Integer. Number of observations in the series.

n_edges Integer. Number of edges in the graph.

adjacency A list of integer vectors, where element `i` contains the indices of all neighbors of node `i`. This is the neighborhood structure used as subbase for topology induction.

out_adjacency (Only if `directed = TRUE`.) A list of integer vectors, where element `i` contains the indices of all forward (later-in-time) neighbors. Used as subbase for the forward Nada topology τ^+ or as input for the Alexandrov topology.

in_adjacency (Only if `directed = TRUE`.) A list of integer vectors, where element `i` contains the indices of all backward (earlier-in-time) neighbors. Used as subbase for the backward Nada topology τ^-.

References

Luque, B., Lacasa, L., Ballesteros, F., & Luque, J. (2009). Horizontal visibility graphs: exact results for random time series. *Physical Review E*, 80(4), 046103. doi:10.1103/PhysRevE.80.046103

Examples

```
series <- c(3, 1, 4, 1, 5, 9, 2, 6)
g <- horizontal_visibility_graph(series)
g$n_edges
head(g$edges)

# Directed graph for bitopological analysis
gd <- horizontal_visibility_graph(series, directed = TRUE)
gd$out_adjacency[[1]]  # forward neighbors of first observation
```

is_topology_connected *Check if a topology is connected using undirected graph approach*

Description

Converts the topology into an undirected graph (two elements share an edge if they appear together in any open set) and checks graph connectivity via depth-first search.

Note: This is a necessary but not sufficient condition for topological connectivity. For an exact check, use `is_topology_connected_exact`.

Usage

```
is_topology_connected(topology)
```

Arguments

`topology`	A list of integer vectors representing the open sets.

Value

`logical` scalar. `TRUE` if the derived graph is connected, `FALSE` otherwise or if the topology is empty.

Examples

```
topology <- list(c(1, 2, 3), c(3, 4, 5))
is_topology_connected(topology)
```

is_topology_connected2 *Check if a topology is connected using directed graph approach*

Description

Converts the topology into a directed graph (sequential edges between sorted elements within each set) and checks reachability via DFS.

Note: This is a necessary but not sufficient condition for topological connectivity. For an exact check, use `is_topology_connected_exact`.

Usage

```
is_topology_connected2(topology)
```

Arguments

`topology`	A list of integer vectors representing the open sets.

Value

`logical` scalar. `TRUE` if all elements are reachable from the minimum element via directed edges, `FALSE` otherwise.

Examples

```
topology <- list(c(1, 2, 3), c(3, 4, 5))
is_topology_connected2(topology)
```

---

is_topology_connected_exact

Check topological connectivity of an existing topology

Description

Checks whether a topological space (V, τ) is connected by verifying the exact definition: no proper non-empty open set has an open complement. Uses multi-word bitset representation and hash lookup.

Usage

```
is_topology_connected_exact(topology, n_elements)
```

Arguments

`topology`	A list of integer vectors representing the open sets.
`n_elements`	Integer. Total number of elements in V.

Value

A `list` with:

connected Logical. `TRUE` if the space is topologically connected.

component1 Integer vector (only if disconnected). One component.

component2 Integer vector (only if disconnected). The complement.

Examples

```
# A connected topology
tau <- list(integer(0), 1:3, c(1L, 2L), c(2L, 3L))
is_topology_connected_exact(tau, 3L)

# A disconnected topology
tau2 <- list(integer(0), 1:4, c(1L, 2L), c(3L, 4L))
is_topology_connected_exact(tau2, 4L)
```

is_topology_connected_manual

Check if all elements are covered by the topology

Description

Checks whether every integer from 1 to the maximum value in the topology appears in at least one open set. This verifies **coverage** ($\bigcup \tau = X$), not topological connectivity.

Note: A space can have full coverage and be maximally disconnected (e.g., the discrete topology). For connectivity, use `is_topology_connected_exact`.

Usage

```
is_topology_connected_manual(topology)
```

Arguments

`topology`	A list of integer vectors representing the open sets.

Value

`logical` scalar. `TRUE` if all elements from 1 to the maximum are present in at least one set.

Examples

```
topology <- list(c(1, 2, 3), c(3, 4, 5))
is_topology_connected_manual(topology)
```

natural_visibility_graph

Construct the Natural Visibility Graph of a time series

Description

Constructs the Natural Visibility Graph (NVG) from a time series. Two observations (t_a, y_a) and (t_b, y_b) with $t_a < t_b$ are connected if and only if every intermediate observation (t_c, y_c) satisfies:

$$y_c < y_a + (y_b - y_a) \cdot \frac{t_c - t_a}{t_b - t_a}$$

The NVG is parameter-free and richer in structure than the HVG (more edges), preserving dynamical properties such as periodicity, chaoticity, and long-range correlations. Computed in $O(n \log n)$ expected time.

Usage

```
natural_visibility_graph(series, directed = FALSE)
```

Arguments

series	Numeric vector representing the time series. The position index is used as the time coordinate.
directed	Logical. If TRUE, also returns directed adjacency lists (out_adjacency and in_adjacency) for constructing forward and backward Nada topologies or Alexandrov topologies. Edges are directed from earlier to later time (default: FALSE).

Details

The algorithm uses divide-and-conquer on the global maximum. The maximum point blocks all cross-edges between the left and right halves, so the problem decomposes cleanly. Edges from the pivot are found by a linear slope scan. Expected time is $O(n \log n)$; worst case (monotone data) is $O(n^2)$.

When directed = TRUE, the function additionally returns directed adjacency lists. By construction, all edges satisfy from < to (earlier time to later time), so the directed graph is a DAG.

Value

A list with the same structure as horizontal_visibility_graph.

References

Lacasa, L., Luque, B., Ballesteros, F., Luque, J., & Nuno, J. C. (2008). From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13), 4972-4975. doi:10.1073/pnas.0709247105

Examples

```
series <- c(3, 1, 4, 1, 5, 9, 2, 6)
g <- natural_visibility_graph(series)
g$n_edges
head(g$edges)

# Directed graph for bitopological analysis
gd <- natural_visibility_graph(series, directed = TRUE)
gd$out_adjacency[[1]]  # forward neighbors of first observation
```

---

simplest_topology *Create the discrete topology (completely disconnected)*

Description

Generates the discrete topology on n elements, where the base consists of all singleton sets $\{\{1\}, \{2\}, \ldots, \{n\}\}$. The full topology is the power set 2^V, but only the base is returned explicitly (enumerating 2^n sets is impractical for large n).

Usage

```
simplest_topology(data)
```

Arguments

data	Numeric vector containing the data points.

Value

A list with:

subbase List of singleton sets.

base Same as subbase (singletons are already a base).

topology_type Character: "discrete".

n Integer. Number of elements.

connected Logical. Always FALSE for $n \geq 2$ (the discrete topology is maximally disconnected).

Examples

```
result <- simplest_topology(c(1, 2, 3, 4, 5))
result$connected
```

---

visualize_topology_thresholds

Visualize and compare different threshold methods

Description

Visualize and compare different threshold methods

Usage

```
visualize_topology_thresholds(data, plot = TRUE)
```

Arguments

data	Numeric vector to analyze.
plot	Logical indicating whether to return plot objects (default: TRUE).

Value

A data.frame with columns method, threshold, and base_size. If plot = TRUE, carries an attribute "plots" containing a list of three ggplot objects.

Examples

```
data <- rnorm(50)
results <- visualize_topology_thresholds(data)
```

Index