

Package: topologyR (via r-universe)

October 9, 2026

Type Package

Title Topological Connectivity Analysis for Numeric Data

Version 0.4.0

Description Builds topologies on the vertex set of a graph from the closed neighbourhoods of its vertices, a closed-neighbourhood variant of the construction of Nada, El Atik and Atef (2018) <[doi:10.1002/mma.4726](https://doi.org/10.1002/mma.4726)>, and computes their connected components exactly through the specialization preorder, without enumerating the topology. For time series it builds the horizontal and natural visibility graphs (Luque et al., 2009, <[doi:10.1103/PhysRevE.80.046103](https://doi.org/10.1103/PhysRevE.80.046103)>; Lacasa et al., 2008, <[doi:10.1073/pnas.0709247105](https://doi.org/10.1073/pnas.0709247105)>) at equally or unequally spaced instants, deciding the natural criterion exactly under IEEE 754 double arithmetic, and from their directed versions the forward and backward topologies, a bitopological space in the sense of Kelly (1963) <[doi:10.1112/plms/s3-13.1.71](https://doi.org/10.1112/plms/s3-13.1.71)>, together with the Alexandrov topology of reachability. Reading a series backwards, with its instants reflected, exchanges the two topologies, so the difference of their numbers of components changes sign.

License GPL (>= 3)

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

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

Encoding UTF-8

Roxygen list(markdown = TRUE)

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

Config/roxygen2/version 8.0.0
Repository https://isadorenabi.r-universe.dev
Date/Publication 2026-10-06 03:29:18 UTC
RemoteUrl https://github.com/IsadoreNabi/topologyR
RemoteRef HEAD
RemoteSha e43f2828573df3bfe2ae5df4763cb004579f571d

Contents

analyze_topology_factors	2
bitopology_invariants	4
calculate_thresholds	7
calculate_topology	8
complete_topology	9
generate_alexandrov_topology	10
generate_bitopology	12
generate_topology	15
horizontal_visibility_graph	18
is_topology_connected	20
is_topology_connected_exact	21
is_topology_connected_manual	22
is_topology_connected2	23
natural_visibility_exactness	24
natural_visibility_graph	27
simplest_topology	30
time_reverse	31
visualize_topology_thresholds	33
Index	34

analyze_topology_factors	<i>Size of the threshold-neighbourhood base for several IQR factors</i>
--------------------------	---

Description

For each factor f , builds the threshold neighbourhoods of the data with radius $h = \text{IQR}/f$ and reports the size of the base they generate and the sizes of its largest and smallest members. It shows how the choice of the factor changes the granularity of a metric-threshold topology; it does not select a factor, because no criterion of optimality is defined.

Usage

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

Arguments

<code>data</code>	Numeric vector with at least two elements and no missing values. No default.
<code>factors</code>	Positive numeric vector of divisors of the interquartile range. Default NULL, which means <code>c(1, 2, 4, 8, 16)</code> .
<code>plot</code>	Logical scalar; if TRUE (default) the result carries a ggplot object in its "plot" attribute.

Details

The neighbourhood of observation i is the set of indices j with $|x_j - x_i| \leq h$. The family counted in `base_size` consists of the empty set, the whole index set and every non-empty intersection of two neighbourhoods, deduplicated. Each neighbourhood collects the observations whose values lie in a closed interval, and an intersection of several such sets is the intersection of the two whose intervals have the largest left and the smallest right endpoint; the pairwise intersections are therefore all the non-empty finite intersections, and together with the whole index set they form the base generated by the neighbourhoods.

Value

A data.frame with one row per factor and the columns `factor` (numeric), `threshold` (numeric, IQR/f), `base_size` (integer, number of sets in the family described in Details, the empty set and the whole index set included), `max_set_size` and `min_set_size` (integers, largest and smallest cardinality among the non-empty intersections of two neighbourhoods). If `plot = TRUE`, the data frame carries an attribute "plot" with a ggplot object.

Methodological notes

These are metric-threshold neighbourhoods on the values, not the graph-induced neighbourhoods of the visibility-graph pipeline; the radius is a tuning choice of the user. Up to version 0.3.0 the two size columns were taken over the whole family, so `min_set_size` was always 0 (the empty set) and `max_set_size` always n (the whole index set); they are now taken over the neighbourhood intersections, which is where the factor acts.

Dependencies

`stats::IQR()` for the radius; 'ggplot2' (Imports) for the optional plot.

See Also

[calculate_topology\(\)](#) for a single threshold, [calculate_thresholds\(\)](#) for data-driven radii, [generate_topology\(\)](#) for graph-induced topologies.

Examples

```
set.seed(1)
results <- analyze_topology_factors(rnorm(50), plot = FALSE)
results
```

bitopology_invariants *Invariants of a pair of forward and backward topologies*

Description

Given the forward topology τ^+ and the backward topology τ^- computed by `generate_topology()`, reports their numbers of connected components and of base sets, two summaries of the difference between the component counts, the pairwise connectedness of the bitopological space, decided exactly, and, when supplied, comparisons with the Alexandrov and the undirected topologies.

Usage

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

Arguments

forward	Result of <code>generate_topology()</code> for the out-neighbour lists, computed with <code>check_connected = TRUE</code> . No default.
backward	Result of <code>generate_topology()</code> for the in-neighbour lists, computed with <code>check_connected = TRUE</code> . No default.
n_elements	Integer scalar: the number of points; must equal the number of points covered by the components. No default.
undirected	Optional result of <code>generate_topology()</code> for the undirected lists. Default NULL.
alexandrov	Optional result of <code>generate_alexandrov_topology()</code> for the out-neighbour lists. Default NULL.

Details

The two base sizes are equal. Let R be a binary relation on a finite non-empty set V , with rows $F(v) = \{w : vRw\}$ and columns $C(v) = \{w : wRv\}$, and let $\mathcal{B}(F)$ be the set of non-empty intersections of non-empty families of rows, likewise $\mathcal{B}(C)$. Then $|\mathcal{B}(F)| = |\mathcal{B}(C)|$. *Proof.* For $A \subseteq V$ let $A' = \bigcap_{v \in A} F(v)$ (with $\emptyset' = V$) and for $B \subseteq V$ let $B^* = \{v : B \subseteq F(v)\} = \bigcap_{w \in B} C(w)$. Both maps reverse inclusion and $A \subseteq A'^*$, $B \subseteq B^{*'}$, so $A' = A'^{**}$ and $B^* = B^{*'}^*$; hence $'$ and $*$ are mutually inverse bijections between the intersections of rows over all A , the intents, and the intersections of columns over all B , the extents, in the terminology of formal concept analysis (Ganter & Wille, 1999), and there are as many of each. $\mathcal{B}(F)$ is the set of intents without V , unless V is a row, and without $\emptyset$, which is an intent exactly when no column is V (the intersection of all rows is the set of w that are related to every v); symmetrically for $\mathcal{B}(C)$ with rows and columns exchanged. The two corrections are the same two events, so the cardinals

agree. The closed forward and backward neighbourhoods are the rows and columns of the reflexive closure of the arc relation, and the engine keeps exactly the non-empty intersections, so `forward_base_size` equals `backward_base_size` whenever both bases are complete. For the same reason `nada_forward_base_gain` equals `nada_backward_base_gain` whenever both are defined.

Comparison with the Alexandrov topology. $\tau_A \subseteq \tau^+$ for every digraph, and the Alexandrov topology of the reversed reachability is contained in τ^- ; both have as many base sets as the digraph has strongly connected components. Each gain is non-negative, because every smallest open set of τ^+ is an intersection of members of the subbase and so belongs to B^+ , and τ^+ , being finer, separates at least as many points as τ_A . The gains count sets of the construction; they do not measure how much finer τ^+ is than τ_A , since the size of an intersection-closed base depends on the subbase that generates it. On a visibility graph the Alexandrov topology is the chain of n sets $\{i, \dots, n\}$, so the gains are $|B^+| - n$.

Pairwise connectedness. (V, τ^+, τ^-) is pairwise disconnected when V is the union of two disjoint non-empty sets, the first open in τ^+ and the second open in τ^- . The function decides it exactly from the two subbases, which are always complete, without enumerating either topology. Let U_x^+ and U_x^- be the smallest open sets of x in τ^+ and τ^- , the intersections of the members of each subbase that contain x , and let H be the digraph on V with an arc $x \rightarrow y$ when $y \in U_x^+$ or $x \in U_y^-$. A set A is open in τ^+ with its complement open in τ^- if and only if no arc of H leaves A : $A \in \tau^+$ means $U_x^+ \subseteq A$ for every $x \in A$, and $V \setminus A \in \tau^-$ means $U_y^- \subseteq V \setminus A$ for every $y \notin A$, that is, $x \in A$ and $x \in U_y^-$ imply $y \in A$. A proper non-empty set that no arc leaves exists if and only if H is not strongly connected, and then a strongly connected component that no arc leaves is one; the function takes the first component that Tarjan's (1972) algorithm completes. The smallest open sets cost the intersections of the members that contain each point, $O(\sum_S |S|^2)$ over the members S of the two subbases (expected, since `intersect()` hashes), and the search costs $O(n + a)$ for the a arcs of H ; with at most n members of at most n points, the total is $O(n^3)$ at worst, and much less when the neighbourhoods are small. The accompanying article decides the property by searching the enumerated topologies (its algorithm 5), with an undecided answer when an enumeration is cut; this criterion gives the same answer whenever that search decides, and never leaves it undecided.

On visibility graphs the answer is known in advance. When every arc goes from a lower to a higher index, as in every directed visibility graph, and $n \geq 2$, the space is pairwise disconnected. For each $k < n$, the final segment $\{k+1, \dots, n\}$ is the union of the closed out-neighbourhoods of its points, each of which lies inside it, so it is open in τ^+ ; in the same way the initial segment $\{1, \dots, k\}$ is open in τ^- ; and the two segments partition V . On the results of `generate_bitopology()` the check therefore carries no information about the series: it answers FALSE, with a witness found by the search (the segments $\{n\}$ and $\{1, \dots, n-1\}$ are always one, but the search can return another). It can answer TRUE on digraphs with cycles, whose topologies are computed with `generate_topology()`.

Value

A list with:

forward_components, backward_components Integers C^+ and C^- : the numbers of connected components, exact even when a base was truncated.

forward_connected, backward_connected Logical.

forward_base_size, backward_base_size Integers $|B^+|$ and $|B^-|$: the numbers of sets in the two bases, or NA when the corresponding base was truncated.

forward_base_complete, backward_base_complete Logical: the base_complete flags of the two topologies.

irreversibility_components Numeric in $[0, 1]$: $I_C = |C^+ - C^-| / \max(C^+, C^-)$.

asymmetry_direction Integer $D = C^- - C^+$.

pairwise List with pairwise_connected (logical, never NA: the property is decided exactly, without enumerating either topology) and, when it is FALSE, copen_forward, a proper non-empty τ^+ -open set whose complement is τ^- -open, and copen_backward, that complement.

resolution NULL unless alexandrov is given; otherwise a list with alexandrov_base_size, alexandrov_components, alexandrov_connected, nada_forward_base_gain and nada_backward_base_gain ($|B^+| - |B_A|$ and $|B^-| - |B_A|$, NA with a truncated base) and the same differences divided by $\max(|B_A|, 1)$ (nada_forward_relative_gain, nada_backward_relative_gain).

undirected_info NULL unless undirected is given; otherwise a list with undirected_components, undirected_connected and undirected_base_size (NA with a truncated base).

Methodological notes

irreversibility_components and asymmetry_direction are descriptive statistics of the pair of topologies of one series. By the time-reversal theorem T1 (see [generate_bitopology\(\)](#)), reading the series backwards with its instants reflected leaves I_C unchanged and changes the sign of D , so both vanish on a palindrome observed on a palindromic grid. Neither proves anything about the reversibility of the process that generated the series, and a positive or negative D is not a prediction of any particular shape of the series. I_C compares counts: two different partitions can have the same number of components.

The connected components are exact whether or not a base was truncated; the base sizes and the gains are not reported for a truncated base, because the number of sets found before the cut is only a lower bound. Up to version 0.2.0 the function also returned irreversibility_base, the normalized difference of the two base sizes, which the theorem above makes identically zero; it was removed in 0.3.0.

Dependencies

Base R only.

References

Ganter, B., & Wille, R. (1999). Formal concept analysis: Mathematical foundations. Springer. <https://doi.org/10.1007/978-3-642-59830-2>

Kelly, J. C. (1963). Bitopological spaces. Proceedings of the London Mathematical Society, s3-13(1), 71-89. <https://doi.org/10.1112/plms/s3-13.1.71>

Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1(2), 146-160. <https://doi.org/10.1137/0201010>

See Also

[generate_bitopology\(\)](#) for the complete pipeline and theorem T1.

Examples

```

g <- horizontal_visibility_graph(c(3, 1, 4, 1, 5, 9, 2, 6), 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
inv$forward_base_size == inv$backward_base_size

small <- generate_bitopology(c(3, 1, 4))
small$invariants$pairwise

```

calculate_thresholds *Candidate radii for threshold neighbourhoods*

Description

Computes five scales of the data that can serve as radii for the threshold neighbourhoods of [calculate_topology\(\)](#) and [visualize_topology_thresholds\(\)](#). They are descriptive scales, not estimates of an optimal radius.

Usage

```
calculate_thresholds(data)
```

Arguments

data Numeric vector with at least two elements and no missing values. No default.

Value

A named list of five numeric scalars:

mean_diff Mean of the gaps between consecutive sorted values.

median_diff Median of the gaps between consecutive sorted values.

sd Standard deviation of the data.

iqr Interquartile range divided by 4.

dbscan The (kn) -th smallest of the $n(n - 1)/2$ pairwise distances, with $k = \lceil \log n \rceil$ and the position capped at the number of distances.

Methodological notes

The dbscan entry keeps its historical name, but it is a single order statistic of all pairwise distances; it is not the distance from any observation to its k -th nearest neighbour.

Dependencies

stats::dist(), stats::median(), stats::sd() and stats::IQR().

See Also

[calculate_topology\(\)](#), [visualize_topology_thresholds\(\)](#).

Examples

```
set.seed(1)
calculate_thresholds(rnorm(100))
```

calculate_topology	<i>Size of the threshold-neighbourhood base for one radius</i>
--------------------	--

Description

Builds the threshold neighbourhoods of the data with a given radius and returns the number of sets in the base they generate, counted as in [analyze_topology_factors\(\)](#).

Usage

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

Arguments

data	Numeric vector with at least two elements and no missing values. No default.
threshold	Non-negative numeric scalar, the radius. No default.

Value

An integer scalar: the number of distinct sets among the empty set, the whole index set and the non-empty intersections of two neighbourhoods (see [analyze_topology_factors\(\)](#) for why these are all the finite intersections).

Methodological notes

The count includes the empty set and the whole index set. Missing values are rejected; up to version 0.3.0 they were silently dropped from every neighbourhood.

Dependencies

Base R only.

See Also

[analyze_topology_factors\(\)](#), [calculate_thresholds\(\)](#).

Examples

```
set.seed(1)
calculate_topology(rnorm(30), threshold = 0.5)
```

complete_topology	<i>The topology induced by the complete graph</i>
-------------------	---

Description

Runs `generate_topology()` on the complete graph over the indices of the data. Every closed neighbourhood of the complete graph is the whole index set, so the result is the indiscrete topology, whose only open sets are the empty set and the whole set, and the space is connected.

Usage

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

Arguments

data	Numeric vector with at least two elements and no missing values; only its length is used. No default.
verify_axioms	Logical scalar passed to <code>generate_topology()</code> . Default FALSE.

Value

The list returned by `generate_topology()`: base $\{V\}$, two open sets when enumerated, one component.

Methodological notes

The construction uses closed neighbourhoods $N[v] = \{v\} \cup N(v)$, and in the complete graph $N[v] = V$ for every vertex. The result is a reference point: the coarsest topology on the set.

Dependencies

The compiled engine of `generate_topology()` (via 'Rcpp').

See Also

`simplest_topology()` for the opposite extreme, the discrete topology.

Examples

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

generate_alexandrov_topology

Alexandrov topology of the reachability preorder of a digraph

Description

Computes the Alexandrov topology of the reachability preorder of a directed graph: its open sets are the sets closed under following arcs, and the smallest open set containing v is the set of vertices reachable from v by directed paths, v included. It accepts any digraph and reports which of three cases it received; `generate_bitopology()` applies it to directed visibility graphs, where it is always the same chain.

Usage

```
generate_alexandrov_topology(
  out_adjacency,
  n_elements,
  max_open_sets = 0L,
  check_connected = TRUE,
  verify_axioms = FALSE,
  expect = c("any", "index_ordered_dag")
)
```

Arguments

<code>out_adjacency</code>	List of length <code>n_elements</code> ; element i is an integer vector with the 1-based indices of the direct successors of vertex i . Indices outside $1, \dots, n$ are ignored. No default.
<code>n_elements</code>	Integer scalar, at least 1: the number of vertices. Must equal <code>length(out_adjacency)</code> . No default.
<code>max_open_sets</code>	Non-negative integer scalar (default 0, no enumeration): the enumeration of the open sets stops as soon as their number exceeds it. There can be up to 2^n of them.
<code>check_connected</code>	Logical scalar (default TRUE): whether to compute the connected components.
<code>verify_axioms</code>	Logical scalar (default FALSE): whether to check, when the enumeration is complete, that the enumerated family contains the empty set and the whole set and is closed under the union and the intersection of any two members, the axioms of a topology on a finite set.
<code>expect</code>	Character scalar: "any" (default) accepts every digraph; "index_ordered_dag" verifies, in $O(m)$, that every arc goes from a lower to a higher index, as in a directed visibility graph, and stops with an error naming the first arc that does not.

Details

Three cases are handled. (1) Every arc goes from a lower to a higher index, as in directed visibility graphs: the reachable sets are propagated in reverse index order as bitsets, $\text{reach}(v) = \{v\} \cup \bigcup_{w \in \text{out}(v)} \text{reach}(w)$, in $O(nm/64)$ word operations. (2) An acyclic digraph in another order: the vertices are first re-indexed along a topological order. (3) A digraph with directed cycles: mutually reachable vertices have the same reachable set, so the topology is computed on the condensation, found with the algorithm of Tarjan (1972), and expanded back; the expansion is a bijection between the open sets of the condensation and those of the input, so every reported field carries over exactly. A set is closed under following arcs if and only if it is closed under following directed paths, which is why propagating along single arcs gives the reachable sets.

Value

A list with the elements of `generate_topology()` and three more:

subbase, base The same list of integer vectors: the distinct sets of reachable vertices, which form the minimal base of the topology. They are not closed under intersection in general (see Methodological notes).

base_complete Always TRUE: the minimal base is computed exactly.

connected, components Connectedness and the connected components, as in `generate_topology()`.

topology, n_open_sets, topology_complete The enumerated open sets and their flags, as in `generate_topology()`.

input_index_ordered Logical: whether every arc went from a lower to a higher index.

input_acyclic Logical: whether the digraph had no directed cycle.

collapsed_classes List of integer vectors: the strongly connected components with more than one vertex, whose vertices the topology cannot tell apart; `list()` when there are none.

Methodological notes

The base is minimal, not intersection-closed. For the arcs $a \rightarrow c$, $a \rightarrow d$, $b \rightarrow c$, $b \rightarrow d$, the reachable sets are $\{a, c, d\}$, $\{b, c, d\}$, $\{c\}$ and $\{d\}$; the first two meet in $\{c, d\}$, which is open, as the union of $\{c\}$ and $\{d\}$, but is not one of them. The number of base sets is the number of strongly connected components.

Relation with the forward topology. For every digraph, the Alexandrov topology is contained in the forward topology generated by the closed forward neighbourhoods $N^+[v]$ (see `generate_bitopology()`): each reachable set is the union of the $N^+[w]$ of its points. It is not contained in the undirected topology in general: on the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, $\{4\}$ is open here and not in the undirected topology, whose smallest open set containing 4 is $\{3, 4\}$.

The connection between finite topologies and preorders goes back to Alexandroff (1937), after whom these spaces are named.

Dependencies

The compiled engine (C++ through 'Rcpp') for the propagation and the components; base R for the condensation.

References

- Alexandroff, P. (1937). Diskrete Räume. Recueil Mathématique (Matematicheskii Sbornik), Nouvelle Série, 2(44)(3), 501-519. <https://www.mathnet.ru/eng/sm5579>
- Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1(2), 146-160. <https://doi.org/10.1137/0201010>

See Also

[generate_topology\(\)](#) for the topologies of closed neighbourhoods, [generate_bitopology\(\)](#) for the time-series pipeline.

Examples

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

cross <- generate_alexandrov_topology(list(c(3L, 4L), c(3L, 4L)), integer(0), integer(0)), 4L)
cross$base

cyc <- generate_alexandrov_topology(list(2L, 3L, 1L), 3L)
cyc$collapsed_classes
```

generate_bitopology *Bitopological analysis of a time series*

Description

Builds the directed visibility graph of a series (horizontal or natural, at equally or unequally spaced instants) and from it four topologies on the observations: the undirected topology τ of the closed neighbourhoods, the forward topology τ^+ of the closed forward neighbourhoods $N^+[v] = \{v\} \cup \{w : v \rightarrow w\}$, the backward topology τ^- of the closed backward neighbourhoods $N^-[v] = \{v\} \cup \{w : w \rightarrow v\}$, and optionally the Alexandrov topology of reachability; then computes their invariants with [bitopology_invariants\(\)](#). The triple (V, τ^+, τ^-) is a bitopological space in the sense of Kelly (1963).

Usage

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

Arguments

series	Numeric vector with at least two observations and no NA, NaN or infinite values. No default.
graph_type	Character scalar: "hvg" (default) for the horizontal visibility graph or "nvg" for the natural one.
max_open_sets	Non-negative integer scalar (default 0, no enumeration), passed to generate_topology() and generate_alexandrov_topology() . The pairwise-connectedness check of bitopology_invariants() does not need the enumeration.
max_base_sets	Positive integer scalar (default 100,000), passed to generate_topology() .
alexandrov	Logical scalar (default TRUE): whether to compute the Alexandrov topology too.
times	NULL (default) for the instants $1, \dots, n$, or a numeric, Date or date-time vector of finite, strictly increasing instants of the same length as series. It matters only for the natural graph.

Details

Every arc of a directed visibility graph goes from the earlier to the later observation, so the out-neighbour lists give τ^+ and the in-neighbour lists τ^- ; the same engine computes the three topologies of closed neighbourhoods. Consecutive observations always see each other, so the digraph contains the path $1 \rightarrow 2 \rightarrow \dots \rightarrow n$, reachability is the total order, and the Alexandrov topology is the same chain of sets $\{i, \dots, n\}$ for every series; it describes the class of visibility graphs and not the series, and alexandrov = FALSE skips it. Because every arc goes forward in time, two more comparisons are constant on the class: $\{n\}$ is open in τ^+ (the last observation has no later neighbour) and not in τ^- (the only member of the backward subbase that contains n is its closed in-neighbourhood, which also contains $n - 1$, since consecutive observations see each other), so the two topologies differ for every series with $n \geq 2$, a reversible one included; and the bitopological space is pairwise disconnected for every such series (see [bitopology_invariants\(\)](#)). What T1 relates is τ^+ of a series to τ^- of its reversal, not τ^+ to τ^- of the same series.

Theorem T1 (time reversal). Let a series x be observed at strictly increasing instants t , write $\tilde{x}_i = x_{n+1-i}$, $\tilde{t}_i = -t_{n+1-i}$ (what [time_reverse\(\)](#) returns) and $r(i) = n + 1 - i$. Under either visibility criterion, $\{i, j\}$ is an edge for $(\tilde{x}, \tilde{t})$ if and only if $\{r(i), r(j)\}$ is an edge for (x, t) ; r carries the closed forward neighbourhood of i for $(\tilde{x}, \tilde{t})$ onto the closed backward neighbourhood of $r(i)$ for (x, t) , and the backward one onto the forward one; hence r is a homeomorphism from $\tau^+(\tilde{x}, \tilde{t})$ onto $\tau^-(x, t)$ and from $\tau^-(\tilde{x}, \tilde{t})$ onto $\tau^+(x, t)$. Consequently $C^+(\tilde{x}, \tilde{t}) = C^-(x, t)$ and $C^-(\tilde{x}, \tilde{t}) = C^+(x, t)$, the two base sizes are exchanged, asymmetry_direction, $D = C^- - C^+$, changes sign, and irreversibility_components, $I_C = |C^+ - C^-| / \max(C^+, C^-)$, is unchanged.

Proof. The horizontal criterion for $i < j$ involves only the values strictly between them, which r carries onto those strictly between $r(j)$ and $r(i)$, with the same values. For the natural criterion, the quantity $D(a, b, k) = x_a(t_b - t_k) + x_b(t_k - t_a) - x_k(t_b - t_a)$ of [natural_visibility_graph\(\)](#) computed for $(\tilde{x}, \tilde{t})$ at $(r(b), r(a), r(k))$ equals $D(a, b, k)$ for (x, t) , term by term, so each condition transfers to the reflected pair. Arcs run from the lower index to the higher one and r inverts that order, so forward closed neighbourhoods become backward ones; a bijection between the subbases of two topologies that commutes with the point map is a homeomorphism of the topologies they generate, and it carries bases and connected components onto each other. $\square$

Value

A list with graph (the directed visibility graph, as returned by `horizontal_visibility_graph()` or `natural_visibility_graph()` with `directed = TRUE`), undirected, forward and backward (results of `generate_topology()`), alexandrov (the result of `generate_alexandrov_topology()`, or NULL) and invariants (the result of `bitopology_invariants()`).

Methodological notes

What T1 says and what it does not. T1 fixes the meaning of the sign of `asymmetry_direction`: reading the series backwards, with its instants reflected, changes it. When the series and the gaps between its instants both read the same in the two directions (a palindrome on a palindromic grid; with `times = NULL`, a palindrome), the reversed series is the same series up to a translation of time and the asymmetry is exactly zero. T1 has no converse: a non-zero asymmetry does not show that the process that generated the series is irreversible, and a zero asymmetry does not show that it is reversible. The probabilistic statement it supports is about laws: if the joint law of the observations together with their instants is invariant under the joint reversal (values read backwards, instants reflected), the law of `asymmetry_direction` is symmetric about zero, the mass at zero included; with `times = NULL`, or with fixed instants whose gaps read the same in both directions, the condition is that the law of the values equals the law of their reversal, and so it is for the horizontal criterion with any fixed instants, which it does not use. For the natural criterion with fixed instants whose gaps are not palindromic, reversing the values alone is another operation, to which this statement does not apply. None of this says anything about stationarity or about the generating process. For a statistical treatment of time irreversibility with visibility graphs, see Lacasa et al. (2012), who estimate it by the Kullback-Leibler divergence between the in- and out-degree distributions of the directed horizontal visibility graph.

Instants. Reversing the values over the same instants is a different operation as soon as the gaps between consecutive instants are not palindromic. The horizontal criterion does not use the instants, so for it the two operations give the same graphs and T1 holds for both; for the natural criterion T1 can fail for the reversal of the values alone: with the values 0, 1, 2 at the instants 0, 1, 3, it creates the edge {1, 3}, which neither the original series nor its reversal with reflected instants has; see `time_reverse()`. With the natural criterion the theorem holds exactly in the computation as well, under the hypotheses of `natural_visibility_exactness`, because the criterion is then decided exactly and negating the instants is exact.

Dependencies

`horizontal_visibility_graph()` or `natural_visibility_graph()`, `generate_topology()`, `generate_alexandrov_topology()` and `bitopology_invariants()`; the compiled engine through 'Rcpp'.

References

- Kelly, J. C. (1963). Bitopological spaces. *Proceedings of the London Mathematical Society*, s3-13(1), 71-89. <https://doi.org/10.1112/plms/s3-13.1.71>
- Lacasa, L., Nuñez, A., Roldán, É., Parrondo, J. M. R., & Luque, B. (2012). Time series irreversibility: A visibility graph approach. *The European Physical Journal B*, 85(6), Article 217. <https://doi.org/10.1140/epjb/e2012-20809-8>

See Also

[bitopology_invariants\(\)](#), [time_reverse\(\)](#), [generate_topology\(\)](#), [generate_alexandrov_topology\(\)](#).

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$invariants$asymmetry_direction

rv <- time_reverse(series)
generate_bitopology(rv$series, graph_type = "hvg")$invariants$asymmetry_direction

t <- c(0, 1, 3, 4, 7, 8, 9, 12)
bn <- generate_bitopology(series, graph_type = "nvg", times = t)
rn <- time_reverse(series, t)
bn$invariants$asymmetry_direction
generate_bitopology(rn$series, graph_type = "nvg", times = rn$times)$invariants$asymmetry_direction
```

generate_topology	<i>Generate the topology induced by the closed neighbourhoods of a graph</i>
-------------------	--

Description

Builds the topology on the vertices $\{1, \dots, n\}$ of a graph that is generated by the closed neighbourhoods $N[v] = \{v\} \cup N(v)$, computes its connected components exactly, and optionally enumerates its open sets. Given undirected adjacency lists it is the undirected construction; given the out-neighbour or in-neighbour lists of a digraph it gives the forward or the backward topology. Use it directly for any graph, or through [generate_bitopology\(\)](#) for time series.

Usage

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

Arguments

adjacency	List of length <code>n_elements</code> ; element <code>i</code> is an integer vector with the 1-based indices of the neighbours (for a directed construction, the out-neighbours or the in-neighbours) of vertex <code>i</code> . Repeated indices, <code>i</code> itself and indices outside $1, \dots, n$ are ignored. No default.
-----------	--

<code>n_elements</code>	Integer scalar, at least 1: the number of vertices. Must equal <code>length(adjacency)</code> . No default.
<code>max_open_sets</code>	Non-negative integer scalar (default 1,000,000). The enumeration of the open sets stops as soon as their number exceeds it; 0 skips the enumeration.
<code>max_base_sets</code>	Positive integer scalar (default 100,000). The closure of the subbase under intersections stops at the first new intersection that makes the base exceed this number of sets. It is not a strict cap on the size of the returned base: the whole subbase is always kept, and so is the intersection that crosses the limit.
<code>check_connected</code>	Logical scalar (default TRUE): whether to compute the connected components.
<code>verify_axioms</code>	Logical scalar (default FALSE): whether to check, independently of the construction, that the enumerated family contains the empty set and the whole set and is closed under the union and the intersection of any two members, which on a finite set are the axioms of a topology. The check runs only when <code>topology_complete</code> is TRUE; it takes $O(\tau ^2 \lceil n/64 \rceil)$ word operations and $O(\tau ^2)$ hash lookups.

Details

The base is computed by wavefront propagation: each new intersection is intersected, in the next round, with every set already in the base, until a round produces nothing new. Every new set is a proper subset of a set produced in the previous round, so the closure ends after at most n rounds; the base itself can have exponentially many sets in n , which is what `max_base_sets` guards against. The components are computed from the signatures $S(x) = \{B \in \mathcal{B} : x \in B\}$: x and y are comparable when $S(x) \subseteq S(y)$ or $S(y) \subseteq S(x)$, and the components are those of the comparability graph, found by breadth-first search in $O(n^2 \lceil |\mathcal{B}|/64 \rceil)$ word operations. The open sets are enumerated by the same wavefront scheme with unions, which can produce up to 2^n sets; when the space is already known to be disconnected, the engine warns that the enumeration is not needed for connectivity.

Value

A list with:

subbase List of integer vectors: the distinct closed neighbourhoods $N[v]$.

base List of integer vectors: the distinct non-empty finite intersections of members of the subbase. When `base_complete` is FALSE it is a subfamily that contains the whole subbase.

base_complete Logical: whether the intersection closure finished within `max_base_sets`.

connected Logical: whether the space is connected; NA if `check_connected` = FALSE.

components List of integer vectors: the connected components, exact whether or not the base is complete (see Methodological notes); an empty list if `check_connected` = FALSE and $n \geq 2$.

iterations_base Integer: rounds of the intersection closure.

topology List of integer vectors with the enumerated open sets, or NULL if `max_open_sets` = 0.

n_open_sets Integer: the number of enumerated open sets, or NA if the enumeration was not requested.

topology_complete Logical: TRUE if the enumerated family is the whole topology; FALSE if the enumeration was requested but cannot be certified complete, because `max_open_sets` cut it or because the base was incomplete; NA if the enumeration was not requested.

iterations_topo Integer: rounds of the union closure.

axioms_ok Logical, present only when `verify_axioms` = TRUE and `topology_complete` is TRUE.

axiom_failure Character, present only if that check fails.

Methodological notes

Closed neighbourhoods. Nada, El Atik and Atef (2018, Definition 3.1) take as subbase the post classes of the adjacency relation, which for a loopless graph are the open neighbourhoods $N(v)$. This function uses the closed neighbourhoods $N[v]$, so that every vertex lies in its own subbase element: the subbase covers the vertex set and the smallest open set containing v is contained in $N[v]$. The two constructions give different topologies; on the complete graph, for instance, the closed variant gives the indiscrete topology and the open one the discrete topology.

Why the components are exact, even from a truncated base. In a finite space every point x has a smallest open set U_x , the intersection of the members of the subbase that contain x . For any family F with subbase $\subseteq F \subseteq \tau$, every member of F that contains x contains y if and only if $y \in U_x$: if $y \in U_x$, every open set containing x contains U_x ; conversely, the members of the subbase that contain x are among those of F , and their intersection is U_x . The relation $x \leq y \iff y \in U_x$ is the specialization preorder. Let $\sim$ be the equivalence generated by comparability. Each class C is open, because it contains U_x for each of its points (every $y \in U_x$ satisfies $x \leq y$), and closed, because its complement is a union of other classes; so every connected component lies inside one class. Each class is connected: if a class split into two non-empty parts A and B , both relatively open, some $x \in A$ and $y \in B$ would be comparable, say $x \leq y$; the open set that cuts out A contains U_x , hence y , a contradiction (the case $y \leq x$ is symmetric). The components are therefore the classes, and since the engine places the whole subbase in the base before the closure can be cut, the signatures it compares always give the same classes.

What the base size measures. The number of sets in the base is a property of the construction, not of the topology: two subbases that generate the same topology can have intersection closures of different sizes.

Dependencies

The compiled engine (C++ through 'Rcpp').

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. <https://doi.org/10.1002/mma.4726>

See Also

`generate_bitopology()` for the time-series pipeline, `generate_alexandrov_topology()` for the Alexandrov topology of reachability, `is_topology_connected_exact()` for an enumerated topology, and the user manual in `system.file("manuals", package = "topologyR")`.

Examples

```

g <- horizontal_visibility_graph(c(3, 1, 4, 1, 5))
topo <- generate_topology(g$adjacency, g$n, max_open_sets = 0L)
topo$connected
topo$components
topo$topology_complete

full <- generate_topology(g$adjacency, g$n, verify_axioms = TRUE)
full$n_open_sets
full$topology_complete
full$axioms_ok

```

horizontal_visibility_graph

Horizontal visibility graph of a time series

Description

Builds the horizontal visibility graph of a series (Luque et al., 2009): two observations are adjacent when every observation strictly between them is strictly lower than both. With `directed = TRUE` it also returns the arcs from the earlier to the later observation of each edge, which [generate_bitopology\(\)](#) turns into the forward and backward topologies. The instants of observation enter only through their order, so `times` is validated and recorded but does not change the graph.

Usage

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

Arguments

<code>series</code>	Numeric vector without NA, NaN or infinite values. A series of length 0 or 1 gives a graph without edges. No default.
<code>directed</code>	Logical scalar (default FALSE): whether to return the directed adjacency lists as well.
<code>times</code>	NULL (default), meaning the instants $1, \dots, n$, or a numeric, Date or date-time vector of finite, strictly increasing instants of the same length as <code>series</code> ; dates count in days and date-times in seconds.

Details

The series is scanned from left to right with a stack whose values decrease strictly from bottom to top, in $O(n)$ time. When observation i arrives, every stacked index j with $y_j \leq y_i$ is popped and joined to i : nothing between j and i reaches y_j , or it would have popped j earlier. The index left on top, if any, is higher than y_i and is joined to i unless a popped value equals y_i , which blocks at height $\min = y_i$. Every other earlier index is blocked: one deeper in the stack by that top, which is higher than y_i ; one popped before by the index that popped it, which lies in between and is at least as high. Each edge is thus reported exactly once, always with `from < to`.

Value

A list with:

edges data.frame with integer columns from and to, from < to, one row per edge.

n Integer: the number of observations.

n_edges Integer: the number of edges.

adjacency List of integer vectors: element *i* holds the neighbours of observation *i*, its open neighbourhood $N(i)$; [generate_topology\(\)](#) adds *i* itself.

times Numeric vector: the instants used.

out_adjacency Only with directed = TRUE: list whose element *i* holds the later neighbours of *i*.

in_adjacency Only with directed = TRUE: list whose element *i* holds the earlier neighbours of *i*.

Methodological notes

The criterion uses comparisons of values and nothing else, so the graph is computed exactly on the numbers received, is invariant under any strictly increasing transformation of the values, and does not depend on the spacing of the instants. Decimal data are converted to binary doubles before they reach the function, and the graph is that of the doubles. A conversion that rounds each decimal to the nearest double keeps the order of the decimals, and keeps distinct any two with at most fifteen significant digits in the normal range (magnitudes from about 2.2×10^{-308} to 1.8×10^{308}), because their relative gap, at least 10^{-15} , exceeds the relative spacing of doubles, at most 2^{-52} ; outside that range distinct decimals can collapse: 2×10^{-324} , 10^{-324} and 2×10^{-324} all become zero, and the edge between the first and the third disappears.

The function stops with an error when the floating-point environment does not provide IEEE 754 double arithmetic (see [natural_visibility_exactness](#) for the test). Under the denormals-are-zero mode, which some libraries switch on, the comparisons would treat every subnormal value as zero; the flush-to-zero mode, which changes results of arithmetic and not these comparisons, is refused as well, because the same test guards the natural criterion, which depends on it.

Dependencies

The compiled engine (C++ through 'Rcpp').

References

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

See Also

[natural_visibility_graph\(\)](#), [generate_bitopology\(\)](#), [time_reverse\(\)](#).

Examples

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

```
head(g$edges)

gd <- horizontal_visibility_graph(series, directed = TRUE)
gd$out_adjacency[[1]]
gd$in_adjacency[[5]]
```

is_topology_connected *Connectivity of the co-occurrence graph of a family of sets*

Description

Builds the co-occurrence graph of a family of sets, in which two elements are adjacent when some member of the family contains both, and reports whether that graph is connected. It is retained from version 0.1.0 as a quick necessary-condition screen; the exact test of topological connectedness is [generate_topology\(\)](#) for a topology given by a subbase and [is_topology_connected_exact\(\)](#) for a fully enumerated topology.

Usage

```
is_topology_connected(topology)
```

Arguments

topology	A list of integer vectors (positive labels), read as a family of subsets of the labels that appear in it. A full topology, a base or a subbase are all admissible. No default.
----------	--

Details

The ground set is taken to be the set of labels that appear in some member, and the graph is traversed from the first of them. Memory grows with the square of the largest label, because the graph is stored as a dense adjacency matrix.

Value

A logical scalar: TRUE if the co-occurrence graph on the labels that appear in the family is connected, FALSE if it is not or if the family is empty.

Methodological notes

For any family of open sets that covers the ground set, a topologically connected space has a connected co-occurrence graph: if the graph split into two non-empty parts, every member would lie inside one part, each part would be a union of members and hence open, and the two parts would disconnect the space. A disconnected graph therefore proves that the space is disconnected. The converse fails: the discrete topology on $\{1, 2\}$, given by its open sets $\{\}, \{1\}, \{2\}$ and $\{1, 2\}$, is disconnected while its co-occurrence graph is connected, and any list that contains the whole ground set makes the graph complete. A TRUE answer is thus inconclusive.

Dependencies

Base R only.

See Also

`is_topology_connected_exact()` for the exact test on an enumerated topology, `generate_topology()` for exact components from a subbase, `is_topology_connected2()` and `is_topology_connected_manual()` for the other legacy diagnostics.

Examples

```
is_topology_connected(list(c(1, 2, 3), c(3, 4, 5)))
is_topology_connected(list(c(1, 2), c(3, 4)))
is_topology_connected(list(integer(0), 1L, 2L, c(1L, 2L)))
```

is_topology_connected_exact

Connectedness of a fully enumerated topology

Description

Decides whether a finite topological space, given by the complete list of its open sets, is connected: it is connected if and only if no proper non-empty open set has an open complement. Use it on the topology element of a result whose `topology_complete` is TRUE; for a topology given by a subbase, `generate_topology()` returns the components without enumerating anything.

Usage

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

Arguments

<code>topology</code>	List of integer vectors: the open sets of a topology on $\{1, \dots, n\}$, in any order, possibly repeated. Indices outside that range are ignored. No default.
<code>n_elements</code>	Integer scalar, at least 1: the number of points. No default.

Details

Each set is encoded as a bitset and stored in a hash set; the function scans the open sets once and looks up each complement, in $O(|\tau|\lceil n/64 \rceil)$ word operations when the hash spreads the sets evenly over the table (the usual uniform-hashing assumption; a fixed hash function gives no worst-case guarantee).

Value

A list with `connected` (logical). When it is FALSE, also `component1`, a proper non-empty open set whose complement is open, and `component2`, that complement (integer vectors). Each of the two is a union of connected components, not necessarily a single component; the field names are kept from earlier versions.

Methodological notes

The function does not check that the list is a topology, that is, that it contains the empty set and the whole set and is closed under unions and intersections. On an arbitrary family it reports whether some proper non-empty member has its complement in the family, which is connectedness only when the family is the complete topology.

Dependencies

The compiled engine (C++ through 'Rcpp').

See Also

[generate_topology\(\)](#) for exact components from a subbase.

Examples

```
chain <- list(integer(0), 1L, 1:2, 1:3)
is_topology_connected_exact(chain, 3L)

split <- list(integer(0), 1:2, 3:4, 1:4)
is_topology_connected_exact(split, 4L)
```

is_topology_connected_manual

Check whether a family of sets covers the labels 1 to its maximum

Description

Checks whether every integer from 1 to the largest label of a family of sets appears in at least one member, that is, whether the family covers the ground set $\{1, \dots, \max\}$. Despite its name, retained from version 0.1.0, it does not test connectedness.

Usage

```
is_topology_connected_manual(topology)
```

Arguments

topology A list of integer vectors (positive labels). No default.

Value

A logical scalar: TRUE if every integer from 1 to the largest label appears in some member, FALSE otherwise or if the family is empty.

Methodological notes

Coverage is necessary for a family to be a base or a subbase of a topology on $\{1, \dots, \max\}$, but it says nothing about connectedness: the discrete topology covers its ground set and is maximally disconnected.

Dependencies

Base R only.

See Also

[is_topology_connected_exact\(\)](#) and [generate_topology\(\)](#) for connectedness.

Examples

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

is_topology_connected2

Label-order reachability within a family of sets

Description

Builds a directed graph with an arc between consecutive labels, in increasing order, inside each member of a family of sets, and reports whether every label that appears in the family is reachable from the smallest one. It is retained from version 0.1.0 for backward compatibility and is not a test of topological connectedness; use [generate_topology\(\)](#) or [is_topology_connected_exact\(\)](#) for that.

Usage

```
is_topology_connected2(topology)
```

Arguments

topology A list of integer vectors (positive labels), read as a family of subsets. No default.

Details

Memory grows with the square of the largest label, because the arcs are stored as a dense matrix.

Value

A logical scalar: TRUE if every label that appears in the family is reachable from the smallest label along the arcs described above, FALSE otherwise or if the family is empty.

Methodological notes

The answer depends on how the elements are labelled, so it is not a topological invariant, and it is neither a necessary nor a sufficient condition for connectedness. The base $\{1, 3\}, \{2, 3\}$ generates the connected topology whose open sets are $\{\}, \{3\}, \{1, 3\}, \{2, 3\}$ and $\{1, 2, 3\}$ (no proper non-empty open set has an open complement), yet label 2 is not reachable from label 1 and the function returns FALSE. The discrete topology on $\{1, 2\}$, listed as $\{\}, \{1\}, \{2\}, \{1, 2\}$, is disconnected, yet the function returns TRUE.

Dependencies

Base R only.

See Also

[generate_topology\(\)](#) and [is_topology_connected_exact\(\)](#) for exact connectedness, [is_topology_connected\(\)](#) for a necessary condition.

Examples

```
is_topology_connected2(list(c(1, 2, 3), c(3, 4, 5)))
is_topology_connected2(list(c(1L, 3L), c(2L, 3L)))
```

natural_visibility_exactness

Exactness of the natural visibility criterion

Description

[natural_visibility_graph\(\)](#) decides every pair by the sign of

$$D(a, b, k) = x_a(t_b - t_k) + x_b(t_k - t_a) - x_k(t_b - t_a)$$

for an observation k between a and b : k lies strictly below the chord joining a to b if and only if $D > 0$. That sign is computed exactly for every finite input, in a floating-point environment that provides IEEE 754 double arithmetic, under the hypotheses stated below; the graph is therefore the natural visibility graph of the doubles received, whatever the order in which floating-point operations round. This page proves it. The computation has two stages, the adaptive scheme of Shewchuk (1997): a floating-point evaluation whose sign is accepted only when the result exceeds a threshold above which that sign is proved correct, and an exact evaluation in integer arithmetic otherwise. The code is in the file `src/exact_predicate.h` of the package sources.

The floating-point stage

D equals $-O$, where $O = L - R$ is the orientation determinant of the points $(t_a, x_a), (t_b, x_b), (t_k, x_k)$, with $L = (t_a - t_k)(x_b - x_k)$ and $R = (x_a - x_k)(t_b - t_k)$. When the six inputs have magnitude at most 2^{510} , the code computes the four differences, the products $l = \text{fl}(t_a - t_k) \text{fl}(x_b - x_k)$ and r likewise, $S = \text{fl}(|l| + |r|)$ and $\text{det} = \text{fl}(l - r)$, an evaluation of O , where $\text{fl}(\cdot)$ denotes the double stored after one operation: its exact result rounded to binary64, either directly or after a first

rounding to the evaluation format (the proof bounds both roundings together). When $S > 2^{-930}$ (about 1.1×10^{-280} ; the code writes it as the hexadecimal literal `0x1p-930`, which binary64 represents exactly, and so does every wider format in which the comparison may be evaluated, so its value does not depend on the compiler) and $|\det| > 8uS$, with $u = 2^{-53}$, it accepts the sign of $\det$ as the sign of O and returns its opposite as the sign of D ; otherwise it goes to the exact stage. Each of the ten values above (four differences, two products, two absolute values, S and $\det$) is stored in a volatile variable, one operation per statement, so it is the result of a single operation rounded to double: the compiler can neither fuse a product into the subtraction nor reorder the operations, and a value computed in a wider format is rounded to double when it is stored.

Proof that the accepted sign is the sign of O . Assume IEEE 754 binary64 doubles in any of the four rounding modes, each operation rounded to binary64 directly, or first to a format with at least 64 significand bits (such as the x87 extended format) or to the x87 extended format with its precision control at 53 bits, and then to binary64 on the store. Under any other evaluation the threshold is not established; the next section says which part of these hypotheses is checked. (o) No operation overflows: the differences have magnitude at most 2^{511} , the products at most 2^{1022} and S at most 2^{1023} , and rounding is monotone, so every rounded value obeys the same bounds. (i) Hence every operation whose exact result has magnitude at least 2^{-1022} returns it with relative error at most $e = 2u(1 + 2^{-10})$, which bounds one directed rounding to double ($2u$) after one rounding to a 64-bit significand (2^{-63}); a 53-bit significand gives an error of at most $2u$ that the store, exact in that range, does not increase; an operation whose exact result is smaller is exact if it is a subtraction of two doubles (the difference is a multiple of 2^{-1074} below 2^{-1022} , hence a double), and otherwise has absolute error below 2^{-1073} , the sum of at most 2^{-1074} from the store and less than 2^{-1074} from a previous rounding in a wider format. (a) Each difference is exact or within relative error e . (b) Each product is $l = L(1 + \theta) + a_l$ with $|\theta| \leq g = (1 + e)^3 - 1 < 6.01u$ and $|a_l| < 2^{-1073}$, the absolute term covering an underflow; likewise r . Hence $|(l - r) - O| \leq g(|L| + |R|) + 2^{-1072}$, and since $|L| \leq (|l| + 2^{-1073})/(1 - g)$ and $g/(1 - g) < 6.02u$, $|(l - r) - O| < 6.02u(|l| + |r|) + 2^{-1071}$. (c) $\det = \text{fl}(l - r)$ has the sign of $l - r$: a non-zero $l - r$ is at least 2^{-1074} in magnitude, a double that every rounding mode keeps on the same side of zero, and $\det = 0$ only if $l = r$; and $|l - r| \geq |\det|/(1 + e)$ (the subtraction is exact when its result is below 2^{-1022}). (d) $S > 2^{-930}$ implies that the exact sum $|l| + |r|$ is at least 2^{-1022} (a smaller sum of two doubles would be exact, and S would equal it), hence at least $S/(1 + e)$, and $S \geq (|l| + |r|)(1 - e)$. If $|\det| > 8uS$, then $|l - r| > 8u(1 - e)(|l| + |r|)/(1 + e) > 7.99u(|l| + |r|)$, and with $|l| + |r| > 2^{-930}/(1 + e) > 1.1 \times 10^{-280}$ the term 2^{-1071} is below $1.97u(|l| + |r|)$; so $|l - r| > |(l - r) - O|$, $l - r$ and O have the same sign, and so does $\det$. $\square$

Inputs above 2^{510} in magnitude go straight to the exact stage. Without that bound, a directed rounding mode can turn an overflow into the largest finite double instead of an infinity (under rounding toward zero, every overflow; under rounding downward or upward, the overflows of one sign), and the relative error, which approaches 1 as the exact result grows, is no longer bounded by $2u$. With M the largest double, $2^{1024} - 2^{971}$, the instants $t_a = -M < t_k = 2^{1023} < t_b = M$ and the values $x_a = 5/4$, $x_k = 0$, $x_b = -1/2$ give $D = -2^{1021} - 3 \cdot 2^{969} < 0$, but under rounding toward zero a filter without the bound computes the difference $t_a - t_k$, of magnitude $M + 2^{1023}$, as $-M$, finds $\det < 0$ far above the threshold, and returns $+1$.

The exact stage

Proof that it returns the sign of D . (1) Decomposition. For a finite non-zero double v , the C function `frexp()` returns, exactly, $f \in [1/2, 1)$ and E with $|v| = f 2^E$, E from -1073 to 1024 . f has at most 53 significant bits (a subnormal v has fewer, and `frexp()` normalizes it), so $m = f 2^{53}$,

computed exactly by `ldexp()`, is an integer with $2^{52} \leq m < 2^{53}$, and $|v| = m 2^e$ with $e = E - 53$, from -1126 to 971 . Zero gives $m = 0$. (2) Products. D is the sum of the six terms $x_a t_b$, $-x_a t_k$, $x_b t_k$, $-x_b t_a$, $-x_k t_b$, $x_k t_a$, each a product of two inputs with the coefficient $+1$ or -1 ; the function that adds them receives, for each term, only whether its coefficient is -1 , so no other coefficient can be passed. Each non-zero term is $\pm(m_x m_y) 2^{e_x + e_y}$ with $m_x m_y < 2^{106}$, and the product $m_x m_y$ is computed exactly in four words of 32 bits: writing $a = a_1 2^{32} + a_0$ and $b = b_1 2^{32} + b_0$ with $a_1, b_1 < 2^{21}$, the four partial products are below 2^{64} , 2^{53} , 2^{53} and 2^{42} , the middle sum is below $3 \cdot 2^{32}$ and the high sum below 2^{33} , so no operation on 64-bit unsigned integers overflows. (3) Alignment. Let $e_{\min}$ and $e_{\max}$ be the smallest and the largest exponent of the non-zero terms and $\Delta = e_{\max} - e_{\min}$. Then $2^{-e_{\min}} D = P - N$, where P and N are the sums of the terms of each sign, each term $\pm(m_x m_y) 2^{e_k}$, with $e_k = e_x + e_y$, contributing the integer $m_x m_y 2^{e_k - e_{\min}}$; $2^{-e_{\min}} D$ has the sign of D . Each shifted term is below $2^{106 + \Delta}$, so P and N are below $6 \cdot 2^{106 + \Delta} < 2^{109 + \Delta}$. (4) Capacity. Each accumulator has $L = \lfloor (\Delta + 109) / 32 \rfloor + 3$ words of 32 bits, at least $\Delta + 174$ bits, so no carry leaves it; a term shifted by $s \leq \Delta$ bits is written into the words $\lfloor s/32 \rfloor$ to $\lfloor s/32 \rfloor + 4$, and $\lfloor \Delta/32 \rfloor + 4 < L$. Since Δ is at most $1942 + 2252 = 4194$, L is at most 137. (5) Comparison. P and N are compared word by word from the most significant one, which orders non-negative integers written in base 2^{32} , and $P = N$ gives $D = 0$. After `frexp()` and `ldexp()`, which are exact, the stage uses integer arithmetic only, so no rounding, overflow or underflow can occur, and neither the rounding mode nor the compiler options matter. $\square$

Compiler options and run-time environment

At compile time the build stops with an error unless doubles have the binary64 format (radix 2, 53 significant digits, its exponent range) and operations on doubles are evaluated in binary64 (`FLT_EVAL_METHOD` 0 or 1) or in a format with at least 64 significand bits (`FLT_EVAL_METHOD` 2 with `LDBL_MANT_DIG` at least 64). It also stops with `-ffast-math` or `-ffinite-math-only`, which let the compiler assume that no infinity or NaN occurs, and, under GCC, with every option that makes the compiler set `__GCC_IEC_559` to 0 (among them `-funsafe-math-optimizations`, `-fassociative-math`, `-freciprocal-math` and `-fno-signed-zeros`). Among the options that the compilers accept, contraction (with any compiler) and the reassociation and reciprocal options of Clang, which does not reveal them through its predefined macros, cannot change the floating-point stage. Each of the ten intermediate values of that stage is a volatile object, and the C++ standard requires every access to a volatile object to be performed as the program writes it: each result is stored into its object, and every operation that uses it reads it from that object, so the operand is the double stored there and not a fused or wider intermediate; each statement that computes one of them performs a single operation; the final test reads S and `det` once and compares $|\text{det}|$ with $2^{-50} S$ through an absolute value, a scaling by a power of two (exact, since $S > 2^{-930}$) and comparisons, which are all exact; and the stage contains no division (its operations are four subtractions, two multiplications, one addition, absolute values, comparisons and a scaling by 2^{-50}), so a reciprocal option has nothing to transform. An accepted option can still act through the floating-point environment it sets at start-up, which the run-time tests below cover. The proof assumes, beyond these checks, a compilation that performs each operation as written, as the C++ standard prescribes for operations on volatile objects. What remains is the floating-point environment at run time, which some libraries change: the flush-to-zero and denormals-are-zero modes (Clang links such a start-up routine into executables built with `-funsafe-math-optimizations`) make subnormal arithmetic inexact and would also change the comparisons of both visibility criteria, and the precision control of the x87 unit can reduce the significand to 24 bits. Before building a graph, the constructors test, with results that are exact in binary64 and therefore independent of the rounding mode, that $(1 + 2^{-52}) - 1 = 2^{-52}$ and $(1 + 2^{-26})^2 = 1 + 2^{-25} + 2^{-52}$ (53 significant bits), that

half of the smallest normal double is still positive and that the smallest subnormal double compares greater than zero; they stop with an error when any test fails. They also compute in the non-stop mode of IEEE 754: they save the floating-point environment, disable every trap (an enabled trap on inexact results or on underflow would otherwise interrupt the filter, whose products can be inexact or underflow) and restore the environment when they return. With the compile-time checks, these tests cover the hypotheses of the proof except one case: a run-time precision strictly between 53 and 64 bits would be neither covered by the proof nor detected, and under it the exactness is not established.

Reproducible checks

In the source repository, `dev/check_filter_constants.py` checks the numerical facts used in the floating-point stage (the constants 6.01, 6.02, 7.99 and 1.97, the bound e , the bound on $|l| + |r|$ from 2^{-930} , the absolute terms, the overflow bounds and $2^{-50} = 8u$) in exact rational arithmetic, and `dev/make_exact_predicate_fixture.py` regenerates, deterministically, the 1,800 sextuples of the test suite in nine regimes (among them extreme exponents, subnormal numbers, points within two units of rounding of a line, and inputs beyond the bound of the floating-point stage), with signs computed in exact rational arithmetic outside the package. The test suite compares the computed sign with each of them.

References

Shewchuk, J. R. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3), 305-363. <https://doi.org/10.1007/PL00009321>

See Also

[natural_visibility_graph\(\)](#), [horizontal_visibility_graph\(\)](#).

natural_visibility_graph

Natural visibility graph of a time series

Description

Builds the natural visibility graph of a series (Lacasa et al., 2008): two observations are adjacent when every observation strictly between them lies strictly below the straight segment that joins them, drawn at the instants of observation, equally spaced unless times says otherwise. The criterion is decided exactly on the numbers received, under the hypotheses on the floating-point arithmetic stated in [natural_visibility_exactness](#). With `directed = TRUE` it also returns the arcs from the earlier to the later observation of each edge, which [generate_bitopology\(\)](#) turns into the forward and backward topologies.

Usage

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

Arguments

series	Numeric vector without NA, NaN or infinite values. A series of length 0 or 1 gives a graph without edges. No default.
directed	Logical scalar (default FALSE): whether to return the directed adjacency lists as well.
times	NULL (default), meaning the instants $1, \dots, n$, or a numeric, Date or date-time vector of finite, strictly increasing instants of the same length as series; dates count in days and date-times in seconds.

Details

Observations $a < b$ are adjacent when, for every k between them,

$$D(a, b, k) = y_a(t_b - t_k) + y_b(t_k - t_a) - y_k(t_b - t_a) > 0.$$

The graph is built by divide and conquer on the maximum. Each level of the recursion makes at most one decision per point of its intervals, so the number of decisions is $O(n^2)$ in the worst case (monotone data, whose maximum is always at an end), $O(n \log n)$ when the maximum of every interval splits it in parts of bounded proportion, and $O(n \log n)$ in expectation when the values are exchangeable and without ties, because the position of each maximum is then uniform, as the pivot of randomized quicksort. Three facts make the construction correct for any strictly increasing instants. (i) If p maximizes y on an interval and $a < p < b$, the chord from a to b takes at t_p a convex combination of y_a and y_b , at most y_p , so p blocks every pair across it. (ii) For $j < p$, every k between them is below the chord from j to p if and only if the slope from j to p is smaller than the slope from each such k to p ; scanning leftwards from p , the smallest of those slopes belongs to the last index found visible, m , so j is visible if and only if $D(j, p, m) > 0$, and symmetrically on the right. (iii) The edges inside each half depend only on the points of that half.

Each decision is therefore the sign of one quantity D , and it is computed exactly under those hypotheses. When the six inputs of a decision are at most 2^{510} in magnitude, so that no intermediate value can overflow, the engine evaluates in floating point the orientation determinant $O = -D$, as the difference of two rounded products l and r , and accepts the sign of the result when its magnitude exceeds $8uS$, where $u = 2^{-53}$ and S is the computed value of $|l| + |r|$: above that threshold the sign is proved correct. Otherwise, near a tie, or when S does not exceed the 2^{-930} , about 1.1×10^{-280} , D is evaluated exactly in integer arithmetic on the binary mantissas and exponents of the six products it contains. This is the adaptive scheme of Shewchuk (1997). The proofs of both stages, and the hypotheses on the arithmetic under which they hold (every IEEE 754 rounding mode; evaluation in binary64, in a format with at least 64 significand bits, or in the x87 format at 53 bits), with what is checked of them, are written in [natural_visibility_exactness](#); every intermediate value of the floating-point evaluation is stored, so that the compiler cannot fuse a product into the subtraction or reorder the operations. A floating-point decision takes a fixed number of operations; an exact one allocates and compares two accumulators of L words of 32 bits, where L grows with the spread of the binary exponents of the six products and is at most 137, so its number of word operations is proportional to L . The function stops with an error when the floating-point environment does not provide IEEE 754 double arithmetic (subnormal numbers flushed to zero, or operations with fewer than 53 significant bits), a condition that some libraries create and under which the proofs do not apply.

Value

A list with the same elements as `horizontal_visibility_graph()`.

Methodological notes

What exact means here. Under the hypotheses of `natural_visibility_exactness`, the graph is the natural visibility graph of the doubles received, independent of the order of floating-point operations. Consequences, under the same hypotheses: reversing the values and reflecting the instants (see `time_reverse()`) exchanges the forward and backward topologies exactly, for every input; and the graph does not change when the values are replaced by $\alpha y + \beta$ and the instants by $\gamma t + \delta$, with $\alpha, \gamma > 0$, provided that the new values and instants are finite doubles equal to these exact images (no rounding), so that the instants stay strictly increasing. Proof: each D computed for the images equals $\alpha\gamma D$, because the coefficients of y_a , y_b and y_k in D add up to zero, so β cancels, and every difference of instants is multiplied by γ , so δ cancels. Versions up to 0.3.0 accepted only equally spaced instants and compared the rounded slopes $\text{fl}(\text{fl}(y_p - y_j)/(p - j))$. When no difference of values overflows and no slope is non-zero and below 2^{-1022} in magnitude, each rounded slope is within relative error $2u + u^2$ of the exact one ($u = 2^{-53}$, rounding to nearest), so the two engines can disagree only on two slopes whose difference is at most about $2u$ times the sum of their magnitudes, that is, when a point lies on a chord or very near it. Outside that range of magnitudes, with values near the largest doubles or slopes in the subnormal range, the rounded slopes of 0.3.0 could also err far from any tie.

Decimal data. A decimal such as 0.1 has no exact binary representation, so points that are collinear in decimal can be slightly below or above the line in binary, and the exact criterion then follows the binary numbers. Suppose the values were recorded with d decimal places, $q_i = m_i/10^d$ with a non-negative integer $d \leq 22$ and integers $|m_i| \leq 2^{50}$, and that each was stored as the double y_i nearest to it. Then `m <- round(10^d * y)` returns the integers m_i , and the graph of the recorded decimals is the graph of `m`, computed exactly; the same applies to the instants. The check `all(abs(m) <= 2^50) && all(m / 10^d == y)` passes in that case, fails whenever some y_i is not the double nearest to a decimal with d places, and, when it passes, $m_i/10^d$ is the only decimal with d places whose nearest double is y_i . Proof, in the rounding to nearest in which R computes: 10^d is an exact double for $d \leq 22$, because $5^{22} < 2^{53}$. Since y_i is nearest to q_i , $y_i = q_i(1 + \delta)$ with $|\delta| < u = 2^{-53}$; the computed product `10^d * y[i]` equals $m_i(1 + \delta)(1 + \delta')$ with $|\delta'| < u$, so it differs from m_i by less than $2^{50}(2u + u^2)$, which is below $1/2$, and `round()` returns m_i ; `m[i] / 10^d`, a single correctly rounded division, returns the double nearest to q_i , that is y_i . When the check passes, y_i is the double nearest to $m_i/10^d$, and no other decimal with d places shares it: two such decimals differ by at least 10^{-d} , while two numbers rounded to the same y_i differ by at most $2u|y_i|$, which is below $10^{-d}/2$. Finally, multiplying every value by $10^d > 0$ multiplies every D by 10^d , which keeps its sign, and the integers m_i are exact doubles. Without the bound the recipe can fail: the decimals 2^{53} , $2^{53} + 1$ and $2^{53} + 2$ are stored as 2^{53} , 2^{53} and $2^{53} + 2$, and rounding cannot recover the middle one.

What the graph keeps of the series. Lacasa et al. (2008) describe the graph as inheriting properties of the series: in the words of their abstract, periodic series convert into regular graphs, random series into random graphs, and fractal series into scale-free networks. These are statements of the abstract, which does not say for which series they hold; they do not hold for every finite series: the periodic series 0, 1, 0, 1, 0 at the equally spaced instants 1, ..., 5 has, under the natural criterion, the edges 1-2, 2-3, 2-4, 3-4 and 4-5, and the degrees 1, 3, 2, 3, 1.

Dependencies

The compiled engine (C++ through 'Rcpp').

References

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

Shewchuk, J. R. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3), 305-363. <https://doi.org/10.1007/PL00009321>

See Also

[horizontal_visibility_graph\(\)](#), [generate_bitopology\(\)](#), [time_reverse\(\)](#).

Examples

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

natural_visibility_graph(c(0, 2, 4), times = c(0, 1, 4))$edges

y <- round(0.1 * (1:40) + 0.3, 1)
natural_visibility_graph(y)$n_edges
m <- round(10 * y)
all(abs(m) <= 2^50) && all(m / 10 == y)
natural_visibility_graph(m)$n_edges
```

simplest_topology	<i>The discrete topology on n points</i>
-------------------	---

Description

Returns the discrete topology on the indices of the data, the finest topology on a set, in which every subset is open. Only the singleton base is returned, because the full topology has 2^n open sets.

Usage

```
simplest_topology(data)
```

Arguments

data	Vector with at least one element; only its length is used. No default.
------	--

Value

A list with:

subbase List of the n singletons.

base The same list: the singletons form the minimal base.

topology_type Character: "discrete".

n Integer, the number of points.

connected Logical: TRUE only when $n = 1$; every singleton is open and closed, so two or more points are disconnected.

Methodological notes

The discrete topology is a reference point: every topology produced by the package is coarser than or equal to it.

Dependencies

Base R only.

See Also

[complete_topology\(\)](#) for the opposite extreme, the indiscrete topology.

Examples

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

time_reverse

Reverse a time series together with its instants

Description

Returns the series read backwards together with the reflection of its instants, the operation under which the forward and backward topologies of a visibility graph are exchanged (theorem T1, stated and proved in [generate_bitopology\(\)](#)). Use it instead of reversing the values alone, which is a different operation as soon as the instants are not equally spaced.

Usage

```
time_reverse(series, times = NULL)
```

Arguments

series	Numeric vector without NA, NaN or infinite values. A series of length 0 or 1 gives a graph without edges. No default.
times	NULL (default), meaning the instants $1, \dots, n$, or a numeric, Date or date-time vector of finite, strictly increasing instants of the same length as series; dates count in days and date-times in seconds.

Details

Reflecting the instants keeps every gap between consecutive observations and reverses their order. Any reflection $c - t_{n+1-i}$ gives the same visibility graphs in exact arithmetic, because both criteria are invariant under translations of the time axis, and $c = t_1 + t_n$ keeps the observation window. The function takes $c = 0$ because negation is exact in floating point, whereas $c - t$ can round. With times = NULL the reflection of $1, \dots, n$ is an exact translation of the same equally spaced grid, so NULL is returned.

Value

A list with series, the reversed values $\tilde{y}_i = y_{n+1-i}$, and times: NULL when times is NULL, and otherwise the numeric vector $\tilde{t}_i = -t_{n+1-i}$, again strictly increasing.

Methodological notes

Reversing the values over the same instants coincides with this operation only when the gaps between consecutive instants read the same in both directions. With the instants 0, 1, 4 and the values 0, 2, 4, the reversed values 4, 2, 0 over the same instants make the first and the third observation visible to each other, while in the original series they are not; with the reflected instants -4, -1, 0 the exchange of the two topologies holds.

Dependencies

Base R only.

See Also

[generate_bitopology\(\)](#) for the theorem and its consequences, [natural_visibility_graph\(\)](#) and [horizontal_visibility_graph\(\)](#).

Examples

```
rv <- time_reverse(c(0, 2, 4), times = c(0, 1, 4))
rv
natural_visibility_graph(rv$series, times = rv$times)$edges
natural_visibility_graph(c(0, 2, 4), times = c(0, 1, 4))$edges
natural_visibility_graph(c(4, 2, 0), times = c(0, 1, 4))$edges
```

`visualize_topology_thresholds`*Compare the candidate radii of `calculate_thresholds()`*

Description

Computes the five candidate radii of `calculate_thresholds()` and the size of the threshold-neighbourhood base that each one generates, and optionally plots them side by side.

Usage

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

Arguments

<code>data</code>	Numeric vector with at least two elements and no missing values. No default.
<code>plot</code>	Logical scalar; if TRUE (default) the result carries three ggplot objects in its "plots" attribute.

Value

A data.frame with the columns `method` (character), `threshold` (numeric) and `base_size` (integer, counted as in `calculate_topology()`). If `plot = TRUE`, it carries an attribute "plots": a list with the elements `threshold`, `base_size` and `scatter`.

Methodological notes

The comparison is descriptive: it shows how the radius, and with it the granularity of the base, depends on the scale chosen.

Dependencies

'ggplot2' (Imports) for the plots; `calculate_thresholds()` and `calculate_topology()` for the numbers.

See Also

`calculate_thresholds()`, `calculate_topology()`, `analyze_topology_factors()`.

Examples

```
set.seed(1)
results <- visualize_topology_thresholds(rnorm(50))
results
```

Index

analyze_topology_factors, [2](#)
analyze_topology_factors(), [8](#), [33](#)

bitopology_invariants, [4](#)
bitopology_invariants(), [12–15](#)

calculate_thresholds, [7](#)
calculate_thresholds(), [3](#), [8](#), [33](#)
calculate_topology, [8](#)
calculate_topology(), [3](#), [7](#), [8](#), [33](#)
complete_topology, [9](#)
complete_topology(), [31](#)

generate_alexandrov_topology, [10](#)
generate_alexandrov_topology(), [4](#),
[13–15](#), [17](#)
generate_bitopology, [12](#)
generate_bitopology(), [5](#), [6](#), [10–12](#), [15](#),
[17–19](#), [27](#), [30–32](#)
generate_topology, [15](#)
generate_topology(), [3–5](#), [9](#), [11–15](#), [19–24](#)

horizontal_visibility_graph, [18](#)
horizontal_visibility_graph(), [14](#), [27](#),
[29](#), [30](#), [32](#)

is_topology_connected, [20](#)
is_topology_connected(), [24](#)
is_topology_connected2, [23](#)
is_topology_connected2(), [21](#)
is_topology_connected_exact, [21](#)
is_topology_connected_exact(), [17](#), [20](#),
[21](#), [23](#), [24](#)
is_topology_connected_manual, [22](#)
is_topology_connected_manual(), [21](#)

natural_visibility_exactness, [14](#), [19](#), [24](#),
[27–29](#)
natural_visibility_graph, [27](#)
natural_visibility_graph(), [13](#), [14](#), [19](#),
[24](#), [27](#), [32](#)

simplest_topology, [30](#)
simplest_topology(), [9](#)

time_reverse, [31](#)
time_reverse(), [13–15](#), [19](#), [29](#), [30](#)

visualize_topology_thresholds, [33](#)
visualize_topology_thresholds(), [7](#), [8](#)