

Package ‘rd2dnet’

October 28, 2026

Type Package

Title Boundary Discontinuity Designs on Networks

Version 0.1.0

Description Estimation and inference for boundary discontinuity designs on street networks. Extends the location-based framework of 'rd2d' (Cattaneo, Titiunik and Yu, <[doi:10.48550/arXiv.2505.07989](https://doi.org/10.48550/arXiv.2505.07989)>) from Euclidean coordinates to network-native coordinates (signed network distance to the boundary and offset along it) and user-defined network metrics (edge costs, travel time, road-class transition penalties, precomputed matrices). Provides side-specific local polynomial estimates of boundary average treatment effect curves with robust bias-corrected inference (Calonico, Cattaneo and Titiunik, 2014, <[doi:10.3982/ECTA11757](https://doi.org/10.3982/ECTA11757)>), uniform confidence bands, weighted and largest aggregate effects, fuzzy designs, covariates, derivative targets, heteroskedasticity-robust, cluster and network Conley (1999, <[doi:10.1016/S0304-4076\(98\)00084-0](https://doi.org/10.1016/S0304-4076(98)00084-0)>) standard errors, a port of the 'rd2d' direct plug-in bandwidth selector, and a network-normalised version of the local polynomial density test of Cattaneo, Jansson and Ma (2020, <[doi:10.1080/01621459.2019.1635480](https://doi.org/10.1080/01621459.2019.1635480)>).

License GPL-3

URL <https://github.com/zwinship/rd2dnet>

BugReports <https://github.com/zwinship/rd2dnet/issues>

Depends R (>= 4.1.0)

Imports graphics, grDevices, methods, Rcpp, stats, utils

LinkingTo Rcpp

Suggests dodgr, igraph, knitr, Matrix, osmdata, rmarkdown, sf, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1**NeedsCompilation** yes

Author Zach Winship [aut, cre, cph],
 Matias D. Cattaneo [ctb, cph] (author of rd2d and rddensity, from which
 the bandwidth selector and density test are ported),
 Rocío Titiunik [ctb, cph] (author of rd2d, from which the bandwidth
 selector is ported),
 Ruiqi Rae Yu [ctb, cph] (author of rd2d, from which the bandwidth
 selector is ported),
 Michael Jansson [ctb, cph] (author of rddensity, from which the density
 test is ported),
 Xinwei Ma [ctb, cph] (author of rddensity, from which the density test
 is ported)

Maintainer Zach Winship <zachtwinship@gmail.com>**Repository** CRAN**Date/Publication** 2026-10-28 10:30:08 UTC**Contents**

boundary	3
boundary_geometry	4
boundary_grid	5
boundary_is_masked	5
boundary_is_treated	6
coords_at	6
density.rd2dnet_coords	7
edge_frame	9
eval_table	9
get_kernel	10
load_mock_midtown	11
metrics	12
mock_midtown	14
mock_true_tau	14
network_coordinates	15
network_distances	16
network_graph	17
network_graph_from_adjacency	18
network_graph_from_igraph	19
network_graph_from_place	20
pairwise_distances	21
project_lonlat	22
rdbwnet	22
rdnet	24
rdnet-methods	27
rect_polygon	28
snap_points	29

<i>boundary</i>	3
wbate	29
with_boundary_crossings	30
Index	32

boundary	<i>Boundary separating treated from control locations</i>
----------	---

Description

A boundary line, the side that is treated, and optional masked stretches.

Usage

```
boundary(
  line,
  treated_region = NULL,
  treated_side = NULL,
  mask = NULL,
  tol = 1e-06
)
```

Arguments

line	The boundary: an n x 2 matrix (or data frame with x/y) of vertices, or an sf LINESTRING, in the same projected coordinates as the network.
treated_region	Polygon containing the treated side: a ring matrix, a list of ring matrices (holes and several parts use the even-odd rule), or an sf polygon. Points inside it, or within tol of it, are treated.
treated_side	Alternatively "left" or "right" of the line's direction.
mask	A list of geometries (polygons as above, or sf geometries such as a park) and/or numeric c(start, end) arc-length intervals along the line. Evaluation points touching them are skipped: use this for stretches with no mass on one side (Central Park on 60th Street).
tol	Distance at which a network node counts as lying on the boundary.

Value

An object of class rd2dnet_boundary.

See Also

[boundary_project\(\)](#), [boundary_is_treated\(\)](#), [boundary_is_masked\(\)](#).

Examples

```
b <- boundary(rbind(c(0, 0), c(100, 0)), treated_side = "left",
              mask = list(rect_polygon(20, 0, 40, 10), c(80, 90)))
b
boundary_is_treated(b, rbind(c(50, 1), c(50, -1)))
boundary_is_masked(b, c(10, 30, 50, 85))
```

boundary_geometry	<i>Boundary geometry operations</i>
-------------------	-------------------------------------

Description

Distances to the boundary line, arc-length positions along it, and points at given positions.

Usage

```
boundary_distance(b, xy)

boundary_project(b, xy)

boundary_interpolate(b, arc)
```

Arguments

b	A boundary() .
xy	Points: an n x 2 matrix, a data frame with x/y, or sf points.
arc	Arc-length positions along the boundary line.

Value

`boundary_distance()`: Euclidean distance from each point to the line. `boundary_project()`: arc-length position of each point's projection onto the line. `boundary_interpolate()`: an n x 2 matrix of coordinates.

Examples

```
b <- boundary(rbind(c(0, 0), c(100, 0)), treated_side = "left")
boundary_distance(b, rbind(c(10, 5), c(120, 0)))
boundary_project(b, rbind(c(10, 5), c(120, 0)))
boundary_interpolate(b, c(0, 50))
```

boundary_grid	<i>Equally spaced positions along the boundary</i>
---------------	--

Description

Equally spaced positions along the boundary

Usage

```
boundary_grid(b, n = NULL, spacing = NULL, start = 0, end = NULL)
```

Arguments

b	A boundary() .
n	Number of positions (default 25) when spacing is not given.
spacing	Distance between consecutive positions.
start, end	Arc-length range (default: the whole line).

Value

Numeric vector of arc-length positions.

Examples

```
b <- boundary(rbind(c(0, 0), c(100, 0)), treated_side = "left")
boundary_grid(b, n = 5)
boundary_grid(b, spacing = 30)
```

boundary_is_masked	<i>Masked stretches of the boundary</i>
--------------------	---

Description

Masked stretches of the boundary

Usage

```
boundary_is_masked(b, arc)
```

Arguments

b	A boundary() .
arc	Arc-length positions along the boundary line.

Value

Logical: whether each arc-length position falls in a masked stretch.

Examples

```
b <- boundary(rbind(c(0, 0), c(100, 0)), treated_side = "left", mask = list(c(80, 90)))
boundary_is_masked(b, c(10, 85))
```

boundary_is_treated	<i>Treatment side of points</i>
---------------------	---------------------------------

Description

Treatment side of points

Usage

```
boundary_is_treated(b, xy)
```

Arguments

- b A [boundary\(\)](#).
- xy Points: an $n \times 2$ matrix, a data frame with x/y, or sf points.

Value

Logical: whether each point lies on the treated side (inside treated_region, within tol of it, or on treated_side of the line).

Examples

```
b <- boundary(rbind(c(0, 0), c(10, 0)), treated_side = "right")
boundary_is_treated(b, rbind(c(5, 1), c(5, -1)))
```

coords_at	<i>Kernel distance and boundary offset at an evaluation point</i>
-----------	---

Description

Kernel distance and boundary offset at an evaluation point

Usage

```
coords_at(coords, j)
```

Arguments

`coords` Output of `network_coordinates()`.
`j` Evaluation point (1-based index into `coords$eval_arc`).

Value

A list with `d`, the network distance from evaluation point `j` to every observation (for kernel weights), and `o`, each observation's signed offset along the boundary from `j` (NA when the boundary cannot be reached).

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 5)
str(coords_at(coords, 3))
```

density.rd2dnet_coords

Density (sorting) test at a network boundary

Description

Tests for a discontinuity in the density of observations at the boundary with the local polynomial density estimator of Cattaneo, Jansson and Ma (as in **rddensity**), with jackknife standard errors and bias-corrected inference, applied to the signed network distance `s`.

Usage

```
## S3 method for class 'rd2dnet_coords'
density(
  x,
  eval_index = NULL,
  window = NULL,
  h = NULL,
  p = 2,
  q = NULL,
  normalize = "network",
  kernel = "triangular",
  weights = NULL,
  bwselect = "common",
  scaleregul = 1,
  ...
)
```

Arguments

<code>x</code>	Output of <code>network_coordinates()</code> .
<code>eval_index, window</code>	Restrict to observations (and streets) whose boundary crossing lies within window (arc length) of evaluation point <code>eval_index</code> (1-based). By default the whole boundary is pooled.
<code>h</code>	Bandwidth, scalar or <code>c(h_left, h_right)</code> , in metric units (street-length units with <code>normalize = "network"</code>). Selected by an MSE plug-in rule per side when omitted.
<code>p, q</code>	Local polynomial order of the CDF fit (the density is its slope) and of the bias-corrected fit used for inference (default <code>p + 1</code>).
<code>normalize</code>	"network" (observations per metre of street) or "none" (density of observations in <code>s</code> , as in <code>rddensity</code>), which also reacts to differences in street supply across the boundary.
<code>kernel</code>	Kernel name or function (see <code>get_kernel()</code>).
<code>weights</code>	Optional non-negative observation weights.
<code>bwselect</code>	"common" (one bandwidth, the smaller of the two side-wise MSE-optimal choices, for both sides) or "each" (separate bandwidths). On street grids the density of <code>s</code> is jagged, and unequal bandwidths let unequal smoothing bias pass for a discontinuity.
<code>scaleregul</code>	Weight on the variance of the bias estimate in the bandwidth rule.
<code>...</code>	Ignored.

Details

With the default `normalize = "network"`, each observation's distance is re-expressed as cumulative street length: $G(a)$ is the total length of street on that side lying within network distance a of the boundary, and the test runs on $\text{sign} * G(|s|)$. The density in that coordinate is observations per metre of street, so jumps in street supply at the boundary (a highway on one side, a dense grid on the other, a park frontage) do not masquerade as sorting. Bandwidths are then in street-length units. Masked stretches of the boundary are always excluded.

Value

An object of class `rdnet_density`: the density jump `diff (f+ - f-, order p)` and `diff_bc` (bias-corrected), its jackknife standard error `se`, `z`, `pvalue`, the descriptive log ratio `theta (log(f+/f-))`, side-wise densities `f_left/f_right`, bandwidths `h_left/h_right`, sample sizes `n_left/n_right` and within the bandwidth `n_left_h/n_right_h`, `p`, `q` and `normalize`.

References

- Cattaneo, M. D., Jansson, M. and Ma, X. (2020). Simple Local Polynomial Density Estimators. *Journal of the American Statistical Association* 115(531): 1449-1455.
- Cattaneo, M. D., Jansson, M. and Ma, X. (2018). Manipulation Testing Based on Density Discontinuity. *Stata Journal* 18(1): 234-261.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 5)
density(coords)
density(coords, normalize = "none")
```

edge_frame	<i>Street segments of a network</i>
------------	-------------------------------------

Description

One row per undirected street segment (the first edge of each node pair): endpoint indices *u* and *v*, geometric length (the *length* column, or the geometry's length), the metric cost of traversing it (NA for a precomputed metric), and its geometry oriented from *u* to *v*.

Usage

```
edge_frame(net)
```

Arguments

net A [network_graph\(\)](#).

Value

A data frame with a list column geometry.

Examples

```
city <- load_mock_midtown()
head(edge_frame(city$network())[, c("u", "v", "length", "cost")])
```

eval_table	<i>Evaluation points of network coordinates</i>
------------	---

Description

Evaluation points of network coordinates

Usage

```
eval_table(coords)
```

Arguments

coords Output of [network_coordinates\(\)](#).

Value

A data frame with one row per evaluation point: requested arc position `arc`, nearest network node id `node`, arc position after snapping onto the network `node_arc`, coordinates `x`, `y`, and `masked`.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 5)
eval_table(coords)
```

get_kernel	<i>Distance-decay kernels</i>
------------	-------------------------------

Description

Kernel functions evaluated at $u = d / h$, with d a network distance.

Usage

```
get_kernel(kernel)
```

Arguments

`kernel` "triangular", "epanechnikov", "uniform", "gaussian", their three-letter abbreviations ("tri", "epa", "uni", "gau"), or a function, which is returned unchanged.

Value

A vectorised function of u .

Examples

```
k <- get_kernel("epanechnikov")
k(c(0, 0.5, 1.5))
```

load_mock_midtown	<i>Mock Midtown city</i>
-------------------	--------------------------

Description

load_mock_midtown() loads the bundled synthetic Midtown-Manhattan-style street grid and 3,000 simulated parcels (no internet needed); it is the same data as the Python package's bundled city. make_mock_midtown() simulates a new city of the same design with R's random number generator.

Usage

```
load_mock_midtown()
```

```
make_mock_midtown(n_obs = 3000, seed = 0, park = TRUE)
```

Arguments

n_obs	Number of simulated parcels.
seed	Random seed (the global random number stream is left untouched).
park	Include the park (and mask its boundary frontage).

Details

Layout (metres): 11 north-south avenues 250 m apart ("primary"), 41 east-west streets 80 m apart numbered 40th-80th ("residential"), a west-side highway ("motorway") joined to the grid by ramps ("motorway_link"), and a park north of the boundary between avenues 3 and 7 with no interior streets or parcels. The boundary is 60th Street ($y = 1600$); the treated zone lies south of it. The outcome is $0.0008 (y - 1600) + 0.6 \cos(x / 700) + 0.2 \log(\text{far}) + \text{tau}(\text{arc}) T + N(0, 1)$ with tau given by [mock_true_tau\(\)](#).

Value

A rd2dnet_mock_city list with nodes, edges, obs, the [boundary\(\)](#) boundary, the park polygon park, boundary_y, true_tau (a function of arc length) and network(metric, ...), a function returning the [network_graph\(\)](#) with the given metric.

See Also

[mock_midtown_obs](#), [mock_midtown_nodes](#), [mock_midtown_edges](#).

Examples

```
city <- load_mock_midtown()
city
head(city$obs)
city$network("length")
small <- make_mock_midtown(n_obs = 500, seed = 1)
```

metrics

*Network metrics***Description**

A metric defines what distance means on a `network_graph()`. Distance is dissimilarity: it does not have to be metres. The coordinates `s` and `o`, the kernel distances and the bandwidths are all in the metric's units.

Usage

```
edge_cost(cost = "length", snap_scale = NULL, name = NULL)

transition_cost(
  edge_cost = "length",
  class_attr = "highway",
  penalty = NULL,
  default_penalty = 0,
  connectors = character(),
  default_class = "unclassified",
  snap_scale = NULL
)

precomputed(matrix, nodes = NULL)

travel_time(
  speed_attr = "speed_kph",
  length_attr = "length",
  default_kph = NULL
)

node_contrast(attr, weight, base = "length", snap_scale = NULL)
```

Arguments

<code>cost</code>	Edge column name or a vectorised function (see Details).
<code>snap_scale</code>	Cost per unit of off-network snapping distance. Defaults to 1 when the cost is "length" and 0 otherwise.
<code>name</code>	Label used when printing.
<code>edge_cost</code>	Base per-edge cost (column name or function), as in <code>edge_cost()</code> .
<code>class_attr</code>	Edge column holding the edge class (OSM's "highway" by default).
<code>penalty</code>	Transition penalties: a function <code>f(from_class, to_class)</code> returning a cost, a data frame with columns <code>from</code> , <code>to</code> and <code>penalty</code> , or a matrix with class names as row (<code>from</code>) and column (<code>to</code>) names. Pairs not given cost <code>default_penalty</code> when the classes differ and 0 otherwise.

default_penalty	Penalty for unlisted pairs of different classes.
connectors	Classes that do not reset the remembered class (ramps, links, crosswalks).
default_class	Class of edges with a missing class value.
matrix	Square matrix; <code>matrix[i, j]</code> is the distance from node <code>nodes[i]</code> to node <code>nodes[j]</code> .
nodes	Node ids for the rows and columns. Defaults to the network's node order.
speed_attr, length_attr	Edge columns with the speed (km/h) and length (m).
default_kph	Speed used where <code>speed_attr</code> is missing.
attr	Node column compared across each edge.
weight	Cost per unit of absolute difference in <code>attr</code> .
base	Edge column with the base cost.

Details

- `edge_cost()`: path cost is the sum of edge costs. `cost` is an edge column name ("length", "travel_time") or a function. A function is called once, vectorised over all edges, as `cost(edges)` or, if it takes two arguments, `cost(edges, nodes)`; `edges` has columns `from` and `to` (node ids) plus every edge attribute, `nodes` has `name`, `x`, `y` and the node attributes. It returns one non-negative cost per edge (Inf blocks an edge).
- `transition_cost()`: edge costs plus a penalty for *changing* edge class (e.g. highway to local street). Edges whose class is in `connectors` (ramps, links) are transparent: the penalty is charged between the last non-connector class and the next one, so highway -> ramp -> highway stays cheap while highway -> ramp -> residential is expensive.
- `precomputed()`: a user-supplied node-by-node dissimilarity matrix. Observations then snap to nodes, since per-edge costs are unknown.
- `travel_time()`: seconds to traverse each edge from a length (m) and a speed (km/h) column.
- `node_contrast()`: $\text{base} + \text{weight} * |\text{attr}(u) - \text{attr}(v)|$, for land-use or other node-level dissimilarity.

Value

An object of class `rd2dnet_metric`.

Examples

```
edge_cost("length")
edge_cost(function(edges) 2 * edges$length, name = "double length")
travel_time()
node_contrast("far", 50)
transition_cost("length", class_attr = "highway",
               penalty = data.frame(from = "motorway", to = "residential", penalty = 2000),
               connectors = c("motorway_link", "trunk_link"))
```

mock_midtown	<i>Mock Midtown data</i>
--------------	--------------------------

Description

The bundled synthetic city used in examples, tests and the vignette (see [load_mock_midtown\(\)](#), which assembles them with the boundary).

Format

mock_midtown_obs: 3,000 simulated parcels with columns

x, y projected coordinates (metres)

treated south of the boundary (60th Street, $y < 1600$)

far floor-area ratio, used as intensity weight

block city block id, for clustering

outcome simulated outcome

mock_midtown_nodes: 450 street intersections with columns name, x, y and a node-level floor-area ratio far.

mock_midtown_edges: street segments with columns from, to, length (metres), highway (road class), speed_kph and travel_time (seconds).

Source

Simulated; identical to the data bundled with the Python package rd2dnet.

mock_true_tau	<i>Mock city treatment effect</i>
---------------	-----------------------------------

Description

The treatment effect used to simulate the mock city's outcome, by arc length along its boundary: $1.5 + 0.75 \sin(2 \pi x / 2500)$ with x the easting.

Usage

```
mock_true_tau(arc)
```

Arguments

arc Arc-length positions along the mock boundary.

Value

Numeric vector of true effects.

Examples

```
mock_true_tau(c(0, 500, 1000))
```

network_coordinates	<i>Network coordinates around a boundary</i>
---------------------	--

Description

Translates point locations into network coordinates around a boundary: the signed network distance s to the boundary (positive on the treated side) and, at every evaluation point b along the boundary, the network distance $d(b)$ used for kernel weights and the signed offset $o(b)$ along the boundary. These replace rd2d's Euclidean $(X1 - b1, X2 - b2)$.

Usage

```
network_coordinates(
  network,
  boundary,
  points,
  treated = NULL,
  eval_points = NULL,
  n_eval = 25,
  snap = "edge",
  lonlat = FALSE,
  split_edges = TRUE
)

## S3 method for class 'rd2dnet_coords'
as.data.frame(x, row.names = NULL, optional = FALSE, j = NULL, ...)
```

Arguments

network	A network_graph() , carrying its metric.
boundary	A boundary() .
points	Observation locations: an $n \times 2$ matrix, a data frame with x/y columns, or sf points, in the network's projected coordinates (or longitude/latitude with lonlat = TRUE).
treated	Optional treatment indicator; defaults to the side of the boundary each point is on.
eval_points	Arc-length positions along the boundary, or a two-column matrix of coordinates to project onto it. Defaults to n_eval equally spaced points spanning the boundary nodes. Each evaluation point is placed on the street segment running along the boundary, or at the nearest boundary node when there is none.
n_eval	Number of default evaluation points.
snap	"edge" (attach to the nearest street segment) or "node".

lonlat	Points are longitude/latitude; project them into the network's crs (requires sf).
split_edges	Insert nodes where edges cross the boundary (recommended).
x	Output of <code>network_coordinates()</code> .
row.names, optional	Ignored (for compatibility with the generic).
j	Optional evaluation point index; adds the columns d and o at that point.
...	Ignored.

Details

Each observation snaps to the street network **on its own side** of the boundary; its off-network depth is charged at the network's `snap_scale`. Repeated locations (panels, sensors) are processed once.

Value

An object of class `rd2dnet_coords`, a list with (among others) `s`, `treated`, `xy`, `crossing_arc` (where each observation reaches the boundary), `eval_arc`, `eval_node_arc`, `eval_xy`, `eval_masked`, `loc` (location id of each observation), `n_obs`, `n_eval`, `n_locations`, and the (boundary-split) network and boundary. Use `coords_at()` for d and o at an evaluation point, `eval_table()` for the evaluation points, and `as.data.frame()` for a per-observation table.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 11)
coords
head(as.data.frame(coords, j = 5))
```

network_distances	<i>Network distances</i>
-------------------	--------------------------

Description

Metric distances between network nodes.

Usage

```
distances_from(net, sources, limit = Inf)

nearest_source(net, sources)

node_distance(net, a, b)
```

Arguments

<code>net</code>	A <code>network_graph()</code> .
<code>sources</code>	Node indices (row numbers of <code>net\$nodes</code>) to search from.
<code>limit</code>	Distances beyond <code>limit</code> are reported as <code>Inf</code> .
<code>a, b</code>	Node ids.

Value

`distances_from()`: a `length(sources) x n_nodes` matrix. `nearest_source()`: a list with, for every node, the distance to the nearest source (`dist`) and that source's node index (`src`, `NA` if unreachable). `node_distance()`: a single distance.

Examples

```
nodes <- data.frame(name = 1:3, x = c(0, 1, 2), y = 0)
net <- network_graph(data.frame(from = 1:2, to = 2:3, length = c(5, 7)), nodes)
distances_from(net, 1)
nearest_source(net, c(1, 3))
node_distance(net, 1, 3)
```

<code>network_graph</code>	<i>Spatial network with a distance metric</i>
----------------------------	---

Description

A street (or any spatial) network together with the metric that defines distance on it.

Usage

```
network_graph(
  edges,
  nodes,
  metric = "length",
  directed = FALSE,
  direction = "both",
  x = "x",
  y = "y",
  crs = NULL,
  snap_scale = NULL
)
```

Arguments

edges	Data frame with one row per edge: columns from and to (node ids, as in the node id column of nodes) plus any edge attributes (length, highway, speed_kph, ...). An optional geometry column holds each edge's shape as a list of $n \times 2$ matrices or an sf LINESTRING column; edges without one are straight segments. Repeated node pairs (multigraphs) are allowed; self-loops are ignored.
nodes	Data frame with one row per node: a node id column (name or id; otherwise ids are the row numbers), projected coordinates x and y, and any node attributes.
metric	How distance is measured: an edge column name ("length"), a vectorised cost function, a metric from metrics such as transition_cost() , or a square node-by-node matrix.
directed	Whether edges are one-way (from -> to).
direction	For directed networks: "both" treats every edge as two-way (pedestrian access, the default); "out" follows edge direction; "in" reverses it.
x, y	Names of the coordinate columns in nodes.
crs	Coordinate reference system label of the projected coordinates (used to project longitude/latitude points).
snap_scale	Metric cost per unit of off-network snapping distance. Defaults to the metric's own default (1 for length, 0 otherwise).

Value

An object of class `rd2dnet_network`.

See Also

[network_graph_from_adjacency\(\)](#), [network_graph_from_igraph\(\)](#), [network_graph_from_place\(\)](#), [network_coordinates\(\)](#).

Examples

```
nodes <- data.frame(name = c("a", "b", "c"), x = c(0, 100, 200), y = 0)
edges <- data.frame(from = c("a", "b"), to = c("b", "c"), length = c(100, 100))
net <- network_graph(edges, nodes)
net
node_distance(net, "a", "c")
```

network_graph_from_adjacency

Build a network from an adjacency matrix

Description

Each nonzero entry $A[i, j]$ becomes an edge with weight (the entry) and length (the straight-line distance between the two nodes).

Usage

```
network_graph_from_adjacency(
  adjacency,
  coords,
  nodes = NULL,
  directed = FALSE,
  metric = "weight",
  ...
)
```

Arguments

adjacency	Square weighted adjacency matrix (base or Matrix).
coords	n x 2 node coordinates.
nodes	Optional node ids (default 1:n).
directed	Whether entries are one-way.
metric	Metric specification, as in network_graph() ; defaults to the matrix entries.
...	Further arguments to network_graph() .

Value

A [network_graph\(\)](#).

Examples

```
A <- rbind(c(0, 2, 0), c(2, 0, 3), c(0, 3, 0))
net <- network_graph_from_adjacency(A, rbind(c(0, 0), c(1, 0), c(2, 0)))
node_distance(net, 1, 3)
```

network_graph_from_igraph

Build a network from an igraph graph

Description

Vertices must carry projected coordinates; vertex names (if any) become node ids and edge attributes become edge columns.

Usage

```
network_graph_from_igraph(g, metric = "weight", x = "x", y = "y", ...)
```

Arguments

<code>g</code>	An igraph graph.
<code>metric</code>	Metric specification, as in <code>network_graph()</code> .
<code>x, y</code>	Vertex attributes holding the coordinates.
<code>...</code>	Further arguments to <code>network_graph()</code> .

Value

A `network_graph()`.

Examples

```
g <- igraph::make_ring(4)
igraph::V(g)$x <- c(0, 1, 1, 0)
igraph::V(g)$y <- c(0, 0, 1, 1)
igraph::E(g)$weight <- 1
network_graph_from_igraph(g)
```

`network_graph_from_place`

Download an OpenStreetMap street network

Description

Downloads streets with **dodgr** (which uses **osmdata**) and projects them to the local UTM zone with **sf**. If a package is missing or the download fails and `offline_fallback` is `TRUE`, the bundled mock Midtown network is returned with a warning, so scripts keep working offline.

Usage

```
network_graph_from_place(
  query = NULL,
  bbox = NULL,
  network_type = "walk",
  metric = "length",
  offline_fallback = TRUE,
  dist = 1000,
  min_area = 250000,
  ...
)
```

Arguments

query	Place name, e.g. "Midtown Manhattan, New York".
bbox	Alternatively a bounding box c(xmin, ymin, xmax, ymax) in longitude/latitude.
network_type	"walk", "bike" or "drive" (or a dodgr weighting profile).
metric	Metric specification, as in network_graph() .
offline_fallback	Return the bundled mock network when the download fails.
dist, min_area	query is geocoded with Nominatim and the bounding box of its top match is downloaded. When that match is a point (many neighbourhoods, such as Midtown Manhattan, are mapped as points) or covers less than min_area square metres (a single building), the streets within dist metres of it are downloaded instead, with a warning.
...	Further arguments to network_graph() .

Value

A [network_graph\(\)](#) with length (m), highway, and travel_time (s) edge columns.

Examples

```
net <- suppressWarnings(network_graph_from_place("Midtown Manhattan, New York"))
net
```

pairwise_distances	<i>Network distances between observations</i>
--------------------	---

Description

Network distances between observations

Usage

```
pairwise_distances(coords, idx, cols = NULL, cutoff = Inf)
```

Arguments

coords	Output of network_coordinates() .
idx	Observation indices for the rows.
cols	Observation indices for the columns (default idx).
cutoff	Distances beyond cutoff may be reported as Inf.

Value

A length(idx) × length(cols) matrix of network distances (including both snapping depths); an observation is at distance 0 from itself.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 5)
round(pairwise_distances(coords, 1:4), 1)
```

project_lonlat	<i>Project longitude/latitude to a planar coordinate system</i>
----------------	---

Description

Project longitude/latitude to a planar coordinate system

Usage

```
project_lonlat(lon, lat, to_crs)
```

Arguments

lon, lat	Longitude and latitude (WGS84).
to_crs	Target coordinate reference system (e.g. "EPSG:32618").

Value

An $n \times 2$ matrix of projected coordinates. Requires **sf**.

Examples

```
project_lonlat(-73.98, 40.76, "EPSG:32618")
```

rdbwnet	<i>Bandwidth selection for network boundary discontinuity designs</i>
---------	---

Description

Selects MSE- or coverage-error-optimal bandwidths (in metric units) at each evaluation point. A port of rd2d's direct plug-in selector (rdbw2d) to network coordinates: a rule-of-thumb pilot, a separate pilot bandwidth for the bias, regularisation (scaleregul) and a higher-order bias term (scalebiasrct).

Usage

```
rdbwnet(
  y,
  coords,
  p = 1,
  deriv = c(0, 0),
  tangvec = NULL,
  covs = NULL,
  kernel = "triangular",
  bwselect = "mserd",
  vce = "hc1",
  cluster = NULL,
  fuzzy = NULL,
  weights = NULL,
  bwcheck = NULL,
  scaleregul = 1,
  scalebiasrcrt = 1
)
```

Arguments

y	Outcome, one value per observation in coords.
coords	Output of <code>network_coordinates()</code> .
p	Polynomial order of the local fit.
deriv, tangvec, covs, kernel, cluster, fuzzy, weights	As in <code>rdnet()</code> .
bwselect	"mserd" (one MSE-optimal bandwidth for both sides), "msetwo" (one per side), "imserd"/"imsetwo" (one bandwidth for all evaluation points), or the coverage-error-optimal "cerrd", "certwo", "icerrd", "icertwo".
vce	"hc0"- "hc3" variance used inside the selector (others fall back to "hc1").
bwcheck	Every bandwidth is at least the distance to the bwcheck-th nearest observation on each side (default 20 + the number of order-p + 1 terms).
scaleregul, scalebiasrcrt	Weights on the bias-variance regularisation and the higher-order bias term.

Details

deriv/tangvec target a derivative jump and covs are partialled out (with coefficients common to both sides, estimated at the pilot bandwidth), as in rd2d. The cer* variants shrink the MSE-optimal bandwidth by rd2d's coverage-error rate factor $n^{(1/(2p+4) - 1/(p+4))}$. Fuzzy designs use the linearised outcome $y / fs - itt / fs^2 * D$, as rd2d does.

Value

A data frame with one row per evaluation point: the control/treated bandwidths h_0/h_1 (and h when they are equal), the pilots dn, bn_0, bn_1 , the MSE constants (V, B, Reg1, Reg2 per side, suffixed 0 for control and 1 for treated), the bounds $hmin_0/hmin_1/hmax_0/hmax_1$, local sample sizes n_0/n_1 , and a status.

References

Cattaneo, M. D., Titiunik, R. and Yu, R. R. (2026). rd2d: Causal Inference in Boundary Discontinuity Designs. arXiv:2505.07989.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 7)
bw <- rdbwnet(city$obs$outcome, coords)
bw[, c("arc", "h0", "h1", "n0", "n1", "status")]
```

rdnet

Boundary discontinuity estimation on a network

Description

Estimates the boundary average treatment effect curve (BATEC) at each evaluation point along the boundary, with robust bias-corrected inference and a uniform confidence band. This carries rd2d's location-based estimator over to network coordinates: (s, o(b)) replace (X1 - b1, X2 - b2) and the network distance d(b) replaces the Euclidean distance in the kernel.

Usage

```
rdnet(
  y,
  coords,
  h = NULL,
  p = 1,
  q = NULL,
  deriv = c(0, 0),
  tangvec = NULL,
  covs = NULL,
  kernel = "triangular",
  bwselect = "msrd",
  vce = "hc1",
  cluster = NULL,
  conley_cutoff = NULL,
  conley_distance = "network",
  conley_resid_scale = 3,
  conley_resid_order = NULL,
  fuzzy = NULL,
  weights = NULL,
  pooled = FALSE,
  level = 95,
  cbands = TRUE,
  repp = 2000,
  min_obs = NULL,
```

```

    scaleregul = 3,
    scalebiasrcrt = 1,
    seed = NULL
)

```

Arguments

y	Outcome, one value per observation in coords. Missing values are skipped.
coords	Output of <code>network_coordinates()</code> .
h	Bandwidth in metric units: a scalar, <code>c(h_control, h_treated)</code> , one value per evaluation point, a $J \times 2$ matrix, or NULL to select with <code>rdbwnet()</code> using <code>bwselect</code> .
p, q	Polynomial order for estimation and for bias-corrected inference (default $p + 1$).
deriv	<code>c(a, c)</code> : estimate the jump in $d^{(a+c)} \mu / ds^a do^c$ instead of the level. <code>c(1, 0)</code> is the jump in the slope across the boundary (a kink), <code>c(0, 1)</code> the jump in the slope along it. Requires $a + c \leq p$.
tangvec	Direction <code>c(t_s, t_o)</code> (or a $J \times 2$ matrix, one per evaluation point): estimate the jump in the directional derivative $t_s d/ds + t_o d/do$.
covs	Covariates (vector or $n \times m$ matrix) entering the local fit with coefficients common to both sides (rd2d's <code>covs_eff</code>). Covariates that are constant in a local sample are dropped there.
kernel	"triangular", "epanechnikov", "uniform", "gaussian" or a function (see <code>get_kernel()</code>).
bwselect	Bandwidth selector used when <code>h</code> is NULL; see <code>rdbwnet()</code> .
vce	"hc0"-"hc3"; "cluster" (with <code>cluster</code>) and "conley" (with <code>conley_cutoff</code>) are implied when those are given.
cluster	Cluster labels, one per observation; implies <code>vce = "cluster"</code> (CR1 with rd2d's $(n - 1)/(n - k) * g/(g - 1)$ factor).
conley_cutoff, conley_distance	Spatial HAC with Bartlett weights over "network" (default) or "euclidean" distance, used for pointwise errors and for the uniform band.
conley_resid_scale, conley_resid_order	Conley residuals come from an auxiliary fit of order <code>conley_resid_order</code> (default <code>q</code>) at <code>conley_resid_scale * h</code> (default 3). Residuals from the estimation fit itself absorb spatially correlated shocks at scales below <code>h</code> , which makes the HAC variance far too small.
fuzzy	Treatment take-up D (0/1 or a share). The estimand becomes the ratio of the outcome jump to the take-up jump at each boundary point.
weights	Non-negative intensity weights (e.g. floor area) multiplying the kernel weights.
pooled	Fit one regression $y \sim 1 + T + \text{poly}(s, o)$ with common slopes instead of side-specific polynomials (levels only).
level	Confidence level in percent.
cbands	Add the uniform confidence band <code>cb_lo/cb_hi</code> to the table. The joint covariance and its critical value are stored either way.

repp	Gaussian draws used to simulate the uniform critical value.
min_obs	Minimum local observations per side (default 20 + the number of order-q terms); points with fewer are reported as "insufficient".
scaleregul, scalebiasrcrt	Bandwidth-selection regularisation and higher-order bias weights, passed to <code>rdbwnet()</code> when <code>h</code> is selected.
seed	Seed for the critical-value simulation (reproducible bands and LBATE). The global random number stream is left untouched.

Value

An object of class `rdnet`, a list with

`table` one row per evaluation point: `eval`, `arc`, `node_arc`, bandwidths `h0`, `h1` (and `h` when equal), local sample sizes `n0`, `n1`, the order-p estimate `tau` and its standard error `se`, the bias-corrected `tau_bc` with robust `se_rb`, the robust interval `ci_lo/ci_hi`, `pvalue`, a status ("ok", "masked", "insufficient", "singular", "no bandwidth", "weak first stage"), and optionally the uniform band `cb_lo/cb_hi`. Fuzzy designs add `itt`, `fs` and `fs_se`.

`cov` the $J \times J$ covariance of `tau_bc` across evaluation points (`rd2d`'s `params_cov`), NA for points not estimated; its diagonal is `se_rb^2`.

`crit` the uniform critical value implied by `cov`.

`bandwidths` the `rdbwnet()` output when `h` was selected.

plus `p`, `q`, `kernel`, `vce`, `level`, `metric`, `fuzzy` and `coords`. Use `wbate()` and `lbate()` for aggregate effects, `summary()`, `plot()`, `coef()`, `vcov()` and `confint()` for the usual summaries.

References

- Cattaneo, M. D., Titiunik, R. and Yu, R. R. (2026). `rd2d`: Causal Inference in Boundary Discontinuity Designs. [arXiv:2505.07989](https://arxiv.org/abs/2505.07989).
- Calonico, S., Cattaneo, M. D. and Titiunik, R. (2014). Robust Nonparametric Confidence Intervals for Regression-Discontinuity Designs. *Econometrica* 82(6): 2295-2326.
- Conley, T. G. (1999). GMM Estimation with Cross Sectional Dependence. *Journal of Econometrics* 92(1): 1-45.

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 9)
res <- rdnet(city$obs$outcome, coords, weights = city$obs$far, seed = 1)
res
ok <- res$table$status == "ok"
wbate(res, as.numeric(ok))
plot(res)
```

Description

Summaries of rdnet results

Usage

```
## S3 method for class 'rdnet'
summary(object, wbate = NULL, lbate = FALSE, ...)

## S3 method for class 'summary.rdnet'
print(x, digits = 3, ...)

## S3 method for class 'rdnet'
print(x, ...)

## S3 method for class 'rdnet'
coef(object, type = c("tau_bc", "tau"), ...)

## S3 method for class 'rdnet'
vcov(object, ...)

## S3 method for class 'rdnet'
confint(object, parm, level = object$level/100, ...)

## S3 method for class 'rdnet'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'rdnet'
plot(x, band = TRUE, true_tau = NULL, ...)
```

Arguments

<code>object, x</code>	An <code>rdnet()</code> result (or its summary).
<code>wbate</code>	Optional WBATE weights (see <code>wbate()</code>) to add an aggregate row.
<code>lbate</code>	Add the LBATE row (see <code>lbate()</code>).
<code>...</code>	Further arguments (passed to <code>plot()</code> for the plot method).
<code>digits</code>	Digits to print.
<code>type</code>	For <code>coef()</code> : "tau_bc" (bias-corrected, default) or "tau".
<code>parm</code>	Evaluation points (indices) for <code>confint()</code> ; default all.
<code>level</code>	Confidence level for <code>confint()</code> as a proportion.
<code>row.names, optional</code>	Ignored.

- band Draw the uniform band.
- true_tau Optional true effects at the evaluation points, drawn dashed.

Value

summary() returns an object of class summary.rdnet; coef() the estimates by evaluation point; vcov() the joint covariance x\$cov; confint() robust intervals tau_bc +/- z se_rb; as.data.frame() the result table; plot() draws the BATEC curve with pointwise robust intervals and the uniform band and returns x invisibly.

rect_polygon	<i>Rectangle polygon</i>
--------------	--------------------------

Description

A convenience constructor for an axis-aligned rectangle, usable wherever a polygon is accepted (treated regions, masks).

Usage

rect_polygon(xmin, ymin, xmax, ymax)

Arguments

xmin, ymin, xmax, ymax
Corners of the rectangle.

Value

A 5 x 2 matrix of ring vertices (closed).

Examples

rect_polygon(0, 0, 10, 5)

snap_points	<i>Attach points to a network</i>
-------------	-----------------------------------

Description

Each point reaches node u at cost cost_u and node v at cost cost_v along its street segment, after an off-network depth (Euclidean units). Points snapped to a node have $u == v$ and zero edge costs.

Usage

```
snap_points(net, xy, allowed = NULL, mode = "edge")
```

Arguments

net	A network_graph() .
xy	Points: an $n \times 2$ matrix, a data frame with x/y, or sf points.
allowed	Optional logical vector over nodes. With mode = "edge" a point may only snap to segments whose two endpoints are allowed.
mode	"edge" (nearest street segment) or "node" (nearest node). Precomputed metrics always snap to nodes.

Value

A list with integer vectors u , v (node indices) and numeric vectors cost_u , cost_v , depth .

Examples

```
nodes <- data.frame(name = 1:2, x = c(0, 100), y = 0)
net <- network_graph(data.frame(from = 1, to = 2, length = 100), nodes)
snap_points(net, rbind(c(30, 10)))
```

wbate	<i>Aggregate boundary treatment effects</i>
-------	---

Description

`wbate()` is the weighted boundary average treatment effect $w' \tau$ (rd2d's WBATE): `weights` has one entry per evaluation point and is normalised to sum to one; points that were not estimated must get weight 0. Its standard error $\sqrt{w' \Sigma w}$ uses the joint covariance `x$cov`, so it accounts for the correlation between nearby points.

Usage

```
wbate(x, weights)
```

```
lbate(x)
```

Arguments

x	An <code>rdnet()</code> result.
weights	One weight per evaluation point.

Details

`lbate()` is the largest boundary average treatment effect $\max_j \tau(b_j)$ (rd2d's LBATE), with interval $[\max_j (\tau_{bc} - \text{crit } se_{rb}), \max_j (\tau_{bc} + \text{crit } se_{rb})]$ over the estimated points, `crit` the uniform critical value, so it inherits the uniform band's joint coverage.

Value

A named numeric vector: `tau`, `tau_bc`, `se_rb`, `ci_lo`, `ci_hi`, `pvalue` (the bias-corrected estimate drives the interval and p-value; `se_rb` and `pvalue` are NA for LBATE).

Examples

```
city <- load_mock_midtown()
coords <- network_coordinates(city$network(), city$boundary, city$obs, n_eval = 9)
res <- rdnet(city$obs$outcome, coords, h = 600, seed = 1)
ok <- res$table$status == "ok"
wbate(res, as.numeric(ok))
lbate(res)
```

with_boundary_crossings

Split edges where they cross the boundary

Description

Inserts a node wherever an edge crosses the boundary line, so that every path between the two sides passes through a node on the boundary. Columns listed in `extensive` are divided in proportion to the split; all others are copied. `network_coordinates()` does this automatically.

Usage

```
with_boundary_crossings(net, b, extensive = c("length", "travel_time"))
```

Arguments

net	A <code>network_graph()</code> .
b	A <code>boundary()</code> .
extensive	Edge columns that scale with length.

Value

A network (the same one when nothing needs splitting), with attribute `"n_crossings_inserted"`.

Examples

```
nodes <- data.frame(name = 1:2, x = 0, y = c(0, 100))
net <- network_graph(data.frame(from = 1, to = 2, length = 100), nodes)
b <- boundary(rbind(c(-10, 40), c(10, 40)), treated_region = rect_polygon(-50, -50, 50, 40))
split <- with_boundary_crossings(net, b)
split$edges
```

Index

- * **datasets**
 - mock_midtown, [14](#)
- as.data.frame.rd2dnet_coords
 - (network_coordinates), [15](#)
- as.data.frame.rdnet (rdnet-methods), [27](#)
- boundary, [3](#)
- boundary(), [4–6](#), [11](#), [15](#), [30](#)
- boundary_distance (boundary_geometry), [4](#)
- boundary_geometry, [4](#)
- boundary_grid, [5](#)
- boundary_interpolate
 - (boundary_geometry), [4](#)
- boundary_is_masked, [5](#)
- boundary_is_masked(), [3](#)
- boundary_is_treated, [6](#)
- boundary_is_treated(), [3](#)
- boundary_project (boundary_geometry), [4](#)
- boundary_project(), [3](#)
- coef.rdnet (rdnet-methods), [27](#)
- confint.rdnet (rdnet-methods), [27](#)
- coords_at, [6](#)
- coords_at(), [16](#)
- density.rd2dnet_coords, [7](#)
- distances_from (network_distances), [16](#)
- edge_cost (metrics), [12](#)
- edge_frame, [9](#)
- eval_table, [9](#)
- eval_table(), [16](#)
- get_kernel, [10](#)
- get_kernel(), [8](#), [25](#)
- lbate (wbate), [29](#)
- lbate(), [26](#), [27](#)
- load_mock_midtown, [11](#)
- load_mock_midtown(), [14](#)
- make_mock_midtown (load_mock_midtown),
[11](#)
- metrics, [12](#), [18](#)
- mock_midtown, [14](#)
- mock_midtown_edges, [11](#)
- mock_midtown_edges (mock_midtown), [14](#)
- mock_midtown_nodes, [11](#)
- mock_midtown_nodes (mock_midtown), [14](#)
- mock_midtown_obs, [11](#)
- mock_midtown_obs (mock_midtown), [14](#)
- mock_true_tau, [14](#)
- mock_true_tau(), [11](#)
- nearest_source (network_distances), [16](#)
- network_coordinates, [15](#)
- network_coordinates(), [7–9](#), [16](#), [18](#), [21](#), [23](#),
[25](#), [30](#)
- network_distances, [16](#)
- network_graph, [17](#)
- network_graph(), [9](#), [11](#), [12](#), [15](#), [17](#), [19–21](#),
[29](#), [30](#)
- network_graph_from_adjacency, [18](#)
- network_graph_from_adjacency(), [18](#)
- network_graph_from_igraph, [19](#)
- network_graph_from_igraph(), [18](#)
- network_graph_from_place, [20](#)
- network_graph_from_place(), [18](#)
- node_contrast (metrics), [12](#)
- node_distance (network_distances), [16](#)
- pairwise_distances, [21](#)
- plot.rdnet (rdnet-methods), [27](#)
- precomputed (metrics), [12](#)
- print.rdnet (rdnet-methods), [27](#)
- print.summary.rdnet (rdnet-methods), [27](#)
- project_lonlat, [22](#)
- rdbwnet, [22](#)
- rdbwnet(), [25](#), [26](#)
- rdnet, [24](#)

`rdnet()`, [23](#), [27](#), [30](#)
`rdnet-methods`, [27](#)
`rect_polygon`, [28](#)

`snap_points`, [29](#)
`summary.rdnet` (`rdnet-methods`), [27](#)

`transition_cost` (`metrics`), [12](#)
`transition_cost()`, [18](#)
`travel_time` (`metrics`), [12](#)

`vcov.rdnet` (`rdnet-methods`), [27](#)

`wbate`, [29](#)
`wbate()`, [26](#), [27](#)
`with_boundary_crossings`, [30](#)