Package ‘stplanr’

September 15, 2023

Type Package
Title Sustainable Transport Planning
Version 1.1.2
Maintainer Robin Lovelace <rob00x@gmail.com>
Description Tools for transport planning with an emphasis on spatial transport data and non-motorized modes. Create geographic "desire lines" from origin-destination (OD) data (building on the 'od' package); calculate routes on the transport network locally and via interfaces to routing services such as <https://cyclestreets.net/>; calculate route segment attributes such as bearing. The package implements the 'travel flow aggregation' method described in Morgan and Lovelace (2020) <doi:10.1177/2399808320942779>. Further information on the package's aim and scope can be found in the vignettes and in a paper in the R Journal (Lovelace and Ellison 2018) <doi:10.32614/RJ-2018-053>.
License MIT + file LICENSE
URL https://github.com/ropensci/stplanr,
https://docs.ropensci.org/stplanr/
BugReports https://github.com/ropensci/stplanr/issues
Depends R (>= 3.5.0)
Imports curl (>= 3.2), data.table, dplyr (>= 0.7.6), htr (>= 1.3.1), jsonlite (>= 1.5), lwgeom (>= 0.1.4), magrittr, methods, nabor (>= 0.5.0), od, pbapply, Rcpp (>= 0.12.1), rlang (>= 0.2.2), sf (>= 0.6.3), sfheaders
Suggests cyclestreets, dodgr (>= 0.2.15), geodist, igraph (>= 1.2.2), knitr (>= 1.20), leaflet, mapsapi, opentripplanner, osrm, pct, markdown (>= 1.10), testthat (>= 2.0.0), tmap
VignetteBuilder knitr
Encoding UTF-8
LazyData yes
RoxygenNote 7.2.3
SystemRequirements GNU make

NeedsCompilation no

Author Robin Lovelace [aut, cre] (<https://orcid.org/0000-0001-5679-6536>),
           Richard Ellison [aut],
           Malcolm Morgan [aut] (<https://orcid.org/0000-0002-9488-9183>),
           Barry Rowlingson [ctb],
           Nick Bearman [ctb],
           Nikolai Berkoff [ctb],
           Scott Chamberlain [rev] (Scott reviewed the package for rOpenSci, see https://github.com/ropensci/onboarding/issues/10),
           Mark Padgham [ctb],
           Zhao Wang [ctb] (<https://orcid.org/0000-0002-4054-0533>),
           Andrea Gilardi [ctb] (<https://orcid.org/0000-0002-9424-7439>)

Repository CRAN

Date/Publication 2023-09-15 08:22:05 UTC

R topics documented:

- stplanr-package ........................................... 4
- angle_diff ................................................. 4
- bbox_scale ................................................ 5
- bind_sf .................................................... 6
- cents_sf ................................................... 7
- destinations_sf .......................................... 7
- flow ....................................................... 8
- flowlines_sf ............................................. 9
- flow_dests ................................................. 9
- geo_bb ..................................................... 10
- geo_bb_matrix ........................................... 11
- geo_buffer ............................................... 11
- geo_code .................................................. 12
- geo_length ............................................... 13
- geo_projected ............................................ 14
- geo_select_aeq ......................................... 14
- geo_toptail .............................................. 15
- gsection .................................................. 16
- islines .................................................... 17
- is_linepoint ............................................. 18
- line2df ................................................... 18
- line2points .............................................. 19
- line_bearing ............................................. 20
- line_breakup ............................................. 21
- line_cast ................................................ 22
- line_midpoint ........................................... 22
- line_segment ............................................ 23
- line_via .................................................. 24
R topics documented:

mats2line ....................................................... 25
n_vertices ..................................................... 25
od2line .......................................................... 26
od2odf ............................................................ 27
odmatrix_to_od .................................................. 28
od_aggregate_from ............................................. 29
od_aggregate_to ................................................ 30
od_coords ........................................................ 31
od_coords2line ................................................... 31
od_data_lines ................................................... 32
od_data_routes .................................................. 33
od_data_sample .................................................. 33
od_id .............................................................. 34
od_id_order ....................................................... 35
od_oneway ........................................................ 35
od_to_odmatrix ................................................... 37
onewaygeo ........................................................ 38
osm_net_example ................................................ 39
overline ........................................................... 39
overline_intersection ............................................ 42
points2flow ....................................................... 43
points2line ....................................................... 43
points2odf ......................................................... 44
quadrant ........................................................... 44
read_table_builder .............................................. 45
rnet_add_node .................................................... 46
rnet_boundary_points ........................................... 47
rnet_breakup_vertices ........................................... 48
rnet_connected .................................................. 50
rnet_cycleway_intersection .................................... 51
rnet_get_nodes ................................................... 51
rnet_group ........................................................ 52
rnet_join ........................................................ 53
rnet_merge ........................................................ 55
rnet_overpass .................................................... 56
rnet_roundabout .................................................. 57
rnet_subset ....................................................... 57
route ............................................................. 58
routes_fast_sf .................................................... 59
routes_slow_sf .................................................... 60
route_average_gradient ....................................... 60
route_bikecitizens .............................................. 61
route_dodgr ....................................................... 62
route_google ...................................................... 63
route_nearest_point ............................................. 63
route_network_sf ................................................ 64
route_network_small .......................................... 64
route_osrm ....................................................... 65
angle_diff

Description

This function was designed to find lines that are close to parallel and perpendicular to some predefined route. It can return results that are absolute (contain information on the direction of turn, i.e. + or - values for clockwise/anticlockwise), bidirectional (which mean values greater than +/- 90 are impossible).

Usage

angle_diff(l, angle, bidirectional = FALSE, absolute = TRUE)
bbox_scale

Arguments

1. **angle**
   - A spatial lines object
   - an angle in degrees relative to North, with 90 being East and -90 being West. (direction of rotation is ignored).

2. **bidirectional**
   - Should the result be returned in a bidirectional format? Default is FALSE. If TRUE, the same line in the opposite direction would have the same bearing.

3. **absolute**
   - If TRUE (the default) only positive values can be returned.

Details

Building on the convention used in the `bearing()` function from the `geosphere` package and in many applications, North is defined as 0, East as 90 and West as -90.

See Also

Other lines: `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

Examples

```r
lib_versions <- sf::sf_extSoftVersion()
lib_versions
# fails on some systems (with early versions of PROJ)
if (lib_versions[3] >= "6.3.1") {
  # Find all routes going North-South
  lines_sf <- od2line(od_data_sample, zones = zones_sf)
  angle_diff(lines_sf[2, ], angle = 0)
  angle_diff(lines_sf[2:3, ], angle = 0)
}
```

bbox_scale

**Scale a bounding box**

Description

Takes a bounding box as an input and outputs a bounding box of a different size, centred at the same point.

Usage

```r
bbox_scale(bb, scale_factor)
```

Arguments

1. **bb**
   - Bounding box object

2. **scale_factor**
   - Numeric vector determining how much the bounding box will grow or shrink. Two numbers refer to extending the bounding box in x and y dimensions, respectively. If the value is 1, the output size will be the same as the input.
See Also

Other geo: `bind_sf()`, `geo_bb_matrix()` , `geo_bb()`, `geo_buffer()`, `geo_length()`, `geo_projected()`, `geo_select_aeq()`, `quadrant()`

Examples

```r
bb <- matrix(c(-1.55, 53.80, -1.50, 53.83), nrow = 2)
bb1 <- bbox_scale(bb, scale_factor = 1.05)
bb2 <- bbox_scale(bb, scale_factor = c(2, 1.05))
bb3 <- bbox_scale(bb, 0.1)
plot(x = bb2[1, ], y = bb2[2, ])
points(bb1[1, ], bb1[2, ])
points(bb3[1, ], bb3[2, ])
points(bb[1, ], bb[2, ], col = "red")
```

---

**bind_sf**

*Rapid row-binding of sf objects*

Description

Rapid row-binding of sf objects

Usage

```r
bind_sf(x)
```

Arguments

- `x` : List of sf objects to combine

Value

An sf data frame

See Also

Other geo: `bbox_scale()`, `geo_bb_matrix()`, `geo_bb()`, `geo_buffer()`, `geo_length()`, `geo_projected()`, `geo_select_aeq()`, `quadrant()`
cents_sf

| cents_sf | Spatial points representing home locations |

Description

These points represent population-weighted centroids of Medium Super Output Area (MSOA) zones within a 1 mile radius of my home when I was writing this package.

Format

A spatial dataset with 8 rows and 5 columns

Details

- geo_code the official code of the zone
- MSOA11NM name zone name
- percent_fem the percent female
- avslope average gradient of the zone

Cents was generated from the data repository pct-data: https://github.com/npct/pct-data. This data was accessed from within the pct repo: https://github.com/npct/pct, using the following code:

See Also

Other data: destinations_sf, flow_dests, flowlines_sf, flow, od_data_lines, od_data_routes, od_data_sample, osm_net_example, read_table_builder(), route_network_sf, route_network_small, routes_fast_sf, routes_slow_sf, zones_sf

Examples

cents_sf

destinations_sf

| destinations_sf | Example destinations data |

Description

This dataset represents trip destinations on a different geographic level than the origins stored in the object cents_sf.

Format

A spatial dataset with 87 features
See Also

Other data: `cents_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

Examples

destinations_sf

---

**flow**  Data frame of commuter flows

Description

This dataset represents commuter flows (work travel) between origin and destination zones. The data is from the UK and is available as open data: [https://wicid.ukdataservice.ac.uk/](https://wicid.ukdataservice.ac.uk/).

Format

A data frame with 49 rows and 15 columns

Details

The variables are as follows:

- Area.of.residence. id of origin zone
- Area.of.workplace id of destination zone
- All. Travel to work flows by all modes
- [,4:15]. Flows for different modes
- id. unique id of flow

Although these variable names are unique to UK data, the data structure is generalisable and typical of flow data from any source. The key variables are the origin and destination ids, which link to the georeferenced spatial objects.

See Also

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`
**flowlines_sf**

Spatial lines dataset of commuter flows

**Description**

Flow data after conversion to a spatial format.

**Format**

A spatial lines dataset with 49 rows and 15 columns

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

---

**flow_dests**

Data frame of invented commuter flows with destinations in a different layer than the origins

**Description**

Data frame of invented commuter flows with destinations in a different layer than the origins

**Usage**

```r
data(flow_dests)
```

**Format**

A data frame with 49 rows and 15 columns

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`
Examples

```r
## Not run:
# This is how the dataset was constructed
flow.dests <- flow
flow.dests$Area.of.workplace <- sample(x = destinations$WZ11CD, size = nrow(flow))
flow.dests <- dplyr::rename(flow.dests, WZ11CD = Area.of.workplace)
devtools::use_data(flow.dests)

## End(Not run)
```

---

### geo_bb

**Flexible function to generate bounding boxes**

#### Description

Takes a geographic object or bounding box as an input and outputs a bounding box, represented as a bounding box, corner points or rectangular polygon.

#### Usage

```r
geo_bb(
  shp,
  scale_factor = 1,
  distance = 0,
  output = c("polygon", "points", "bb")
)
```

#### Arguments

- **shp**: Spatial object
- **scale_factor**: Numeric vector determining how much the bounding box will grow or shrink. Two numbers refer to extending the bounding box in x and y dimensions, respectively. If the value is 1, the output size will be the same as the input.
- **distance**: Distance in metres to extend the bounding box by
- **output**: Type of object returned (polygon by default)

#### See Also

- `bb_scale`
- Other geo: `bbox_scale()`, `bind_sf()`, `geo_bb_matrix()`, `geo_buffer()`, `geo_length()`, `geo_projected()`, `geo_select_aeq()`, `quadrant()`
Examples

```r
shp <- routes_fast_sf
shp_bb <- geo_bb(shp, distance = 100)
plot(shp_bb, col = "red", reset = FALSE)
plot(geo_bb(routes_fast_sf, scale_factor = 0.8), col = "green", add = TRUE)
plot(routes_fast_sf$geometry, add = TRUE)
geo_bb(shp, output = "point")
```

**geo_bb_matrix**

*Create matrix representing the spatial bounds of an object*

**Description**

Converts a range of spatial data formats into a matrix representing the bounding box

**Usage**

```r
geo_bb_matrix(shp)
```

**Arguments**

- `shp`: Spatial object

**See Also**

Other geo: `bbox_scale()`, `bind_sf()`, `geo_bb()`, `geo_buffer()`, `geo_length()`, `geo_projected()`, `geo_select_aeq()`, `quadrant()`

**Examples**

```r
geo_bb_matrix(routes_fast_sf)
geo_bb_matrix(cents_sf[1, ])
geo_bb_matrix(c(-2, 54))
geo_bb_matrix(sf::st_coordinates(cents_sf))
```

**geo_buffer**

*Perform a buffer operation on a temporary projected CRS*

**Description**

This function solves the problem that buffers will not be circular when used on non-projected data.

**Usage**

```r
geo_buffer(shp, dist = NULL, width = NULL, ...)
```
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shp</td>
<td>A spatial object with a geographic CRS (e.g. WGS84) around which a buffer should be drawn</td>
</tr>
<tr>
<td>dist</td>
<td>The distance (in metres) of the buffer (when buffering simple features)</td>
</tr>
<tr>
<td>width</td>
<td>The distance (in metres) of the buffer (when buffering shp objects)</td>
</tr>
<tr>
<td>...</td>
<td>Arguments passed to the buffer (see <code>?sf::st_buffer</code> for details)</td>
</tr>
</tbody>
</table>

## Details

Requires recent version of PROJ (>= 6.3.0). Buffers on shp objects with geographic (lon/lat) coordinates can also be done with the s2 package.

## See Also

Other geo: `bbox_scale()`, `bind_sf()`, `geo_bb_matrix()`, `geo_bb()`, `geo_length()`, `geo_projected()`, `geo_select_aeq()`, `quadrant()`

## Examples

```r
lib_versions <- sf::sf_extSoftVersion()
lib_versions
if (lib_versions[3] >= "6.3.1") {
  buff_sf <- geo_buffer(routes_fast_sf, dist = 50)
  plot(buff_sf$geometry)
  geo_buffer(routes_fast_sf$geometry, dist = 50)
}
```

---

**geo_code**

Convert text strings into points on the map

## Description

Generate a lat/long pair from data using Google’s geolocation API.

## Usage

```r
geo_code(
  address,  # Text string to geocode
  service = "nominatim",  # Service to use (default: "nominatim")
  base_url = "https://maps.google.com/maps/api/geocode/json",  # Base URL for geocoding request
  return_all = FALSE,  # Whether to return all results (default: FALSE)
  pat = NULL  # Pattern for matching results (default: NULL)
)
```
Arguments

address Text string representing the address you want to geocode
service Which service to use? Nominatim by default
base_url The base url to query
return_all Should the request return all information returned by Google Maps? The default is FALSE: to return only two numbers: the longitude and latitude, in that order
pat Personal access token

Examples

## Not run:
geo_code(address = "Hereford")
geo_code("LS7 3HB")
geo_code("hereford", return_all = TRUE)
# needs api key in .Renviron
geo_code("hereford", service = "google", pat = Sys.getenv("GOOGLE"), return_all = TRUE)

## End(Not run)

geo_length Calculate line length of line with geographic or projected CRS

Description

Takes a line (represented in sf or sp classes) and returns a numeric value representing distance in meters.

Usage

geo_length(shp)

Arguments

shp A spatial line object

See Also

Other geo: bbox_scale(), bind_sf(), geo_bb_matrix(), geo_bb(), geo_buffer(), geo_projected(),
geo_select_aeq(), quadrant()

Examples

lib_versions <- sf::sf_extSoftVersion()
lib_versions
if (lib_versions[3] >= "6.3.1") {
  geo_length(routes_fast_sf)
}

geo_projected

*Perform GIS functions on a temporary, projected version of a spatial object*

**Description**

This function performs operations on projected data.

**Usage**

```r
geo_projected(shp, fun, crs, silent, ...)
```

**Arguments**

- `shp`: A spatial object with a geographic (WGS84) coordinate system
- `fun`: A function to perform on the projected object (e.g. from the sf package)
- `crs`: An optional coordinate reference system (if not provided it is set automatically by `geo_select_aeq()`)
- `silent`: A binary value for printing the CRS details (default: TRUE)
- `...`: Arguments to pass to `fun`

**See Also**

Other geo: `bbox_scale()`, `bind_sf()`, `geo_bb_matrix()`, `geo_bb()`, `geo_buffer()`, `geo_length()`,
`geo_select_aeq()`, `quadrant()`

**Examples**

```r
lib_versions <- sf::sf_extSoftVersion()
lib_versions
# fails on some systems (with early versions of PROJ)
if (lib_versions[3] >= "6.3.1") {
  shp <- routes_fast_sf[2:4,]
  geo_projected(shp, sf::st_buffer, dist = 100)
}
```

geo_select_aeq

*Select a custom projected CRS for the area of interest*

**Description**

This function takes a spatial object with a geographic (WGS84) CRS and returns a custom projected CRS focused on the centroid of the object. This function is especially useful for using units of metres in all directions for data collected anywhere in the world.
**geo_toptail**

Usage

geo_select_aeq(shp)

Arguments

shp A spatial object with a geographic (WGS84) coordinate system

Details

The function is based on this stackexchange answer: https://gis.stackexchange.com/questions/121489

See Also

Other geo: `bbox_scale()`, `bind_sf()`, `geo_bb_matrix()`, `geo_bb()`, `geo_buffer()`, `geo_length()`, `geo_projected()`, `quadrant()`

Examples

```r
shp <- zones_sf
geo_select_aeq(shp)
```

---

**geo_toptail**

*Clip the first and last n metres of SpatialLines*

Description

Takes lines and removes the start and end point, to a distance determined by the user.

Usage

geo_toptail(l, toptail_dist, ...)

Arguments

l An sf object representing lines
toptail_dist The distance (in metres) to top and tail the line by. Can either be a single value or a vector of the same length as the SpatialLines object.
...
Arguments passed to sf::st_buffer()

Details

Note: see the function `toptailgs()` in stplanr v0.8.5 for an implementation that uses the geosphere package.
gsection

Function to split overlapping SpatialLines into segments

description

Divides SpatialLinesDataFrame objects into separate Lines. Each new Lines object is the aggregate of a single number of aggregated lines.

Usage

gsection(sl, buff_dist = 0)

Arguments

sl SpatialLinesDataFrame with overlapping Lines to split by number of overlapping features.

buff_dist A number specifying the distance in meters of the buffer to be used to crop lines before running the operation. If the distance is zero (the default) touching but non-overlapping lines may be aggregated.

See Also

Other rnet: islines(), overline(), rnet_breakup_vertices(), rnet_group()
Examples

```r
lib_versions <- sf::sf_extSoftVersion()
lib_versions
# fails on some systems (with early versions of PROJ)
if (lib_versions[3] >= "6.3.1") {
  sl <- routes_fast_sf[2:4, ]
  rsec <- gsection(sl)
  length(rsec) # sections
  plot(rsec, col = seq(length(rsec)))
  rsec <- gsection(sl, buff_dist = 50)
  length(rsec) # 4 features: issue
  plot(rsec, col = seq(length(rsec)))
}
```

islines

Do the intersections between two geometries create lines?

Description

This is a function required in `overline()`. It identifies whether sets of lines overlap (beyond shared points) or not.

Usage

```r
islines(g1, g2)
```

Arguments

- `g1`: A spatial object
- `g2`: A spatial object

See Also

Other rnet: `gsection()`, `overline()`, `rnet_breakup_vertices()`, `rnet_group()`

Examples

```r
## Not run:
# sf implementation
islines(routes_fast_sf[2, ], routes_fast_sf[3, ])
islines(routes_fast_sf[2, ], routes_fast_sf[22, ])

## End(Not run)
```
is_linepoint  Identify lines that are points

Description

OD matrices often contain 'intrazonal' flows, where the origin is the same point as the destination. This function can help identify such intrazonal OD pairs, using 2 criteria: the total number of vertices (2 or fewer) and whether the origin and destination are the same.

Usage

is_linepoint(l)

Arguments

l  A spatial lines object

Details

Returns a boolean vector. TRUE means that the associated line is in fact a point (has no distance). This can be useful for removing data that will not be plotted.

See Also

Other lines: `angle_diff()`, `geo_toptail()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

Examples

islp <- is_linepoint(flowlines_sf)
nrow(flowlines_sf)
sum(islp)
# Remove invisible 'linepoints'
nrow(flowlines_sf[!islp, ])

line2df  Convert geographic line objects to a data.frame with from and to coordinates

Description

This function returns a data frame with fx and fy and tx and ty variables representing the beginning and end points of spatial line features respectively.
Usage

```r
line2df(l)
```

Arguments

- `l` A spatial lines object

See Also

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

Examples

```r
line2df(routes_fast_sf[5:6,]) # beginning and end of routes
```

---

**line2points**

*Convert a spatial (linestring) object to points*

Description

The number of points will be double the number of lines with `line2points`. A closely related function, `line2pointsn` returns all the points that were line vertices. The points corresponding with a given line, i, will be `(2*i):((2*i)+1)`. The last function, `line2vertices`, returns all the points that are vertices but not nodes. If the input `l` object is composed by only 1 LINESTRING with 2 POINTS, then it returns an empty `sf` object.

Usage

```r
line2points(l, ids = rep(1:nrow(l)))
line2pointsn(l)
line2vertices(l)
```

Arguments

- `l` An `sf` object or a `SpatialLinesDataFrame` from the older `sp` package
- `ids` Vector of ids (by default `1:nrow(l)`)

See Also

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`
Examples

```r
l <- routes_fast_sf[2,]
lpoints <- line2points(l)
plot(l$geometry)
plot(lpoints, add = TRUE)
# test all vertices:
plot(l$geometry)
lpoints2 <- line2pointsn(l)
plot(lpoints2$geometry, add = TRUE)

# extract only internal vertices
l_internal_vertices <- line2vertices(l)
plot(sf::st_geometry(l), reset = FALSE)
plot(l_internal_vertices, add = TRUE)

# The boundary points are missing
```

---

`line_bearing`  
*Find the bearing of straight lines*

**Description**

This function returns the bearing (in degrees relative to north) of lines.

**Usage**

```r
line_bearing(l, bidirectional = FALSE)
```

**Arguments**

- `l`  
  A spatial lines object

- `bidirectional`  
  Should the result be returned in a bidirectional format? Default is FALSE. If TRUE, the same line in the opposite direction would have the same bearing

**Details**

Returns a boolean vector. TRUE means that the associated line is in fact a point (has no distance). This can be useful for removing data that will not be plotted.

**See Also**

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`
Examples

```
lib_versions <- sf::sf_extSoftVersion()
lib_versions
# fails on some systems (with early versions of PROJ)
if (lib_versions[3] >= "6.3.1") {
  bearings_sf_1_9 <- line_bearing(flowlines_sf[1:5, ])
  bearings_sf_1_9 # lines of 0 length have NaN bearing
  line_bearing(flowlines_sf[1:5, ], bidirectional = TRUE)
}
```

Description

This function breaks up a LINESTRING geometries into smaller pieces.

Usage

```
line_breakup(l, z)
```

Arguments

- `l`: An sf object with LINESTRING geometry
- `z`: An sf object with POLYGON geometry or a number representing the resolution of grid cells used to break up the linestring objects

Value

An sf object with LINESTRING geometry created after breaking up the input object.

See Also

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

Examples

```
library(sf)
z <- zones_sf$geometry
l <- routes_fast_sf$geometry[2]
l_split <- line_breakup(l, z)
l
l_split
sf::st_length(l)
sum(sf::st_length(l_split))
plot(z)
plot(l, add = TRUE, lwd = 9, col = "grey")
plot(l_split, add = TRUE, col = 1:length(l_split))
```
**line_cast**  
Convert multilinestring object into linestrings

**Description**
Without losing vertices

**Usage**
```r
line_cast(x)
```

**Arguments**
- `x` : Linestring object

---

**line_midpoint**  
Find the mid-point of lines

**Description**
Find the mid-point of lines

**Usage**
```r
line_midpoint(l, tolerance = NULL)
```

**Arguments**
- `l` : A spatial lines object
- `tolerance` : The tolerance used to break lines at vertices. See `lwgeom::st_linesubstring()`.

**See Also**
Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line-bearing()`, `line_breakup()`, `line_segment()`, `line_via()`, `mats2line()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

**Examples**
```r
l = routes_fast_sf[2:5, ]
plot(l$geometry, col = 2:5)
midpoints = line_midpoint(l)
plot(midpoints, add = TRUE)
```
line_segment

Divide an sf object with LINestring geometry into regular segments

Description

This function keeps the attributes

Usage

\[
\text{line_segment}(l, \ n\_segments = NA, \ segment\_length = NA)
\]

Arguments

\(l\)  
A spatial lines object

\(n\_segments\)  
The number of segments to divide the line into

\(segment\_length\)  
The approximate length of segments in the output (overrides \(n\_segments\) if set)

See Also

Other lines: \(\text{angle}\_diff()\), \(\text{geo}\_toptail()\), \(\text{is}\_linepoint()\), \(\text{line}\_2df()\), \(\text{line}\_2points()\), \(\text{line}\_bearing()\), \(\text{line}\_breakup()\), \(\text{line}\_midpoint()\), \(\text{line}\_via()\), \(\text{mats}\_2line()\), \(\text{n}\_vertices()\), \(\text{onewaygeo()\), \(\text{points}\_2line()\), \(\text{toptail\_buff()\)

Examples

\[
l \leftarrow \text{routes}\_fast\_sf[2, ]
\]
\[
l\_seg2 \leftarrow \text{line}\_segment}(l = 1, \ n\_segments = 2)
\]
\[
l\_seg3 \leftarrow \text{line}\_segment}(l = 1, \ n\_segments = 3)
\]
\[
l\_seg\_100 \leftarrow \text{line}\_segment}(l = 1, \ \text{segment}\_length = 100)
\]
\[
l\_seg\_2000 \leftarrow \text{line}\_segment}(l = 1, \ \text{segment}\_length = 2000)
\]
\[
\text{plot}(\text{sf}:\text{st}\_geometry(l\_seg2), \ \text{col} = 1:2, \ \text{lwd} = 5)
\]
\[
\text{plot}(\text{sf}:\text{st}\_geometry(l\_seg3), \ \text{col} = 1:3, \ \text{lwd} = 5)
\]
\[
\text{plot}(\text{sf}:\text{st}\_geometry(l\_seg\_100), \ \text{col} = \text{seq}(\text{nrow}(l\_seg\_100)), \ \text{lwd} = 5)
\]
\[
\text{plot}(\text{sf}:\text{st}\_geometry(l\_seg\_2000), \ \text{col} = \text{seq}(\text{nrow}(l\_seg\_100)), \ \text{lwd} = 5)
\]

# Multiple lines
\[
l \leftarrow \text{routes}\_fast\_sf[2:4, ]
\]
\[
l\_seg\_multi = \text{line}\_segment}(l, \ \text{segment}\_length = 1000)
\]
\[
\text{plot}(\text{sf}:\text{st}\_geometry(l\_seg\_multi), \ \text{col} = \text{seq}(\text{nrow}(l\_seg\_100)), \ \text{lwd} = 5)
line_via

Add geometry columns representing a route via intermediary points

Description
Takes an origin (A) and destination (B), represented by the linestring l, and generates 3 extra geometries based on points p:

Usage
line_via(l, p)

Arguments
l A spatial lines object
p A spatial points object

Details
1. From A to P1 (P1 being the nearest point to A)
2. From P1 to P2 (P2 being the nearest point to B)
3. From P2 to B

See Also
Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), line2points(), line_bearing(), line_breakup(), line_midpoint(), line_segment(), mats2line(), n_vertices(), onewaygeo(), points2line(), toptail_buff()

Examples
library(sf)
l <- flowlines_sf[2:4,]
p <- destinations_sf
lv <- line_via(l, p)
lv
# library(mapview)
# mapview(lv) +
#   mapview(lv$leg_orig, col = "red")
plot(lv[3], lwd = 9, reset = FALSE)
plot(lv$leg_orig, col = "red", lwd = 5, add = TRUE)
plot(lv$leg_via, col = "black", add = TRUE)
plot(lv$leg_dest, col = "green", lwd = 5, add = TRUE)
**mats2line**

*Convert 2 matrices to lines*

**Description**

Convert 2 matrices to lines

**Usage**

```r
mats2line(mat1, mat2, crs = NA)
```

**Arguments**

- `mat1`: Matrix representing origins
- `mat2`: Matrix representing destinations
- `crs`: Number representing the coordinate system of the data, e.g. 4326

**See Also**

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`

**Examples**

```r
m1 <- matrix(c(1, 2, 1, 2), ncol = 2)
m2 <- matrix(c(9, 9, 9, 1), ncol = 2)
l <- mats2line(m1, m2)
class(l)
l
lsf <- sf::st_sf(l, crs = 4326)
class(lsf)
plot(lsf)
# mapview::mapview(lsf)
```

---

**n_vertices**

*Retrieve the number of vertices in sf objects*

**Description**

Returns a vector of the same length as the number of sf objects.

**Usage**

```r
n_vertices(l)
```
Arguments

1. An sf object with LINestring geometry

See Also

Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), linepoints(), line_bearing(), line_breakup(), line_midpoint(), line_segment(), line_via(), mats2line(), onewaygeo(), points2line(), toptail_buff()

Examples

```r
l = routes_fast_sf
n_vertices(l)
```

```r
n_vertices(zones_sf)
```

---

**od2line**  
Convert origin-destination data to spatial lines

**Description**

Origin-destination (‘OD’) flow data is often provided in the form of 1 line per flow with zone codes of origin and destination centroids. This can be tricky to plot and link-up with geographical data. This function makes the task easier.

**Usage**

```r
od2line(
  flow,  
  zones, 
  destinations = NULL, 
  zone_code = names(zones)[1], 
  origin_code = names(flow)[1], 
  dest_code = names(flow)[2], 
  zone_code_d = NA, 
  silent = FALSE
)
```

**Arguments**

- `flow`: A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the `zones`. Thus in `cents_sf()`, the first column is `geo_code`. This corresponds to the first two columns of `flow()`.
- `zones`: A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.
- `destinations`: A spatial object representing destinations of travel flows.
zone_code  Name of the variable in zones containing the ids of the zone. By default this is the first column names in the zones.

origin_code  Name of the variable in flow containing the ids of the zone of origin. By default this is the first column name in the flow input dataset.

dest_code  Name of the variable in flow containing the ids of the zone of destination. By default this is the second column name in the flow input dataset or the first column name in the destinations if that is set.

zone_code_d  Name of the variable in destinations containing the ids of the zone. By default this is the first column names in the destinations.

silent  TRUE by default, setting it to TRUE will show you the matching columns

Details
Origin-destination (OD) data is often provided in the form of 1 line per OD pair, with zone codes of the trip origin in the first column and the zone codes of the destination in the second column (see the vignette("stplanr-od")) for details. od2line() creates a spatial (linestring) object representing movement from the origin to the destination for each OD pair. It takes data frame containing origin and destination cones (flow) that match the first column in a a spatial (polygon or point) object (zones).

See Also
Other od: od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_id_order(), od_id, od_oneway(), od_to_odmatrix(), odmatrix_to_od(), points2flow(), points2odf()

Examples
od_data <- stplanr::flow[1:20, ]
l <- od2line(flow = od_data, zones = cents_sf)
plot(sf::st_geometry(cents_sf))
plot(l, lwd = l$All / mean(l$All), add = TRUE)
Arguments

flow  A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in `cents_sf()`, the first column is `geo_code`. This corresponds to the first two columns of `flow()`.

zones  A spatial object representing origins (and destinations if no separate destinations object is provided) of travel.

Details

Origin-destination (OD) data is often provided in the form of 1 line per OD pair, with zone codes of the trip origin in the first column and the zone codes of the destination in the second column (see the vignette("stplanr-od")) for details. `od2odf()` creates an 'origin-destination data frame', with columns containing origin and destination codes (flow) that match the first column in a a spatial (polygon or point `sf`) object (zones). The function returns a data frame with coordinates for the origin and destination.

See Also

Other od: `od2line()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_id_order()`, `od_id_od_oneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

Examples

```r
od2odf(flow[1:2, ], zones_sf)
```

---

### odmatrix_to_od

Convert origin-destination data from wide to long format

Description

This function takes a matrix representing travel between origins (with origin codes in the rownames of the matrix) and destinations (with destination codes in the colnames of the matrix) and returns a data frame representing origin-destination pairs.

Usage

```r
odmatrix_to_od(odmatrix)
```

Arguments

- `odmatrix` A matrix with row and columns representing origin and destination zone codes and cells representing the flow between these zones.
od_aggregate_from

Details

The function returns a data frame with rows ordered by origin and then destination zone code values and with names orig, dest and flow.

See Also

Other od: od2line(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_id_order(), od_id_od_oneway(), od_to_odmatrix(), points2flow(), points2odf()

Examples

odmatrix <- od_to_odmatrix(flow)
odmatrix_to_od(odmatrix)
flow[1:9, 1:3]
odmatrix_to_od(od_to_odmatrix(flow[1:9, 1:3]))

od_aggregate_from

Summary statistics of trips originating from zones in OD data

Description

This function takes a data frame of OD data and returns a data frame reporting summary statistics for each unique zone of origin.

Usage

od_aggregate_from(flow, attrib = NULL, FUN = sum, ..., col = 1)

Arguments

flow A data frame representing origin-destination data. The first two columns of this data frame should correspond to the first column of the data in the zones. Thus in cents_sf(), the first column is geo_code. This corresponds to the first two columns of flow().

attrib character, column names in sl to be aggregated

FUN A function to summarise OD data by

... Additional arguments passed to FUN

col The column that the OD dataset is grouped by (1 by default, the first column usually represents the origin)

Details

It has some default settings: the default summary statistic is sum() and the first column in the OD data is assumed to represent the zone of origin. By default, if attrib is not set, it summarises all numeric columns.
See Also

Other od: \texttt{od2line()}, \texttt{od2odf()}, \texttt{od_aggregate_from()}, \texttt{od_coords2line()}, \texttt{od_coords()}, \texttt{od_id_order()}, \texttt{od_id}, \texttt{od_oneway()}, \texttt{od_to_odmatrix()}, \texttt{odmatrix_to_od()}, \texttt{points2flow()}, \texttt{points2odf()}

Examples

\texttt{od_aggregate_from(flow)}
Create matrices representing origin-destination coordinates

**Description**

This function takes a wide range of input data types (spatial lines, points or text strings) and returns a matrix of coordinates representing origin (fx, fy) and destination (tx, ty) points.

**Usage**

```r
od_coords(from = NULL, to = NULL, l = NULL)
```

**Arguments**

- **from**: An object representing origins (if lines are provided as the first argument, from is assigned to `l`)
- **to**: An object representing destinations
- **l**: Only needed if from and to are empty, in which case this should be a spatial object representing desire lines

**See Also**

Other od: `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_id_order()`, `od_id`, `od_oneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

**Examples**

```r
od_coords(from = c(0, 52), to = c(1, 53)) # lon/lat coordinates
od_coords(cents_sf[1:3, ], cents_sf[2:4, ]) # sf points
# od_coords("Hereford", "Leeds") # geocode locations
od_coords(flowlines_sf[1:3, ])
```

Convert origin-destination coordinates into desire lines

**Description**

Convert origin-destination coordinates into desire lines

**Usage**

```r
od_coords2line(odc, crs = 4326, remove_duplicates = TRUE)
```
Arguments

- `odc` A data frame or matrix representing the coordinates of origin-destination data. The first two columns represent the coordinates of the origin (typically longitude and latitude) points; the third and fourth columns represent the coordinates of the destination (in the same CRS). Each row represents travel from origin to destination.
- `crs` A number representing the coordinate reference system of the result, 4326 by default.
- `remove_duplicates` Should rows with duplicated rows be removed? TRUE by default.

See Also

Other od: `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords()`, `od_id_order()`, `od_id.od_oneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

Examples

```r
odf <- od_coords(l = flowlines_sf)
odlines <- od_coords2line(odf)
odlines <- od_coords2line(odf, crs = 4326)
plot(odlines)
x_coords <- 1:3
n <- 50
d <- data.frame(lapply(1:4, function(x) sample(x_coords, n, replace = TRUE)))
names(d) <- c("fx", "fy", "tx", "ty")
l <- od_coords2line(d)
plot(l)
nrow(l)
l_with_duplicates <- od_coords2line(d, remove_duplicates = FALSE)
plot(l_with_duplicates)
nrow(l_with_duplicates)
```

---

**od_data_lines**

Example of desire line representations of origin-destination data from UK Census

Description

Derived from `od_data_sample` showing movement between points represented in `cents_sf`

Format

A data frame (tibble) object
See Also

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

Examples

```
od_data_lines
```

---

### `od_data_routes`

**Example segment-level route data**

**Description**

See `data-raw/generate-data.Rmd` for details on how this was created. The dataset shows routes between origins and destinations represented in `od_data_lines`.

**Format**

A data frame (tibble) object

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

**Examples**

```
od_data_routes
```

---

### `od_data_sample`

**Example of origin-destination data from UK Census**

**Description**

See `data-raw/generate-data.Rmd` for details on how this was created.

**Format**

A data frame (tibble) object

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_routes`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`
od_id

Combine two ID values to create a single ID number

Description

Combine two ID values to create a single ID number

Usage

od_id_szudzik(x, y, ordermatters = FALSE)

don_id_max_min(x, y)

don_id_character(x, y)

Arguments

x a vector of numeric, character, or factor values

y a vector of numeric, character, or factor values

ordermatters logical, does the order of values matter to pairing, default = FALSE

Details

In OD data it is common to have many 'oneway' flows from "A to B" and "B to A". It can be useful to group these and have a single ID that represents pairs of IDs with or without directionality, so they contain 'twoway' or bi-directional values.

don_id* functions take two vectors of equal length and return a vector of IDs, which are unique for each combination but the same for twoway flows.

- the Szudzik pairing function, on two vectors of equal length. It returns a vector of ID numbers.

This function superseeds od_id_order as it is faster on large datasets

See Also

Other od: od2line(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_id_order(), od_oneway(), od_to_odmatrix(), odmatrix_to_od(), points2flow(), points2odf()

Examples

\[(d <- od_data_sample[2:9, 1:2])\]
\[(id <- od_id_character(d[[1]], d[[2]]))\]
\[duplicated(id)\]
\[od_id_szudzik(d[[1]], d[[2]])\]
\[od_id_max_min(d[[1]], d[[2]])\]
**od_id_order**

Generate ordered ids of OD pairs so lowest is always first. This function is slow on large datasets, see szudzik_pairing for faster alternative.

Usage

```r
od_id_order(x, id1 = names(x)[1], id2 = names(x)[2])
```

Arguments

- `x`: A data frame or SpatialLinesDataFrame, representing an OD matrix.
- `id1`: Optional (it is assumed to be the first column) text string referring to the name of the variable containing the unique id of the origin.
- `id2`: Optional (it is assumed to be the second column) text string referring to the name of the variable containing the unique id of the destination.

See Also

Other od: `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_id`, `od_oneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

Examples

```r
x <- data.frame(id1 = c(1, 1, 2, 2, 3), id2 = c(1, 2, 3, 1, 4))
od_id_order(x) # 4th line switches id1 and id2 so stplanr.key is in order
```

**od_oneway**

Aggregate od pairs they become non-directional.

Description

For example, sum total travel in both directions.
Usage

\[
\text{od_oneway}\left(x, \text{attrib} = \text{names}(x[-c(1:2)])[\text{vapply}(x[-c(1:2)], \text{is.numeric}, \text{TRUE})], \text{id1} = \text{names}(x)[1], \text{id2} = \text{names}(x)[2], \text{stplanr.key} = \text{NULL}\right)
\]

Arguments

- **x** A data frame or SpatialLinesDataFrame, representing an OD matrix
- **attrib** A vector of column numbers or names, representing variables to be aggregated. By default, all numeric variables are selected. aggregate
- **id1** Optional (it is assumed to be the first column) text string referring to the name of the variable containing the unique id of the origin
- **id2** Optional (it is assumed to be the second column) text string referring to the name of the variable containing the unique id of the destination
- **stplanr.key** Optional key of unique OD pairs regardless of the order, e.g., as generated by \text{od_id_max_min()} or \text{od_id_szudzik()}

Details

Flow data often contains movement in two directions: from point A to point B and then from B to A. This can be problematic for transport planning, because the magnitude of flow along a route can be masked by flows the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will by heavily under-estimated for OD pairs which have similar amounts of travel in both directions. Flows in both direction are often represented by overlapping lines with identical geometries which can be confusing for users and are difficult to plot.

Value

\text{oneway} outputs a data frame (or \text{sf} data frame) with rows containing results for the user-selected attribute values that have been aggregated.

See Also

Other \text{od}: \text{od2line()}, \text{od2odf()}, \text{od_aggregate_from()}, \text{od_aggregate_to()}, \text{od_coords2line()}, \text{od_coords()}, \text{od_id_order()}, \text{od_id, od_to_odmatrix()}, \text{odmatrix_to_od()}, \text{points2flow()}, \text{points2odf()}

Examples

\[
\begin{align*}
\text{(od_min} & \leftarrow \text{od_data_sample}[c(1, 2, 9), 1:6]) \\
\text{(od_oneway} & \leftarrow \text{od_oneway(od_min)}) \\
& \text{# (od_oneway_old} = \text{onewayid(od_min, attrib = 3:6))} \text{# old implementation} \\
& \text{nrow(od_oneway)} < \text{nrow(od_min)} \text{# result has fewer rows}
\end{align*}
\]
od_to_odmatrix

Convert origin-destination data from long to wide format

Description
This function takes a data frame representing travel between origins (with origin codes in name_orig, typically the 1st column) and destinations (with destination codes in name_dest, typically the second column) and returns a matrix with cell values (from attrib, the third column by default) representing travel between origins and destinations.

Usage
od_to_odmatrix(flow, attrib = 3, name_orig = 1, name_dest = 2)

Arguments
flow A data frame representing flows between origin and destinations
attrib A number or character string representing the column containing the attribute data of interest from the flow data frame
name_orig A number or character string representing the zone of origin
name_dest A number or character string representing the zone of destination

See Also
Other od: od2line(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_id_order(), od_id, od_oneway(), odmatrix_to_od(), points2flow(), points2odf()

Examples
od_to_odmatrix(flow)
od_to_odmatrix(flow[1:9, ])
od_to_odmatrix(flow[1:9, ], attrib = "Bicycle")
onewaygeo

Aggregate flows so they become non-directional (by geometry - the slow way)

Description

Flow data often contains movement in two directions: from point A to point B and then from B to A. This can be problematic for transport planning, because the magnitude of flow along a route can be masked by flows the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will by heavily under-estimated for OD pairs which have similar amounts of travel in both directions.

Usage

onewaygeo(x, attrib)

Arguments

x 
A dataset containing linestring geometries

attrib 
A text string containing the name of the line’s attribute to aggregate or a numeric vector of the columns to be aggregated

Details

This function aggregates directional flows into non-directional flows, potentially halving the number of lines objects and reducing the number of overlapping lines to zero.

Value

onewaygeo outputs a SpatialLinesDataFrame with single lines and user-selected attribute values that have been aggregated. Only lines with a distance (i.e. not intra-zone flows) are included

See Also

Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), line2points(), line_bearing(), line_breakup(), line_midpoint(), line_segment(), line_via(), mats2line(), n_vertices(), points2line(), toptail_buff()
**osm_net_example**

---

**Example of OpenStreetMap road network**

---

**Description**

Example of OpenStreetMap road network

**Format**

An sf object

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

**Examples**

osm_net_example

---

**overline**

Convert series of overlapping lines into a route network

---

**Description**

This function takes a series of overlapping lines and converts them into a single route network.

This function is intended as a replacement for overline() and is significantly faster especially on large datasets. However, it also uses more memory.

**Usage**

```r
overline(
    sl,
    attrib,
    ncores = 1,
    simplify = TRUE,
    regionalise = 1e+09,
    quiet = ifelse(nrow(sl) < 1000, TRUE, FALSE),
    fun = sum
)
```

```r
overline2(
    sl,
    attrib,
```
```r
ncores = 1,
simplify = TRUE,
regionalise = 1e+07,
quiet = ifelse(nrow(sl) < 1000, TRUE, FALSE),
fun = sum
)
```

### Arguments

- **sl**  
  A spatial object representing routes on a transport network

- **attrib**  
  character, column names in sl to be aggregated

- **ncores**  
  integer, how many cores to use in parallel processing, default = 1

- **simplify**  
  logical, if TRUE group final segments back into lines, default = TRUE

- **regionalise**  
  integer, during simplification regionalisation is used if the number of segments exceeds this value

- **quiet**  
  Should the the function omit messages? NULL by default, which means the output will only be shown if sl has more than 1000 rows.

- **fun**  
  Named list of functions to summaries the attributes by? sum is the default. list(sum = sum, average = mean) will summarise all attributes by sum and mean.

### Details

The function can be used to estimate the amount of transport 'flow' at the route segment level based on input datasets from routing services, for example linestring geometries created with the route() function.

The `overline()` function breaks each line into many straight segments and then looks for duplicated segments. Attributes are summed for all duplicated segments, and if simplify is TRUE the segments with identical attributes are recombined into linestrings.

The following arguments only apply to the sf implementation of `overline()`:

- **ncores**, the number of cores to use in parallel processing
- **simplify**, should the final segments be converted back into longer lines? The default setting is TRUE. simplify = FALSE results in straight line segments consisting of only 2 vertices (the start and end point), resulting in a data frame with many more rows than the simplified results (see examples).
- **regionalise** the threshold number of rows above which regionalisation is used (see details).

For sf objects Regionalisation breaks the dataset into a 10 x 10 grid and then performed the simplification across each grid. This significantly reduces computation time for large datasets, but slightly increases the final file size. For smaller datasets it increases computation time slightly but reduces memory usage and so may also be useful.

A known limitation of this method is that overlapping segments of different lengths are not aggregated. This can occur when lines stop halfway down a road. Typically these errors are small, but some artefacts may remain within the resulting data.
For very large datasets nrow(x) > 1000000, memory usage can be significant. In these cases it is possible to overline subsets of the dataset, rbind the results together, and then overline again, to produce a final result.

Multicore support is only enabled for the regionalised simplification stage as it does not help with other stages.

Value

An sf object representing a route network

Author(s)

Barry Rowlingson
Malcolm Morgan

References


See Also

Other rnet: gsection(), islines(), rnet_breakup_vertices(), rnet_group()

Examples

```r
sl <- routes_fast_sf[2:4, ]
sl$All <- flowlines_sf$All[2:4]
sl <- overline(sl = sl, attrib = "All")
nrow(sl)
nrow(rnet)
plot(rnet)

rnet_mean <- overline(sl, c("All", "av_incline"), fun = list(mean = mean, sum = sum))
plot(rnet_mean, lwd = rnet_mean$All_sum / mean(rnet_mean$All_sum))

rnet_sf_raw <- overline(sl, attrib = "length", simplify = FALSE)
nrow(rnet_sf_raw)
summary(n_vertices(rnet_sf_raw))
plot(rnet_sf_raw)

rnet_sf_raw$n <- 1:nrow(rnet_sf_raw)
plot(rnet_sf_raw[10:25, ])
```
overline_intersection  

Convert series of overlapping lines into a route network

Description

This function takes overlapping LINESTRINGs stored in an sf object and returns a route network composed of non-overlapping geometries and aggregated values.

Usage

overline_intersection(sl, attrib, fun = sum)

Arguments

sl  
An sf LINESTRING object with overlapping elements

attrib  
character, column names in sl to be aggregated

fun  
Named list of functions to summaries the attributes by?  sum is the default.  list(sum = sum, average = mean) will summarise all attributes by sum and mean.

Examples

routes_fast_sf$value <- 1
sl <- routes_fast_sf[4:6, ]
 attrib <- c("value", "length")
 rnet <- overline_intersection(sl = sl, attrib)
 plot(rnet, lwd = rnet$value)
 # A larger example
 sl <- routes_fast_sf[4:7, ]
 rnet <- overline_intersection(sl = sl, attrib = c("value", "length"))
 plot(rnet, lwd = rnet$value)
 rnet_sf <- overline(routes_fast_sf[4:7, ], attrib = c("value", "length"))
 plot(rnet_sf, lwd = rnet_sf$value)
 # An even larger example (not shown, takes time to run)
 # rnet = overline_intersection(routes_fast_sf, attrib = c("value", "length"))
 # rnet_sf <- overline(routes_fast_sf, attrib = c("value", "length"), buff_dist = 10)
 # plot(rnet$geometry, lwd = rnet$value * 2, col = "grey")
 # plot(rnet_sf$geometry, lwd = rnet_sf$value, add = TRUE)
points2flow

Convert a series of points into geographical flows

Description
Takes a series of geographical points and converts them into a spatial (linestring) object representing the potential flows, or 'spatial interaction', between every combination of points.

Usage
points2flow(p)

Arguments
p A spatial (point) object

See Also
Other od: od2line(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_id_order(), od_id, od_oneway(), od_to_odmatrix(), odmatrix_to_od(), points2odf()

Examples
flow_sf <- points2flow(cents_sf[1:4, ])
plot(flow_sf)

points2line

Convert a series of points, or a matrix of coordinates, into a line

Description
This function makes that makes the creation of sf objects with LINESTRING geometries easy.

Usage
points2line(p)

Arguments
p A spatial (points) object or matrix representing the coordinates of points.

See Also
Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), line2points(), line_bearing(), line_breakup(), line_midpoint(), line_segment(), line_via(), mats2line(), n_vertices(), onewaygeo(), toptail_buff()
**Examples**

```r
l_sf <- points2line(cents_sf)
plot(l_sf)
```

**points2odf**

*Convert a series of points into a dataframe of origins and destinations*

**Description**

Takes a series of geographical points and converts them into a dataframe representing the potential flows, or 'spatial interaction', between every combination of points.

**Usage**

```r
points2odf(p)
```

**Arguments**

- `p`  
  A spatial points object

**See Also**

Other od: `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_id_order()`, `od_id_odoneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`

**Examples**

```r
points2odf(cents_sf)
```

**quadrant**

*Split a spatial object into quadrants*

**Description**

Returns a character vector of NE, SE, SW, NW corresponding to north-east, south-east quadrants respectively. If number_out is TRUE, returns numbers from 1:4, respectively.

**Usage**

```r
quadrant(x, cent = NULL, number_out = FALSE)
```
Arguments

x Object of class sf

cent The centrepoint of the region of interest. Quadrants will be defined based on this point. By default this will be the geographic centroid of the zones.

number_out Should the result be returned as a number?

See Also

Other geo: bbox_scale(), bind_sf(), geo_bb_matrix(), geo_bb(), geo_buffer(), geo_length(), geo_projected(), geo_select_aeq()

Examples

x = zones_sf
(quads <- quadrant(x))
plot(x$geometry, col = factor(quads))
Details

The Australian Bureau of Statistics (ABS) provides customised tables for census and other datasets in a format that is difficult to use in R because it contains rows with additional information. This function imports the original (unzipped) TableBuilder files in .csv or .xlsx format before creating an R dataframe with the data.

Note: we recommend using the readabs package for this purpose.

See Also

Other data: cents_sf, destinations_sf, flow_dests, flowlines_sf, flow, od_data_lines, od_data_routes, od_data_sample, osm_net_example, route_network_sf, route_network_small, routes_fast_sf, routes_slow_sf, zones_sf

---

rnet_add_node  Add a node to route network

Description

Add a node to route network

Usage

rnet_add_node(rnet, p)

Arguments

rnet  A route network of the type generated by overline()
p  A point represented by an sf object the will split the route

Examples

sample_routes <- routes_fast_sf[2:6, NULL]
sample_routes$value <- rep(1:3, length.out = 5)
rnet <- overline2(sample_routes, attrib = "value")
p <- sf::st_sfc(sf::st_point(c(-1.540, 53.826)), crs = sf::st_crs(rnet))
r_split <- route_split(rnet, p)
plot(rnet$geometry, lwd = rnet$value * 5, col = "grey")
plot(p, cex = 9, add = TRUE)
plot(r_split, col = 1:nrow(r_split), add = TRUE, lwd = r_split$value)
rnet_boundary_points

Get points at the beginner and end of linestrings

**Description**

Get points at the beginner and end of linestrings

**Usage**

```r
rnet_boundary_points(rnet)
```

```r
rnet_boundary_df(rnet)
```

```r
rnet_boundary_unique(rnet)
```

```r
rnet_boundary_points_lwgeom(rnet)
```

```r
rnet_duplicated_vertices(rnet, n = 2)
```

**Arguments**

- `rnet`  
  An sf or sfc object with LINESTRING geometry representing a route network.

- `n`  
  The minimum number of times a vertex must be duplicated to be returned

**Examples**

```r
has_sfheaders <- requireNamespace("sfheaders", quietly = TRUE)
if(has_sfheaders) {
  rnet <- rnet_roundabout
  bp1 <- rnet_boundary_points(rnet)
  bp2 <- line2points(rnet) # slower version with lwgeom
  bp3 <- rnet_boundary_points_lwgeom(rnet) # slower version with lwgeom
  bp4 <- rnet_boundary_unique(rnet)
  nrow(bp1)
  nrow(bp3)
  identical(sort(sf::st_coordinates(bp1)), sort(sf::st_coordinates(bp2)))
  identical(sort(sf::st_coordinates(bp3)), sort(sf::st_coordinates(bp4)))
  plot(rnet$geometry)
  plot(bp3, add = TRUE)
}
```
rnet_breakup_vertices  
*Break up an sf object with LINestring geometry.*

**Description**

This function breaks up a LINestring geometry into multiple LINestring(s). It is used mainly for preserving routability of an object that is created using Open Street Map data. See details, stplanr/issues/282, and stplanr/issues/416.

**Usage**

```
rnet_breakup_vertices(rnet, verbose = FALSE)
```

**Arguments**

- `rnet`: An sf or sfc object with LINestring geometry representing a route network.
- `verbose`: Boolean. If TRUE, the function prints additional messages.

**Details**

A LINestring geometry is broken-up when one of the two following conditions are met:

1. two or more LINERINGs share a POINT which is a boundary point for some LINERING(s), but not all of them (see the rnet_roundabout example);
2. two or more LINERINGs share a POINT which is not in the boundary of any LINERING (see the rnet_cycleway_intersection example).

The problem with the first example is that, according to algorithm behind SpatialLinesNetwork(), two LINERINGs are connected if and only if they share at least one point in their boundaries. The roads and the roundabout are clearly connected in the "real" world but the corresponding LINERING objects do not share two distinct boundary points. In fact, by Open Street Map standards, a roundabout is represented as a closed and circular LINERING, and this implies that the roundabout is not connected to the other roads according to SpatialLinesNetwork() definition. By the same reasoning, the roads in the second example are clearly connected in the "real" world, but they do not share any point in their boundaries. This function is used to solve this type of problem.

**Value**

An sf or sfc object with LINestring geometry created after breaking up the input object.

**See Also**

Other rnet: gsection(), islines(), overline(), rnet_group()
Examples

library(sf)
def_par <- par(no.readonly = TRUE)
par(mar = rep(0, 4))

# Check the geometry of the roundabout example. The dots represent the
# boundary points of the LINESTRINGS. The "isolated" red point in the
# top-left is the boundary point of the roundabout, and it is not shared
# with any other street.
plot(st_geometry(rnet_roundabout), lwd = 2, col = rainbow(nrow(rnet_roundabout)))
boundary_points <- st_geometry(line2points(rnet_roundabout))
points_cols <- rep(rainbow(nrow(rnet_roundabout)), each = 2)
plot(boundary_points, pch = 16, add = TRUE, col = points_cols, cex = 2)

# Clean the roundabout example.
rnet_roundabout_clean <- rnet_breakup_vertices(rnet_roundabout)
plot(st_geometry(rnet_roundabout_clean), lwd = 2, col = rainbow(nrow(rnet_roundabout_clean)))
boundary_points <- st_geometry(line2points(rnet_roundabout_clean))
points_cols <- rep(rainbow(nrow(rnet_roundabout_clean)), each = 2)
plot(boundary_points, pch = 16, add = TRUE, col = points_cols)
# The roundabout is now routable since it was divided into multiple pieces
# (one for each colour), which, according to SpatialLinesNetwork() function,
# are connected.

# Check the geometry of the overpasses example. This example is used to test
# that this function does not create any spurious intersection.
plot(st_geometry(rnet_overpass), lwd = 2, col = rainbow(nrow(rnet_overpass)))
boundary_points <- st_geometry(line2points(rnet_overpass))
points_cols <- rep(rainbow(nrow(rnet_overpass)), each = 2)
plot(boundary_points, pch = 16, add = TRUE, col = points_cols, cex = 2)
# At the moment the network is not routable since one of the underpasses is
# not connected to the other streets.

# Check interactively.
# mapview::mapview(rnet_overpass)

# Clean the network. It should not create any spurious intersection between
# roads located at different heights.
rnet_overpass_clean <- rnet_breakup_vertices(rnet_overpass)
plot(st_geometry(rnet_overpass_clean), lwd = 2, col = rainbow(nrow(rnet_overpass_clean)))
# Check interactively.
# mapview::mapview(rnet_overpass)

# Check the geometry of the cycleway_intersection example. The black dots
# represent the boundary points and we can see that the two roads are not
# connected according to SpatialLinesNetwork() function.
plot(
  rnet_cycleway_intersection$geometry,
  lwd = 2,
  col = rainbow(nrow(rnet_cycleway_intersection)),
  cex = 2
)
rnet_connected

Keep only segments connected to the largest group in a network

Description

This function takes an sf object representing a road network and returns only the parts of the network that are in the largest group.

Usage

rnet_connected(rnet)

Arguments

rnet

An sf object representing a road network

Value

An sf object representing the largest group in the network

Examples

rnet <- rnet_breakup_vertices(stplanr::osm_net_example)
rnet_largest_group <- rnet_connected(rnet)
plot(rnet$geometry)
plot(rnet_largest_group$geometry)
**rnet_cycleway_intersection**

*Example of cycleway intersection data showing problems for SpatialLinesNetwork objects*

---

**Description**

See data-raw/rnet_cycleway_intersection for details on how this was created.

**Format**

A sf object

**Examples**

```r
rnet_cycleway_intersection
```

---

**rnet_get_nodes**

*Extract nodes from route network*

---

**Description**

Extract nodes from route network

**Usage**

```r
rnet_get_nodes(rnet, p = NULL)
```

**Arguments**

- `rnet`: A route network of the type generated by `overline()`
- `p`: A point represented by an sf object the will split the route

**Examples**

```r
rnet_get_nodes(route_network_sf)
```
rnet_group

Assign segments in a route network to groups

Description

This function assigns linestring features, many of which in an sf object can form route networks, into groups. By default, the function igraph::clusters() is used to determine group membership, but any igraph::cluster*() function can be used. See examples and the web page igraph.org/r/doc/communities.html for more information. From that web page, the following clustering functions are available:

Usage

rnet_group(rnet, ...)

## Default S3 method:
rnet_group(rnet, ...)

## S3 method for class 'sfc'
rnet_group(
  rnet,
  cluster_fun = igraph::clusters,
  d = NULL,
  as.undirected = TRUE,
  ...
)

## S3 method for class 'sf'
rnet_group(
  rnet,
  cluster_fun = igraph::clusters,
  d = NULL,
  as.undirected = TRUE,
  ...
)

Arguments

rnet An sf, sfc, or sfNetwork object representing a route network.

... Arguments passed to other methods.

cluster_fun The clustering function to use. Various clustering functions are available in the igraph package. Default: igraph::clusters().

d Optional distance variable used to classify segments that are close (within a certain distance specified by d) to each other but not necessarily touching

as.undirected Coerce the graph created internally into an undirected graph with igraph::as.undirected()? TRUE by default, which enables use of a wider range of clustering functions.
rnet_join

Join route networks

Description
This is a spatial join function that enables adding columns to a 'target' route network from a 'source' route network that contains the base geometry, e.g. from OSM

Usage
rnet_join(
  rnet_x,
  rnet_y,
  dist = 5,
  length_y = TRUE,
  key_column = 1,
  subset_x = TRUE,
  dist_subset = NULL,
  segment_length = 0,
Arguments

- **rnet_x**: Target route network, the output will have the same geometries as features in this object.
- **rnet_y**: Source route network. Columns from this route network object will be copied across to the new network.
- **dist**: The buffer width around rnet_y in meters. 1 m by default.
- **length_y**: Add a new column called length_y? Useful when joining based on length of segments (e.g. weighted mean). TRUE by default.
- **key_column**: The index of the key (unique identifier) column in rnet_x.
- **subset_x**: Subset the source route network by the target network before creating buffers? This can lead to faster and better results. Default: TRUE.
- **dist_subset**: The buffer distance in m to apply when breaking up the source object rnet_y. Default: 5.
- **segment_length**: Should the source route network be split? 0 by default, meaning no splitting. Values above 0 split the source into linestrings with a max distance. Around 5 (m) may be a sensible default for many use cases, the smaller the value the slower the process.
- **endCapStyle**: Type of buffer. See ?sf::st_buffer for details
- **contains**: Should the join be based on sf::st_contains or sf::st_intersects? TRUE by default. If FALSE the centroid of each segment of rnet_y is used for the join. Note: this can result in incorrectly assigning values on sideroads, as documented in #520.
- **max_angle_diff**: The maximum angle difference between x and y nets for a value to be returned... Additional arguments passed to rnet_subset.

Details

The output is an sf object containing polygons representing buffers around the route network in rnet_x. The examples below demonstrate how to join attributes from a route network object created with the function `overline()` onto OSM geometries.

Note: The main purpose of this function is to join an ID from rnet_x onto rnet_y. Subsequent steps, e.g. with `dplyr::inner_join()` are needed to join the attributes back onto rnet_x. There are rarely 1-to-1 relationships between spatial network geometries so we take care when using this function.

See #505 for details and a link to an interactive example of inputs and outputs shown below.
rnet_merge

**Examples**

```r
library(sf)
library(dplyr)
# Uncomment for interactive examples:
plot(st_geometry(route_network_small))
plot(osm_net_example$geometry, lwd = 5, col = "grey", add = TRUE)
plot(route_network_small["flow"], add = TRUE)

rnetj = rnet_join(osm_net_example, route_network_small, dist = 9)
rnetj2 = rnet_join(osm_net_example, route_network_small, dist = 9, segment_length = 10)

# library(mapview)
# mapview(rnetj, zcol = "flow") +
# mapview(rnetj2, zcol = "flow") +
# mapview(route_network_small, zcol = "flow")

plot(sf::st_geometry(rnetj))
plot(rnetj["flow"], add = TRUE)
plot(rnetj2["flow"], add = TRUE)
plot(route_network_small["flow"], add = TRUE)

summary(rnetj2$length_y)

rnetj_summary = rnetj2 %>%
  filter(!is.na(length_y)) %>%
  sf::st_drop_geometry() %>%
  group_by(osm_id) %>%
  summarise(
    flow = weighted.mean(flow, length_y, na.rm = TRUE),
  )

osm_joined_rnet = dplyr::left_join(osm_net_example, rnetj_summary)

plot(sf::st_geometry(route_network_small))
plot(route_network_small["flow"], lwd = 3, add = TRUE)

# Improve fit between geometries and performance by subsetting rnet_x
osm_subset = rnet_subset(osm_net_example, route_network_small, dist = 5)

osm_joined_rnet = dplyr::left_join(osm_subset, rnetj_summary)

plot(route_network_small["flow"])

# plot(osm_joined_rnet[,c("flow")])
# mapview(joined_network) +
# mapview(route_network_small)
```

---

**rnet_merge**

*Merge route networks, keeping attributes with aggregating functions*

**Description**

Merge route networks, keeping attributes with aggregating functions

**Usage**

```
rnet_merge(rnet_x, rnet_y, dist = 5, funs = NULL, sum_flows = TRUE, ...)```
Arguments

|rnet_x| Target route network, the output will have the same geometries as features in this object. |
|rnet_y| Source route network. Columns from this route network object will be copied across to the new network. |
|dist| The buffer width around rnet_y in meters. 1 m by default. |
|funs| A named list of functions to apply to named columns, e.g.: list(flow = sum, length = mean). The default is to sum all numeric columns. |
|sum_flows| Should flows be summed? TRUE by default. |

Value

An sf object with the same geometry as rnet_x

Examples

```
# The source object:
rnet_y = route_network_small["flow"]
# The target object
rnet_x = rnet_subset(osm_net_example[1], rnet_y)
plot(rnet_x$geometry, lwd = 5)
plot(rnet_y$geometry, add = TRUE, col = "red", lwd = 2)
rnet_y$quietness = rnorm(nrow(rnet_y))
funs = list(flow = sum, quietness = mean)
rnet_merged = rnet_merge(rnet_x[1], rnet_y[c("flow", "quietness")], dist = 9, segment_length = 20, funs = funs)
plot(rnet_y$geometry, lwd = 5, col = "lightgrey")
plot(rnet_merged["flow"], add = TRUE, lwd = 2)
```

```
# # Larger example
# system("gh release list")
# system("gh release upload v1.0.2 rnet_x")
# List the files released in v1.0.2:
# system("gh release download v1.0.2")
# rnet_x = sf::read_sf("rnet_x_ed.geojson")
# rnet_y = sf::read_sf("rnet_y_ed.geojson")
# rnet_merged = rnet_merge(rnet_x, rnet_y, dist = 9, segment_length = 20, funs = funs)
```

rnet_overpass  Example of overpass data showing problems for SpatialLinesNetwork objects

Description

See data-raw/rnet_overpass.R for details on how this was created.
**rnet_roundabout**

**Format**

A sf object

**Examples**

```
rnet_overpass
```

```
rnet_roundabout  # Example of roundabout data showing problems for SpatialLinesNetwork objects
```

**Description**

See `data-raw/rnet_roundabout.R` for details on how this was created.

**Format**

A sf object

**Examples**

```
rnet_roundabout
```

```
rnet_subset  # Subset one route network based on overlaps with another
```

**rnet_subset**

**Subset one route network based on overlaps with another**

**Description**

Subset one route network based on overlaps with another

**Usage**

```
rnet_subset(
  rnet_x,
  rnet_y,
  dist = 10,
  crop = TRUE,
  min_length = 20,
  rm_disconnected = TRUE
)
```
route

Plan routes on the transport network

Description

Takes origins and destinations, finds the optimal routes between them and returns the result as a spatial (sf or sp) object. The definition of optimal depends on the routing function used.

Usage

```r
route(
  from = NULL,
  to = NULL,
  l = NULL,
  route_fun = cyclestreets::journey,
  wait = 0,
  n_print = 10,
  list_output = FALSE,
  cl = NULL,
  ...
)
```
Arguments

from  
An object representing origins (if lines are provided as the first argument, from is assigned to l)

to  
An object representing destinations

l  
A spatial (linestring) object

route_fun  
A routing function to be used for converting the lines to routes

wait  
How long to wait between routes? 0 seconds by default, can be useful when sending requests to rate limited APIs.

n_print  
A number specifying how frequently progress updates should be shown

list_output  
If FALSE (default) assumes spatial (linestring) object output. Set to TRUE to save output as a list.

c1  
Cluster

...  
Arguments passed to the routing function

See Also

Other routes: `route_dodgr()`, `route_osrm()`

Examples

# Todo: add examples

---

`routes_fast_sf`  
_Spatial lines dataset of commuter flows on the travel network_

Description

Simulated travel route allocated to the transport network representing the 'fastest' between `cents_sf` objects.

Usage

`routes_fast_sf`

Format

A spatial lines dataset with 49 rows and 15 columns

See Also

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_sf`, `route_network_small`, `routes_slow_sf`, `zones_sf`
routes_slow_sf  
*Spatial lines dataset of commuter flows on the travel network*

**Description**

Simulated travel route allocated to the transport network representing the 'quietest' between cents_sf.

**Format**

A spatial lines dataset 49 rows and 15 columns

**See Also**

Other data: cents_sf, destinations_sf, flow_dests, flowlines_sf, flow, od_data_lines, od_data_routes, od_data_sample, osm_net_example, read_table_builder(), route_network_sf, route_network_small, routes_fast_sf, zones_sf

---

route_average_gradient

*Return average gradient across a route*

**Description**

This function assumes that elevations and distances are in the same units.

**Usage**

```r
route_average_gradient(elevations, distances)
```

**Arguments**

- `elevations`  
  Elevations, e.g. those provided by the cyclestreets package
- `distances`  
  Distances, e.g. those provided by the cyclestreets package

**See Also**

Other route_funs: route_rolling_average(), route_rolling_diff(), route_rolling_gradient(), route_sequential_dist(), route_slope_matrix(), route_slope_vector()

**Examples**

```r
r1 <- od_data_routes[od_data_routes$route_number == 2, ]
elevations <- r1$elevations
distances <- r1$distances
route_average_gradient(elevations, distances) # an average of a 4% gradient
```
**route_bikecitizens**  Get a route from the BikeCitizens web service

---

**Description**

See bikecitizens.net for an interactive version of the routing engine used by BikeCitizens.

**Usage**

```r
route_bikecitizens(
  from = NULL,
  to = NULL,
  base_url = "https://map.bikecitizens.net/api/v1/locations/route.json",
  cocode = "gb-leeds",
  routing_profile = "balanced",
  bike_profile = "citybike",
  from_lat = 53.8265,
  from_lon = -1.576195,
  to_lat = 53.80025,
  to_lon = -1.51577
)
```

**Arguments**

- `from`: A numeric vector representing the start point
- `to`: A numeric vector representing the end point
- `base_url`: The base URL for the routes
- `cocode`: The city code for the routes
- `routing_profile`: What type of routing to use?
- `bike_profile`: What type of bike?
- `from_lat`: Latitude of origin
- `from_lon`: Longitude of origin
- `to_lat`: Latitude of destination
- `to_lon`: Longitude of destination

**Details**

See the bikecitizens.R file in the data-raw directory of the package’s development repository for details on usage and examples.
route_dodgr

Route on local data using the dodgr package

Description

Route on local data using the dodgr package

Usage

route_dodgr(from = NULL, to = NULL, l = NULL, net = NULL)

Arguments

from An object representing origins (if lines are provided as the first argument, from is assigned to l)
to An object representing destinations
l A spatial (linestring) object
net sf object representing the route network

See Also

Other routes: route_osrm(), route()

Examples

if (requireNamespace("dodgr")) {
  from <- c(-1.5327, 53.8006) # from <- geo_code("pedallers arms leeds")
  to <- c(-1.5279, 53.8044) # to <- geo_code("gzing")
  # next 4 lines were used to generate `stplanr::osm_net_example`
  # pts <- rbind(from, to)
  # colnames(pts) <- c("X", "Y")
  # net <- dodgr::dodgr_streetnet(pts = pts, expand = 0.1)
  # osm_net_example <- net[c("highway", "name", "lanes", "maxspeed")]
  r <- route_dodgr(from, to, net = osm_net_example)
  plot(osm_net_example$geometry)
  plot(r$geometry, add = TRUE, col = "red", lwd = 5)
}
route_google

Find shortest path using Google services

Description

Find the shortest path using Google’s services. See the mapsapi package for details.

Usage

\[
\text{route.google(from, to, mode = "walking", key = Sys.getenv("GOOGLE"), ...)}
\]

Arguments

from: An object representing origins (if lines are provided as the first argument, from is assigned to \( l \))

to: An object representing destinations

mode: Mode of transport, walking (default), bicycling, transit, or driving

key: Google key. By default it is Sys.getenv("GOOGLE"). Set it with: usethis::edit_r_environ().

...: Arguments passed to the routing function

Examples

```r
# Not run:
from <- "university of leeds"
to <- "pedallers arms leeds"
r <- route(from, to, route_fun = cyclestreets::journey)
plot(r)
# r_google <- route(from, to, route_fun = mapsapi::mp_directions) # fails
r_google1 <- route_google(from, to)
plot(r_google1)
r_google <- route(from, to, route_fun = route_google)
```

route_nearest_point

Find nearest route to a given point

Description

This function was written as a drop-in replacement for sf::st_nearest_feature(), which only works with recent versions of GEOS.

Usage

\[
\text{route_nearest_point(r, p, id_out = FALSE)}
\]
Arguments

- **r**: The input route object from which the nearest route is to be found
- **p**: The point whose nearest route will be found
- **id_out**: Should the index of the matching feature be returned? FALSE by default

Examples

```r
r <- routes_fast_sf[2:6, NULL]
p <- sf::st_sfc(sf::st_point(c(-1.540, 53.826)), crs = sf::st_crs(r))
route_nearest_point(r, p, id_out = TRUE)
r_nearest <- route_nearest_point(r, p)
plot(r$geometry)
plot(p, add = TRUE)
plot(r_nearest, lwd = 5, add = TRUE)
```

---

**route_network_sf**  
*Spatial lines dataset representing a route network*

**Description**

The flow of commuters using different segments of the road network represented in the `flowlines_sf()` and `routes_fast_sf()` datasets

**Format**

A spatial lines dataset 80 rows and 1 column

**See Also**

Other data: `cents_sf`, `destinations_sf`, `flow_dests`, `flowlines_sf`, `flow`, `od_data_lines`, `od_data_routes`, `od_data_sample`, `osm_net_example`, `read_table_builder()`, `route_network_small`, `routes_fast_sf`, `routes_slow_sf`, `zones_sf`

---

**route_network_small**  
*Spatial lines dataset representing a small route network*

**Description**

The flow between randomly selected vertices on the `osm_net_example`. See `data-raw/route_network_small.R` for details.

**Format**

A spatial lines dataset with one column: flow
**route_osrm**

Plan routes on the transport network using the OSRM server

### Description

This function is a simplified and (because it uses GeoJSON not binary polyline format) slower R interface to OSRM routing services compared with the excellent `osrm::osrmRoute()` function (which can be used via the `route()` function).

### Usage

```r
route_osrm(
  from,  # An object representing origins (if lines are provided as the first argument, from is assigned to l)
  to,    # An object representing destinations
  osrm.server = "https://routing.openstreetmap.de/",  # The base URL of the routing server. getOption("osrm.server") by default.
  osrm.profile = "foot"  # The routing profile to use, e.g. "car", "bike" or "foot" (when using the routing.openstreetmap.de test server), getOption("osrm.profile") by default.
)
```

### Arguments

- `from`:
  - An object representing origins (if lines are provided as the first argument, from is assigned to l)
- `to`:
  - An object representing destinations
- `osrm.server`:
  - The base URL of the routing server. getOption("osrm.server") by default.
- `osrm.profile`:
  - The routing profile to use, e.g. "car", "bike" or "foot" (when using the routing.openstreetmap.de test server), getOption("osrm.profile") by default.
- `profile`:
  - Which routing profile to use? One of "foot" (default) "bike" or "car" for the default open server.

### See Also

Other routes: `route_dodgr()`, `route()`

### Examples

```r
# Examples no longer working due to API being down
# l1 = od_data_lines[49,]
# l1m = od_coords(l1)
# from = l1m[, 1:2]
# to = l1m[, 3:4]
```
route_rolling_average

Return smoothed averages of vector

Description

This function calculates a simple rolling mean in base R. It is useful for calculating route characteristics such as mean distances of segments and changes in gradient.

Usage

route_rolling_average(x, n = 3)

Arguments

x  Numeric vector to smooth
n  The window size of the smoothing function. The default, 3, will take the mean of values before, after and including each value.

See Also

Other route_funs: route_average_gradient(), route_rolling_diff(), route_rolling_gradient(), route_sequential_dist(), route_slope_matrix(), route_slope_vector()

Examples

y <- od_data_routes$elevations[od_data_routes$route_number == 2]
y
route_rolling_average(y)
route_rolling_average(y, n = 1)
route_rolling_average(y, n = 2)
route_rolling_average(y, n = 3)
route_rolling_diff  

Return smoothed differences between vector values

Description

This function calculates a simple rolling mean in base R. It is useful for calculating route characteristics such as mean distances of segments and changes in gradient.

Usage

route_rolling_diff(x, lag = 1, abs = TRUE)

Arguments

- **x**: Numeric vector to smooth
- **lag**: The window size of the smoothing function. The default, 3, will take the mean of values before, after and including each value.
- **abs**: Should the absolute (always positive) change be returned? True by default

See Also

Other route_funs: route_average_gradient(), route_rolling_average(), route_rolling_gradient(), route_sequential_dist(), route_slope_matrix(), route_slope_vector()

Examples

```r
r1 <- od_data_routes[od_data_routes$route_number == 2, ]
y <- r1$elevations
route_rolling_diff(y, lag = 1)
route_rolling_diff(y, lag = 2)
r1$elevations_diff_1 <- route_rolling_diff(y, lag = 1)
r1$elevations_diff_n <- route_rolling_diff(y, lag = 1, abs = FALSE)
d <- cumsum(r1$distances) - r1$distances / 2
diff_above_mean <- r1$elevations_diff_1 + mean(y)
diff_above_mean_n <- r1$elevations_diff_n + mean(y)
plot(c(0, cumsum(r1$distances)), c(y, y[length(y)]), ylim = c(80, 130))
lines(c(0, cumsum(r1$distances)), c(y, y[length(y)]))
points(d, diff_above_mean)
points(d, diff_above_mean_n, col = "blue")
abline(h = mean(y))
```
route_rolling_gradient

Calculate rolling average gradient from elevation data at segment level

Description

Calculate rolling average gradient from elevation data at segment level

Usage

route_rolling_gradient(elevations, distances, lag = 1, n = 2, abs = TRUE)

Arguments

elevations  Elevations, e.g. those provided by the cyclestreets package
distances   Distances, e.g. those provided by the cyclestreets package
lag         The window size of the smoothing function. The default, 3, will take the mean of values before, after and including each value.
n         The window size of the smoothing function. The default, 3, will take the mean of values before, after and including each value.
abs         Should the absolute (always positive) change be returned? True by default

See Also

Other route_funs: route_average_gradient(), route_rolling_average(), route_rolling_diff(), route_sequential_dist(), route_slope_matrix(), route_slope_vector()

Examples

r1 <- od_data_routes[od_data_routes$route_number == 2, ]
y <- r1$elevations
distances <- r1$distances
route_rolling_gradient(y, distances)
route_rolling_gradient(y, distances, abs = FALSE)
route_rolling_gradient(y, distances, n = 3)
route_rolling_gradient(y, distances, n = 4)
r1$elevations_diff_1 <- route_rolling_diff(y, lag = 1)
r1$rolling_gradient <- route_rolling_gradient(y, distances, n = 2)
r1$rolling_gradient3 <- route_rolling_gradient(y, distances, n = 3)
r1$rolling_gradient4 <- route_rolling_gradient(y, distances, n = 4)
d <- cumsum(r1$distances) - r1$distances / 2
diff_above_mean <- r1$elevations_diff_1 + mean(y)
par(mfrow = c(2, 1))
plot(NULL, c(0, cumsum(r1$distances)), c(y, mean(y)), ylim = c(80, 130))
lines(NULL, c(0, cumsum(r1$distances)), c(y, mean(y)))
points(d, diff_above_mean)
abline(h = mean(y))
route_sequential_dist

Calculate the sequential distances between sequential coordinate pairs

Description
Calculate the sequential distances between sequential coordinate pairs

Usage
route_sequential_dist(m, lonlat = TRUE)

Arguments
m Matrix containing coordinates and elevations
lonlat Are the coordinates in lon/lat order? TRUE by default

See Also
Other route_funs: route_average_gradient(), route_rolling_average(), route_rolling_diff(), route_rolling_gradient(), route_slope_matrix(), route_slope_vector()

Examples
x <- c(0, 2, 3, 4, 5, 9)
y <- c(0, 0, 0, 0, 0, 1)
m <- cbind(x, y)
route_sequential_dist(m)

route_slope_matrix

Calculate the gradient of line segments from a matrix of coordinates

Description
Calculate the gradient of line segments from a matrix of coordinates

Usage
route_slope_matrix(m, e = m[, 3], lonlat = TRUE)
route_slope_vector

Arguments

m
Matrix containing coordinates and elevations

e
Elevations in same units as x (assumed to be metres)

lonlat
Are the coordinates in lon/lat order? TRUE by default

See Also

Other route_funs: route_average_gradient(), route_rolling_average(), route_rolling_diff(), route_rolling_gradient(), route_sequential_dist(), route_slope_matrix()

Examples

x <- c(0, 2, 3, 4, 5, 9)
y <- c(0, 0, 0, 0, 0, 9)
z <- c(1, 2, 2, 4, 3, 1) / 10
m <- cbind(x, y, z)
plot(x, z, ylim = c(-0.5, 0.5), type = "l")
(gx <- route_slope_vector(x, z))
(gxy <- route_slope_matrix(m, lonlat = FALSE))
abline(h = 0, lty = 2)
points(x[-length(x)], gx, col = "red")
points(x[-length(x)], gxy, col = "blue")
title("Distance (in x coordinates) elevation profile",
    sub = "Points show calculated gradients of subsequent lines")

route_slope_vector Calculate the gradient of line segments from distance and elevation vectors

Description

Calculate the gradient of line segments from distance and elevation vectors

Usage

route_slope_vector(x, e)

Arguments

x
Vector of locations

e
Elevations in same units as x (assumed to be metres)

See Also

Other route_funs: route_average_gradient(), route_rolling_average(), route_rolling_diff(), route_rolling_gradient(), route_sequential_dist(), route_slope_matrix()
Examples

```r
x <- c(0, 2, 3, 4, 5, 9)
e <- c(1, 2, 2, 4, 3, 1) / 10
route_slope_vector(x, e)
```

---

### route_split

**Split route in two at point on or near network**

**Description**

Split route in two at point on or near network

**Usage**

```r
route_split(r, p)
```

**Arguments**

- `r`: An `sf` object with one feature containing a linestring geometry to be split
- `p`: A point represented by an `sf` object the will split the route

**Value**

An `sf` object with 2 feature

**Examples**

```r
sample_routes <- routes_fast_sf[2:6, NULL]
r <- sample_routes[2, ]
p <- sf::st_sfc(sf::st_point(c(-1.540, 53.826)), crs = sf::st_crs(r))
plot(r$geometry, lwd = 9, col = "grey")
plot(p, add = TRUE)
r_split <- route_split(r, p)
plot(r_split, col = c("red", "blue"), add = TRUE)
```

---

### route_split_id

**Split route based on the id or coordinates of one of its vertices**

**Description**

Split route based on the id or coordinates of one of its vertices

**Usage**

```r
route_split_id(r, id = NULL, p = NULL)
```
toptail_buff

Arguments

\( r \)  
An sf object with one feature containing a linestring geometry to be split

\( id \)  
The index of the point on the number to be split

\( p \)  
A point represented by an sf object the will split the route

Examples

```r
sample_routes <- routes_fast_sf[2:6, 3]
r <- sample_routes[2, ]
id <- round(n_vertices(r) / 2)
r_split <- route_split_id(r, id = id)
plot(r$geometry, lwd = 9, col = "grey")
plot(r_split, col = c("red", "blue"), add = TRUE)
```

stplanr-deprecated  
Deprecated functions in stplanr

Description

These functions are depreciated and will be removed:

toptail_buff  
Clip the beginning and ends of sf LINESTRING objects

Description

Takes lines and removes the start and end point, to a distance determined by the nearest buff polygon border.

Usage

toptail_buff(l, buff, ...)

Arguments

\( l \)  
An sf object representing lines

\( buff \)  
An sf object with POLYGON geometry to buffer the linestring.

\( ... \)  
Arguments passed to sf::st_buffer()

See Also

Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), line2points(), line_bearing(), line_breakup(), line_midpoint(), line_segment(), line_via(), mats2line(), n_vertices(), onewaygeo(), points2line()
Examples

```r
l <- routes_fast_sf
buff <- zones_sf
r_toptail <- toptail_buff(l, buff)
nrow(l)
nrow(r_toptail)
plot(zones_sf$geometry)
plot(l$geometry, add = TRUE)
plot(r_toptail$geometry, lwd = 5, add = TRUE)
```

zones_sf  |  Spatial polygons of home locations for flow analysis.

Description

These correspond to the cents_sf data.

Details

- `geo_code`. the official code of the zone

See Also

Other data: `cents_sf, destinations_sf, flow_dests, flowlines_sf, flow, od_data_lines, od_data_routes, od_data_sample, osm_net_example, read_table_builder(), route_network_sf, route_network_small, routes_fast_sf, routes_slow_sf`

Examples

```r
library(sf)
zones_sf
plot(zones_sf)
```
## Index

### datasets
- cents_sf, 7
- destinations_sf, 7
- flow, 8
- flow_dests, 9
- flowlines_sf, 9
- od_data_lines, 32
- od_data_routes, 33
- od_data_sample, 33
- osm_net_example, 39
- rnet_cycleway_intersection, 51
- rnet_overpass, 56
- rnet_roundabout, 57
- route_network_sf, 64
- route_network_small, 64
- routes_fast_sf, 59
- routes_slow_sf, 60
- zones_sf, 73

### data
- cents_sf, 7
- destinations_sf, 7
- flow, 8
- flow_dests, 9
- flowlines_sf, 9
- od_data_lines, 32
- od_data_routes, 33
- od_data_sample, 33
- osm_net_example, 39
- read_table_builder, 45
- route_network_sf, 64
- route_network_small, 64
- routes_fast_sf, 59
- routes_slow_sf, 60
- zones_sf, 73

### geo
- bbox_scale, 5
- bind_sf, 6
- geo_bb, 10
- geo_bb_matrix, 11
- geo_buffer, 11
- geo_length, 13
- geo_projected, 14
- geo_select_aeq, 14
- quadrant, 44

### lines
- angle_diff, 4
- geo_toptail, 15
- is_linepoint, 18
- line2df, 18
- line2points, 19
- line_bearing, 20
- line_breakup, 21
- line_midpoint, 22
- line_segment, 23
- line_via, 24
- mats2line, 25
- n_vertices, 25
- onewaygeo, 38
- points2line, 43
- toptail_buff, 72

### nodes
- geo_code, 12

### od
- od2line, 26
- od2odf, 27
- od_aggregate_from, 29
- od_aggregate_to, 30
- od_coords, 31
- od_coords2line, 31
- od_id, 34
- od_id_order, 35
- od_oneway, 35
- od_to_odmatrix, 37
- odmatrix_to_od, 28
- points2flow, 43
- points2odf, 44

### package
- stplanr-package, 4
INDEX

* rnet
  gsection, 16
  islines, 17
  overline, 39
  rnet_breakup_vertices, 48
  rnet_group, 52
* route_funs
  route_average_gradient, 60
  route_rolling_average, 66
  route_rolling_diff, 67
  route_rolling_gradient, 68
  route_sequential_dist, 69
  route_slope_matrix, 69
  route_slope_vector, 70
* routes
  route, 58
  route_dodgr, 62
  route_osrm, 65

angle_diff, 4, 16, 18–26, 38, 43, 72

bb2poly (geo_bb), 10
bbox_scale, 5, 6, 10–15, 45
bind_sf, 6, 6, 10–15, 45
cents_sf, 7, 8, 9, 33, 39, 46, 59, 60, 64, 65, 73
cents_sf(), 26, 28–30
destinations_sf, 7, 7, 8, 9, 33, 39, 46, 59, 60, 64, 65, 73
dplyr::inner_join(), 54
flow, 7, 8, 8, 9, 33, 39, 46, 59, 60, 64, 65, 73
flow(), 26, 28–30
flow_dests, 7–9, 9, 33, 39, 46, 59, 60, 64, 65, 73
flowlines_sf, 7–9, 9, 33, 39, 46, 59, 60, 64, 65, 73
flowlines_sf(), 64
geo_bb, 6, 10, 11–15, 45
geo_bb_matrix, 6, 10, 11, 12–15, 45
geo_buffer, 6, 10, 11, 11, 13–15, 45
geo_code, 12
geo_length, 6, 10–12, 13, 14, 15, 45
geo_projected, 6, 10–13, 14, 15, 45
geo_select_aeq, 6, 10–14, 14, 45
geo_select_aeq(), 14
gprojected (geo_projected), 14
gsection, 16, 17, 41, 48, 53
igraph::as.undirected(), 52
igraph::clusters(), 52
is_linepoint, 5, 16, 18, 19–26, 38, 43, 72
islines, 16, 17, 41, 48, 53
line2df, 5, 16, 18, 18, 19–26, 38, 43, 72
line2points, 5, 16, 18, 19, 19, 20–26, 38, 43, 72
line2pointsn (line2points), 19
line2vertices (line2points), 19
line Bearing, 5, 16, 18, 19, 20, 21–26, 38, 43, 72
line_breakup, 5, 16, 18–20, 21, 22–26, 38, 43, 72
line_cast, 22
line_midpoint, 5, 16, 18–21, 22, 23–26, 38, 43, 72
line_segment, 5, 16, 18–22, 23, 24–26, 38, 43, 72
line_via, 5, 16, 18–23, 24, 25, 26, 38, 43, 72
lwgeom::st_linesubstring(), 22
mats2line, 5, 16, 18–24, 25, 26, 38, 43, 72
n_vertices, 5, 16, 18–25, 26, 38, 43, 72
od2line, 26, 28–32, 34–37, 43, 44
od2df, 27, 27, 29–32, 34–37, 43, 44
od_aggregate_from, 27–29, 29, 30–32, 34–37, 43, 44
od_aggregate_to, 27–30, 31, 32, 34–37, 43, 44
od_coords, 27–30, 31, 32, 34–37, 43, 44
od_coords2line, 27–31, 31, 34–37, 43, 44
od_data_lines, 7–9, 32, 33, 39, 46, 59, 60, 64, 65, 73
od_data_routes, 7–9, 33, 33, 39, 46, 59, 60, 64, 65, 73
od_data_sample, 7–9, 33, 33, 39, 46, 59, 60, 64, 65, 73
od_id, 27–32, 34, 35–37, 43, 44
od_id_character (od_id), 34
od_id_max_min (od_id), 34
od_id_max_min(), 36
od_id_order, 27–32, 34, 35, 36, 37, 43, 44
od_id_szudzik (od_id), 34
od_id_szudzik(), 36
<table>
<thead>
<tr>
<th>Function</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>od_oneway</td>
<td>27–32, 34, 35, 37, 43, 44</td>
</tr>
<tr>
<td>od_to_odmatrix</td>
<td>27–32, 34–36, 37, 43, 44</td>
</tr>
<tr>
<td>odmatrix_to_od</td>
<td>27, 28, 28, 30–32, 34–37, 43, 44</td>
</tr>
<tr>
<td>onewaygeo</td>
<td>5, 16, 18–26, 38, 43, 72</td>
</tr>
<tr>
<td>osm_net_example</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>osrm::osrmRoute()</td>
<td>65</td>
</tr>
<tr>
<td>overline()</td>
<td>16, 17, 39, 48, 53</td>
</tr>
<tr>
<td>overline2()</td>
<td>17, 54</td>
</tr>
<tr>
<td>overline_intersection</td>
<td>42</td>
</tr>
<tr>
<td>points2flow</td>
<td>27–32, 34–37, 43, 44</td>
</tr>
<tr>
<td>points2line</td>
<td>5, 16, 18–26, 38, 43, 72</td>
</tr>
<tr>
<td>points2odf</td>
<td>27–32, 34–37, 43, 44</td>
</tr>
<tr>
<td>quadrant</td>
<td>6, 10–15, 44</td>
</tr>
<tr>
<td>read_table_builder</td>
<td>7–9, 33, 39, 45, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>rnet_add_node</td>
<td>46</td>
</tr>
<tr>
<td>rnet_boundary_df</td>
<td>(rnet_boundary_points), 47</td>
</tr>
<tr>
<td>rnet_boundary_points</td>
<td>47</td>
</tr>
<tr>
<td>rnet_boundary_points_lwgeom</td>
<td>(rnet_boundary_points), 47</td>
</tr>
<tr>
<td>rnet_boundary_unique</td>
<td>(rnet_boundary_points), 47</td>
</tr>
<tr>
<td>rnet_breakup_vertices</td>
<td>16, 17, 41, 48, 53</td>
</tr>
<tr>
<td>rnet_connected</td>
<td>50</td>
</tr>
<tr>
<td>rnet_cycleway_intersection</td>
<td>51</td>
</tr>
<tr>
<td>rnet_duplicated_vertices</td>
<td>(rnet_boundary_points), 47</td>
</tr>
<tr>
<td>rnet_get_nodes</td>
<td>51</td>
</tr>
<tr>
<td>rnet_group</td>
<td>16, 17, 41, 48, 52</td>
</tr>
<tr>
<td>rnet_join</td>
<td>53</td>
</tr>
<tr>
<td>rnet_merge</td>
<td>55</td>
</tr>
<tr>
<td>rnet_overpass</td>
<td>56</td>
</tr>
<tr>
<td>rnet_roundabout</td>
<td>57</td>
</tr>
<tr>
<td>rnet_subset</td>
<td>57</td>
</tr>
<tr>
<td>route()</td>
<td>65</td>
</tr>
<tr>
<td>route_average_gradient</td>
<td>60, 66–70</td>
</tr>
<tr>
<td>route_bikecitizens</td>
<td>61</td>
</tr>
<tr>
<td>route_dodgr</td>
<td>59, 62, 65</td>
</tr>
<tr>
<td>route_google</td>
<td>63</td>
</tr>
<tr>
<td>route_nearest_point</td>
<td>63</td>
</tr>
<tr>
<td>route_network_sf</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>route_network_small</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>route_osrm</td>
<td>59, 62, 65</td>
</tr>
<tr>
<td>route_rolling_average</td>
<td>60, 66, 67–70</td>
</tr>
<tr>
<td>route_rolling_diff</td>
<td>60, 66, 67, 68–70</td>
</tr>
<tr>
<td>route_rolling_gradient</td>
<td>60, 66, 67, 68, 69, 70</td>
</tr>
<tr>
<td>route_sequential_dist</td>
<td>60, 66–68, 69, 70</td>
</tr>
<tr>
<td>route_slope_matrix</td>
<td>60, 66–70, 70</td>
</tr>
<tr>
<td>route_slope_vector</td>
<td>60, 66–70, 70</td>
</tr>
<tr>
<td>route_split</td>
<td>71</td>
</tr>
<tr>
<td>route_split_id</td>
<td>71</td>
</tr>
<tr>
<td>routes_fast_sf</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>routes_fast_sf()</td>
<td>64</td>
</tr>
<tr>
<td>routes_slow_sf</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
<tr>
<td>stplanr (stplanr-package)</td>
<td>4</td>
</tr>
<tr>
<td>stplanr-deprecated</td>
<td>72</td>
</tr>
<tr>
<td>stplanr-package</td>
<td>4</td>
</tr>
<tr>
<td>toptail (geo_toptail)</td>
<td>15</td>
</tr>
<tr>
<td>toptail_buff</td>
<td>5, 16, 18–26, 38, 43, 72</td>
</tr>
<tr>
<td>zones_sf</td>
<td>7–9, 33, 39, 46, 59, 60, 64, 65, 73</td>
</tr>
</tbody>
</table>