Package ‘stplanr’

July 22, 2021

Type Package
Title Sustainable Transport Planning
Version 0.8.3
Maintainer Robin Lovelace <rob00x@gmail.com>
Description Tools for transport planning with an emphasis on spatial transport data and non-motorized modes. Enables common transport planning tasks including: downloading and cleaning transport datasets; creating geographic “desire lines” from origin-destination (OD) data; route assignment, locally and via interfaces to routing services such as <https://cyclestreets.net/> and calculation of route segment attributes such as bearing. The package implements the ‘travel flow aggregation’ method described in Morgan and Lovelace (2020) <doi:10.1177/239980320942779>. 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

BugReports https://github.com/ropensci/stplanr/issues
LazyData yes

Depends R (>= 3.5.0)
Imports sp (>= 1.3.1), curl (>= 3.2), dplyr (>= 0.7.6), htr (>= 1.3.1), jsonlite (>= 1.5), stringr (>= 1.3.1), maptools (>= 0.9.3), raster (>= 2.6.7), rgeos (>= 0.3.28), methods, geosphere (>= 1.5.7), Rcpp (>= 0.12.1), igraph (>= 1.2.2), nabor (>= 0.5.0), rlang (>= 0.2.2), lwgeom (>= 0.1.4), sf (>= 0.6.3), magrittr, sfheaders, data.table, pbapply

LinkingTo RcppArmadillo (>= 0.9.100.5.0), Rcpp (>= 0.12.18)

Suggests testthat (>= 2.0.0), knitr (>= 1.20), rmarkdown (>= 1.10), dodgr (>= 0.0.3), stats19, cyclestreets, leaflet, rgdal, pcat, tmap, bench, openxlsx (>= 4.1.0), osrm, geodist, mapsapi

VignetteBuilder knitr

URL https://github.com/ropensci/stplanr,
https://docs.ropensci.org/stplanr/
SystemRequirements GNU make
RoxygenNote 7.1.1
Encoding UTF-8
NeedsCompilation yes
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],
Andrea Gilardi [ctb] (<https://orcid.org/0000-0002-9424-7439>)
Repository CRAN
Date/Publication 2021-07-22 04:40:02 UTC

R topics documented:

stplanr-package ........................................... 5
angle_diff .................................................. 5
as_sf_fun .................................................... 6
bbox_scale ................................................... 7
calc_catchment ............................................ 7
calc_catchment_sum ....................................... 10
calc_moving_catchment .................................. 12
calc_network_catchment .................................. 13
c_local ....................................................... 15
cents .......................................................... 16
destination_zones ........................................ 17
dist_google .................................................. 18
find_network_nodes ....................................... 19
flow ........................................................... 21
flowlines .................................................... 22
flow_dests ................................................... 22
geo_bb ....................................................... 23
geo_bb_matrix ............................................. 24
geo_buffer ................................................... 25
geo_code ..................................................... 26
geo_length ................................................... 27
geo_projected .............................................. 27
geo_select_aeq .......................................... 28
geo_toptail ................................................ 29
gsection ..................................................... 30
islines ....................................................... 31
is_linepoint ............................................... 31
R topics documented:

- line2df .................................................. 32
- line2points .......................................... 33
- line2route ............................................ 34
- line2routeRetry ...................................... 35
- lineLabels ............................................ 36
- line_bearing ......................................... 37
- line_breakup ......................................... 38
- line_length .......................................... 39
- line_midpoint ........................................ 39
- line_sample .......................................... 40
- line_segment ......................................... 40
- line_via ................................................ 41
- l_poly .................................................. 42
- mats2line ............................................. 43
- nearest_cyclestreets ............................... 43
- nearest_google ...................................... 44
- n_sample_length ..................................... 45
- n_vertices ............................................ 46
- od2line ................................................ 47
- od2odf ................................................ 48
- odmatrix_to_od ....................................... 49
- od_aggregate_from .................................. 50
- od_aggregate_to ..................................... 51
- od_coords ............................................. 52
- od_coords2line ....................................... 52
- od_data_lines ....................................... 53
- od_data_routes ...................................... 54
- od_data_sample ...................................... 54
- od_dist ................................................. 54
- od_id ................................................... 55
- od_id_order .......................................... 56
- od_oneway ............................................ 57
- od_to_odmatrix ....................................... 59
- onewaygeo ............................................. 59
- osm_net_example ..................................... 61
- overline .............................................. 61
- overline_intersection ................................ 64
- overline_spatial ..................................... 65
- plot,sfNetwork,ANY-method ........................ 65
- plot,SpatialLinesNetwork,ANY-method .......... 66
- points2flow .......................................... 67
- points2line .......................................... 67
- points2odf ........................................... 68
- quadrant .............................................. 69
- read_table_builder .................................. 70
- reproject ............................................. 71
- rnet_add_node ....................................... 71
- rnet_boundary_points ............................... 72
R topics documented:

rnet_breakup_vertices ................................................. 73
rnet_cycleway_intersection ........................................ 75
rnet_get_nodes ....................................................... 75
rnet_group ............................................................ 76
rnet_overpass ......................................................... 78
rnet_roundabout ....................................................... 78
route ................................................................. 78
routes_fast ............................................................ 80
routes_slow ............................................................ 81
route_average_gradient ............................................. 81
route_bikecitizens .................................................... 82
route_cyclestreets .................................................... 83
route_dodgr ............................................................ 84
route_google ........................................................... 85
route_local ............................................................. 86
route_nearest_point .................................................. 87
route_network ......................................................... 87
route_osrm ............................................................. 88
route_rolling_average ............................................... 89
route_rolling_diff .................................................... 90
route_rolling_gradient ............................................... 91
route_sequential_dist ................................................ 92
route_slope_matrix ................................................... 92
route_slope_vector ................................................... 93
route_split ............................................................ 94
route_split_id ........................................................ 94
route_transportapi_public .......................................... 95
sfNetwork-class ...................................................... 96
sln2points ............................................................ 97
sln_add_node .......................................................... 97
sln_clean_graph ...................................................... 98
SpatialLinesNetwork .................................................. 98
SpatialLinesNetwork-class .......................................... 100
stplanr-deprecated .................................................. 100
summary.sfNetwork-method ......................................... 100
summary.SpatialLinesNetwork-method ............................. 101
sum_network_links ................................................... 101
sum_network_routes .................................................. 102
toptailgs .............................................................. 104
toptail_buff ............................................................ 105
update_line_geometry ................................................. 105
weightfield ............................................................. 106
writeGeoJSON .......................................................... 107
zones ................................................................. 108

Index 109
Description

The stplanr package provides functions to access and analyse data for transportation research, including origin-destination analysis, route allocation and modelling travel patterns.

Interesting functions

- `overline()` - Aggregate overlaying route lines and data intelligently
- `calc_catchment()` - Create a 'catchment area' to show the areas serving a destination
- `route_cyclestreets()` - Finds the fastest routes for cyclists between two places.

Author(s)

Robin Lovelace <rob00x@gmail.com>

See Also

https://github.com/ropensci/stplanr

angle_diff

Calculate the angular difference between lines and a predefined bearing

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)

Arguments

- l: A spatial lines object
- angle: an angle in degrees relative to North, with 90 being East and -90 being West.
- 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
- absolute: If TRUE (the default) only positive values can be returned
Details

Building on the convention used in `bearing()` 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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptailBuff()`, `toptailgs()`, `update_line_geometry()`

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)
  a <- angle_diff(flowlines, angle = 0, bidirectional = TRUE, absolute = TRUE)
  plot(flowlines)
  plot(flowlines[a < 15, ], add = TRUE, lwd = 3, col = "red")
  # East-West
  plot(flowlines[a > 75, ], add = TRUE, lwd = 3, col = "green")
}
```

---

**as_sf_fun**  
*Convert functions support sf/sp*

**Description**

Convert functions support sf/sp

**Usage**

`as_sf_fun(input, FUN, ...)`

**Arguments**

- `input`: Input object - an sf or sp object
- `FUN`: A function that works on sp/sf data
- `...`: Arguments passed to FUN
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

bbox_scale(bb, scale_factor)

Arguments

bb  Bounding box 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.

See Also

Other geo: geo_bb_matrix(), geo_bb(), quadrant(), reproject()

Examples

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")

calc_catchment

Calculate catchment area and associated summary statistics.

Description

Calculate catchment area and associated summary statistics.
Usage

calc_catchment(
  polygonlayer,
  targetlayer,
  calccols,
  distance = 500,
  projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667 ",
    "+lat_0=0 +lon_0=10 +x_0=0 +y_0=0 +ellps=GRS80",
    " +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"),
  retainAreaProportion = FALSE,
  dissolve = FALSE,
  quadsegs = NULL
)

Arguments

polygonlayer | A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.

targetlayer | A SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame, SpatialPolygons, SpatialLines or SpatialPoints object containing the specifications of the facility for which the catchment area is being calculated. If the object contains more than one facility (e.g., multiple cycle paths) the aggregate catchment area will be calculated.

calccols | A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area. If dissolve = FALSE, all other variables in the original SpatialPolygonsDataFrame for zones that fall partly or entirely within the catchment area will be included in the returned SpatialPolygonsDataFrame but will not be adjusted for the proportion within the catchment area.

distance | Defines the size of the catchment area as the distance around the targetlayer in the units of the projection (default = 500 metres)

projection | The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

retainAreaProportion | Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

dissolve | Boolean value. If TRUE collapses the underlying zones within the catchment area into a single region with statistics for the whole catchment area.
quadsegs  Number of line segments to use to approximate a quarter circle. Parameter passed to buffer functions, default is 5 for sp and 30 for sf.

Details

Calculates the catchment area of a facility (e.g., cycle path) using straight-line distance as well as summary statistics from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns a SpatialPolygonsDataFrame.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline.spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
salincome <- as(sf::read_sf("smallsa1.shp"), "Spatial")
testcycleway <- as(sf::read_sf("testcycleway.shp"), "Spatial")
cway_catch <- calc_catchment(
polygonlayer = salincome,
targetlayer = testcycleway,
calccols = c("Total"),
distance = 800,
projection = "austalbers",
dissolve = TRUE)
)
plot(salincome)
plot(cway_catch, add = TRUE, col = "green")
plot(testcycleway, col = "red", add = TRUE)
salincome <- sf::read_sf("smallsa1.shp")
testcycleway <- sf::read_sf("testcycleway.shp")
f <- list.files(".", "testcycleway|smallsa1")
file.remove(f)
cway_catch <- calc_catchment(
polygonlayer = salincome,
targetlayer = testcycleway,
calccols = c("Total"),
distance = 800,
projection = "austalbers",
dissolve = TRUE)
)
plot(salincome$geometry)
plot(testcycleway$geometry, col = "red", add = TRUE)
plot(cway_catch["Total"], add = TRUE)
```
### Description

Calculate summary statistics for catchment area.

### Usage

```r
calc_catchment_sum(
  polygonlayer,  # A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
  targetlayer,   # A SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame, SpatialPolygons, SpatialLines or SpatialPoints object containing the specifications of the facility for which the catchment area is being calculated. If the object contains more than one facility (e.g., multiple cycle paths) the aggregate catchment area will be calculated.
  calccols,      # A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area.
  distance = 500,  # Defines the size of the catchment area as the distance around the targetlayer in the units of the projection (default = 500 metres)
  projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667",
    " +lat_0=0 +lon_0=10 +x_0=0 +y_0=0",
    " +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"),
  retainAreaProportion = FALSE,  # A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
  quadsegs = NA)  # A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
)
```

### Arguments

- **polygonlayer**: A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.
- **targetlayer**: A SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialPointsDataFrame, SpatialPolygons, SpatialLines or SpatialPoints object containing the specifications of the facility for which the catchment area is being calculated. If the object contains more than one facility (e.g., multiple cycle paths) the aggregate catchment area will be calculated.
- **calccols**: A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area.
- **distance**: Defines the size of the catchment area as the distance around the targetlayer in the units of the projection (default = 500 metres)
- **projection**: The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).
calc_catchment_sum

retainAreaProportion

Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

quadsegs

Number of line segments to use to approximate a quarter circle. Parameter passed to buffer functions, default is 5 for sp and 30 for sf.

Details

Calculates the summary statistics for a catchment area of a facility (e.g., cycle path) using straight-line distance from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns either a single value if calccols is of length = 1, or a named vector otherwise.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), lines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method.plot, sfNetwork, ANY-method.rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

## Not run:

data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
salincome <- rgdal::readOGR(".", "smallsa1")
testcycleway <- rgdal::readOGR(".", "testcycleway")
calc_catchment_sum(  
polygonlayer = salincome,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

calc_catchment_sum(  
polygonlayer = salincome,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)

## End(Not run)
calc_moving_catchment  Calculate summary statistics for all features independently.

Description
Calculate summary statistics for all features independently.

Usage

calc_moving_catchment(
  polygonlayer,
  targetlayer,
  calccols,
  distance = 500,
  projection = "worldalbers",
  retainAreaProportion = FALSE
)

Arguments

polygonlayer  A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.

targetlayer  A SpatialPolygonsDataFrame, SpatialLinesDataFrame or SpatialPointsDataFrame object containing the specifications of the facilities and zones for which the catchment areas are being calculated.

calccols  A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment areas.

distance  Defines the size of the catchment areas as the distance around the targetlayer in the units of the projection (default = 500 metres)

projection  The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

retainAreaProportion  Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).
calc_network_catchment

**Details**

Calculates the summary statistics for a catchment area of multiple facilities or zones using straight-line distance from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns the original source dataframe with additional columns with summary variables.

**See Also**

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), isolines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

**Examples**

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"))
unzip(file.path(data_dir, "testcycleway.zip"))
salincome <- readOGR(".", "smallsa1")
testcycleway <- readOGR(".", "testcycleway")
calc_moving_catchment(
  polygonlayer = salincome,
  targetlayer = testcycleway,
  calccols = c("Total"),
  distance = 800,
  projection = "austalbers"
)
## End(Not run)
```

calc_network_catchment

*Calculate catchment area and associated summary statistics using network.*

**Description**

Calculate catchment area and associated summary statistics using network.

**Usage**

```r
calc_network_catchment(
  sln,
  polygonlayer,
  targetlayer,
  calccols,
  maximpedance = 1000,
)```

calc_network_catchment

distance = 100,
projection = paste0("+proj=aea +lat_1=90 +lat_2=-18.416667",
" +lat_0=0 +lon_0=10 +x_0=0 +y_0=0",
" +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"),
retainAreaProportion = FALSE,
dissolve = FALSE
)

Arguments

sln The SpatialLinesNetwork to use.
polygonlayer A SpatialPolygonsDataFrame containing zones from which the summary statistics for the catchment variable will be calculated. Smaller polygons will increase the accuracy of the results.

targetlayer A SpatialPolygonsDataFrame, SpatialLinesDataFrame or SpatialPointsDataFrame object containing the specifications of the facilities and zones for which the catchment areas are being calculated.
calccols A vector of column names containing the variables in the polygonlayer to be used in the calculation of the summary statistics for the catchment area. If dissolve = FALSE, all other variables in the original SpatialPolygonsDataFrame for zones that fall partly or entirely within the catchment area will be included in the returned SpatialPolygonsDataFrame but will not be adjusted for the proportion within the catchment area.

maximpedance The maximum value of the network’s weight attribute in the units of the weight (default = 1000).
distance Defines the additional catchment area around the network in the units of the projection. (default = 100 metres)

projection The proj4string used to define the projection to be used for calculating the catchment areas or a character string 'austalbers' to use the Australian Albers Equal Area projection. Ignored if the polygonlayer is projected in which case the targetlayer will be converted to the projection used by the polygonlayer. In all cases the resulting object will be reprojected to the original coordinate system and projection of the polygon layer. Default is an Albers Equal Area projection but for more reliable results should use a local projection (e.g., Australian Albers Equal Area project).

retainAreaProportion Boolean value. If TRUE retains a variable in the resulting SpatialPolygonsDataFrame containing the proportion of the original area within the catchment area (Default = FALSE).

dissolve Boolean value. If TRUE collapses the underlying zones within the catchment area into a single region with statistics for the whole catchment area.

Details

Calculates the catchment area of a facility (e.g., cycle path) using network distance (or other weight variable) as well as summary statistics from variables available in a SpatialPolygonsDataFrame with census tracts or other zones. Assumes that the frequency of the variable is evenly distributed throughout the zone. Returns a SpatialPolygonsDataFrame.
ca_local

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

```r
## Not run:
data_dir <- system.file("extdata", package = "stplanr")
unzip(file.path(data_dir, "smallsa1.zip"), exdir = tempdir())
unzip(file.path(data_dir, "testcycleway.zip"), exdir = tempdir())
unzip(file.path(data_dir, "sydroads.zip"), exdir = tempdir())
sa1income <- readOGR(tempdir(), "smallsa1")
testcycleway <- readOGR(tempdir(), "testcycleway")
sydroads <- readOGR(tempdir(), "roads")
sydnetwork <- SpatialLinesNetwork(sydroads)
calc_network_catchment(
  sln = sydnetwork,
  polygonlayer = sa1income,
  targetlayer = testcycleway,
  calccols = c("Total"),
  maximpedance = 800,
  distance = 200,
  projection = "austalbers",
  dissolve = TRUE
)
## End(Not run)
```

ca_local

SpatialPointsDataFrame representing road traffic deaths

Description

This dataset represents the type of data downloaded and cleaned using stplanr functions. It represents a very small sample (with most variables stripped) of open data from the UK’s Stats19 dataset.

Usage

data(ca_local)

Format

A SpatialPointsDataFrame with 11 rows and 2 columns
Examples

```r
## Not run:
# Generate data
ac <- read_stats19_ac()
cia <- read_stats19_ca()
ve <- read_stats19_ve()
library(dplyr)
ca_ac <- inner_join(ca, ac)
ca_cycle <- ca_ac %>%
  filter(Casualty_Severity == "Fatal" & !is.na(Latitude)) %>%
  select(Age = Age_of_Casualty, Mode = Casualty_Type, Longitude, Latitude)
ca_sp <- sp::SpatialPointsDataFrame(coords = ca_cycle[3:4], data = ca_cycle[1:2])
data("route_network")
proj4string(ca_sp) <- proj4string(route_network)
bb <- bb2poly(route_network)
ca_local <- ca_sp[bb, ]
## End(Not run)
```

---

cents 

**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 of my home when I was writing this package.

Usage

data(cents)

Format

A spatial dataset with 8 rows and 5 variables

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:
Examples

```r
## Not run:
cents
plot(cents)

## End(Not run)
```

---

**destination_zones**

**Example destinations data**

### Description

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

### Usage

```r
data(destination_zones)
```

### Format

A spatial dataset with 87 features

### See Also

Other example data: `flow_dests`, `flowlines`, `flow`, `route_network`, `routes_fast`, `routes_slow`

### Examples

```r
## Not run:
# This is how the dataset was constructed - see
# https://cowz.geodata.soton.ac.uk/download/
download.file(
  "https://cowz.geodata.soton.ac.uk/download/files/COWZ_EW_2011_BFC.zip",
  "COWZ_EW_2011_BFC.zip"
)
unzip("COWZ_EW_2011_BFC.zip")
wz <- raster::shapefile("COWZ_EW_2011_BFC.shp")
to_remove <- list.files(pattern = "COWZ", full.names = TRUE, recursive = TRUE)
file.remove(to_remove)
proj4string(wz)
wz <- sp::spTransform(wz, proj4string(zones))
destination_zones <- wz[zones, ]
plot(destination_zones)
devtools::use_data(destination_zones)
head(destination_zones@data)
destinations <- rgeos::gCentroid(destinations, byid = TRUE)
destinations <- sp::SpatialPointsDataFrame(destinations, destination_zones@data)
```
devtools::use_data(destinations, overwrite = TRUE)
destinations_sf <- sf::st_as_sf(destinations)
devtools::use_data(destinations_sf)

## End(Not run)

dist_google Return travel network distances and time using the Google Maps API

Description

Return travel network distances and time using the Google Maps API

Usage

dist_google(
  from, to,
  google_api = Sys.getenv("GOOGLEDIST"),
  g_units = "metric",
  mode = c("bicycling", "walking", "driving", "transit"),
  arrival_time = ""
)

Arguments

from Two-column matrix or data frame of coordinates representing latitude and longitude of origins.
to Two-column matrix or data frame of coordinates representing latitude and longitude of destinations.
google_api String value containing the Google API key to use.
g_units Text string, either metric (default) or imperial.
mode Text string specifying the mode of transport. Can be bicycling (default), walking, driving or transit
arrival_time Time of arrival in date format.

Details

Absent authorization, the google API is limited to a maximum of 100 simultaneous queries, and so will, for example, only returns values for up to 10 origins times 10 destinations.

Details

Estimate travel times accounting for the road network - see https://developers.google.com/maps/documentation/distance-matrix/overview Note: Currently returns the json object returned by the Google Maps API and uses the same origins and destinations.
find_network_nodes

Find graph node ID of closest node to given coordinates

Description

Find graph node ID of closest node to given coordinates

Examples

## Not run:
# Distances from one origin to one destination
from <- c(-46.3, -23.4)
to <- c(-46.4, -23.4)
dist_google(from = from, to = to, mode = "walking") # not supported on last test
dist_google(from = from, to = to, mode = "driving")
dist_google(from = c(0, 52), to = c(0, 53))

data("cents")
# Distances from between all origins and destinations
dists_cycle <- dist_google(from = cents, to = cents)
dists_drive <- dist_google(cents, cents, mode = "driving")
dists_trans <- dist_google(cents, cents, mode = "transit")
dists_trans_am <- dist_google(cents, cents,
mode = "transit",
arrival_time = strptime("2016-05-27 09:00:00",
format = "%Y-%m-%d %H:%M:%S", tz = "BST"
)
)
# Find out how much longer (or shorter) cycling takes than walking
summary(dists_cycle$duration / dists_trans$duration)
# Difference between travelling now and for 9am arrival
summary(dists_trans_am$duration / dists_trans$duration)

do f <- points2odf(cents)
do f <- cbind(do f, dists)
head(do f)
flow <- points2flow(cents)
# show the results for duration (thicker line = shorter)
plot(flow, lwd = mean(odf$duration) / odf$duration)
dist_google(c("Hereford"), c("Weobley", "Leominster", "Kington"))
dist_google(c("Hereford"), c("Weobley", "Leominster", "Kington"),
mode = "transit", arrival_time = strptime("2016-05-27 17:30:00",
format = "%Y-%m-%d %H:%M:%S", tz = "BST"
)
)

## End(Not run)
find_network_nodes

Usage

find_network_nodes(sln, x, y = NULL, maxdist = 1000)

Arguments

sln SpatialLinesNetwork to search.

x Either the x (longitude) coordinate value, a vector of x values, a dataframe or matrix with (at least) two columns, the first for coordinate for x (longitude) values and a second for y (latitude) values, or a named vector of length two with values of 'lat' and 'lon'. The output of geo_code() either as a single result or as multiple (using rbind() ) can also be used.

y Either the y (latitude) coordinate value or a vector of y values.

maxdist The maximum distance within which to match the nodes to coordinates. If the SpatialLinesNetwork is projected then distance should be in the same units as the projection. If longlat, then distance is in metres. Default is 1000.

Value

An integer value with the ID of the node closest to (x,y) with a value of NA the closest node is further than maxdist from (x,y). If x is a vector, returns a vector of Node IDs.

Details

Finds the node ID of the closest point to a single coordinate pair (or a set of coordinates) from a SpatialLinesNetwork.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), gsection(), islines(), lineLabels(), overline.spatial(), overline(), plot,SpatialLinesNetwork,ANY-method,plot,sfNetwork,ANY-method,rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
find_network_nodes(sln, -1.516734, 53.828)
Description

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

Usage

data(flow)

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 cents georeferenced spatial objects.

See Also

Other example data: destination_zones, flow_dests, flowlines, route_network, routes_fast, routes_slow

Examples

## Not run:
# This is how the dataset was constructed - see
# https://github.com/npct/pct - if download to ~/repos
flow <- readRDS("~/repos/pct/pct-data/national/flow.Rds")
data(cents)
o <- flow$Area.of.residence %in% cents$geo_code[-1]
d <- flow$Area.of.workplace %in% cents$geo_code[-1]
flow <- flow[o & d, ] # subset flows with o and d in study area
library(devtools)
flow$id <- paste(flow$Area.of.residence, flow$Area.of.workplace)
use_data(flow, overwrite = TRUE)

# Convert flows to spatial lines dataset
flowlines <- od2line(flow = flow, zones = cents)
# use_data(flowlines, overwrite = TRUE)

# Convert flows to routes
routes_fast <- line2route(l = flowlines, plan = "fastest")
routes_slow <- line2route(l = flowlines, plan = "quietest")

use_data(routes_fast)
use_data(routes_slow)
routes_fast_sf <- sf::st_as_sf(routes_fast)
routes_slow_sf <- sf::st_as_sf(routes_slow)

## End(Not run)

---

**flowlines**

*spatial lines dataset of commuter flows*

**Description**

Flow data after conversion to a spatial format with `od2line()` (see `flow()`).

**Format**

A spatial lines dataset with 49 rows and 15 columns

**See Also**

Other example data: `destination_zones`, `flow_dests`, `flow`, `route_network`, `routes_fast`, `routes_slow`

---

**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**

data(flow_dests)
geo_bb

Format
A data frame with 49 rows and 15 columns

See Also
Other example data: destination_zones, flowlines, flow, route_network, routes_fast, routes_slow

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

Arguments

shp Spatial object (from sf or sp packages)
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)
geo_bb_matrix

Create matrix representing the spatial bounds of an object

Usage

geo_bb_matrix(shp)

Arguments

shp Spatial object (from sf or sp packages)

See Also

Other geo: bbox_scale(), geo_bb(), quadrant(), reproject()

Examples

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

Description

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

Examples

# Simple features implementation:
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(geo_bb(routes_fast_sf, output = "points"), add = TRUE)
plot(routes_fast_sf$geometry, add = TRUE)
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

geo_buffer(shp, dist = NULL, width = NULL, ...)

Arguments

- **shp**: A spatial object with a geographic CRS (e.g. WGS84) around which a buffer should be drawn.
- **dist**: The distance (in metres) of the buffer (when buffering simple features).
- **width**: The distance (in metres) of the buffer (when buffering sp objects).
- **...**: Arguments passed to the buffer (see \rgeos::gBuffer or \sf::st_buffer for details).

Details

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

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)
  # on legacy sp objects (not tested)
  # buff_sp <- geo_buffer(routes_fast, width = 100)
  # class(buff_sp)
  # plot(buff_sp, col = "red")
}
```
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,
  service = "nominatim",
  base_url = "https://maps.google.com/maps/api/geocode/json",
  return_all = FALSE,
  pat = 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**: The API key used. By default this is set to NULL and this is usually aquired automatically through a helper, api_pat().

See Also

Other nodes: `nearest_google()`

Examples

```r
## 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

Examples

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

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. the the rgeos or sf packages)

 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, e.g. byid = TRUE if the function is rgeos::gLength()
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 focussed 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.

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

Examples

sp::bbox(routes_fast)
new_crs <- geo_select_aeq(routes_fast)
rf_projected <- sp::spTransform(routes_fast, new_crs)
sp::bbox(rf_projected)
line_length <- rgeos::gLength(rf_projected, byid = TRUE)
plot(line_length, rf_projected$length)
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**

```r
geo_toptail(l, toptail_dist, ...)
```

**Arguments**

- `l`: A SpatialLines object
- `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 rgeos::gBuffer()

**Details**

Note: `toptailgs()` is around 10 times faster, but only works on data with geographic CRS's due to its reliance on the geosphere package.

**See Also**

Other lines: `angle_diff()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

**Examples**

```r
lib_versions <- sf::sf_extSoftVersion()
lib_versions
# dont test due to issues with sp classes on some set-ups
if (lib_versions[3] >= "6.3.1") {
  # l <- routes_fast[2:4, ] # to run with sp classes
  l <- routes_fast_sf[2:4, ]
  l_top_tail <- geo_toptail(l, 300)
  l_top_tail
  plot(sf::st_geometry(l_top_tail))
  plot(sf::st_geometry(geo_toptail(l, 600)), lwd = 9, add = TRUE)
}
```
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: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), islines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

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)))
  # dont test due to issues with sp classes on some set-ups
  # sl <- routes_fast[2:4, ]
  # rsec <- gsection(sl)
  # rsec_buff <- gsection(sl, buff_dist = 1)
  # plot(sl[1], lwd = 9, col = 1:nrow(sl))
  # plot(rsec, col = 5 + (1:length(rsec)), add = TRUE, lwd = 3)
  # plot(rsec_buff, col = 5 + (1:length(rsec_buff)), add = TRUE, lwd = 3)
}
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

`islines(g1, g2)`

Arguments

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

See Also

Other rnet: `SpatialLinesNetwork`, `calc_catchment_sum()`, `calc_catchment()`, `calc_moving_catchment()`, `calc_network_catchment()`, `find_network_nodes()`, `gsection()`, `lineLabels()`, `overline_spatial()`, `overline()`, `plot`, `SpatialLinesNetwork`, `ANY-method`, `plot`, `sfNetwork`, `ANY-method`, `rnet_breakup_vertices()`, `rnet_group()`, `sln2points()`, `sum_network_links()`, `sum_network_routes()`

Examples

```r
## Not run:
# Load data
rnet <- overline(routes_fast[c(2, 3, 22), ], attrib = "length")
plot(rnet)
lines(routes_fast[22, ], col = "red")  # line without overlaps
islines(routes_fast[2, ], routes_fast[3, ])
islines(routes_fast[2, ], routes_fast[22, ])
# 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

```r
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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

Examples

```r
data(flowlines)
islp <- is_linepoint(flowlines)
nrow(flowlines)
sum(islp)
# Remove invisible 'linepoints'
nrow(flowlines[!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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()` `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`
Examples

```
data(flowlines)
line2df(flowlines[5,]) # beginning and end of a single straight line
line2df(flowlines) # on multiple lines
line2df(routes_fast[5:6,]) # beginning and end of routes
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

```
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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

Examples

```
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

---

**line2route**  
*Convert straight OD data (desire lines) into routes*

**Description**  
Convert straight OD data (desire lines) into routes

**Usage**

```r
line2route(
  l,
  route_fun = stplanr::route_cyclestreets,
  n_print = 10,
  list_output = FALSE,
  l_id = NA,
  time_delay = 0,
  ...
)
```

**Arguments**

- `l` A spatial (linestring) object
- `route_fun` A routing function to be used for converting the straight lines to routes. `od2line()` is an example.
- `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.
- `l_id` Character string naming the id field from the input lines data, typically the origin and destination ids pasted together. If absent, the row name of the straight lines will be used.
- `time_delay` Number or seconds to wait between each query
- `...` Arguments passed to the routing function, e.g. `route_cyclestreets()`

**Details**

See `route_cyclestreets()` and other route functions for details.

A parallel implementation of this was available until version 0.1.8.
line2routeRetry

Convert straight spatial (linestring) object from flow data into routes retrying on connection (or other) intermittent failures

Description

Convert straight spatial (linestring) object from flow data into routes retrying on connection (or other) intermittent failures

Usage

line2routeRetry(lines, pattern = "^Error: ", n_retry = 3, ...)
lineLabels

Details

See line2route() for the version that is not retried on errors.

See Also

Other routes: line2route(), route_dodgr(), route_local(), route_osrm(), route_transportapi_public(), route()

Examples

```r
## Not run:
data(flowlines)
rf_list <- line2routeRetry(flowlines[1:2, ], pattern = "nonexistenceerror", silent = F)
## End(Not run)
```

Description

This function adds labels to lines plotted using base graphics. Largely for illustrative purposes, not designed for publication-quality graphics.

Usage

```r
lineLabels(sl, attrib)
```

Arguments

- `sl` A SpatialLinesDataFrame with overlapping elements
- `attrib` A text string corresponding to a named variable in `sl`

Author(s)

Barry Rowlingson

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()
line_bearing

Find the bearing of straight lines

Description

This is a simple wrapper around the geosphere function `bearing()` to return 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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

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") {
  bearings_sf_1_9 <- line_bearing(flowlines_sf[1:5, ])
  bearings_sf_1_9 # lines of 0 length have NaN bearing
  bearings_sp_1_9 <- line_bearing(flowlines[1:5, ])
  bearings_sp_1_9
  plot(bearings_sf_1_9, bearings_sp_1_9)
  line_bearing(flowlines_sf[1:5, ], bidirectional = TRUE)
  line_bearing(flowlines[1:5, ], bidirectional = TRUE)
}
```
line_breakup  

*Break up line objects into shorter segments*

**Description**

This function breaks up a LINESTRING geometries into smaller pieces.

**Usage**

```r
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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

**Examples**

```r
library(sf)
z <- zones_sf$geometry
l <- routes_fast_sf$geometry[2]
l_split <- line_breakup(l, z)
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_length

**Calculate length of lines in geographic CRS**

**Description**

Calculate length of lines in geographic CRS

**Usage**

```r
line_length(l, byid = TRUE)
```

**Arguments**

- `l`: A spatial lines object
- `byid`: Logical determining whether the length is returned per object (default is true)

### line_midpoint

**Find the mid-point of lines**

**Description**

This is a wrapper around `SpatialLinesMidPoints()` that allows it to find the midpoint of lines that are not projected, which have a lat/long CRS.

**Usage**

```r
line_midpoint(l)
```

**Arguments**

- `l`: A spatial lines object

**See Also**

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

**Examples**

```r
data(routes_fast)
line_midpoint(routes_fast[2:5, ])
```
**line_sample**

Sample n points along lines with density proportional to a weight

**Description**

Sample n points along lines with density proportional to a weight

**Usage**

```r
line_sample(l, n, weights)
```

**Arguments**

- `l` The SpatialLines object along which to create sample points
- `n` The total number of points to sample
- `weights` The relative probabilities of lines being samples

**See Also**

Other lines: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buffer()`, `toptailgs()`, `update_line_geometry()`

**Examples**

```r
l <- flowlines[2:5, ]
n <- 100
l_lengths <- line_length(l)
weights <- l$All
p <- line_sample(l, 50, weights)
plot(p)
p <- line_sample(l, 50, weights = 1:length(l))
plot(p)
```

**line_segment**

Divide SpatialLines dataset into regular segments

**Description**

Divide SpatialLines dataset into regular segments

**Usage**

```r
line_segment(l, n_segments, segment_length = NA)
```
line_via

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: `angle_diff()`, `geo_toptail()`, `is_linepoint()`, `line2df()`, `line2points()`, `line_bearing()`, `line_breakup()`, `line_midpoint()`, `line_sample()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

Examples

```r
data(routes_fast)
l <- routes_fast[2, ]
library(sp)
l_seg2 <- line_segment(l = l, n_segments = 2)
plot(l_seg2, col = l_seg2$group, lwd = 50)
```

Description

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

Usage

```r
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_sample()`, `line_segment()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`
Examples

```r
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)
```

---

**l_poly**

*Line polygon*

**Description**

This dataset represents road width for testing.

**Usage**

```r
data(l_poly)
```

**Format**

A SpatialPolygon

**Examples**

```r
## Not run:
l <- routes_fast[13,]
l_poly <- geo_projected(l, rgeos::gBuffer, 8)
plot(l_poly)
plot(routes_fast, add = TRUE)
# allocate road width to relevant line
devtools::use_data(l_poly)
## End(Not run)
```
Convert 2 matrices to lines

**Description**

Convert 2 matrices to lines

**Usage**

`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_sample()`, `line_segment()`, `line_via()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `points2line()`, `toptailBuff()`, `toptailgs()`, `update_line Geometry()`

**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)
```

Generate nearest point on the route network of a point using the CycleStreets.net

**Description**

Generate nearest point on the route network of a point using the CycleStreets.net

**Usage**

`nearest_cyclestreets(shp = NULL, lat, lng, pat = api_pat("cyclestreet"))`
Arguments

- **shp**
  A spatial object

- **lat**
  Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.

- **lng**
  Numeric vector containing longitude coordinate for each coordinate to map.

- **pat**
  The API key used. By default this is set to NULL and this is usually acquired automatically through a helper, api_pat().

Details

Retrieve coordinates of the node(s) on the network mapped from coordinates passed to functions.

Note: there is now a dedicated cyclestreets package: https://github.com/Robinlovelace/cyclestreets

Examples

```r
## Not run:
nearest_cyclestreets(53, 0.02, pat = Sys.getenv("CYCLESTREETS"))
nearest_cyclestreets(cents[1, ], pat = Sys.getenv("CYCLESTREETS"))
nearest_cyclestreets(cents_sf[1, ], pat = Sys.getenv("CYCLESTREETS"))
## End(Not run)
```

---

**nearest_google**

*Generate nearest point on the route network of a point using the Google Maps API*

**Description**

Generate nearest point on the route network of a point using the Google Maps API

**Usage**

```r
nearest_google(lat, lng, google_api)
```

**Arguments**

- **lat**
  Numeric vector containing latitude coordinate for each coordinate to map. Also accepts dataframe with latitude in the first column and longitude in the second column.

- **lng**
  Numeric vector containing longitude coordinate for each coordinate to map.

- **google_api**
  String value containing the Google API key to use.

**Details**

Retrieve coordinates of the node(s) on the network mapped from coordinates passed to functions.
n_sample_length

See Also

Other nodes: geo_code()

Examples

```r
## Not run:
nearest_google(lat = 50.333, lng = 3.222, google_api = "api_key_here")

## End(Not run)
```

```r
n_sample_length(n, l_lengths, weights)
```

Arguments

- **n**: Sum of integer values returned
- **l_lengths**: Numeric vector of line lengths
- **weights**: Relative probabilities of samples on lines

See Also

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

Examples

```r
n <- 10
l_lengths <- 1:5
weights <- 9:5
(res <- n_sample_length(n, l_lengths, weights))
sum(res)
n <- 100
l_lengths <- c(12, 22, 15, 14)
weights <- c(38, 10, 44, 34)
(res <- n_sample_length(n, l_lengths, weights))
sum(res)
# more examples:
n_sample_length(5, 1:5, c(0.1, 0.9, 0, 0, 0))
```
n_sample_length(5, 1:5, c(0.5, 0.3, 0.1, 0, 0))
l <- flowlines[2:6,]
l_lengths <- line_length(l)
n <- n_sample_length(10, l_lengths, weights = l$All)

---

n_vertices

Retrieve the number of vertices from a SpatialLines or SpatialPolygons object

Description

Returns a vector of the same length as the number of lines, with the number of vertices per line or polygon.

Usage

n_vertices(l)

Arguments

l

A SpatialLines or SpatialPolygons object

Details

See https://gis.stackexchange.com/questions/58147/ for more information.

See Also

Other lines: angle_diff(), geo_toptail(), is_linepoint(), line2df(), line2points(), line_bearing(), line_breakup(), line_midpoint(), line_sample(), line_segment(), line_via(), mats2line(), n_sample_length(), onewaygeo(), points2line(), toptail_buff(), toptailgs(), update_line_geometry()

Examples

n_vertices(routes_fast)
n_vertices(routes_fast_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

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
)

od2line2(flow, zones)

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(), 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: dist_google(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_coords(), od_dist(), 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)
l <- od2line(flow = od_data, zones = cents)
# When destinations are different
head(destinations[1:5])
o_d.data2 <- flow.dests[1:12, 1:3]
o_d.data2
flowlines_dests <- od2line(od_data2, cents_sf, destinations = destinations_sf)
flowlines_dests
plot(flowlines_dests)

od2odf

Extract coordinates from OD data

Description

Extract coordinates from OD data

Usage

od2odf(flow, zones)

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(), 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', based on a data frame containing origin and destination cones (flow) that match the first column in a spatial (polygon or point) object (zones). The function returns a data frame with coordinates for the origin and destination.

See Also

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

Examples

data(flow)
data(zones)
od2odf(flow[1:2, ], zones)

---

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

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.

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

#### Examples

```
od_aggregate_from(flow)
```
Summary statistics of trips arriving at destination 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 destination.

Usage

od_aggregate_to(flow, attrib = NULL, FUN = sum, ..., col = 2)

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()`, 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: it assumes the destination ID column is the 2nd and the default summary statistic is `sum()`. By default, if `attrib` is not set, it summarises all numeric columns.

See Also

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

Examples

```
od_aggregate_to(flow)
```
od_coords

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

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: dist_google(), od2line(), od2odf(), od_aggregate_from(), od_aggregate_to(), od_coords2line(), od_dist(), od_id, od_oneway(), od_to_odmatrix(), odmatrix_to_od(), points2flow(), points2odf()

Examples

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

od_coords2line

Convert origin-destination coordinates into desire lines

Description

Convert origin-destination coordinates into desire lines

Usage

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: `dist_google()`, `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords()`, `od_dist()`, `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

**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

**Examples**

```r
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

**Examples**

```r
od_data_sample
```

### od_dist

**Quickly calculate Euclidean distances of od pairs**

**Description**

It is common to want to know the Euclidean distance between origins and destinations in OD data. You can calculate this by first converting OD data to `SpatialLines` data, e.g. with `od2line()`. However this can be slow and overkill if you just want to know the distance. This function is a few orders of magnitude faster.

**Usage**

```r
od_dist(flow, zones)
```
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()`, 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

Note: this function assumes that the zones or centroids in `cents` have a geographic (lat/lon) CRS.

See Also

Other od: `dist_google()`, `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

data(flow)
data(cents)
od_dist(flow, cents)

---

**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)
```

```
od_id_max_min(x, y)
```

```
od_id_character(x, y)
```

Arguments

- **x** a vector of numeric, character, or factor values
- **y** a vector of numeric, character, or factor values
- **ordematters** 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 an have a single ID that represents pairs of IDs with or without directionality, so they contain 'twoway' or bi-directional values.

od_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

od_oneway

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

Examples

```r
(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]])
n <- 100
ids <- as.character(runif(n, 1e4, 1e7 - 1))
# benchmark of methods:
x <- data.frame(
id1 = rep(ids, times = n),
id2 = rep(ids, each = n),
val = 1,
stringsAsFactors = FALSE
)
bench::mark(
check = FALSE, iterations = 10,
od_id_order(x),
od_id_character(x$id1, x$id2),
od_id_szudzik(x$id1, x$id2),
od_id_max_min(x$id1, x$id2)
)
```

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
od_oneway

Description
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
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.

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
od_oneway(

  x,
  attrib = names(x[-c(1:2)])[vapply(x[-c(1:2)], is.numeric, TRUE)],
  id1 = names(x)[1],
  id2 = names(x)[2],
  stplanr.key = NULL
)

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.
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 `od_id_max_min()` or `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 in the other direction. If only the largest flow in either direction is captured in an analysis, for example, the true extent of travel will be heavily underestimated for OD pairs which have similar amounts of travel in both directions. Flows in both directions are often represented by overlapping lines with identical geometries (see `flowlines()`) which can be confusing for users and are difficult to plot.

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

See Also
Other od: `dist_google()`, `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_dist()`, `od_id`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

Examples
```
(od_min <- od_data_sample[, c(1, 2, 9), 1:6])
(od_oneway <- od_oneway(od_min))
# (od_oneway_old = onewayid(od_min, attrib = 3:6)) # old implementation
nrow(od_oneway) < nrow(od_min) # result has fewer rows
sum(od_min$all) == sum(od_oneway$all) # but the same total flow
od_oneway(od_min, attrib = "all")
attrib <- which(vapply(flow, is.numeric, TRUE))
flow_oneway <- od_oneway(flow, attrib = attrib)
colSums(flow_oneway[attrib]) == colSums(flow[attrib]) # test if the colSums are equal
# Demonstrate the results from oneway and onewaygeo are identical
flow_oneway_geo <- onewaygeo(flowlines, attrib = attrib)
flow_oneway_sf <- od_oneway(flowlines_sf)
par(mfrow = c(1, 2))
plot(flow_oneway_geo, lwd = flow_oneway_geo$All / mean(flow_oneway_geo$All))
plot(flow_oneway_sf$geometry, lwd = flow_oneway_sf$All / mean(flow_oneway_sf$All))
par(mfrow = c(1, 1))
(od_max_min <- od_oneway(od_min, stplanr.key = od_id_character(od_min[[1]], od_min[[2]])))
cor(od_max_min$all, od_oneway$all)
# benchmark performance
# benchmark(mark(check = FALSE, iterations = 3,
# onewayid(flowlines_sf, attrib),
# od_oneway(flowlines_sf)
# )
```
**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**

```r
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: `dist_google()`, `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_dist()`, `od_id`, `od_oneway()`, `odmatrix_to_od()`, `points2flow()`, `points2odf()`

**Examples**

```r
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. Flows in both direction are often represented by overlapping lines with identical geometries (see `flowlines()`) which can be confusing for users and are difficult to plot.
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_sample(), line_segment(), line_via(), mats2line(), n_sample_length(), n_vertices(), points2line(), toptail_buff(), toptailgs(), update_line_geometry()

Examples

plot(flowlines[1:30, ], lwd = flowlines$On.foot[1:30])
singlines <- onewaygeo(flowlines[1:30, ], attrib = which(names(flowlines) == "On.foot"))
plot(singlines, lwd = singlines$On.foot / 2, col = "red", add = TRUE)
## Not run:
plot(flowlines, lwd = flowlines$All / 10)
singlelines <- onewaygeo(flowlines, attrib = 3:14)
plot(singlelines, lwd = singlelines$All / 20, col = "red", add = TRUE)
sum(singlelines$All) == sum(flowlines$All)
nrow(singlelines)
singlelines_sf <- onewaygeo(flowlines_sf, attrib = 3:14)
sum(singlelines_sf$All) == sum(flowlines_sf$All)
summary(singlelines$All == singlelines_sf$All)
## End(Not run)
Example of OpenStreetMap road network

Format
An sf object

Examples
osm_net_example

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+05,
  quiet = ifelse(nrow(sl) < 1000, TRUE, FALSE),
  fun = sum
)
```
```r
overline2(
  sl,
  attrib,
  ncores = 1,
  simplify = TRUE,
  regionalise = 1e+05,
  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 regonalisation 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
overline

Author(s)
Barry Rowlingson
Malcolm Morgan

References

See Also
Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), ilines(), linelabels(), overline_spatial(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples
sl <- routes_fast_sf[2:4,]
s1$All <- flowlines$All[2:4]
rlen <- overline(sl = sl, attrib = "All")
nrow(sl)
nrow(rlen)
plot(rlen)
rlen_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))
rlen_sf_raw <- overline(sl, attrib = "length", simplify = FALSE)
nrow(rnet_sf_raw)
summary(n_vertices(rnet_sf_raw))
plot(rnet_sf_raw)
rlen_sf_raw$n <- 1:nrow(rnet_sf_raw)
plot(rnet_sf_raw[10:25,])
# legacy implementation based on sp data
# sl <- routes_fast[2:4,]
# rnet1 <- overline(sl = sl, attrib = "length")
# rnet2 <- overline(sl = sl, attrib = "length", buff_dist = 1)
# plot(rnet1, lw = rnet1$length / mean(rnet1$length))
# plot(rnet2, lw = rnet2$length / mean(rnet2$length))
**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**

```r
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**

```r
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)
```
**overline_spatial**

*Spatial aggregation of routes represented with sp classes*

**Description**

This function, largely superseded by sf implementations, still works but is not particularly fast.

**Usage**

```r
overline_spatial(sl, attrib, fun = sum, na.zero = FALSE, buff_dist = 0)
```

**Arguments**

- `sl` : SpatialLinesDataFrame with overlapping Lines to split by number of overlapping features.
- `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.
- `na.zero` : Sets whether aggregated values with a value of zero are removed.
- `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: `SpatialLinesNetwork,calc_catchment_sum(),calc_catchment(),calc-moving_catchment(),calc-network-catchment(),find-network-nodes(),gsection(),islines(),lineLabels(),overline(),plot,SpatialLinesNetwork,ANY-method,plot,snNetwork,ANY-method,rnet-breakup-vertices(),rnet-group(),sln2points(),sum-network-links(),sum-network-routes()`

---

**plot, sfNetwork, ANY-method**  

*Plot an sfNetwork*

**Description**

Plot an sfNetwork

**Usage**

```r
## S4 method for signature 'sfNetwork,ANY'
plot(x, component = "sl", ...)
```
plot,SpatialLinesNetwork,ANY-method

Arguments

- `x` The sfNetwork to plot
- `component` The component of the network to plot. Valid values are "sl" for the geographic (sf) representation or "graph" for the graph representation.
- `...` Arguments to pass to relevant plot function.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline.spatial(), overline(), plot,SpatialLinesNetwork,ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()

Examples

```r
sln_sf <- SpatialLinesNetwork(route_network_sf)
plot(sln_sf)
```

Description

Plot a SpatialLinesNetwork

Usage

```r
## S4 method for signature 'SpatialLinesNetwork,ANY'
plot(x, component = "sl", ...)
```

Arguments

- `x` The SpatialLinesNetwork to plot
- `component` The component of the network to plot. Valid values are "sl" for the geographic (SpatialLines) representation or "graph" for the graph representation.
- `...` Arguments to pass to relevant plot function.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline.spatial(), overline(), plot,SpatialLinesNetwork,ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links(), sum_network_routes()
**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**

```r
points2flow(p)
```

**Arguments**

- `p`  
  A spatial (point) object

**See Also**

Other od: `dist_google()`, `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od coorden()`, `od_dist()`, `od_id`, `od_oneaway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2odf()``

**Examples**

```r
data(cents)
plot(cents)
flow <- points2flow(cents)
plot(flow, add = TRUE)
flow_sf <- points2flow(cents_sf)
plot(flow_sf)
```

**points2line**  

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

**Description**

This is a simple wrapper around `spLines()` that makes the creation of `SpatialLines` objects easy and intuitive.

**Usage**

```r
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_sample()`, `line_segment()`, `line_via()`, `mats2line()`, `n_sample_length()`, `n_vertices()`, `onewaygeo()`, `toptail_buff()`, `toptailgs()`, `update_line_geometry()`

**Examples**

```r
p <- matrix(1:4, ncol = 2)
library(sp)
l <- points2line(p)
plot(l)
l <- points2line(cents)
plot(l)
p <- line2points(routes_fast)
l <- points2line(p)
plot(l)
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: `dist_google()`, `od2line()`, `od2odf()`, `od_aggregate_from()`, `od_aggregate_to()`, `od_coords2line()`, `od_coords()`, `od_dist()`, `od_id`, `od_oneway()`, `od_to_odmatrix()`, `odmatrix_to_od()`, `points2flow()`
quadrant

Examples

```r
data(cents)
df <- points2odf(cents)
cents_centroids <- rgeos::gCentroid(cents, byid = TRUE)
df2 <- points2odf(cents_centroids)
df3 <- points2odf(cents_sf)
```

quadrant

Split a spatial object into quadrants

Description

Split a spatial object (initially tested on SpatialPolygons) into quadrants.

Usage

```r
quadrant(sp_obj, number_out = FALSE)
```

Arguments

- **sp_obj**: Spatial object
- **number_out**: Should the output be numbers from 1:4 (FALSE by default)

Details

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.

See Also

Other geo: `bbox_scale()`, `geo_bb_matrix()`, `geo_bb()`, `reproject()`

Examples

```r
data(zones)
sp_obj <- zones
(quads <- quadrant(sp_obj))
plot(sp_obj, col = factor(quads))
points(rgeos::gCentroid(sp_obj), col = "white")
# edge cases (e.g. when using rasters) lead to NAs
sp_obj <- raster::rasterToPolygons(raster::raster(ncol = 3, nrow = 3))
(quads <- quadrant(sp_obj))
plot(sp_obj, col = factor(quads))
```
read_table_builder

Import and format Australian Bureau of Statistics (ABS) TableBuilder files

Description

Import and format Australian Bureau of Statistics (ABS) TableBuilder files

Usage

read_table_builder(dataset, filetype = "csv", sheet = 1, removeTotal = TRUE)

Arguments

dataset
Either a dataframe containing the original data from TableBuilder or a character string containing the path of the unzipped TableBuilder file.

filetype
A character string containing the filetype. Valid values are 'csv', 'legacycsv' and 'xlsx' (default = 'csv'). Required even when dataset is a dataframe. Use 'legacycsv' for csv files derived from earlier versions of TableBuilder for which csv outputs were csv versions of the xlsx files. Current csv output from TableBuilder follow a more standard csv format.

sheet
An integer value containing the index of the sheet in the xlsx file (default = 1).

removeTotal
A boolean value. If TRUE removes the rows and columns with totals (default = TRUE).

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.

Examples

data_dir <- system.file("extdata", package = "stplanr")
t1 <- read_table_builder(file.path(data_dir, "SA1Population.csv"))
if (requireNamespace("openxlsx")) {
  t2 <- read_table_builder(file.path(data_dir, "SA1Population.xlsx"),
    filetype = "xlsx", sheet = 1, removeTotal = TRUE
  )
} else {
  f <- file.path(data_dir, "SA1Population.csv")
  sa1pop <- read.csv(f, stringsAsFactors = TRUE, header = FALSE)
  t3 <- read_table_builder(sa1pop)
reproject

Reproject lat/long spatial object so that they are in units of 1m

Description

Many GIS functions (e.g. finding the area)

Usage

reproject(shp, crs = geo_select_aeq(shp))

Arguments

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

crs An optional coordinate reference system (if not provided it is set automatically by geo_select_aeq()).

See Also

Other geo: bbox_scale(), geo_bb_matrix(), geo_bb(), quadrant()

Examples

data(routes_fast)
rf_aeq <- reproject(routes_fast[1:3, ])
rf_osgb <- reproject(routes_fast[1:3, ], 27700)

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

```r
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)

rnet_boundary_df(rnet)

rnet_boundary_unique(rnet)

rnet_boundary_points_lwgeom(rnet)

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 time 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)
```
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 sfNetwork 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 LINESTRINGS share a POINT which is a boundary point for some LINESTRING(s), but not all of them (see the rnet_roundabout example);
2. two or more LINESTRINGS share a POINT which is not in the boundary of any LINESTRING (see the rnet_cycleway_intersection example).

The problem with the first example is that, according to algorithm behind SpatialLinesNetwork(), two LINESTRINGS 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 LINESTRING objects do not share two distinct boundary points. In fact, by Open Street Map standards, a roundabout is represented as a closed and circular LINESTRING, 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: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc-moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline_spatial(), overline(), plot(), SpatialLinesNetwork ANY-method.plot, sfNetwork ANY-method, rnet_group(), sln2points(), sum_network_links(), sum_network_routes()
Examples

```r
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.
rm_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.
rm_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)
)
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

rnet_cycleway_intersection

rnet_get_nodes Extract nodes from route network

Description

Extract nodes from route network

Usage

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

```r
rnet_group(rnet, ...)
```

#### 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.

Details

cluster_edge_betweenness, cluster_fast_greedy, cluster_label_prop, cluster_leading_eigen, cluster_louvain, cluster_optimal, cluster_spinglass, cluster_walktrap

Value

If the input rnet is an sf/sfc object, it returns an integer vector reporting the groups of each network element. If the input is an sfNetwork object, it returns an sfNetwork object with an extra column called rnet_group representing the groups of each network element. In the latter case, the connectivity of the spatial object is derived from the sfNetwork object.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot.sfNetwork, ANY-method, rnet_breakup_vertices(), sln2points(), sum_network_links(), sum_network_routes()

Examples

rnet <- rnet_breakup_vertices(stplanr::osm_net_example)
rnet$group <- rnet_group(rnet)
plot(rnet[["group"]]
# mapview::mapview(rnet["group"])
rnet$group_25m <- rnet_group(rnet, d = 25)
plot(rnet[["group_25m"]])
rnet$group_walktrap <- rnet_group(rnet, igraph::cluster_walktrap)
plot(rnet[["group_walktrap"]])
rnet$group_louvain <- rnet_group(rnet, igraph::cluster_louvain)
plot(rnet[["group_louvain"]])
rnet$group_fast_greedy <- rnet_group(rnet, igraph::cluster_fast_greedy)
plot(rnet[["group_fast_greedy"]])

# show sfNetwork implementation
sfn <- SpatialLinesNetwork(rnet)
sfn <- rnet_group(sfn)
plot(sfn@sl["rnet_group"])

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.

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

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

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 Only needed if from and to are empty, in which case this should be a spatial object representing desire lines
route_fun A routing function to be used for converting the straight lines to routes od2line()
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.
cl Cluster
... Arguments passed to the routing function, e.g. route_cyclestreets()

See Also

Other routes: line2routeRetry(), line2route(), route_dodgr(), route_local(), route_osrm(), route_transportapi_public()

Examples

library(sf)
l = od_data_lines[2, ]
r_walk = route(l = l, route_fun = route_osrm, osrm.profile = "foot")
r_bike = route(l = l, route_fun = route_osrm, osrm.profile = "bike")
plot(r_walk$geometry)
plot(r_bike$geometry, col = "blue", add = TRUE)

# r_bc = route(l = l, route_fun = route_bikecitizens)
# plot(r_bc)
# route(l = l, route_fun = route_bikecitizens, wait = 1)
library(osrm)
```r
r_osrm <- route(
  l = l,
  route_fun = osrmRoute,
  returnclass = "sf"
)

nrow(r_osrm)

plot(r_osrm)

sln <- stplanr::SpatialLinesNetwork(route_network_sf)

# calculate shortest paths
plot(sln)

plot(l$geometry, add = TRUE)

r_local <- stplanr::route(
  l = l,
  route_fun = stplanr::route_local,
  sln = sln
)

plot(r_local["all"], add = TRUE, lwd = 5)
```

---

**routes_fast**

*spatial lines dataset of commuter flows on the travel network*

**Description**

Simulated travel route allocated to the transport network representing the ‘fastest’ between `cents()` objects with `od2line()` (see `flow()`).

**Usage**

```r
data(routes_fast)
```

**Format**

A spatial lines dataset with 49 rows and 15 columns

**See Also**

Other example data: `destination_zones`, `flow_dests`, `flowlines`, `flow`, `route_network`, `routes_slow`
routes_slow

**spatial lines dataset of commuter flows on the travel network**

**Description**
Simulated travel route allocated to the transport network representing the 'quietest' between `cents()` objects with `od2line()` (see `flow()`).

**Usage**
```r
data(routes_slow)
```

**Format**
A spatial lines dataset 49 rows and 15 columns

**See Also**
Other example data: `destination_zones`, `flow_dests`, `flowlines`, `flow`, `route_network`, `routes_fast`

**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](https://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",
    cccode = "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
- **cccode**: 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

#### Examples

```r
route_bikecitizens()
ldf <- od_coords(stplanr::od_data_lines[2, ])
r <- route_bikecitizens(ldf)
plot(r)
```
route_cyclestreets

Plan a single route with CycleStreets.net

Description

Provides an R interface to the CycleStreets.net cycle planning API, a route planner made by cyclists for cyclists. The function returns a SpatialLinesDataFrame object representing an estimate of the fastest, quietest or most balance route. Currently only works for the United Kingdom and part of continental Europe, though other areas may be requested by contacting CycleStreets. See https://www.cyclestreets.net/api/ for more information.

Usage

route_cyclestreets(
  from, 
  to, 
  plan = "fastest", 
  silent = TRUE, 
  pat = NULL, 
  base_url = "https://www.cyclestreets.net", 
  reporterrors = TRUE, 
  save_raw = "FALSE"
)

Arguments

from Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.

to Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.

plan Text string of either "fastest" (default), "quietest" or "balanced"

silent Logical (default is FALSE). TRUE hides request sent.

pat The API key used. By default this is set to NULL and this is usually acquired automatically through a helper, api_pat().

base_url The base url from which to construct API requests (with default set to main server)

reporterrors Boolean value (TRUE/FALSE) indicating if cyclestreets (TRUE by default) should report errors (FALSE by default).

save_raw Boolean value which returns raw list from the json if TRUE (FALSE by default).
Details

This function uses the online routing service CycleStreets.net to find routes suitable for cyclists between origins and destinations. Requires an internet connection, a CycleStreets.net API key and origins and destinations within the UK (and various areas beyond) to run.

Note that if from and to are supplied as character strings (instead of lon/lat pairs), Google’s geocoding services are used via geo_code().

You need to have an api key for this code to run. Loading a locally saved copy of the api key text string before running the function, for example, will ensure it is available on any computer:

mytoken <- readLines("~/Dropbox/dotfiles/cyclestreets-api-key-rl") Sys.setenv(CYCLESTREETS = mytoken)

if you want the API key to be available in future sessions, set it using the .Renviron file with

usethis::edit_r_environ()

Read more about the .Renviron here: ?.Renviron

See Also

line2route

Examples

```r
## Not run:
from <- c(-1.55, 53.80) # geo_code("leeds")
to <- c(-1.76, 53.80) # geo_code("bradford uk")
json_output <- route_cyclestreets(from = from, to = to, plan = "quietest", save_raw = TRUE)
str(json_output) # what does cyclestreets give you?
rf_lb <- route_cyclestreets(from, to, plan = "fastest")
rf_lb@data
plot(rf_lb)
(rf_lb$length / (1000 * 1.61)) / # distance in miles
(rf_lb$time / (60 * 60)) # time in hours - average speed here: ~8mph
## End(Not run)
```

route_dodgr

Route on local data using the dodgr package

Description

Route on local data using the dodgr package

Usage

```r
route_dodgr(from = NULL, to = NULL, l = NULL, net = NULL)
```
route_google

Find shortest path using Google services

Description

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

Usage

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, e.g. route_cyclestreets()
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)

## End(Not run)
```

---

**route_local**

Plan a route with local data

Description

This function returns the shortest path between locations in, or near to, segments on a `SpatialLinesNetwork`.

Usage

```r
route_local(sln, from, to, l = NULL, ...)
```

Arguments

- `sln`: The `SpatialLinesNetwork` or `sfNetwork` to use.
- `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.
- `...`: Arguments to pass to `sum_network_links`.

See Also

Other routes: `line2routeRetry()`, `line2route()`, `route_dodgr()`, `route_osrm()`, `route_transportapi_public()`, `route()`

Examples

```r
from <- c(-1.535181, 53.82534)
to <- c(-1.52446, 53.80949)
sln <- SpatialLinesNetwork(route_network_sf)
r <- route_local(sln, from, to)
plot(sln)
plot(r$geometry, add = TRUE, col = "red", lwd = 5)
```
**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**

```r
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**  

Spatial lines dataset representing a route network

**Description**

The flow of commuters using different segments of the road network represented in the flowlines() and routes_fast() datasets.

**Usage**

```r
data(route_network)
```
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

route_osrm(
  from,
  to,
  osrm.server = "https://routing.openstreetmap.de/",
  osrm.profile = "foot"
)

Arguments

from  An object representing origins (if lines are provided as the first argument, from is assigned to 1)
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.

Examples

## Not run:
# Generate route network
route_network <- overline(routes_fast, "All", fun = sum)
route_network_sf <- sf::st_as_sf(route_network)
## End(Not run)
route_rolling_average

See Also

Other routes: line2route_retry(), line2route(), route_dodgr(), route_local(), route_transportapi_public(), route()

Examples

```r
l1 = od_data_lines[49,]
l1m = od_coords(l1)
from = l1m[,1:2]
to = l1m[,3:4]
r_foot = route_osrm(from, to)
r_bike = route_osrm(from, to, osrm.profile = "bike")
r_car = route_osrm(from, to, osrm.profile = "car")
plot(r_foot$geometry, lwd = 9, col = "grey")
plot(r_bike, col = "blue", add = TRUE)
plot(r_car, col = "red", add = TRUE)
```

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

```r
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

```r
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

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_rolling_average(), 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(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)
abline(h = mean(y))
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)

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_slope_matrix(m)

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)

rg <- r1$rolling_gradient
rg[is.na(rg)] <- 0
plot(c(0, d), c(0, rg), ylim = c(0, 0.2))
points(c(0, d), c(0, r1$rolling_gradient3), col = "blue")
points(c(0, d), c(0, r1$rolling_gradient4), col = "grey")
par(mfrow = c(1, 1))
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_vector()

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()
route_split

Split route in two at point on or near network

Description
Split route in two at point on or near network

Usage
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 features

Examples
```
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
route_split_id(r, id = NULL, p = NULL)
route_transportapi_public

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)
```

---

Plan a single route with TransportAPI.com

Description

Provides an R interface to the TransportAPI.com public transport API. The function returns a SpatialLinesDataFrame object representing the public route. Currently only works for the United Kingdom. See https://developer.transportapi.com/documentation for more information.

Usage

```r
route_transportapi_public(
  from,
  to,
  silent = FALSE,
  region = "southeast",
  modes = NA,
  not_modes = NA
)
```

Arguments

from Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth.
to Text string or coordinates (a numeric vector of length = 2 representing latitude and longitude) representing a point on Earth. This represents the destination of the trip.
silent Logical (default is FALSE). TRUE hides request sent.
region String for the active region to use for journey plans. Possible values are ’southeast’ (default) or ’tfl’.
modes Vector of character strings containing modes to use. Default is to use all modes.
not_modes Vector of character strings containing modes not to use. Not used if modes is set.

Details

This function uses the online routing service TransportAPI.com to find public routes between origins and destinations. It does not require any key to access the API.

Note that if from and to are supplied as character strings (instead of lon/lat pairs), Google’s geocoding services are used via geo_code.

Note: there is now a dedicated transportAPI package: https://github.com/ITSLeeds/transportAPI

See Also

line2route

Other routes: line2routeRetry(), line2route(), route_dodgr(), route_local(), route_osrm(), route()

Examples

```r
## Not run:
# Plan the 'public' route from Hereford to Leeds
rqh <- route_transportapi_public(from = "Hereford", to = "Leeds")
plot(rqh)
## End(Not run)

# Aim plan public transport routes with transportAPI
```

---

sfNetwork-class

An S4 class representing a (typically) transport network

Description

This class uses a combination of a sf layer and an igraph object to represent transport networks that can be used for routing and other network analyses.

Slots

- sl A sf line layer with the geometry and other attributes for each link the in network.
- g The graph network corresponding to sl.
- nb A list containing vectors of the nodes connected to each node in the network.
- weightfield A character vector containing the variable (column) name from the SpatialLines-DataFrame to be used for weighting the network.
sln2points
Generate spatial points representing nodes on a SpatialLinesNetwork or sfNetwork.

Description
Generate spatial points representing nodes on a SpatialLinesNetwork or sfNetwork.

Usage
sln2points(sln)

Arguments
sln The SpatialLinesNetwork or sfNetwork to use.

See Also
Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sum_network_links(), sum_network_routes()

Examples
data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
(sln_nodes <- sln2points(sln))
plot(sln)
plot(sln_nodes, add = TRUE)

sln_add_node
Add node to spatial lines object

Description
Add node to spatial lines object

Usage
sln_add_node(sln, p)

Arguments
sln A spatial lines (sfNetwork) object created by SpatialLinesNetwork
p A point represented by an sf object the will split the route
Examples

```r
sample_routes <- routes_fast_sf[2:6, NULL]
sample_routes$value <- rep(1:3, length.out = 5)
chunk <- overline2(sample_routes, attrib = "value")
sln <- SpatialLinesNetwork(chunk)
p <- sf::st_sfc(sf::st_point(c(-1.540, 53.826)), crs = sf::st_crs(chunk))
sln_nodes <- sln2points(sln)
sln_new <- sln_add_node(sln, p)
route <- route_local(sln_new, p, sln_nodes[9, ])
plot(sln)
plot(sln_nodes, pch = as.character(1:nrow(sln_nodes)), add = TRUE)
plot(route$geometry, lwd = 9, add = TRUE)
```

---

`sln_clean_graph`  
Clean spatial network - return an sln with a single connected graph

**Description**

See https://github.com/ropensci/stplanr/issues/344

**Usage**

`sln_clean_graph(sln)`

**Arguments**

- `sln`  
  A spatial lines (sfNetwork) object created by SpatialLinesNetwork

**Value**

An sfNetwork object

---

`SpatialLinesNetwork`  
Create object of class SpatialLinesNetwork or sfNetwork

**Description**

Creates a new SpatialLinesNetwork (for SpatialLines) or sfNetwork (for sf) object that can be used for routing analysis within R.

**Usage**

`SpatialLinesNetwork(sl, uselonglat = FALSE, tolerance = 0)`
Arguments

- **sl** A SpatialLines or SpatialLinesDataFrame containing the lines to use to create the network.
- **uselonglat** A boolean value indicating if the data should be assumed to be using WGS84 latitude/longitude coordinates. If FALSE or not set, uses the coordinate system specified by the SpatialLines object.
- **tolerance** A numeric value indicating the tolerance (in the units of the coordinate system) to use as a tolerance with which to match nodes.

Details

This function is used to create a new SpatialLinesNetwork from an existing SpatialLines or SpatialLinesDataFrame object. A typical use case is to represent a transport network for routing and other network analysis functions. This function and the corresponding SpatialLinesNetwork class is an implementation of the SpatialLinesNetwork developed by Edzer Pebesma and presented on RPubs. The original implementation has been rewritten to better support large (i.e., detailed city-size) networks and to provide additional methods useful for conducting transport research following on from the initial examples provided by Janoska (2013).

References


See Also

Other rnet: `calc_catchment_sum()`, `calc_catchment()`, `calcMoving_catchment()`, `calc_network_catchment()`, `find_network_nodes()`, `gsection()`, `islines()`, `lineLabels()`, `overline_spatial()`, `overline()`, `plot`, `SpatialLinesNetwork`, `ANY-method.plot`, `sfNetwork`, `ANY-method.rnet_breakup_vertices()`, `rnet_group()`, `sln2points()`, `sum_network_links()`, `sum_network_routes()`

Examples

```r
# dont test due to issues with s2 dependency
sln_sf <- SpatialLinesNetwork(route_network_sf)
plot(sln_sf)
shortpath <- sum_network_routes(sln_sf, 1, 50, sumvars = "length")
plot(shortpath$geometry, col = "red", lwd = 4, add = TRUE)
```
SpatialLinesNetwork-class

An S4 class representing a (typically) transport network

Description

This class uses a combination of a SpatialLinesDataFrame and an igraph object to represent transport networks that can be used for routing and other network analyses.

Slots

- sl  A SpatialLinesDataFrame with the geometry and other attributes for each link the in network.
- g  The graph network corresponding to sl.
- nb  A list containing vectors of the nodes connected to each node in the network.
- weightfield  A character vector containing the variable (column) name from the SpatialLinesDataFrame to be used for weighting the network.

stplanr-deprecated

Deprecated functions in stplanr

Description

These functions are depreciated and will be removed:

summary,sfNetwork-method

Print a summary of a sfNetwork

Description

Print a summary of a sfNetwork

Usage

## S4 method for signature 'sfNetwork'
summary(object, ...)

Arguments

- object  The sfNetwork
- ...  Arguments to pass to relevant summary function.
Examples

data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
summary(sln)

summary, SpatialLinesNetwork-method

Print a summary of a SpatialLinesNetwork

Description

Print a summary of a SpatialLinesNetwork

Usage

## S4 method for signature 'SpatialLinesNetwork'
summary(object, ...)

Arguments

object The SpatialLinesNetwork
...
Arguments to pass to relevant summary function.

Examples

data(routes_fast)
rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
summary(sln)

sum_network_links Summarise links from shortest paths data

Description

Summarise links from shortest paths data

Usage

sum_network_links(sln, routedata)
sum_network_routes

Arguments

sln                        The SpatialLinesNetwork or sfNetwork to use.

routedata                  A dataframe where the first column contains the Node ID(s) of the start of the
                           routes, the second column indicates the Node ID(s) of the end of the routes,
                           and any additional columns are summarised by link. If there are no additional
                           columns, then overlapping routes are counted.

Details

Find the shortest path on the network between specified nodes and returns a SpatialLinesDataFrame
or sf containing the path(s) and summary statistics of each one.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc Moving_catchment(),
calc_network catchment(), find_network_nodes(), gsection(), islines(), lineLabels(),
overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method,
rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_routes()

Examples

sln_sf <- SpatialLinesNetwork(route_network_sf)
plot(sln_sf)
nodes_df <- data.frame(
  start = rep(c(1, 2, 3, 4, 5), each = 4),
  end = rep(c(50, 51, 52, 33), times = 5)
)
weightfield(sln_sf) # field used to determine shortest path
library(sf)
shortpath_sf <- sum_network_links(sln_sf, nodes_df)
plot(shortpath_sf["count"], lwd = shortpath_sf$count, add = TRUE)
Arguments

sln  The SpatialLinesNetwork or sfNetwork to use.
start  Integer of node indices where route starts.
end  Integer of node indices where route ends.
sumvars  Character vector of variables for which to calculate summary statistics. The default value is weightfield(sln).
combinations  Boolean value indicating if all combinations of start and ends should be calculated. If TRUE then every start Node ID will be routed to every end Node ID. This is faster than passing every combination to start and end. Default is FALSE.

Details

Find the shortest path on the network between specified nodes and returns a SpatialLinesDataFrame (or an sf object with LINESTRING geometry) containing the path(s) and summary statistics of each one.

The start and end arguments must be integers representing the node index. To find which node is closest to a geographic point, use find_nearest_node().

If the start and end node are identical, the function will return a degenerate line with just two (identical) points. See #444.

See Also

Other rnet: SpatialLinesNetwork, calc_catchment_sum(), calc_catchment(), calc_moving_catchment(), calc_network_catchment(), find_network_nodes(), gsection(), islines(), lineLabels(), overline_spatial(), overline(), plot, SpatialLinesNetwork, ANY-method, plot, sfNetwork, ANY-method, rnet_breakup_vertices(), rnet_group(), sln2points(), sum_network_links()

Examples

sln <- SpatialLinesNetwork(route_network)
weightfield(sln)  # field used to determine shortest path
shortpath <- sum_network_routes(sln, start = 1, end = 50, sumvars = "length")
plot(shortpath, col = "red", lwd = 4)
plot(sln, add = TRUE)

# with sf objects
sln <- SpatialLinesNetwork(route_network_sf)
weightfield(sln)  # field used to determine shortest path
shortpath <- sum_network_routes(sln, start = 1, end = 50, sumvars = "length")
plot(sf::st_geometry(shortpath), col = "red", lwd = 4)
plot(sln, add = TRUE)

# find shortest path between two coordinates
sf::st_bbox(sln@sln)
start_coords <- c(-1.546, 53.826)
end_coords <- c(-1.519, 53.816)
plot(sln)
plot(sf::st_point(start_coords), cex = 3, add = TRUE, col = "red")
plot(sf::st_point(end_coords), cex = 3, add = TRUE, col = "blue")
nodes <- find_network_nodes(sln, rbind(start_coords, end_coords))
shortpath <- sum_network_routes(sln, nodes[1], nodes[2])
plot(sf::st_geometry(shortpath), col = "darkred", lwd = 3, add = TRUE)

# degenerate path
sum_network_routes(sln, start = 1, end = 1)

toptailgs

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. Uses the geosphere::distHaversine function and requires coordinates in WGS84 (lng/lat).

Usage

toptailgs(l, toptail_dist, tail_dist = NULL)

Arguments

l
A SpatialLines object
toptail_dist
The distance (in metres) to top the line by. Can be either a single value or a vector of the same length as the SpatialLines object. If tail_dist is missing, is used as the tail distance.
tail_dist
The distance (in metres) to tail the line by. Can be either a single value or a vector of the same length as the SpatialLines object.

See Also

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

Examples

data("routes_fast")
rf <- routes_fast[2:3, ]
r_toptail <- toptailgs(rf, toptail_dist = 300)
plot(rf, lwd = 3)
plot(r_toptail, col = "red", add = TRUE)
plot(cents, add = TRUE)
toptail_buff

Clip the beginning and ends SpatialLines to the edge of SpatialPolygon borders

Description

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

Usage

toptail_buff(l, buff, ...)

Arguments

l

An sf LINESTRING object

buff

An sf POLYGON object to act as the buffer

... Argument passed to rgeos::gBuffer()

See Also

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

Examples

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)

update_line_geometry

Update line geometry

Description

Take two SpatialLines objects and update the geometry of the former with that of the latter, retaining the data of the former.

Usage

update_line_geometry(l, nl)
Arguments

l  A SpatialLines object, whose geometry is to be modified
nl  A SpatialLines object of the same length as l to provide the new geometry

See Also

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

Examples

data(flowlines)
l <- flowlines[2:5,]
nl <- routes_fast
nrow(l)
nrow(nl)
l <- l[!is_linepoint(l),]
names(l)
names(routes_fast)
l_newgeom <- update_line_geometry(l, nl)
plot(l, lwd = l$All / mean(l$All))
plot(l_newgeom, lwd = l$All / mean(l$All))
names(l_newgeom)

weightfield  Get or set weight field in SpatialLinesNetwork

Description

Get or set value of weight field in SpatialLinesNetwork

Usage

weightfield(x)

weightfield(x, varname) <- value

weightfield(x, varname) <- value

## S4 method for signature 'SpatialLinesNetwork'
weightfield(x)

## S4 method for signature 'sfNetwork'
weightfield(x)

## S4 replacement method for signature 'SpatialLinesNetwork,ANY'
writeGeoJSON

weightfield(x) <- value

## S4 replacement method for signature 'sfNetwork,ANY'
weightfield(x) <- value

## S4 replacement method for signature 'SpatialLinesNetwork,character'
weightfield(x, varname) <- value

## S4 replacement method for signature 'sfNetwork,character'
weightfield(x, varname) <- value

Arguments

x SpatialLinesNetwork to use
varname The name of the variable to set/use.
value Either the name of the variable to use as the weight field or a dataframe or vector containing the weights to use if varname is passed to the replacement function. If the dataframe contains multiple columns, the column with the same name as varname is used, otherwise the first column is used.

Details

These functions manipulate the value of weightfield in a SpatialLinesNetwork. When changing the value of weightfield, the weights of the graph network are updated with the values of the corresponding variables.

Examples

# with sp objects
data(routes_fast)
  rnet <- overline(routes_fast, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
  weightfield(sln) <- "length"
  weightfield(sln, "randomnum") <- sample(1:10, size = nrow(sln@sl), replace = TRUE)
data(routes_fast_sf)
  rnet <- overline(routes_fast_sf, attrib = "length")
sln <- SpatialLinesNetwork(rnet)
  weightfield(sln) <- "length"
  sln@sl$randomnum <- sample(1:10, size = nrow(sln@sl), replace = TRUE)
  weightfield(sln) <- "randomnum"
# todo: show the difference that it makes

writeGeoJSON

Write to geojson easily

Description

Provides a user-friendly wrapper for sf::st_write(). Note, geojson_write from the geojsonio package provides the same functionality https://github.com/ropensci/geojsonio.
Usage

writeGeoJSON(shp, filename)

Arguments

shp Spatial data object
filename File name of the output geojson

zones Spatial polygons of home locations for flow analysis.

Description

Note: we recommend using the zones_sf data.

Details

These correspond to the cents_sf data.

• geo_code. the official code of the zone

Examples

library(sf)
zones_sf
plot(zones_sf)
Index

* datasets
  ca_local, 15
  cents, 16
  destination_zones, 17
  flow, 21
  flow_dests, 22
  flowlines, 22
  l_poly, 42
  od_data_lines, 53
  od_data_routes, 54
  od_data_sample, 54
  osm_net_example, 61
  rnet_cycleway_intersection, 75
  rnet_overpass, 78
  rnet_roundabout, 78
  route_network, 87
  routes_fast, 80
  routes_slow, 81
  zones, 108

* data
  read_table_builder, 70

* example data
  destination_zones, 17
  flow, 21
  flow_dests, 22
  flowlines, 22
  route_network, 87
  routes_fast, 80
  routes_slow, 81

* geo
  bbox_scale, 7
  geo_bb, 23
  geo_bb_matrix, 24
  quadrant, 69
  reproject, 71

* lines
  angle_diff, 5
  geo_toptail, 29
  is_linepoint, 31
  line2df, 32
  line2points, 33
  line_bearing, 37
  line_breakup, 38
  line_midpoint, 39
  line_sample, 40
  line_segment, 40
  line_via, 41
  mats2line, 43
  n_sample_length, 45
  n_vertices, 46
  onewaygeo, 59
  points2line, 67
  toptail_buff, 105
  toptailgs, 104
  update_line_geometry, 105

* nodes
  geo_code, 26
  nearest_google, 44

* od
  dist_google, 18
  od2line, 47
  od2odf, 48
  od_aggregate_from, 50
  od_aggregate_to, 51
  od_coords, 52
  od_coords2line, 52
  od_dist, 54
  od_id, 55
  od_oneway, 57
  od_to_odmatrix, 59
  odmatrix_to_od, 49
  points2flow, 67
  points2odf, 68

* package
  stplanr-package, 5

* rnet
  calc_catchment, 7
  calc_catchment_sum, 10
calc_moveingcatchment, 12
calc_networkcatchment, 13
find_network_nodes, 19
gsection, 30
islines, 31
lineLabels, 36
overline, 61
overline_spatial, 65
plot, sfNetwork, ANY-method, 65
plot, SpatialLinesNetwork, ANY-method, 66
rnet_breakup_vertices, 73
rnet_group, 76
sln2points, 97
SpatialLinesNetwork, 98
sum_network_links, 101
sum_network_routes, 102
* route funs
    route_average_gradient, 81
    route_rolling_average, 89
    route_rolling_diff, 90
    route_rolling_gradient, 91
    route_sequential_dist, 92
    route_slope_matrix, 92
    route_slope_vector, 93
* routes
    line2route, 34
    line2routeRetry, 35
    route, 78
    route_dodgr, 84
    route_local, 86
    route_osrm, 88
    route_transportapi_public, 95
angle_diff, 5, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
as_sf_fun, 6
as_sp_fun (as_sf_fun), 6
bb2poly (geo_bb), 23
bbox_scale, 7, 24, 69, 71
bearing(), 6, 37
call_local, 15
calc_catchment, 7, 11, 13, 15, 20, 30, 31, 36, 63, 65, 73, 77, 97, 99, 102, 103
calc_catchment(), 5
calc_catchment_sum, 9, 10, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
calc_movingcatchment, 9, 11, 12, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
calc_networkcatchment, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
cents, 16
cents(), 21, 47, 48, 50, 51, 55, 80, 81
cents_sf (cents), 16
destination_zones, 17, 21–23, 80, 81, 88
destinations (destination_zones), 17
destinations_sf (destination_zones), 17
dist_google, 18, 48–53, 55, 56, 58, 59, 67, 68
destinationzones, 17, 21–23, 80, 81, 88
destinations (destination_zones), 17
destinations_sf (destination_zones), 17
dist_google, 18, 48–53, 55, 56, 58, 59, 67, 68
find_network_nodes, 9, 11, 13, 15, 19, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
flow, 17, 21, 22, 23, 80, 81, 88
flow(), 22, 47, 48, 50, 51, 55, 80, 81
flow_dests, 17, 21, 22, 23, 80, 81, 88
flowlines, 17, 21, 22, 23, 80, 81, 88
flowlines(), 58, 59, 87
flowlines_sf (flowlines), 22
ggeo_bb, 7, 23, 24, 69, 71
ggeo_bb_matrix, 7, 24, 24, 69, 71
ggeo_buffer, 25
ggeo_code, 26, 45
ggeo_length, 27
ggeo_projected, 27
ggeo_select_aeq, 28
ggeo_select_aeq(), 27, 71
ggeo_toptail, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
gprojected (geo_projected), 27
gsection, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
iggraph::as_undirected(), 77
iggraph::clusters(), 77
is_linepoint, 6, 29, 31, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
islines, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
l_poly, 42
line2df, 6, 29, 31, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
INDEX

line2points, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
line2pointsn(line2points), 33
line2route, 34, 36, 79, 85, 86, 89, 96
line2route(), 36
line2routeRetry, 35, 35, 79, 85, 86, 89, 96
line2vertices(line2points), 33
line_bearing, 6, 29, 32, 33, 37, 38–41, 43, 45, 46, 60, 68, 104–106
line_breakup, 6, 29, 32, 33, 37, 38–41, 43, 45, 46, 60, 68, 104–106
line_length, 39
line_midpoint, 6, 29, 32, 33, 37, 38, 39, 40, 41, 43, 45, 46, 60, 68, 104–106
line_sample, 6, 29, 32, 33, 37–39, 40, 41, 43, 45, 46, 60, 68, 104–106
line_segment, 6, 29, 32, 33, 37–40, 40, 41, 43, 45, 46, 60, 68, 104–106
line_via, 6, 29, 32, 33, 37–41, 41, 43, 45, 46, 60, 68, 104–106
lineLabels, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
mats2line, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
n_sample_length, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
n_vertices, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 68, 104–106
nearest_cyclestreets, 43
nearest_google, 26, 44
od2line, 19, 47, 49–53, 55, 56, 58, 59, 67, 68
od2line(), 22, 34, 54, 79–81
od2line2(od2line), 47
od2odf, 19, 48, 49, 49–53, 55, 56, 58, 59, 67, 68
od2odf(), 47
od_aggregate_from, 19, 48, 49, 50, 51–53, 55, 56, 58, 59, 67, 68
od_aggregate_to, 19, 48–50, 51, 52, 53, 55, 56, 58, 59, 67, 68
od_coords, 19, 48–51, 52, 53, 55, 56, 58, 59, 67, 68
od_coords2line, 19, 48–52, 52, 55, 56, 58, 59, 67, 68
od_data_lines, 53
od_data_routes, 54
od_data_sample, 54
od_dist, 19, 48–53, 54, 56, 58, 59, 67, 68
od_id, 19, 48–53, 55, 55, 58, 59, 67, 68
od_id_character(od_id), 55
od_id_max_min(od_id), 55
od_id_max_min(), 58
od_id_order, 56
od_id_szudzik(od_id), 55
od_id_szudzik(), 58
od_oneway, 19, 48–53, 55, 56, 57, 59, 67, 68
od_to_odmatrix, 19, 48–53, 55, 56, 58, 59, 67, 68
odmatrix_to_od, 19, 48, 49, 49, 50–53, 55, 56, 58, 69, 67, 68
onewaygeo, 6, 29, 32, 33, 37–41, 43, 45, 46, 59, 68, 104–106
osm_net_example, 61
osrm::osrmRoute(), 88
overline, 9, 11, 13, 15, 20, 30, 31, 36, 61, 65, 66, 73, 77, 97, 99, 102, 103
overline(), 5, 31
overline2(overline), 61
overline_intersection, 64
overline spatial, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103
plot, sfNetwork, ANY-method, 65
plot, SpatialLinesNetwork, ANY-method, 66
points2flow, 19, 48–53, 55, 56, 58, 59, 67, 68
points2line, 6, 29, 32, 33, 37–41, 43, 45, 46, 60, 67, 104–106
points2odf, 19, 48–53, 55, 56, 58, 59, 67, 68
quadrant, 7, 24, 69, 71
read_table_builder, 70
reproject, 7, 24, 69, 71
rnet_add_node, 71
rnet_boundary_df
(rnet_boundary_points), 72
rnet_boundary_points, 72
rnet_boundary_points_lwgeom
(rnet_boundary_points), 72
rnet_boundary_unique
(rnet_boundary_points), 72
rnet_breakup_vertices, 9, 11, 13, 15, 20, 30, 31, 36, 63, 65, 66, 73, 77, 97, 99, 102, 103