Package 'traipse'

October 14, 2022

Title Shared Tools for Tracking Data

Version 0.3.0

Description A collection of commonly used tools for animal movement and other tracking data. Variously distance, angle, bearing, distance-to, bearing-to and speed are provided for geographic data that can be used directly or within 'tidyverse' syntax. Distances and bearings are calculated using modern geodesic methods as provided by Charles F. F. Karney (2013) <doi:10.1007/s00190-012-0578-z> via the 'geodist' and 'geosphere' packages.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Depends R (>= 2.10)

Imports magrittr, geodist, geosphere, stats

URL https://github.com/Trackage/traipse

BugReports https://github.com/Trackage/traipse/issues

Suggests covr, testthat (>= 2.1.0), dplyr, tibble, tidyr, spelling

Language en-US

NeedsCompilation no

Author Michael Sumner [aut, cre] (<https://orcid.org/0000-0002-2471-7511>)

Maintainer Michael Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2022-10-10 07:40:02 UTC

R topics documented:

`track_angle` *Track angle*

Description

Calculate internal track angle on longitude, latitude input vectors. The unit of angle is degrees.

Usage

```
track_angle(x, y)
```

Arguments

`x`	longitude
`y`	latitude

Details

By convention the first and last values are set to `NA` missing value, because the angle applies to the location between each previous and next location.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(angle = track_angle(lon, lat))`.

The maximum possible value is 180 and the minimum is 0.

Value

a numeric vector of the relative internal angle between sequential locations in degrees, see Details

Examples

```
track_angle(trips0$x, trips0$y)[1:10]

## maximum value
track_angle(c(0, 0, 0), c(0, 1, 2))
## minimum value
track_angle(c(0, 0, 0), c(0, 1, 0))
```

track_bearing *Track bearing*

Description

Calculate sequential bearing on longitude, latitude input vectors. The unit of bearing is degrees.

Usage

```
track_bearing(x, y)
```

Arguments

x	longitude
y	latitude

Details

By convention the last value is set to NA missing value, because the bearing applies to the segment extending from the current location.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(turn = track_bearing(lon, lat))`.

Absolute bearing is relative to North (0), and proceeds clockwise positive and anti-clockwise negative `N = 0, E = 90, S = +/-180, W = -90`.

The last value will be NA as the bearing is relative to the first point of each segment.

Value

a numeric vector of absolute bearing in degrees, see Details

Examples

```
track_bearing(trips0$x, trips0$y)[1:10]
```

track_bearing_to *Track bearing to location/s*

Description

Calculate geodesic bearing to a location or locations based on longitude, latitude (from) input vectors and longitude, latitude (to) input vectors. The unit of bearing is degrees. The *to* values may be a single value or individual to each *from* location.

Usage

```
track_bearing_to(x, y, to_x, to_y)
```

Arguments

x	longitude
y	latitude
to_x	longitude vector of *to* location/s
to_y	latitude vector of *to* locations/s

Details

No missing values are required as padding, but input data with NAs will incur an NA in the output.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(bearing_to = track_bearing_to(lon, lat, to_lon, to_lat))`.

Absolute bearing is relative to North (0), and proceeds clockwise positive and anti-clockwise negative `N = 0, E = 90, S = +/-180, W = -90`.

There is no NA padding in the output value (though missing values in the input will be mirrored in the output).

Value

a numeric vector of absolute bearing-to in degrees, see Details

Examples

```
track_bearing_to(trips0$x, trips0$y, to_x = 147, to_y = -42)[1:10]
# N E S W
track_bearing_to(0,0, c(0, 10, 0, -10), c(5, 0, -5, 0))

# maximum and minimum value are the same direction (due south)
track_bearing(c(0, -0.00001), c(0, -1))
track_bearing(c(0,  0.00001), c(0, -1))

# the absolute minimum is north
track_bearing(c(0, 0), c(0, 1))
```

---

track_distance *Track distance*

Description

Calculate geodesic distance on longitude, latitude input vectors. The unit of distance is metres.

Usage

```
track_distance(x, y)
```

Arguments

x	longitude
y	latitude

Details

By convention the first value is set to NA missing value, because the distance applies to each sequential pair of locations.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(distance = track_distance(lon, lat))`

Value

numeric vector of distances between sequential pairs of x, y in metres, see Details

Examples

```
track_distance(trips0$x, trips0$y)[1:10]
```

---

track_distance_to *Track distance to location/s*

Description

Calculate geodesic distance to a location or locations based on longitude, latitude (from) input vectors and longitude, latitude (to) input vectors. The unit of distance is metres. The *to* values may be a single value or individual to each *from* location.

Usage

```
track_distance_to(x, y, to_x, to_y)
```

Arguments

x	longitude
y	latitude
to_x	longitude vector of *to* location/s
to_y	latitude vector of *to* locations/s

Details

No missing values are required as padding, but input data with NAs will incur an NA in the output.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(distance = track_distance_to(lon, lat, to_lon, to_lat))`

Value

a numeric vector of distance-to values in metres

Examples

```
track_distance_to(trips0$x, trips0$y, to_x = 147, to_y = -42)[1:10]
```

`track_grid` *Track grid*

Description

Computes the cell a track location point falls in on a grid.

Usage

```
track_grid(x, y, dimension, extent = NULL)
```

Arguments

`x`	longitude or x
`y`	latitude or y
`dimension`	grid size 'nx', 'ny' 2 element vector (ncol, nrow)
`extent`	grid extent, if not supplied we use the range of the data input

Details

A grid is defined by a 'dimension' ('ncol', 'nrow') and 'extent' ('xmin', 'xmax', 'ymin', 'ymax'). The cell index returned is in 'raster order', this is by top row, left to right and down as per 'rasterImage'. This is aligned with usage in the Github organization 'hypertidy' packages 'vaster' and 'ximage', and is how other raster packages work.

This function doesn't care if the x,y input values are longitude latitude or x, y and it makes no difference at all. No account of movement between points is made.

Value

cell index of each input point in the grid specification

Examples

```
dimension <- c(50, 35)
extent <- c(range(trips0$x), range(trips0$y))
cells <- track_grid(trips0$x, trips0$y, dimension = dimension, extent = extent)
plot(extent[1:2], extent[3:4], asp = 1, type = "n")
tab <- tabulate(cells, nbin = prod(dimension))
rasterImage(matrix(1 - (tab/max(tab)), dimension[2L], byrow = TRUE),
extent[1L], extent[3L], extent[2L], extent[4L], interpolate = FALSE)
points(trips0$x, trips0$y, pch = ".", col = "firebrick")
```

`track_intermediate` *Track intermediate points*

Description

Calculate great circle intermediate points on longitude, latitude input vectors. A spherical model is used, from the geosphere package.

Usage

```
track_intermediate(x, y, date = NULL, distance = NULL, duration = NULL)
```

Arguments

`x`	longitude
`y`	latitude
`date`	optional input date-time in POSIXct
`distance`	optional minimum distance (metres) between interpolated points
`duration`	optional minimum duration (seconds) between interpolated point, if set then `distance` must be `NULL` and `date` must be input

Details

This function returns a list of data frames, with a data frame of interpolated locations for every interval between input locations. There is a final empty data frame to ensure the list is the same length as the inputs. See embedded usage of the tidyr function 'unnest()' for ease of use.

To use on multiple track ids, use a grouped data frame with tidyverse code like `inter <- data %>% group_by(id) %>% mutat`

Then, un-nest this result for further use (the 'inter' above retains the information about the parent locations for custom usage if needed), so the final location of each group has invalid intermediates: `dd <- inter %>% slice(-1) %>% unnest()`

Value

a list of data frames of intermediate points (for use with `unnest()` from tidyr)

Examples

```
track_intermediate(trips0$x[1:10], trips0$y[1:10], distance = 15000)

track_intermediate(trips0$x[1:10], trips0$y[1:10], date = trips0$date,
                                  distance = 1500)

inter_time <- track_intermediate(trips0$x[1:10], trips0$y[1:10],
                             date = trips0$date, duration = 1800)
```

`track_query` *Query track data for arbitrary locations*

Description

Latent positions may be queried using arbitrary date-time values. The only method (for now) is 'linear', but default should be 'geodesic'. In time we include more methods to match the GeoPandas implementation.

Usage

```
track_query(x, y, date = NULL, query, type = "linear")
```

Arguments

`x`	longitude
`y`	latitude
`date`	date-time in POSIXct (or can be ignore, for relative index-time)
`query`	required argument, date-time values to return inferred x, y positions for
`type`	linear, geodesic, rhumb, forward, backward, nearest (also need open/closed intervals)

Details

If `date` is not included, time itself is treated as the obvious index on n-locations so simple relative time, and `query` is expected to match this.

We use group_modify to keep the id groups: `trips0 %>% group_by(id) %>% group_modify(~track_query(.x$x, .x$y, query = c(4.5, 6.7)))`

Value

data frame of 'x,y,date' of inferred positions

Examples

```
track_query(trips0$x[1:10], trips0$y[1:10], query = c(4.5, 5.5, 6.5))
track_query(trips0$x[1:10], trips0$y[1:10], trips0$date[1:10], query = trips0$date[1:10] + 10)
s <- seq(min(trips0$date), max(trips0$date), by = "1 hour")
```

track_speed *Track speed*

Description

Calculate speed (m/s) based on geodesic distance with longitude, latitude, date-time input vectors. The unit of speed is metres per second.

Usage

```
track_speed(x, y, date)
```

Arguments

x	longitude
y	latitude
date	date-time in POSIXct

Details

By convention the first value is set to NA missing value, because the difference applies to each sequential pair of locations.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(speed = track_speed(lon, lat, date))`

Value

numeric vector of sequential distances in metres per second, see Details

Examples

```
track_speed(trips0$x, trips0$y, trips0$date)[1:10]
```

track_time *Track time duration*

Description

Calculate time duration based on sequential difference of date-time input. The unit of time duration is seconds.

Usage

```
track_time(date)
```

Arguments

date	date-time in POSIXct

Details

By convention the first value is set to NA missing value, because the difference applies to each sequential pair of locations.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(duration = track_time(date))`

Value

numeric vector of duration between sequential date-time values in seconds, see Details

Examples

```
track_time(trips0$date)[1:10]
```

---

track_turn *Track turn angle*

Description

Calculate relative track turning angle on longitude, latitude input vectors. The unit of turn angle is degrees.

Usage

```
track_turn(x, y)
```

Arguments

x	longitude
y	latitude

Details

By convention the last value is set to NA missing value, because the angle applies to the relative turn from the current location.

To use this on multiple track ids, use a grouped data frame with tidyverse code like `data %>% group_by(id) %>% mutate(turn = track_turn(lon, lat))`.

The maximum possible value is 180 degrees and the minimum is -180, although these particular values are a special case and will probably always be positive. Turn angle is a signed quantity with negative values for a left turn and positive values for a right turn.

Value

a numeric vector of absolute turn angles, in degrees

Examples

```
track_turn(trips0$x, trips0$y)[1:10]

## maximum turn angle
track_turn(c(0, 0, 0), c(0, 1, 0))
## minimum turn angle
track_turn(c(0, 0, 0), c(0, 1, 2))
```

---

trips0 *Simulated track data*

Description

`trips0` is an ungrouped data frame of x, y, date, id

Index