Package: traipse (via r-universe)

November 5, 2026

Title Shared Tools for Tracking Data

Version 0.4.0.9000

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 'geographiclib' package.

License GPL-3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Depends R (>= 2.10)

Imports geodist, geographiclib, methods, stats

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

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

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

Language en-US

Config/roxygen2/version 8.1.0

Repository https://australianantarcticdivision.r-universe.dev

Date/Publication 2026-10-26 06:17:28 UTC

RemoteUrl https://github.com/AustralianAntarcticDivision/traipse

RemoteRef HEAD

RemoteSha 1a88384615203d58319ffd437f1b1fa942ca2771

Contents

as_track . . . 2
TRACK . . . 3
track_angle . . . 4
track_bearing . . . 5
track_bearing_to . . . 6
track_cells . . . 7
track_distance . . . 8
track_distance_to . . . 9
track_fixes . . . 9
track_grid . . . 10
track_intermediate . . . 11
track_metrics . . . 12
track_query . . . 13
track_speed . . . 14
track_speedfilter . . . 14
track_time . . . 16
track_turn . . . 16
trips0 . . . 17

Index **18**

`as_track` *Convert to a track model*

Description

`as_track()` makes a `TRACK()` from a data frame or a 'trip' object.

Usage

```
as_track(x, ...)

## S3 method for class 'TRACK'
as_track(x, ...)

## S3 method for class 'data.frame'
as_track(x, ..., xcol = "x", ycol = "y", date = "date", id = "id")

## S3 method for class 'trip'
as_track(x, ...)
```

Arguments

`x`	a data frame, or a 'trip' object
`...`	passed to `TRACK()` (`crs`, `longlat`, `segment`, `policy`)
`xcol`, `ycol`, `date`, `id`	names of the columns

Details

For a data frame, give the names of the coordinate, date-time and id columns; every other column is kept as fix data. For a 'trip' object the columns are read from the object, without loading 'sp'.

Value

A track model, see `TRACK()`.

Examples

```
as_track(trips0)
```

TRACK *Track model*

Description

`TRACK()` builds a track model from columns of coordinates, date-times and track ids. A track model is a list of keyed tables: `tracks` (one row per track, key `.track`) and `fixes` (one row per location, key `.fix`, ordered by date-time within track). Segments between consecutive fixes are derived on demand with `track_segments()`. Ids are 0-based, as in the mesh models of 'meshcore'.

Usage

```
TRACK(
  x,
  y,
  date,
  id,
  data = NULL,
  crs = NA_character_,
  longlat = NULL,
  segment = NULL,
  policy = c("error", "repair")
)
```

Arguments

`x`, `y`	coordinates
`date`	date-times (converted with `as.POSIXct()`)
`id`	track ids (a single value is used for every fix)
`data`	optional data frame of other columns, one row per fix
`crs`	coordinate reference system, carried as a string
`longlat`	logical, are `x` and `y` longitude and latitude? See Details.
`segment`	`"geodesic"` or `"planar"`, see Details
`policy`	`"error"` or `"repair"`, see Details

Details

A fix is identified by its track and its date-time, not by its position: two fixes at the same coordinates are different fixes. This is recorded as `rung = "sequential"` in `meta`.

The rules are those of the 'trip' package: no missing values in the coordinates, date-times or ids, records ordered by date-time within track, no duplicated records, and no duplicated date-times within a track. `policy = "error"` stops when a rule is broken. `policy = "repair"` removes records with missing values, orders the records, removes duplicated records, and moves duplicated date-times forward by one second until they are unique, and records what it did in `meta$log`.

`longlat` says whether the coordinates are longitude and latitude, which decides how distances are measured. When it is `NULL` it is taken from `crs`: `TRUE` when `crs` is missing or names a longitude and latitude system, `FALSE` otherwise. `segment` says what the line between two fixes is: `"geodesic"` (the default for longitude and latitude) or `"planar"` (straight in the coordinate system, the default otherwise). `track_cells()` needs planar segments.

Value

A track model: a list with `meta`, `tracks` and `fixes`, with class `"TRACK"`.

See Also

`as_track()`, `track_fixes()`, `track_segments()`, `track_cells()`

Examples

```
tr <- TRACK(trips0$x, trips0$y, trips0$date, trips0$id)
tr
head(track_segments(tr))
```

---

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 `NA`s 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_cells *Track segments in grid cells, and time spent*

Description

`track_cells()` overlays the segments of a track model on a regular grid. It returns one row per piece of a segment inside one cell, with the segment parameters where the piece starts and ends (`p0`, `p1`, from 0 at the first fix to 1 at the second) and the time spent in the piece (`dt`, seconds), assuming constant speed along the segment. A segment with no movement puts all its time in the cell of its fix.

Usage

```
track_cells(x, dimension, extent = NULL, ...)

track_time_spent(x, dimension, extent = NULL, ...)
```

Arguments

`x`	a track model, or anything `as_track()` accepts
`dimension`	grid size `c(ncol, nrow)`, or a 'meshcore' `GRID`
`extent`	grid extent `c(xmin, xmax, ymin, ymax)`, the range of the fixes if `NULL`
`...`	unused

Details

`track_time_spent()` sums that time by cell.

The overlay is exact: the pieces come from where each segment crosses the grid lines, which is arithmetic on the extent and resolution. It needs planar segments (`segment = "planar"` in `TRACK()`); for longitude and latitude data that means straight lines in degrees, which is what the 'trip' package has always assumed for time spent.

The grid is given as for `track_grid()` by `dimension` (ncol, nrow) and `extent` (xmin, xmax, ymin, ymax), or as a 'meshcore' `GRID` model in `dimension`. Cell ids in `.cell` are 0-based and row-major from the top left cell (the GDAL and 'meshcore' order), so `.cell + 1` is the index that `track_grid()` returns. Pieces outside the grid are dropped.

Value

`track_cells()`: a data frame with `.seg`, `.cell`, `p0`, `p1` and `dt`. `track_time_spent()`: a data frame with `.cell` and `time` (seconds), one row per cell with time in it.

Examples

```
tr <- TRACK(trips0$x, trips0$y, trips0$date, trips0$id, segment = "planar")
tc <- track_cells(tr, c(50, 35))
head(tc)
ts <- track_time_spent(tr, c(50, 35))
v <- numeric(50 * 35)
v[ts$.cell + 1] <- ts$time
m <- matrix(v, 35, 50, byrow = TRUE)  ## first row is the top of the grid
image(t(m[nrow(m):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_fixes *Tables of a track model*

Description

`track_fixes()` and `track_tracks()` return the stored tables of a `TRACK()`. `track_segments()` derives the segments: one row per pair of consecutive fixes within a track, keyed by `.seg`, with the two fix keys (`.fix0`, `.fix1`), their coordinates and date-times, and the duration `dt` in seconds.

Usage

```
track_fixes(x, ...)

track_tracks(x, ...)

track_segments(x, ...)
```

Arguments

x	a track model, or anything as_track() accepts
...	unused

Value

A data frame.

Examples

```
tr <- as_track(trips0)
head(track_fixes(tr))
track_tracks(tr)
head(track_segments(tr))
```

---

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. An exact ellipsoidal model is used, from the geographiclib 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_metrics *Segment metrics*

Description

Distance, duration, speed and bearing for every segment of a track model. For longitude and latitude, distance is geodesic in metres and bearing is the initial geodesic azimuth in degrees, as from `track_distance()` and `track_bearing()`. Otherwise distance is Euclidean in the units of the coordinates, and bearing is the planar direction clockwise from the y axis.

Usage

```
track_metrics(x, ...)
```

Arguments

`x`	a track model, or anything `as_track()` accepts
`...`	unused

Value

`track_segments()` with columns `distance`, `speed` (distance per second) and `bearing`.

Examples

```
head(track_metrics(trips0))
```

`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_speedfilter *Speed and angle filters for track models*

Description

`track_speedfilter()` flags fixes that imply a speed of travel above `max_speed`, with the iterative running root mean square of McConnell et al. (1992). `track_sda()` adds the distance and angle tests of Freitas et al. (2008). Both are ports of `speedfilter()` and `sda()` from the 'trip' package; the fixes are not removed, the result says which pass.

Usage

```
track_speedfilter(x, max_speed, ...)

track_sda(x, smax, ang = c(15, 25), distlim = c(2.5, 5), pre = NULL, ...)
```

Arguments

x	a track model, or anything as_track() accepts
max_speed, smax	maximum speed (km/h for longitude and latitude)
...	unused
ang	minimum turning angles in degrees, one per pass
distlim	maximum step lengths in km, one per pass
pre	optional logical vector, one per fix, of fixes already rejected (FALSE) by another filter

Details

Speeds are in km/h and distances in km for longitude and latitude (geodesic distance), as in 'trip'; otherwise in coordinate units per hour and coordinate units. track_sda() needs longitude and latitude.

Two differences from 'trip' 1.10.0. The speed over two segments (from a fix to the one after next) uses the distance between those two fixes; 'trip' measured it along the wrong pairs of fixes for longitude and latitude data, so its flags can differ there. And where 'trip' could loop forever on a track whose remaining fixes are too few for the running mean, the filter stops for that track.

Value

A data frame with .fix and the logical ok, FALSE where the fix fails.

References

McConnell, B. J., Chambers, C. and Fedak, M. A. (1992) Foraging ecology of southern elephant seals in relation to the bathymetry and productivity of the Southern Ocean. Antarctic Science 4:393-398.

Freitas, C., Lydersen, C., Fedak, M. A. and Kovacs, K. M. (2008) A simple new algorithm to filter marine mammal Argos locations. Marine Mammal Science 24:315-325. doi:10.1111/j.1748-7692.2007.00180.x

Examples

```
tr <- as_track(trips0)
table(track_speedfilter(tr, max_speed = 20)$ok)
table(track_sda(tr, smax = 20)$ok)
```

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

as.POSIXct(), *3*
as_track, 2
as_track(), *4*, *7*, *10*, *12*, *15*

TRACK, 3
TRACK(), *2, 3*, *7*, *9*
track_angle, 4
track_bearing, 5
track_bearing(), *12*
track_bearing_to, 6
track_cells, 7
track_cells(), *4*
track_distance, 8
track_distance(), *12*
track_distance_to, 9
track_fixes, 9
track_fixes(), *4*
track_grid, 10
track_grid(), *7*
track_intermediate, 11
track_metrics, 12
track_query, 13
track_sda *(*track_speedfilter*)*, 14
track_segments *(*track_fixes*)*, 9
track_segments(), *3, 4*, *12*
track_speed, 14
track_speedfilter, 14
track_time, 16
track_time_spent *(*track_cells*)*, 7
track_tracks *(*track_fixes*)*, 9
track_turn, 16
trips0, 17