Package 'amt'

September 12, 2018

Type Package

Title Animal Movement Tools

Version 0.0.5.0

Description Manage and analyze animal movement data. The functionality of 'amt' includes methods to calculate track statistics (e.g. step lengths, speed, or turning angles), prepare data for fitting habitat selection analyses (resource and step-selection functions), and simulation of space-use from fitted step-selection functions.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 2.10),

Imports broom, circular, dplyr (>= 0.7.0), fitdistrplus, FNN, geosphere, graphics, grDevices, KernSmooth, lazyeval, leaflet, lubridate, magrittr, maptools, methods, purrr, raster, Rcpp (>= 0.12.7), rgeos, rlang, sf, sp, survival, stats, tibble, tidyr, utils, velox

Suggests adehabitatLT, bcpa, ctmm, devtools, move, moveHMM, spacetime, testthat, trajectories, knitr, Rdpack, rmarkdown

LinkingTo Rcpp

RoxygenNote 6.0.1

RdMacros Rdpack

VignetteBuilder knitr

NeedsCompilation yes

Author Johannes Signer [aut, cre]

Maintainer Johannes Signer <jsigner@gwdg.de>

Repository CRAN

Date/Publication 2018-09-12 19:10:06 UTC

R topics documented:

amt-package *amt: Animal Movement Tools*

Description

Manage and analyze animal movement data. The functionality of 'amt' includes methods to calculate track statistics (e.g. step lengths, speed, or turning angles), prepare data for fitting habitat selection analyses (resource and step-selection functions), and simulation of space-use from fitted step-selection functions.

Author(s)

Maintainer: Johannes Signer <jsigner@gwdg.de>

adjust_param *Adjust parameters*

Description

Functions for parameter adjustment after fitting an integrated step-selection function (iSSF).

Usage

```
adjust_shape(tentative, modifier = 0)

adjust_scale(tentative, modifier = 0)

adjust_kappa(tentative, modifier = 0)
```

Arguments

tentative	[numeric] The tentative parameter estimate.
modifier	[numeric=0] The modifier to adjust the tentative estimate.

Details

The shape and scale parameter of a gamma distribution, and the concentration parameter (=kappa) of a von-Mises distribution can be adjusted. The following adjustments are possible:

1. The shape parameter of a gamma distribution fitted to the observed step lengths, can be adjusted with the coefficient for the log of the step lenghts.
2. The scale parameter of a gamma distribution fitted to the observed step lengths, can be adjusted with the coefficient for the step lenghts.
3. The concentration parameter of fa von Mises distribution fitted to the observed turning angle, can be adjusted with the coefficient for the cosine of turning angles.

References

Avgar T, Potts JR, Lewis MA and Boyce MS (2016). "Integrated step selection analysis: bridging the gap between resource selection and animal movement." *Methods in Ecology and Evolution*.

Examples

```
# Using the deer data set
data(deer)
data(sh_forest)

# first prepare the data and fit a model
m1 <- deer %>% steps_by_burst() %>%
  random_steps() %>%
  extract_covariates(sh_forest) %>%
  mutate(sh.forest = factor(sh.forest)) %>%
  fit_clogit(case_ ~ sh.forest * log(sl_) + sl_ + strata(step_id_))



# Investigate and adjust parameters ---------------------------------------

sl_params(m1)['shape', 'est']
# adjust shape with the log of the step length
sh1 <- adjust_shape(sl_params(m1)['shape', 'est'],
             modifier = coef(m1)['log(sl_)'])

sl_params(m1)['scale', 'est']
# adjust shape with the step length
sc1 <- adjust_shape(sl_params(m1)['scale', 'est'],
             modifier = coef(m1)['sl_'])

# Up to now we have ignored the interaction with forest
# this means the above assumes that forest = 2 (= non forest)

sl_params(m1)['shape', 'est']
# adjust shape with the log of the step length
sh2 <- adjust_shape(tentative = sl_params(m1)['shape', 'est'],
                    modifier = coef(m1)['log(sl_)'] + coef(m1)['sh.forest2:log(sl_)'])

# The modified shape paremeter differ for forest and non forest.
# The shape for steps that end in forest are lower.
sh1
sh2

# This can be best seen when plotting the tentative Gamma distribution (black) and
# adding lines for Gamma distributions with adjusted shape paremeters
# for open areas (red) and forested areas (green).
## Not run:
plot_sl(m1)
curve(dgamma(x, shape = sh1, scale = sc1), col = "red", add = TRUE, from = 0.1)
curve(dgamma(x, shape = sh2, scale = sc1), col = "forestgreen", add = TRUE, from = 0.1)
```

```
## End(Not run)
```

`animal_sim` *Simulated movement data of 1 animal.*

Description

32400 simulated relocations of an animal

Usage

```
animal_sim
```

Format

A `data_frame` with 32400 rows and 3 variables:

x_ the x-coordinate

y_ the y-coordinate

t_ the timestamp

`as_rad` *Converts angles to radians*

Description

Converts angles to radians

Usage

```
as_rad(x)
```

Arguments

`x`	`[numeric]` Angles in degrees.

Examples

```
as_rad(seq(-180, 180, 30))

# The default unit of turning angles is degrees.
data(deer)
deer %>% steps() %>% mutate(ta_ = as_rad(ta_))
```

as_track *Coerce to track*

Description

Coerce other classes (currently implemented: `SpatialPoints`) to a `track_xy`.

Usage

```
as_track(x, ...)

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

Arguments

`x`	`[SpatialPoints]` Object to be converted to a track.
`...`	Further arguments, none implemented.

Examples

```
xy <- sp::SpatialPoints(cbind(c(1, 3, 2, 1), c(3, 2, 2, 1)))
as_track(xy)
```

bbox *Get bounding box of a track.*

Description

Get bounding box of a track.

Usage

```
bbox(x, ...)

## S3 method for class 'track_xy'
bbox(x, spatial = TRUE, buffer = NULL, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
spatial	[logical(1)=FALSE] Whether or not to return a SpatialPolygons-object or not.
buffer	[numeric(0)=NULL]{NULL, >0} An optional buffer of the bounding box.

Examples

```
data(deer)
bbox(deer)
bbox(deer, buffer = 100)
```

---

centroid *Calculate the centroid of a track.*

Description

Calculate the centroid of a track.

Usage

```
centroid(x, ...)

## S3 method for class 'track_xy'
centroid(x, spatial = FALSE, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
spatial	[logical(1)=FALSE] Whether or not to return a SpatialPoints-object.

Examples

```
data(deer)
centroid(deer)
```

coercion *Coerce a track to other formats.*

Description

Several other packages provides methods to analyse movement data, and amt provides coercion methods to some packages

Usage

```
as_sp(x, ...)

## S3 method for class 'steps_xy'
as_sp(x, end = TRUE, ...)

as_move(x, ...)

## S3 method for class 'track_xy'
as_move(x, ...)

## S3 method for class 'track_xyt'
as_move(x, ...)

as_ltraj(x, ...)

## S3 method for class 'track_xy'
as_ltraj(x, id = "animal_1", ...)

## S3 method for class 'track_xyt'
as_ltraj(x, ...)

as_bcpa(x, ...)

## S3 method for class 'track_xyt'
as_bcpa(x, ...)

as_moveHMM(x, ...)

## S3 method for class 'track_xy'
as_moveHMM(x, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.

end	[logical(1)=TRUE] For steps, should the end or start points be used?
id	[numeric,character,factor] Animal id(s).

Examples

```
data(deer)
as_move(deer)
as_move(deer, id = "foo")
data(deer)
as_ltraj(deer)
as_ltraj(deer, id = "animal_3")
data(deer)
d <- as_bcpa(deer)
## Not run:
bcpa1 <- bcpa::WindowSweep(d, "Theta", K = 2, windowsize = 50)
plot(bcpa1, type = "flat", clusterwidth = 1)

## End(Not run)
# Fit HMM with two states
data(deer)
dm <- as_moveHMM(deer)
## Not run:
mu0 <- rep(mean(dm$step, na.rm = TRUE), 2) # step mean (two parameters: one for each state)
sigma0 <- rep(sd(dm$step, na.rm = TRUE), 2) # step SD
zeromass0 <- c(0.1, 0.05) # step zero-mass
stepPar0 <- c(mu0, sigma0, zeromass0)
angleMean0 <- c(pi, pi) # angle mean
kappa0 <- c(1, 1) # angle concentration
anglePar0 <- c(angleMean0, kappa0) ## call to fitting function
m1 <- fitHMM(data = dm, nbStates = 2,
       stepPar0 = stepPar0, anglePar0 = anglePar0, formula = ~ 1)

## End(Not run)
```

---

coords *Coordinates of a track.*

Description

Coordinates of a track.

Usage

```
coords(x, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.

Value

[data_frame]
The coordinates.

Examples

```
data(deer)
coords(deer)
```

---

crs *Coordinate References System (CRS)*

Description

Check if an object has a coordinate reference system (has_crs) and returns the proj4string with get_crs of the coordinate reference system.

Usage

```
get_crs(x, ...)

has_crs(x, ...)
```

Arguments

x	[any] Object to check.
...	Further arguments, none implemented.

Examples

```
data(deer)
has_crs(deer)
get_crs(deer)
```

cum_ud *Calculate a cumulative UD*

Description

Calculate the cumulative utilization distribution (UD).

Usage

```
cumulative_ud(x, ...)

## S3 method for class 'RasterLayer'
cumulative_ud(x, ...)

## S3 method for class 'kde'
cumulative_ud(x, ...)
```

Arguments

x	[RasterLayer] Containing the Utilization Distribution (UD).
...	Further arguments, none implemented.

Value

[RasterLayer]
The cumulative UD.

Note

This function is typically used to obtain isopleths.

deer *Relocations of 1 red deer*

Description

826 GPS relocations of one red deer from northern Germany. The data is already resampled to a regular time interval of 6 hours and the coordinate reference system is transformed to epsg:3035.

Usage

```
deer
```

Format

A track_xyt

x_ the x-coordinate

y_ the y-coordinate

t_ the timestamp

burst_ the burst a particular points belongs to.

Source

Verein für Wildtierforschung Dresden und Göttingen e.V.

`dist_cent` *Distance to center*

Description

Distances to frequently used areas. `distance_to_center` calculates the distance to the home-range center (i.e., the centroid of the `x` and `y` coordinates). `distance_to_centers` calculates the distance to `top_n` most frequently used cells. Note, that the results of `distance_to_center` is different to `distance_to_centers` with `top_n = 1`, since in the first case the distance to the centroid is calculated and in the second case the distance to the raster cell with the most relocations.

Usage

```
distance_to_center(x, ...)

## S3 method for class 'track_xy'
distance_to_center(x, trast, square = TRUE, ...)

distance_to_centers(x, ...)

## S3 method for class 'track_xy'
distance_to_centers(x, trast, top_n = 10, square = TRUE,
  ...)
```

Arguments

`x`	`[track_xy, track_xyt]` A track created with `mk_track` or `track`.
`...`	Further arguments, none implemented.
`trast`	`[RasterLayer]` A template.
`square`	`[logical(1)]` Should the distance be squared?
`top_n`	`[integer(1)]` To how many centers should the distance be calculated?

Value

RasterLayer

Examples

```
data(deer)
r <- raster::raster(bbox(deer, buffer = 100), res = 40)
d1 <- distance_to_center(deer, r)
d2 <- distance_to_centers(deer, r, top_n = 1)
d3 <- distance_to_centers(deer, r, top_n = 10)
```

---

extract_covariates *Extract covariate values*

Description

Extract the covariate values at relocations, or at the beginning or end of steps.

Usage

```
extract_covariates(x, ...)

## S3 method for class 'track_xy'
extract_covariates(x, covariates, ...)

## S3 method for class 'random_points'
extract_covariates(x, covariates, ...)

## S3 method for class 'steps_xy'
extract_covariates(x, covariates, where = "end", ...)

extract_covariates_along(x, ...)

## S3 method for class 'steps_xy'
extract_covariates_along(x, covariates, ...)
```

Arguments

x	[track_xy, track_xyt, steps] Either a track created with mk_track or track, or steps.
...	Further arguments, none implemented.
covariates	[RasterLayer,RasterStack,RasterBrick] The (environmental) covariates.
where	[character(1)="end"]{"start", "end", "both"} For steps this determines if the covariate values should be extracted at the beginning or the end of a step. or end.

Details

extract_covariates_along extracts the covariates along a straight line between the start and the end point of a (random) step. It returns a list, which in most cases will have to be processed further.

Examples

```
data(deer)
data(sh_forest)
deer %>% extract_covariates(sh_forest)
deer %>% steps %>% extract_covariates(sh_forest)
deer %>% steps %>% extract_covariates(sh_forest, where = "start")
data(deer) # relocation
data("sh_forest") # env covar

p1 <- deer %>% steps() %>% random_steps() %>%
  extract_covariates(sh_forest) %>% # extract at the endpoint
  mutate(for_path = extract_covariates_along(., sh_forest))  %>%
  # 1 = forest, lets calc the fraction of forest along the path
  mutate(for_per = purrr::map_dbl(for_path, ~ mean(. == 1)))
```

---

filter_min_n_burst *Filter bursts by number of relocations*

Description

Only retain bursts with a minimum number (= min_n) of relocations.

Usage

```
filter_min_n_burst(x, ...)

## S3 method for class 'track_xy'
filter_min_n_burst(x, min_n = 3, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
min_n	[numeric(1)=3] Indicating the minimum number of relocations (=fixes per burst).

`fit_clogit` *Fit a conditional logistic regression*

Description

This function is a wrapper around `stats::glm`, making it usable in a piped workflow.

Usage

```
fit_clogit(data, formula, more = NULL, summary_only = FALSE, ...)

fit_ssf(data, formula, more = NULL, summary_only = FALSE, ...)

fit_issf(data, formula, more = NULL, summary_only = FALSE, ...)
```

Arguments

`data`	`[data.frame]` The data used to fit a model.
`formula`	`[formula]` The model formula.
`more`	`[list]` Optional list that is passed on the output.
`summary_only`	`[logical(1)=FALSE]` If `TRUE` only a `broom::tidy` summary of the model is returned.
`...`	Additional arguments, passed to `survival::clogit`.

`fit_logit` *Fit logistic regression*

Description

Tihs function is a wrapper around `stats::glm` for piped workflows.

Usage

```
fit_logit(data, formula, ...)

fit_rsf(data, formula, ...)
```

Arguments

data	[data.frame] The data used to fit a model.
formula	[formula] The model formula.
...	Further arguments passed to stats::glm.

---

fit_sl_dist *Fit a statistical distribution to step lengths.*

Description

Fit a statistical distribution to step lengths.

Usage

```
fit_sl_dist(.tbl, x, ...)

fit_sl_dist_base(x, na.rm = TRUE, distr = "gamma", ...)
```

Arguments

.tbl	[track_xy,track_xyt] A track.
x	[expression] The name of the column containing step lengths, usually sl_.
...	Further arguments, none implemented.
na.rm	[logical(1)] Should NA be removed?
distr	[character(1)] Name of the distribution, currently only gamma-distribution is supported.

Examples

```
data(deer)
stps <- steps_by_burst(deer)
fit_sl_dist(stps, sl_)
```

fit_ta_dist *Fit a statistical distribution to the turn angles of a track*

Description

Fit a statistical distribution to the turn angles of a track

Usage

```
fit_ta_dist(.tbl, x, ...)

fit_ta_dist_base(x, na.rm = TRUE, distr = "vonmises", ...)
```

Arguments

.tbl	[track_xy,track_xyt] A track.
x	[expression] The name of the column containing turn angles, usually ta_.
...	Further arguments, none implemented.
na.rm	[logical(1)] Should NA be removed?
distr	[character(1)] Name of the distribution, currently only vonmises-distribution is supported.

from_to *Duration of tracks*

Description

Function that returns the start (from), end (to), and the duration (from_to) of a track.

Usage

```
from_to(x, ...)

## S3 method for class 'track_xyt'
from_to(x, ...)

from(x, ...)

## S3 method for class 'track_xyt'
from(x, ...)
```

```
to(x, ...)

## S3 method for class 'track_xyt'
to(x, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.

Examples

```
data(deer)
from(deer)
to(deer)
from_to(deer)
```

---

get_kappa *Get* kappa

Description

Convenience function to extract kappa and its SE.

Usage

```
get_kappa(x, ...)
```

Arguments

x	[list] Fitted turn angle distribution.
...	Further arguments, none implemented.

---

habitat *Habitat of simulated data*

Description

A Gauss-Random-Field simulation of a landscape.

Usage

```
habitat
```

Format

A RasterLayer

habitat_kernel *Simulate UD from fitted SSF*

Description

Function to obtain a habitat kernel from a fitted (i)SSF.

Usage

```
habitat_kernel(coef, resources, exp = TRUE)

movement_kernel(scale, shape, template, quant = 0.99)

simulate_ud(movement_kernel, habitat_kernel, start, n = 100000L)

simulate_tud(movement_kernel, habitat_kernel, start, n = 100, n_rep = 5000)
```

Arguments

coef	[list] Vector with coeffiecients, not yet implemented.
resources	[RasterLayer, RasterStack] The resources.
exp	A logical scalar, indicating whether or not the resulting habitat kernel should be exponentiated. This is usually TRUE.
scale, shape	[numeric](1) Scale and scale parameter of the gamma distribution of step lengths.
template	[RasterLayer,RasterStack] A raster serving as template for the simulations.
quant	A numeric scalar, quantile of the step-length distribution that is the maximum movement distance.
movement_kernel	[RasterLayer] The movement kernel.
habitat_kernel	[RasterLayer] The habitat kernel.
start	[numeric(2)] Starting point of the simulation.
n	[integer(1)=1e5] The number of simulation steps.
n_rep	[integer(1)=5e3]{>0} The number of times the animal walks of the final position. The mean of all replicates is returned.

Details

`movement_kernel()`: calculates a movement kernel from a fitted (i)SSF. The method is currently only implemented for the gamma distribution.

The habitat kernel is calculated by multiplying resources with their corresponding coefficients from the fitted (i)SSF.

`simulate_ud()`: simulates a utilization distribution (UD) from a fitted Step-Selection Function.

`simulate_tud()`: Is a conviencience wrapper arround `simulate_ud` to simulate transition UDs (i.e., starting at the same position many times and only simulate for a short time).

Value

The habitat kernel, as `RasterLayer`.

Note

This functions are still experimental and should be used with care. If in doubt, please contact the author.

Author(s)

Johannes Signer (jmsigner@gmail.com)

References

Avgar T, Potts JR, Lewis MA and Boyce MS (2016). "Integrated step selection analysis: bridging the gap between resource selection and animal movement." *Methods in Ecology and Evolution*. Signer J, Fieberg J and Avgar T (2017). "Estimating Utilization Distributions from fitted Step-Selection Functions." *Ecosphere*.

`hr_area` *Home ranges*

Description

Functions to calculate animal home ranges from a `track_xy*`, and to work with home ranges. `hr_mcp`, `hr_kde`, and `hr_locoh` calculate the minimum convex polygon, kernel density, and local convex hull home range respectively. `hr_area` extracts the area of an home range, `hr_isopleths` returns the isopleth as a `SpatialPolygonsDataFrame`.

Usage

```
hr_area(x, ...)

hr_isopleths(x, ...)

hr_kde(x, ...)
```

```
## S3 method for class 'track_xy'
hr_kde(x, h = hr_kde_ref(x), trast = raster(as_sp(x),
  nrow = 100, ncol = 100), ...)

hr_kde_ref(x, ...)

## S3 method for class 'track_xy'
hr_kde_ref(x, rescale = "none", ...)

hr_locoh_k(x, ...)

## S3 method for class 'track_xy'
hr_locoh_k(x, n = 10, levels = 0.95,
  rand_buffer = 1e-05, ...)

hr_mcp(x, ...)

## S3 method for class 'track_xy'
hr_mcp(x, levels = 0.95, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
h	[numeric(2)] The bandwidth for kernel density estimation.
trast	[RasterLayer] A template raster for kernel density home-ranges.
rescale	[character(1)] Rescaling method for reference bandwidth calucation. Must be one of "unitvar", "xvar", or "none".
n	[integer(1)] The number of neighbors used when calculating local convex hulls.
levels	[numeric] The isopleth levels used for calculating home ranges. Should be `0 < level < 1`.
rand_buffer	[numeric(1)] Random buffer to avoid polygons with area 0 (if coordinates are numerically identical).

Details

The implementation of the reference bandwidth calculation is based on Worton (1989). If variances differ greatly, it is advaisable to rescale the data using `rescale = "unitvar"` the data is suspected to multimodal other bandwidth estimation methods may be more suitable.

References

Worton, B. J. (1989). Kernel methods for estimating the utilization distribution in home-range studies. *Ecology, 70(1)*, 164-168.

Examples

```
data(deer)


# MCP ---------------------------------------------------------------------
mcp1 <- hr_mcp(deer)
hr_area(mcp1)

# calculated MCP at different levels
mcp1 <- hr_mcp(deer, levels = seq(0.3, 1, 0.1))
hr_area(mcp1)

# CRS are inherited
get_crs(deer)
mcps <- hr_mcp(deer, levels = c(0.5, 0.95, 1))
has_crs(mcps)

# Local Convex Hull (LoCoH) -----------------------------------------------
locoh1 <- hr_locoh_k(deer)
hr_area(locoh1)

# calculated MCP at different levels
locoh <- hr_locoh_k(deer, levels = seq(0.3, 1, 0.1))
hr_area(locoh)

# CRS are inherited
get_crs(deer)
get_crs(locoh1)

# Kernel density estimaiton (KDE) -----------------------------------------
kde1 <- hr_kde(deer)
hr_area(kde1)
get_crs(kde1)
```

`inspect` *Inspect a track*

Description

Provides a very basic interface to `leaflet` and lets the user inspect relocations on an interactive map.

Usage

```
inspect(x, ...)

## S3 method for class 'track_xy'
inspect(x, popup = NULL, cluster = TRUE, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
popup	[character(nrow(x))] Optional labels for popups.
cluster	[logical(1)] If TRUE points are clustered at lower zoom levels.

Value

An interactive leaflet map.

Note

Important, x requires a valid coordinate reference system.

See Also

leaflet::leaflet()

Examples

```
data(sh)
x <- track(x = sh$x, y = sh$y, crs = sp::CRS("+init=epsg:31467"))

## Not run:
inspect(x)
inspect(x, cluster = FALSE)
inspect(x, popup = 1:nrow(x), cluster = FALSE)

## End(Not run)
```

`movement_metrics` *Movement metrics*

Description

Functions to calculate metrics such as straightness, mean squred displacement (msd), intensity use, sinuosity, mean turn angle correlation (`tac`) of a track.

Usage

```
straightness(x, ...)

cum_dist(x, ...)

tot_dist(x, ...)

msd(x, ...)

intensity_use(x, ...)

sinuosity(x, ...)

tac(x, ...)
```

Arguments

`x`	`[track_xy, track_xyt]` A track created with `mk_track` or `track`.
`...`	Further arguments, none implemented.

Details

The intensity use is calculated by dividing the total movement distance (`tot_dist`) by the square of the area of movement (= minimum convex polygon 100).

References

Abrahms B, Seidel DP, Dougherty E, Hazen EL, Bograd SJ, Wilson AM, McNutt JW, Costa DP, Blake S, Brashares JS and others (2017). "Suite of simple metrics reveals common movement syndromes across vertebrate taxa." *Movement ecology*, **5**(1), pp. 12. Almeida PJ, Vieira MV, Kajin M, Forero-Medina G and Cerqueira R (2010). "Indices of movement behaviour: conceptual background, effects of scale and location errors." *Zoologia (Curitiba)*, **27**(5), pp. 674–680. Swihart RK and Slade NA (1985). "Testing for independence of observations in animal movements." *Ecology*, **66**(4), pp. 1176–1184.

Examples

```
data(deer)

tot_dist(deer)
cum_dist(deer)
straightness(deer)
msd(deer)
intensity_use(deer)
```

`plot_sl` *Plot step-length distribution*

Description

Plot step-length distribution

Usage

```
plot_sl(x, ...)
```

Arguments

`x`	`[fit_clogit]` A fitted step selection.
`...`	Further arguments, none implemented.

`random_points` *Generate random points*

Description

Functions to generate random points within an animals home range. This is usually the first step for investigating habitat selection via Resource Selection Functions (RSF).

Usage

```
random_points(x, ...)

## S3 method for class 'mcp'
random_points(x, n = 100, type = "random", ...)

## S3 method for class 'kde'
random_points(x, n = 100, type = "random", ...)

## S3 method for class 'track_xy'
random_points(x, level = 1, hr = "mcp", factor = 10,
  type = "random", ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	[any] None implemented.
n	[integer(1)] The number of random points.
type	[character(1)] Argument passed to sp::spsample type. The default is random.
level	[numeric(1)] Home-range level of the minimum convex polygon, used for generating the background samples.
hr	[character(1)] The home range estimator to be used. Currently only MCP is implemented.
factor	[numeric(1)] Determines the number of random points that are generated. If factor == 1 the number of presence points is equal to the number of observed points.

Note

For objects of class track_xyt the timestamp (t_) is lost.

Examples

```
data(deer)

# track_xyt -----------------------------------------------------------------
# Default settings
rp1 <- random_points(deer)
rp2 <- random_points(deer, hr = "kde") # we need a larger template raster for kde

## Not run:
plot(rp1)
plot(rp2)

## End(Not run)

# rp2 does not make sense, because the `trast` is to small.
# this can be overcome by increasing the trast manually.
trast <- raster(bbox(deer, buffer = 2000), res = 30)
rp3 <- random_points(deer, hr = "kde", trast = trast) # we need a larger template raster

 ## Not run:
plot(rp2)
plot(rp3)

## End(Not run)
```

```
# Only one random point for each observed point
rp <- random_points(deer, factor = 1)
## Not run:
plot(rp)

## End(Not run)

# Within a home range ---------------------------------------------------
hr <- hr_mcp(deer, level = 1)

# 100 random point within the home range
rp <- random_points(hr, n = 100)
## Not run:
plot(rp)

## End(Not run)

# 100 regular point within the home range
rp <- random_points(hr, n = 100, type = "regular")
## Not run:
plot(rp)

## End(Not run)
# 100 hexagonal point within the home range
rp <- random_points(hr, n = 100, type = "hexagonal")
## Not run:
plot(rp)

## End(Not run)
```

`random_steps` *Generate Random Steps*

Description

Function to generate a given number of random steps for each observed step.

Usage

```
random_steps(x, ...)

## S3 method for class 'steps_xy'
random_steps(x, n_control = 10, sl_distr = "gamma",
  ta_distr = "vonmises", random.error = 0.001, ...)
```

Arguments

x	Steps.
...	Further arguments, none implemented.
n_control	[integer(1)=10]{>1} The number of control steps paired with each observed step.
sl_distr	[character(1)='gamma']{'gamma'} The distribution to be fitted to the empirical distribution of step lengths.
ta_distr	[character(1)='vonmises']{'vonmises', 'unif'} The distribution to be fitted to the empirical distribution of turn angles.
random.error	[numeric(1)=0.001]{>0} Upper limit for a uniformly distributed random error (between 0 and random.error) to be added to step lengths, to avoid step lengths of length 0.

---

remove_capture *Removes Capture Effects*

Description

Removing relocations at the beginning and/or end of a track, that fall within a user specified period.

Usage

```
remove_capture_effect(x, ...)

## S3 method for class 'track_xyt'
remove_capture_effect(x, start, end, ...)
```

Arguments

x	An object of class track_xyt.
...	Further arguments, none implemented.
start	A lubirdate::Period, indicating the time period to be removed at the beginning of the track.
end	A lubirdate::Period, indicating the time period to be removed at the end of the track.

Examples

```
library(lubridate)
n <- 10
df <- track(
  x = cumsum(rnorm(n)),
  y = cumsum(rnorm(n)),
  t = ymd_hm("2017-01-01 00:00") +
   hours(seq(0, by = 24, length.out = n))
```

```
)

df
remove_capture_effect(df, start = days(1))
remove_capture_effect(df, end = days(2))
remove_capture_effect(df, start = days(1), end = days(2))
```

`sh` *Relocations of 1 red deer*

Description

1500 GPS relocations of one red deer from northern Germany.

Usage

```
sh
```

Format

A data frame with 1500 rows and 4 variables:

x_epsg31467 the x-coordinate

y_epsg31467 the y-coordinate

day the day of the relocation

time the hour of the relocation

Source

Verein für Wildtierforschung Dresden und Göttingen e.V.

`sh_forest` *Forest cover*

Description

Forest cover for the home range of one red deer in northern Germany.

Usage

```
sh_forest
```

Format

A RasterLAyer

1 forest

2 non-forest

Source

JRC

References

A. Pekkarinen, L. Reithmaier, P. Strobl (2007): "Pan-European Forest/Non-Forest mapping with Landsat ETM+ and CORINE Land Cover 2000 data.

---

sl_distr *Step-length distribution*

Description

Returns the name of the distribution (e.g., gamma) fitted to the distribution of step lengths.

Usage

```
sl_distr(x, ...)

## S3 method for class 'list'
sl_distr(x, ...)

## S3 method for class 'fit_clogit'
sl_distr(x, ...)

## S3 method for class 'random_steps'
sl_distr(x, ...)
```

Arguments

x	An object that contains either a fitted conditional logistic regression, random steps or just a fitted distribution.
...	Further arguments, none implemented.

sl_params *Step lengths parameters*

Description

Returns the parameter of the distribution (e.g., gamma) fitted to the distribution of step lengths.

Usage

```
sl_params(x, ...)

## S3 method for class 'fit_clogit'
sl_params(x, ...)

## S3 method for class 'random_steps'
sl_params(x, ...)

sl_shape(x, ...)

sl_scale(x, ...)

## S3 method for class 'fitdist'
sl_params(x, alpha = 0.05, ...)

## S3 method for class 'random_steps'
ta_params(x, ...)
```

Arguments

x	An object that contains either a fitted conditional logistic regression, random steps or just a fitted distribution.
...	Further arguments, none implemented.
alpha	[numeric(1)=0.05]{0-1} Alpha value for calculating 1-alpha confidence intervals.

speed *Speed*

Description

Obtain the speed of a track.

Usage

```
speed(x, ...)

## S3 method for class 'track_xyt'
speed(x, append_na = TRUE, ...)
```

Arguments

x	A track_xyt.
...	Further arguments, none implemented.
append_na	[logical(1)=TRUE] Should an NA be appended at the end.

Value

[numeric]
The speed in m/s.

---

steps *Functions to create and work with steps*

Description

step_lengths can be use to calculate step lengths of a track. direction_abs and direction_rel calculate the absolute and relative direction of steps. steps converts a track_xy* from a point representation to a step representation and automatically calculates step lengths and relative turning angles.

Usage

```
direction_abs(x, ...)

## S3 method for class 'track_xy'
direction_abs(x, degrees = TRUE, full_circle = FALSE,
  zero_dir = "E", clockwise = FALSE, append_last = TRUE, lonlat = FALSE,
  ...)

direction_rel(x, ...)

## S3 method for class 'track_xy'
direction_rel(x, lonlat = FALSE, degrees = TRUE,
  append_last = TRUE, zero_dir = "E", ...)

step_lengths(x, ...)

## S3 method for class 'track_xy'
```

```
step_lengths(x, lonlat = FALSE, append_last = TRUE, ...)

steps_by_burst(x, ...)

## S3 method for class 'track_xyt'
steps_by_burst(x, lonlat = FALSE, degrees = TRUE,
  keep_cols = NULL, ...)

steps(x, ...)

## S3 method for class 'track_xy'
steps(x, lonlat = FALSE, keep_cols = NULL,
  degrees = TRUE, ...)

## S3 method for class 'track_xyt'
steps(x, lonlat = FALSE, degrees = TRUE,
  keep_cols = NULL, diff_time_units = "auto", ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented
degrees	[logical(1)=TRUE] Should turn angles be calculated in degrees or radians? If TRUE angles are returned in degrees, otherwise in radians.
full_circle	[logical(1)=FALSE] If TRUE angles are returned between 0 and 360 degrees or 0 and $2pi$ (depending on the value of degrees), otherwise angles are between -180 and 180 or $-pi$ and pi.
zero_dir	[character(1)='E'] Indicating the zero direction. Must be either N, E, S, or W.
clockwise	[logical(1)=FALSE] Should angles be calculated clock or anti-clockwise?
append_last	[logical(1)=TRUE] If TRUE an NA is appended at the end of all angles.
lonlat	[logical(1)=TRUE] Should geographical or planar coordinates be used? If TRUE geographic distances are calculated.
keep_cols	[character(1)=NULL]{'start', 'end', 'both'} Should columns with attribute information be transfered to steps? If keep_cols = 'start' the attributes from the starting point are use, otherwise the columns from the end points are used.
diff_time_units	[character(1)='auto'] The unit for time differences, see ?difftime.

Details

step_lengths calculates the step lengths between points a long the path. The last value returned is NA, because no observed step is 'started' at the last point. If lonlat = TRUE, step_lengths() wraps raster::pointDistance().

Value

[numeric]
For step_lengths() and direction_* a numeric vector.
[data.frame]
For steps and steps_by_burst, containing the steps.

Examples

```
# Absolute directions

xy <- data_frame(
  x = c(1, 4, 8, 8, 12, 8, 0, 0, 4, 2),
  y = c(0, 0, 0, 8, 12, 12, 12, 8, 4, 2))
trk <- mk_track(xy, x, y)

# append last
direction_abs(trk, append_last = TRUE)
direction_abs(trk, append_last = FALSE)

# degrees
direction_abs(trk, degrees = FALSE)
direction_abs(trk, degrees = TRUE)

# full circle or not
direction_abs(trk, degrees = TRUE, full_circle = TRUE)
direction_abs(trk, degrees = TRUE, full_circle = FALSE)

# direction of 0
direction_abs(trk, full_circle = TRUE, zero_dir = "N")
direction_abs(trk, full_circle = TRUE, zero_dir = "E")
direction_abs(trk, full_circle = TRUE, zero_dir = "S")
direction_abs(trk, full_circle = TRUE, zero_dir = "W")

# clockwise or not
direction_abs(trk, full_circle = TRUE, zero_dir = "N", clockwise = FALSE)
direction_abs(trk, full_circle = TRUE, zero_dir = "N", clockwise = TRUE)

# Bearing (i.e. azimuth): only for lon/lat
direction_abs(trk, full_circle = FALSE, zero_dir = "N", lonlat = FALSE, clockwise = TRUE)
direction_abs(trk, full_circle = FALSE, zero_dir = "N", lonlat = TRUE, clockwise = TRUE)

# How do results compare to other packages
# adehabitatLT
```

```
df <- adehabitatLT::as.ltraj(data.frame(x = xy$x, y = xy$y), typeII = FALSE, id = 1)
df[[1]]$abs.angle
amt::direction_abs(trk, degrees = FALSE, full_circle = FALSE)

# bcpa
df <- bcpa::MakeTrack(xy$x, xy$y, lubridate::now() +  lubridate::hours(1:10))
bcpa::GetVT(df)$Phi
direction_abs(trk, degrees = FALSE, full_circle = FALSE, append_last = FALSE)

# move
m <- move::move(xy$x, xy$y, lubridate::now() + lubridate::hours(1:10),
 proj = sp::CRS("+init=epsg:4326"))
move::angle(m)
direction_abs(trk, degrees = TRUE, full_circle = FALSE, zero_dir = "N",
  clockwise = TRUE, append_na = FALSE, lonlat = TRUE)

# trajectories
t1 <- trajectories::Track(
  spacetime::STIDF(sp::SpatialPoints(cbind(xy$x, xy$y)),
  lubridate::now(tzone = "UTC") + lubridate::hours(1:10), data = data.frame(1:10)))

t1[["direction"]]
direction_abs(trk, degrees = TRUE, full_circle = TRUE, zero_dir = "N",
  clockwise = TRUE, append_last = FALSE)

# moveHMM (only rel. ta)
df <- data.frame(ID = 1, x = xy$x, y = xy$y)
moveHMM::prepData(df, type = "UTM")
# How do results compare to other packages
xy <- data_frame(
  x = c(1, 4, 8, 8, 12, 8, 0, 0, 4, 2),
  y = c(0, 0, 0, 8, 12, 12, 12, 8, 4, 2))
trk <- mk_track(xy, x, y)
# adehabitatLT
df <- adehabitatLT::as.ltraj(data.frame(x = xy$x, y = xy$y), typeII = FALSE, id = 1)
df[[1]]$rel.angle
amt::direction_rel(trk, degrees = FALSE, full_circle = FALSE)

# trajectories
t1 <- trajectories::Track(
  spacetime::STIDF(sp::SpatialPoints(cbind(xy$x, xy$y)),
  lubridate::now() + lubridate::hours(1:10), data = data.frame(1:10)))

t1[["direction"]]
direction_abs(trk, degrees = TRUE, full_circle = TRUE, zero_dir = "N",
  clockwise = TRUE, append_last = FALSE)

# moveHMM (only rel. ta)
df <- data.frame(ID = 1, x = xy$x, y = xy$y)
moveHMM::prepData(df, type = "UTM")

trk
```

```
# step_lengths ------------------------------------------------------------
xy <- data_frame(
  x = c(0, 1, 2),
  y = c(0, 1, 2)
)
xy <- mk_track(xy, x, y)

step_lengths(xy, lonlat = FALSE)
step_lengths(xy, lonlat = TRUE) # in m, but coords are assumed in degrees


# creating steps ----------------------------------------------------------

# Create some dummy data
library(lubridate)
df <- data_frame(
  x = runif(10),
  y = runif(10),
  a = runif(10),
  t = now() + hours(c(1:2, 5:6, 9:10, 14:17)),
  b = 3,
  c = a + 30
)

library(amt)
make_track(df, x, y, t, a, b, c) %>% steps(keep_cols = "start")

make_track(df, x, y, a = a, b = b, c = c) %>% steps(keep_cols = "end")

make_track(df, x, y, t, a, b, c) %>%
  track_resample(rate = hours(1), tolerance = minutes(5)) %>%
  steps_by_burst(keep_cols = "start")

make_track(df, x, y, t, a, b, c) %>%
  track_resample(rate = hours(1), tolerance = minutes(5)) %>%
  steps_by_burst(keep_cols = NULL)
```

`summarize_sampling_rate` *Returns a summary of sampling rates*

Description

Returns a summary of sampling rates

Usage

```
summarize_sampling_rate(x, ...)
```

```
## S3 method for class 'track_xyt'
summarize_sampling_rate(x, time_unit = "auto",
  summarize = TRUE, as_tibble = TRUE, ...)
```

Arguments

x	A track_xyt.
...	Further arguments, none implemented.
time_unit	A character. The time unit in which the sampling rate is calculated. Acceptable values are sec, min, hour, day, and auto. If auto (the default) is used, the optimal unit is guessed.
summarize	A logical. If TRUE a summary is returned, otherwise raw sampling intervals are returned.
as_tibble	A logical. Should result be returned as tibble or as table.

Value

Depending on summarize and as_tibble, a vector, table or tibble.

Examples

```
data(deer)
amt::summarize_sampling_rate(deer)
```

---

ta_distr *Turn angle distribution*

Description

Returns the name of the distribution (e.g., von Mises) fitted to the distribution of turn angles.

Usage

```
ta_distr(x, ...)

## S3 method for class 'list'
ta_distr(x, ...)

## S3 method for class 'fit_clogit'
ta_distr(x, ...)

## S3 method for class 'random_steps'
ta_distr(x, ...)
```

Arguments

x	[fit_clogit(1),random_steps(1)] An object that contains either a fitted conditional logistic regression, random steps or just a fitted distribution.
...	Further arguments, none implemented.

---

ta_params *Turn angle parameters*

Description

Returns the parameter of the distribution (e.g., von Mises) fitted to the distribution of turn angles.

Usage

```
ta_params(x, ...)

## S3 method for class 'fit_clogit'
ta_params(x, ...)

ta_kappa(x, ...)
```

Arguments

x	[fit_clogit(1),random_steps(1)] An object that contains either a fitted conditional logistic regression, random steps or just a fitted distribution.
...	Further arguments, none implemented.

---

time_of_day *Time of the day when a fix was taken*

Description

A convenience wrapper around `maptools::sunriset` and `maptools::crepuscule` to extract if a fix was taken during day or night (optionally also include dawn and dusk).

Usage

```
time_of_day(x, ...)

## S3 method for class 'track_xyt'
time_of_day(x, solar.dep = 6,
  include.crepuscule = FALSE, ...)

## S3 method for class 'steps_xyt'
time_of_day(x, solar.dep = 6,
  include.crepuscule = FALSE, where = "end", ...)
```

Arguments

x	[track_xyt,steps_xyt] A track or steps.
...	Further arguments, none implemented.
solar.dep	[numeric(1,n)=6] The angle of the sun below the horizon in degrees. Passed to `maptools::crepuscule`.
include.crepuscule	[logical(1)=TRUE] Should dawn and dusk be included.
where	[character(1)="end"]{"start", "end", "both"} For `steps`, should the start, end or both time points be used?

Examples

```
data(deer)
deer %>% time_of_day()
deer %>% steps_by_burst %>% time_of_day()
deer %>% steps_by_burst %>% time_of_day(where = "start")
deer %>% steps_by_burst %>% time_of_day(where = "end")
deer %>% steps_by_burst %>% time_of_day(where = "both")
```

`track` *Create a* `track_*`

Description

Constructor to crate a track, the basic building block of the `amt` package. A `track` is usually created from a set of `x` and `y` coordinates, possibly time stamps, and any number of optional columns, such as id, sex, age, etc.

Usage

```
mk_track(tbl, .x, .y, .t, ..., crs = NULL, order_by_ts = TRUE,
  check_duplicates = FALSE)

make_track(tbl, .x, .y, .t, ..., crs = NULL, order_by_ts = TRUE,
  check_duplicates = FALSE)

track(x, y, t, ..., crs = NULL)
```

Arguments

tbl	data.frame The data.frame from which a track should be created.
.x, .y, .t	[expression(1)] Unquoted variable names of columns containing the x and y coordinates, and optionally a time stamp.
...	[expression] Additional columns from tbl to be used in a track. Columns should be provided in the form of key = val (e.g., for ids this may look like this id = c(1, 1, 1, 2, 2, 2 for three points for ids 1 and 2 each).
crs	[sp::CRS] An optional coordinate reference system of the points.
order_by_ts	[logical(1)] Should relocations be ordered by time stamp, default is TRUE.
check_duplicates	[logical(1)=FALSE] Should it be checked if there are duplicated time stamp, default is FALSE.
x, y	[numeric] The x and y coordinates.
t	[POSIXct] The time stamp.

Value

If t was provided an object of class track_xyt is returned otherwise a track_xy.

---

track_methods *Track Methods*

Description

Methods to work with a track. Function to calculate the absolute direction of a movement track. 0 is north.

Usage

```
velocity(x, ...)

## S3 method for class 'track_xyt'
velocity(x, ...)

nsd(x, ...)

## S3 method for class 'track_xy'
nsd(x, ...)

diff_x(x, ...)

## S3 method for class 'track_xy'
diff_x(x, ...)

diff_y(x, ...)

## S3 method for class 'track_xy'
diff_y(x, ...)
```

Arguments

x	A track_xyt.
...	Further arguments, none implemented.

`track_resample` *Resample track*

Description

Function to resample a track at a predefined sampling rate within some tolerance.

Usage

```
track_resample(x, ...)

## S3 method for class 'track_xyt'
track_resample(x, rate = hours(2),
  tolerance = minutes(15), start = 1, ...)
```

Arguments

x	A `track_xyt`.
...	Further arguments, none implemented.
rate	A lubridate `Period`, that indicates the sampling rate.

tolerance	A lubridate Period, that indicates the tolerance of deviations of the sampling rate.
start	A integer scalar, that gives the relocation at which the sampling rate starts.

---

transform_coords *Transform CRS*

Description

Transforms the CRS for a track.

Usage

```
transform_coords(x, ...)

## S3 method for class 'track_xy'
transform_coords(x, crs_to, crs_from, ...)

transform_crs(x, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with mk_track or track.
...	Further arguments, none implemented.
crs_to	[sp::CRS(1)] Coordinate reference system the data should be transformed to, see sp::CRS.
crs_from	[sp::CRS(1)] Coordinate reference system the data are currently in, see sp::CRS. If crs_from is missing, the crs-attribute of the track is used.

See Also

sp::spTransform

Examples

```
data(deer)
get_crs(deer)

# project to geographical coordinates (note the CRS is taken automatically from the object deer).
d1 <- transform_coords(deer, sp::CRS("+init=epsg:4326"))
```

Index