Package: amt (via r-universe)

August 29, 2024

Type Package

Title Animal Movement Tools

Version 0.3.0.0

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

License GPL-3

URL https://github.com/jmsigner/amt

Depends R (>= 4.1)

Imports checkmate, circular, ctmm, data.table, dplyr (>= 0.7.0), fitdistrplus, FNN, graphics, grDevices, KernSmooth, lubridate, MASS, methods, purrr, Rdpack, rlang, sf, sfheaders, stats, survival, terra, tibble, tidyr (>= 1.0.0), utils

Suggests adehabitatLT, broom, ggplot2, ggraph, geosphere, knitr, leaflet, moveHMM, rmarkdown, sessioninfo, suncalc, tidygraph, tinytest, units

VignetteBuilder knitr

RdMacros Rdpack

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.0

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

RemoteUrl https://github.com/jmsigner/amt

RemoteRef HEAD

RemoteSha d7ffc4ef1e0e576e69a1d2d13dacdf77aaf59396

Contents

`amt_fisher` *GPS tracks from four fishers*

Description

This file includes spatial data from 4 fisher (**Pekania pennanti**). These location data were collected via a 105g GPS tracking collar (manufactured by E-obs GmbH) and programmed to record the animal's location every 10 minutes, continuously. The data re projected in NAD84 (epsg: 5070). The data usage is permitted for exploratory purposes. For other purposes please get in contact (Scott LaPoint).

Usage

```
amt_fisher
```

Format

A `tibble` with 14230 rows and 5 variables:

x_ the x-coordinate

y_ the y-coordinate

t_ the timestamp

sex the sex of the animal

id the id of the animal

name the name of the animal

Source

https://www.datarepository.movebank.org/handle/10255/move.330

References

For more information, contact Scott LaPoint `sdlapoint@gmail.com`

amt_fisher_covar *Environmental data for fishers*

Description

A list with three entries that correspond to the following three layer: land use, elevation and population density.

Usage

```
amt_fisher_covar
```

Format

A list with three where each entry is a SpatRast.

Source

https://lpdaac.usgs.gov/dataset_discovery/aster/aster_products_table
http://dup.esrin.esa.it/page_globcover.php
http://sedac.ciesin.columbia.edu/data/collection/gpw-v3/sets/browse

as.data.frame.uhc_data *Coerce a* uhc_data *object to* data.frame

Description

Coerces uhc_data from list to data.frame

Usage

```
## S3 method for class 'uhc_data'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	[uhc_data] An object of class uhc_data, as returned by the function prep_uhc().
row.names	Included for consistency with generic as.data.frame(). Currently ignored.
optional	Included for consistency with generic as.data.frame(). Currently ignored.
...	Included for consistency with generic as.data.frame(). Currently ignored.

Details

This coercion aims to keep all of the information contained in the uhc_data list in the resulting data.frame representation. Factors are converted to numeric, but the levels are retained in the column "label".

Value

Returns a `data.frame` with columns:

- `var`: The name of the variable
- `x`: The x-coordinate of the density plot (the value of `var`).
- `y`: The y-coordinate of the density plot (the probability density for a numeric `var` and the proportion for a factor `var`).
- `dist`: The distribution represented. Either `"U"` for used, `"A"` for available, or `"S"` for sampled.
- `iter`: The iteration number if `dist == "S"`.
- `label`: The label if `var` is a factor.

Author(s)

Brian J. Smith

See Also

`prep_uhc()`, `conf_envelope()`

---

as_sf_lines *Export track to lines*

Description

Exports a track to (multi)lines from the `sf` package.

Usage

```
as_sf_lines(x, ...)
```

Arguments

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

Value

A `tibble` with a `sfc`-column

as_sf_points *Coerces a track to points*

Description

Coerces a track to points from the `sf` package.

Usage

```
as_sf_points(x, ...)

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

Arguments

`x`	`[track_xy, track_xyt]` A track created with `make_track`.
`...`	Further arguments, none implemented.
`end`	`[logical(1)=TRUE]` For steps, should the end or start points be used?

Value

A data `data.frame` with a `sfc`-column

as_track *Coerce to track*

Description

Coerce other classes to a `track_xy`.

Usage

```
as_track(x, ...)

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

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

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

Arguments

x	Object to be converted to a track.
...	Further arguments, none implemented.

Value

An object of class `track_xy(t)`

---

available_distr *Display available distributions for step lengths and turn angles.*

Description

Display available distributions for step lengths and turn angles.

Usage

```
available_distr(which_dist = "all", names_only = FALSE, ...)
```

Arguments

which_dist	`[char(1)="all"]{"all", "ta", "sl"}` Should `all` distributions be returned, or only distributions for turn angles (`ta`) or step lengths (`sl`).
names_only	`[logical(1)=FALSE]` Indicates if only the names of distributions should be returned.
...	none implemented.

Value

A `tibble` with the purpose of the distribution (turn angles [ta] or step length [sl]) and the distribution name.

---

bandwidth_pi `hr_kde_pi` *wraps* `KernSmooth::dpik` *to select bandwidth for kernel density estimation the plug-in-the-equation method in two dimensions.*

Description

This function calculates bandwidths for kernel density estimation by wrapping `KernSmooth::dpik`. If `correct = TURE`, the bandwidth is trasformed with power 5/6 to correct for using an univariate implementation for bivariate data (Gitzen et. al 2006).

Usage

```
hr_kde_pi(x, ...)

## S3 method for class 'track_xy'
hr_kde_pi(x, rescale = "none", correct = TRUE, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with make_track.
...	Further arguments, none implemented.
rescale	[character(1)] Rescaling method for reference bandwidth calculation. Must be one of "unitvar", "xvar", or "none".
correct	Logical scalar that indicates whether or not the estimate should be correct for the two dimensional case.

Value

The bandwidth, the standardization method and correction.

References

Gitzen, R. A., Millspaugh, J. J., & Kernohan, B. J. (2006). Bandwidth selection for fixed-kernel analysis of animal utilization distributions. *Journal of Wildlife Management*, 70(5), 1334-1344.

See Also

KernSmooth::dpik

---

bandwidth_ref *Reference bandwidth*

Description

Calculate the reference bandwidth for kernel density home-range range estimates.

Usage

```
hr_kde_ref(x, ...)

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

Arguments

x	[track_xy, track_xyt] A track created with make_track.
...	Further arguments, none implemented.
rescale	[character(1)] Rescaling method for reference bandwidth calculation. Must be one of "unitvar", "xvar", or "none".

Value

The estimated bandwidth in x and y direction.

---

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, ...)

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

Arguments

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

Value

If spatial = FALSE a named vector of length four with the extent of the bounding box. Otherwise a SpatialPolygon or a simple freature polygon with the bounding box.

Examples

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

# For steps
deer |> steps_by_burst() |> bbox(spatial = FALSE)
deer |> steps_by_burst() |> bbox(buffer = 100, spatial = FALSE)
deer |> steps_by_burst() |> random_steps() |> bbox(spatial = FALSE)

# Further manipulations are possible
deer |> bbox() |> sf::st_transform(4326)
```

---

calculate_sdr *Calculate SDR*

Description

Calculates squared displacement rate for a given speed and duration

Usage

```
calculate_sdr(speed = 50, time, speed_unit = c("km/h", "m/s"))
```

Arguments

speed	[numeric] A speed given in either km/h or m/s.
time	[Period] A lubridate Period for which the speed can be sustained.
speed_unit	[character] The unit in which speed is given. Should be either "km/h" or "m/s".

Value

Returns a numeric vector (of length 1) with the SDR in m^2/s.

Author(s)

Johannes Signer and Brian J. Smith

See Also

get_displacement()

Examples

```
# Assume a cheetah can sprint 100 km/h for 60 seconds
calculate_sdr(speed = 100, time = seconds(60), speed_unit = "km/h")
# 46296.3 m^2/s

# What is expected displacement in 1 h at that SDR?
get_displacement(46296.3, hours(1))
# 12909.95 m = 12.9 km/h (much slower than sprint speed!)
```

---

calc_w *Calculate* w(x)

Description

Calculates the value of the exponential habitat selection function

Usage

```
calc_w(f, b, newdata)
```

Arguments

f	[formula] A model formula.
b	[numeric] A named vector of coefficients.
newdata	[data.frame] A data.frame to predict eHSF values.

Details

This is actually like to be w(x) * phi(x) for an iSSF.

---

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 make_track.
...	Further arguments, none implemented.
spatial	[logical(1)=FALSE] Whether or not to return a SpatialPoints-object.

Value

The centroid of a track as numeric vector if spatial = FALSE, otherwise as SpatialPoints.

Examples

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

---

check_time_unit *Calculate Speed*

Description

Calculates speed

Usage

```
check_time_unit(tu)
```

Arguments

tu	The time_unit parameter to check.

Details

Calculate Change in NSD

Calculates change in NSD

Check time_unit Parameter

Internal function to check time_unit parameter in various cleaning functions.

coercion *Coerce a track to other formats.*

Description

Several other packages provides methods to analyze movement data, and `amt` provides coercion methods to some packages.

Usage

```
as_sf(x, ...)

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

as_sp(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_telemetry(x, ...)

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

as_moveHMM(x, ...)

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

Arguments

x	`[track_xy, track_xyt]` A track created with `make_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).

Value

An object of the class to which coercion is performed to.

`conf_envelope` *Create confidence envelopes from a* `uhc_data_frame`

Description

Simplifies sampled distributions in a `uhc_data_frame` to confidence envelopes

Usage

```
conf_envelope(x, levels = c(0.95, 1))
```

Arguments

`x`	`[uhc_data]` An object of class `uhc_data_frame`, as returned by the function `as.data.frame.uhc_data()`.
`levels`	`[numeric]` A numeric vector specifying the desired confidence levels. Defaults to `c(0.95, 1)` to create 95% and 100% confidence intervals.

Details

This can dramatically improve plotting time for UHC plots by simplifying the many sampled lines down to the boundaries of a polygon.

Value

Returns a `data.frame` with columns:

- `var`: The name of the variable
- `x`: The x-coordinate of the density plot (the value of `var`).
- `label`: If `var` is a `factor`, the label for the value given by `x`.
- `U`: The y-coordinate of the density plot for the use distribution.
- `A`: The y-coordinate of the density plot for the availability distribution.
- `CI*_lwr`: The lower bound of the confidence envelope for the corresponding confidence level.
- `CI*_upr`: The upper bound of the confidence envelope for the corresponding confidence level.

Author(s)

Brian J. Smith

See Also

`prep_uhc()`, `plot.uhc_envelopes()`

convert_angles *Converts angles to radians*

Description

Converts angles to radians

Usage

```
as_rad(x)

as_degree(x)
```

Arguments

x	[numeric] Angles in degrees or rad.

Value

A numeric vector with the converted angles.

Examples

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

# The default unit of turning angles is rad.
data(deer)
deer |> steps() |> mutate(ta_ = as_degree(ta_))
```

coords *Coordinates of a track.*

Description

Coordinates of a track.

Usage

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

Arguments

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

Value

[tibble]
The coordinates.

Examples

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

---

cum_ud *Calculate a cumulative UD*

Description

Calculate the cumulative utilization distribution (UD).

Usage

```
hr_cud(x, ...)

## S3 method for class 'SpatRaster'
hr_cud(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.

diff *Difference in x and y*

Description

Difference in x and y coordinates.

Usage

```
diff_x(x, ...)

diff_y(x, ...)
```

Arguments

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

Value

Numeric vector

distributions *Functions create statistical distributions*

Description

make_distributions creates a distribution suitable for using it with integrated step selection functions

Usage

```
make_distribution(name, params, vcov = NULL, ...)

make_exp_distr(rate = 1)

make_hnorm_distr(sd = 1)

make_lnorm_distr(meanlog = 0, sdlog = 1)

make_unif_distr(min = -pi, max = pi)

make_vonmises_distr(kappa = 1, vcov = NULL)

make_gamma_distr(shape = 1, scale = 1, vcov = NULL)
```

Arguments

name	[char(1)] Short name of distribution. See available_distr() for all currently implemented distributions.
params	[list] A named list with parameters of the distribution.
vcov	[matrix] A matrix with variance and covariances.
...	none implemented.
rate	[double(1)>0] The rate of the exponential distribution.
sd	[double(1)>0] The standard deviation of the half-normal distribution.
meanlog	[double(1)>0] The standard deviation of the half-normal distribution.
sdlog	[double(1)>0] The standard deviation of the half-normal distribution.
min	[double(1)] The minimum of the uniform distribution.
max	[double(1)] The minimum of the uniform distribution.

kappa	[double(1)>=0] Concentration parameter of the von Mises distribution.
shape, scale	[double(1)>=0] Shape and scale of the Gamma distribution

Value

A list of class amt_distr that contains the name (name) and parameters (params) of a distribution.

---

distr_name *Name of step-length distribution and turn-angle distribution*

Description

Name of step-length distribution and turn-angle distribution

Usage

```
sl_distr_name(x, ...)

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

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

ta_distr_name(x, ...)

ta_distr_name(x, ...)

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

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

Arguments

x	Random steps or fitted model
...	None implemented.

Value

Character vector of length 1.

extent *Extent of a track*

Description

Obtain the extent of a track in x y or both directions

Usage

```
extent_x(x, ...)

extent_y(x, ...)

extent_both(x, ...)

extent_max(x, ...)
```

Arguments

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

Value

Numeric vector with the extent.

Extract.uhc_data *Subset a* uhc_data *object*

Description

Subset a uhc_data object

Usage

```
## S3 method for class 'uhc_data'
x[i]
```

Arguments

x	[uhc_data] A uhc_data object to subset.
i	[numeric or character] A numeric vector to subset variables by position or a character vector to subset variables by name.

`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, ...)

extract_covariates_var_time(x, ...)

## S3 method for class 'track_xyt'
extract_covariates_var_time(
  x,
  covariates,
  when = "any",
  max_time,
  name_covar = "time_var_covar",
  ...
)

## S3 method for class 'steps_xyt'
extract_covariates_var_time(
  x,
  covariates,
  when = "any",
  max_time,
  name_covar = "time_var_covar",
  where = "end",
  ...
)
```

Arguments

x	[track_xy, track_xyt, steps] Either a track created with mk_track or track, or steps.
...	Additional arguments passed to terra::extract().
covariates	[SpatRaster] The (environmental) covariates. For extract_covariates_var_time the argument covariates need to have a z-column (i.e. the time stamp).
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.
when	[character(1)="any"]{"any", "before", "after"} Specifies for for extract_covariates_var_time whether to look before, after or in both direction (any) for the temporally closest environmental raster.
max_time	[Period(1)] The maximum time difference between a relocation and the corresponding raster. If no rasters are within the specified max. distance NA is returned.
name_covar	[character(1)="time_var_covar"] The name of the new column.

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.

Value

A tibble with additional columns for covariate values.

Examples

```
data(deer)
sh_forest <- get_sh_forest()
mini_deer <- deer[1:20, ]
mini_deer |> extract_covariates(sh_forest)
mini_deer |> steps() |> extract_covariates(sh_forest)


# Illustration of extracting covariates along the a step
mini_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, function(x) mean(x == 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 make_track.
...	Further arguments, none implemented.
min_n	[numeric(1)=3] Indicating the minimum number of relocations (=fixes per burst).

Value

A tibble of class track_xy(t).

fit_clogit *Fit a conditional logistic regression*

Description

This function is a wrapper around survival::clogit, 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.

Value

A list with the following entries

- model: The model output.
- sl_: The step length distribution.
- ta_: The turn angle distribution.

---

fit_ctmm *Fit a continuous time movement model with* ctmm

Description

Fit a continuous time movement model with ctmm

Usage

```
fit_ctmm(x, model, uere = NULL, ...)
```

Arguments

x	[track_xyt] A track created with make_track that includes time.
model	[character(1)="bm"]{"iid", "bm","ou","ouf", "auto"} The autocorrelation model that should be fit to the data. iid corresponds to uncorrelated independent data, bm to Brownian motion, ou to an Ornstein-Uhlenbeck process, ouf to an Ornstein-Uhlenbeck forage process. auto will use model selection with AICc to find the best model.
uere	User Equivalent Range Error, see ?ctmm::uere for more details.
...	Additional parameters passed to ctmm::ctmm.fit or ctmm::ctmm.select for model = "auto"

Value

An object of class ctmm from the package ctmm.

References

C. H. Fleming, J. M. Calabrese, T. Mueller, K.A. Olson, P. Leimgruber, W. F. Fagan, "From fine-scale foraging to home ranges: A semi-variance approach to identifying movement modes across spatiotemporal scales", The American Naturalist, 183:5, E154-E167 (2014).

Examples

```
data(deer)
mini_deer <- deer[1:20, ]
m1 <- fit_ctmm(mini_deer, "iid")
summary(m1)
```

`fit_distr` *Fit distribution to data*

Description

Wrapper to fit a distribution to data. Currently implemented distributions are the exponential distribution (`exp`), the gamma distribution (`gamma`) and the von Mises distribution (`vonmises`).

Usage

```
fit_distr(x, dist_name, na.rm = TRUE)
```

Arguments

`x`	`[numeric(>1)]` The observed data.
`dist_name`	`[character(1)]{"exp", "gamma", "unif", "vonmises"}` The name of the distribution.
`na.rm`	`[logical(1)=TRUE]` Indicating whether `NA` should be removed before fitting the distribution.

Value

An `amt_distr` object, which consists of a list with the `name` of the distribution and its parameters (saved in `params`).

Examples

```
set.seed(123)
dat <- rexp(1e3, 2)
fit_distr(dat, "exp")
```

fit_logit *Fit logistic regression*

Description

This function is a wrapper around `stats::glm` for a 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`.

Value

A list with the model output.

flag_defunct_clusters *Flag Defunct Clusters*

Description

Flags defunct clusters at the end of a track

Usage

```
flag_defunct_clusters(x, zeta, eta, theta, ...)

## S3 method for class 'track_xyt'
flag_defunct_clusters(x, zeta, eta, theta, ...)
```

Arguments

`x`	`[track_xyt]` A `track_xyt` object.
`zeta`	`[numeric]` See details.
`eta`	`[numeric]` See details.
`theta`	`[numeric]` See details.
`...`	Addtional arguments. None currently implemented.

Details

Locations at the end of a trajectory may represent a dropped collar or an animal mortality. In some cases, the device may be recording locations for quite some time that are not biologically meaningful. This function aims to flag those locations at the end of the trajectory that belong to a mortality (or similar) cluster. The first location at the cluster remains unflagged, but all subsequent locations are flagged.

The algorithm detects steps that represent zero movement, within a precision threshold given by `zeta`. That is, if `zeta = 5` (units determined by CRS; typically meters), all points that differ by less than 5 will be considered zero movement. Consecutive steps of zero movement (within the tolerance) form a cluster. The parameter `eta` gives the cutoff for the minimum number of zero steps to be considered a cluster. Finally, the algorithm requires that clusters persist without a non-zero step for a minimum amount of time, given by `theta`.

Value

Returns `x` (a `track_xyt`) with a flagging column added (`x$defunct_cluster_`).

Author(s)

Brian J. Smith and Johannes Signer, based on code by Tal Avgar

See Also

`flag_duplicates()`, `flag_fast_steps()`, `flag_roundtrips()`

`flag_duplicates` *Flag Low Quality Duplicates*

Description

Flags locations with duplicate timestamps by DOP and distance

Usage

```
flag_duplicates(x, gamma, time_unit = "mins", DOP = "dop", ...)

## S3 method for class 'track_xyt'
flag_duplicates(x, gamma, time_unit = "mins", DOP = "dop", ...)
```

Arguments

`x`	`[track_xyt]` A `track_xyt` object.
`gamma`	`[numeric or Period]` The temporal tolerance defining duplicates. See details below. If `numeric`, its units are defined by `time_unit`. If `Period`, `time_unit` is ignored.
`time_unit`	`[character]` Character string giving time unit for `gamma`. Should be `"secs"`, `"mins"`, or `"hours"`. Ignored if `class(gamma) == "Period"`.

DOP	[character] A character string giving the name of the column containing the dilution of precision (DOP) data. See details below.
...	Addtional arguments. None currently implemented.

Details

Locations are considered duplicates if their timestamps are within gamma of each other. However, the function runs sequentially through the track object, so that only timestamps after the focal point are flagged as duplicates (and thus removed from further consideration). E.g., if gamma = minutes(5), then all locations with timestamp within 5 minutes after the focal location will be considered duplicates.

When duplicates are found, (1) the location with the lowest dilution of precision (given by DOP column) is kept. If there are multiple duplicates with equally low DOP, then (2) the one closest in space to previous location is kept. In the event of exact ties in DOP and distance, (3) the first location is kept. This is unlikely unless there are exact coordinate duplicates.

In the case that the first location in a trajectory has a duplicate, there is no previous location with which to calculate a distance. In that case, the algorithm skips to (3) and keeps the first location.

In the event your data.frame does not have a DOP column, you can insert a dummy with constant values such that all duplicates will tie, and distance will be the only criterion (e.g., x$dop <- 1). In the event you do have an alternate measure of precision where larger numbers are more precise (e.g., number of satellites), simply multiply that metric by -1 and pass it as if it were DOP.

Internally, the function drops duplicates as it works sequentially through the data.frame. E.g., if location 5 was considered a duplicate of location 4 – and location 4 was higher quality – then location 5 would be dropped. The function would then move on to location 6 (since 5 was already dropped). However, the object returned to the user has all the original rows of x – i.e., locations are flagged rather than removed.

Value

Returns x (a track_xyt) with a flagging column added (x$duplicate_).

Author(s)

Brian J. Smith, based on code by Johannes Signer and Tal Avgar

See Also

flag_fast_steps(), flag_roundtrips(), flag_defunct_clusters()

---

flag_fast_steps *Flag Fast Steps*

Description

Flags locations that imply SDR exceeding a threshold

Usage

```
flag_fast_steps(x, delta, time_unit = "secs", ...)

## S3 method for class 'track_xyt'
flag_fast_steps(x, delta, time_unit = "secs", ...)
```

Arguments

x	[track_xyt] A track_xyt object.
delta	[numeric] The threshold SDR over which steps are flagged. See details.
time_unit	[character] Character string giving time unit. Should be "secs", "mins", or "hours". See details.
...	Addtional arguments. None currently implemented.

Details

Locations are flagged if the SDR from the previous location to the current location exceeds delta. Internally, flagged locations are dropped from future consideration.

The time_unit should be the same time unit with which the SDR threshold was calculated. SDR is typically calculated in m^2/s, so time_unit defaults to "secs". The spatial unit is determined by the CRS, which should typically be in meters.

Value

Returns x (a track_xyt) with a flagging column added (x$fast_step_).

Author(s)

Brian J. Smith, based on code by Johannes Signer and Tal Avgar

See Also

flag_duplicates(), flag_roundtrips(), flag_defunct_clusters()

---

flag_roundtrips *Flag Fast Round Trips*

Description

Flags locations that imply fast round trips

Usage

```
flag_roundtrips(x, delta, epsilon, time_unit = "secs", ...)

## S3 method for class 'track_xyt'
flag_roundtrips(x, delta, epsilon, time_unit = "secs", ...)
```

Arguments

x	[track_xyt] A track_xyt object.
delta	[numeric] The threshold SDR for flagging. Locations that imply both legs of a round trip with SDR > delta/epsilon are flagged. See details.
epsilon	[numeric] A scaling factor to create the threshold for flagging.
time_unit	[character] Character string giving time unit. Should be "secs", "mins", or "hours". See details.
...	Addtional arguments. None currently implemented.

Details

Locations implying a single fast step can be flagged using flag_fast_steps(). However, it is more likely that a single location is imprecise if it implies an unrealistically fast out-and-back round trip. In that case, the user might be willing to scale the threshold SDR. In this function, delta gives the base SDR and epsilon is the scaling factor, such that locations are considered for flagging if the SDR from the previous location (location i - 1) to the focal location (i) [call it sdr1] and the focal location (i) to the next location (i + 1) [call it sdr2] both have SDR > delta/epsilon.

In that case, the SDR from the previous location (i - 1) to the next location (i + 1) is computed; i.e., the SDR assuming we omit the focal location (i) [call it sdr3]. The remaining locations should be closer together than they are to the omitted location. Thus the focal location is flagged if (sdr1 > epsilon * sdr3) & (sdr2 > epsilon * sdr3).

Note that epsilon both *decreases* delta in the out-and-back case and *increases* sdr3 (between the remaining neighbors).

Internally, flagged locations are dropped from future consideration.

The time_unit should be the same time unit with which the SDR threshold was calculated. SDR is typically calculated in m^2/s, so time_unit defaults to "secs". The spatial unit is determined by the CRS, which should typically be in meters. The epsilon parameter is unitless.

Value

Returns x (a track_xyt) with a flagging column added (x$fast_roundtrip_).

Author(s)

Brian J. Smith, based on code by Johannes Signer and Tal Avgar

See Also

flag_duplicates(), flag_fast_steps(), flag_defunct_clusters()

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 make_track.
...	Further arguments, none implemented.

Value

A vector of class POSIXct.

get_amt_fisher_covars *Helper function to get fisher covars*

Description

The current version of terra (1.7.12) requires SpatRasters to be wrapped in order to be saved locally. This function unwraps the covariates for the fisher data and returns a list.

Usage

```
get_amt_fisher_covars()
```

Value

A list with covariates

`get_crs` *Obtains the Coordinate Reference Systems*

Description

Returns the `proj4string` of an object.

Usage

```
get_crs(x, ...)
```

Arguments

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

Value

The `proj4string` of the CRS.

Examples

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

`get_displacement` *Calculate Expected Displacement*

Description

Calculates expected displacement for a given SDR and time span

Usage

```
get_displacement(delta, time_span)
```

Arguments

`delta`	`[numeric]` A squared displacement rate (SDR), such as that returned by `calculate_sdr()`. Units assumed to be `m^2/s`.
`time_span`	`[Period]` A `lubridate Period` giving the time span for which to calculate expected displacement.

Value

Returns a numeric vector (of length 1) with the expected displacement in meters.

Author(s)

Johannes Signer and Brian J. Smith

See Also

`calculate_sdr()`

`get_distr` *Obtain the step length and/or turn angle distributions from random steps or a fitted model.*

Description

Obtain the step length and/or turn angle distributions from random steps or a fitted model.

Usage

```
sl_distr(x, ...)

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

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

ta_distr(x, ...)

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

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

Arguments

x	Random steps or fitted model
...	None implemented.

Value

An amt distribution

get_max_dist *Get the maximum distance*

Description

Helper function to get the maximum distance from a fitted model.

Usage

```
get_max_dist(x, ...)

## S3 method for class 'fit_clogit'
get_max_dist(x, p = 0.99, ...)
```

Arguments

x	[fitted_issf] A fitted integrated step-selection function.
...	Further arguments, none implemented.
p	[numeric(1)]{0.99} The quantile of the step-length distribution.

get_sh_forest *Helper function to get forest cover*

Description

The current version of terra (1.7.12) requires SpatRasters to be wrapped in order to be saved locally. This function unwraps the the forest layer and returns a SpatRast.

Usage

```
get_sh_forest()
```

Value

A SpatRast with forest cover.

has_crs *Check for Coordinate Reference Systems (CRS)*

Description

Checks if an object has a CRS.

Usage

```
has_crs(x, ...)
```

Arguments

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

Value

Logic vector of length 1.

Examples

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

hr_akde *Home ranges*

Description

Functions to calculate animal home ranges from a `track_xy*`. `hr_mcp`, `hr_kde`, and `hr_locoh` calculate the minimum convex polygon, kernel density, and local convex hull home range respectively.

Usage

```
hr_akde(x, ...)

## S3 method for class 'track_xyt'
hr_akde(
  x,
  model = fit_ctmm(x, "iid"),
  keep.data = TRUE,
  trast = make_trast(x),
  levels = 0.95,
  wrap = FALSE,
```

```
  ...
)

hr_kde(x, ...)

## S3 method for class 'track_xy'
hr_kde(
  x,
  h = hr_kde_ref(x),
  trast = make_trast(x),
  levels = 0.95,
  keep.data = TRUE,
  wrap = FALSE,
  ...
)

hr_locoh(x, ...)

## S3 method for class 'track_xy'
hr_locoh(
  x,
  n = 10,
  type = "k",
  levels = 0.95,
  keep.data = TRUE,
  rand_buffer = 1e-05,
  ...
)

hr_mcp(x, ...)

hr_od(x, ...)
```

Arguments

x	[track_xy, track_xyt] A track created with make_track.
...	Further arguments, none implemented.
model	A continuous time movement model. This can be fitted either with ctmm::ctmm.fit or fit_ctmm.
keep.data	[logic(1)] Should the original tracking data be included in the estimate?
trast	[SpatRast] A template raster for kernel density home-ranges.
levels	[numeric] The isopleth levels used for calculating home ranges. Should be 0 < level < 1.
wrap	[logical(1)] If TRUE the UD is wrapped (see terra::wrap()).

h	[numeric(2)] The bandwidth for kernel density estimation.
n	[integer(1)] The number of neighbors used when calculating local convex hulls.
type	k, r or a. Type of LoCoH.
rand_buffer	[numeric(1)] Random buffer to avoid polygons with area 0 (if coordinates are numerically identical).

Value

A hr-estimate.

References

Worton, B. J. (1989). Kernel methods for estimating the utilization distribution in home-range studies. *Ecology, 70(1)*, 164-168. C. H. Fleming, W. F. Fagan, T. Mueller, K. A. Olson, P. Leimgruber, J. M. Calabrese, "Rigorous home-range estimation with movement data: A new autocorrelated kernel-density estimator", Ecology, 96:5, 1182-1188 (2015).

Fleming, C. H., Fagan, W. F., Mueller, T., Olson, K. A., Leimgruber, P., & Calabrese, J. M. (2016). Estimating where and how animals travel: an optimal framework for path reconstruction from autocorrelated tracking data. Ecology, 97(3), 576-582.

Examples

```
data(deer)
mini_deer <- deer[1:100, ]

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

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

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

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

# akde
data(deer)
mini_deer <- deer[1:20, ]
ud1 <- hr_akde(mini_deer) # uses an iid ctmm
```

```
ud2 <- hr_akde(mini_deer, model = fit_ctmm(deer, "ou")) # uses an OU ctmm

# od

data(deer)
ud1 <- hr_od(deer) # uses an iid ctmm
ud2 <- hr_akde(deer, model = fit_ctmm(deer, "ou")) # uses an OU ctmm
```

`hr_area` *Home-range area*

Description

Obtain the area of a home-range estimate, possible at different isopleth levels.

Usage

```
hr_area(x, ...)

## S3 method for class 'hr'
hr_area(x, units = FALSE, ...)

## S3 method for class 'SpatRaster'
hr_area(x, level = 0.95, ...)

## S3 method for class 'akde'
hr_area(x, units = FALSE, ...)
```

Arguments

`x`	An object of class `hr`
`...`	Further arguments, none implemented.
`units`	`[logic(1)]` Should areas be returned as units? If `FALSE` areas are returned as numeric values.
`level`	The level at which the area will be calculated.

Value

A `tibble` with the home-range level and the area.

`hr_isopleths` *Home-range isopleths*

Description

Obtain the isopleths of a home-range estimate, possible at different isopleth levels.

Usage

```
hr_isopleths(x, ...)

## S3 method for class 'PackedSpatRaster'
hr_isopleths(x, levels, descending = TRUE, ...)

## S3 method for class 'SpatRaster'
hr_isopleths(x, levels, descending = TRUE, ...)

## S3 method for class 'mcp'
hr_isopleths(x, descending = TRUE, ...)

## S3 method for class 'locoh'
hr_isopleths(x, descending = TRUE, ...)

## S3 method for class 'hr_prob'
hr_isopleths(x, descending = TRUE, ...)

## S3 method for class 'akde'
hr_isopleths(x, conf.level = 0.95, descending = TRUE, ...)
```

Arguments

`x`	An object of class `hr`
`...`	Further arguments, none implemented.
`levels`	`[numeric]` The isopleth levels used for calculating home ranges. Should be `0 < level < 1`.
`descending`	`[logical = TRUE]` Indicating if levels (and thus the polygons) should be ordered in descending (default) or not.
`conf.level`	The confidence level for isopleths for `aKDE`.

Value

A `tibble` with the home-range level and a simple feature columns with the isoploth as multipolygon.

hr_kde_lscv *Least square cross validation bandwidth*

Description

Use least square cross validation (lscv) to estimate bandwidth for kernel home-range estimation.

Usage

```
hr_kde_lscv(
  x,
  range = do.call(seq, as.list(c(hr_kde_ref(x) * c(0.1, 2), length.out = 100))),
  which_min = "global",
  rescale = "none",
  trast = make_trast(x)
)
```

Arguments

x	[track_xy, track_xyt] A track created with make_track.
range	numeric vector with different candidate h values.
which_min	A character indicating if the global or local minimum should be searched for.
rescale	[character(1)] Rescaling method for reference bandwidth calculation. Must be one of "unit-var", "xvar", or "none".
trast	A template raster.

Details

hr_kde_lscv calculates least square cross validation bandwidth. This implementation is based on Seaman and Powell (1996). If whichMin is "global" the global minimum is returned, else the local minimum with the largest candidate bandwidth is returned.

Value

vector of length two.

References

Seaman, D. E., & Powell, R. A. (1996). An evaluation of the accuracy of kernel density estimators for home range analysis. *Ecology, 77(7)*, 2075-2085.

`hr_kde_ref_scaled` *Select a bandwidth for Kernel Density Estimation*

Description

Use two dimensional reference bandwidth to select a bandwidth for kernel density estimation. Find the smallest value for bandwidth (h) that results in n polygons (usually n=1) contiguous polygons at a given level.

Usage

```
hr_kde_ref_scaled(
  x,
  range = hr_kde_ref(x)[1] * c(0.01, 1),
  trast = make_trast(x),
  num.of.parts = 1,
  levels = 0.95,
  tol = 0.1,
  max.it = 500L
)
```

Arguments

`x`	A `track_xy*`.
`range`	Numeric vector, indicating the lower and upper bound of the search range. If `range` is to large with regard to `trast`, the algorithm will fail.
`trast`	A template `RasterLayer`.
`num.of.parts`	Numeric numeric scalar, indicating the number of contiguous polygons desired. This will usually be one.
`levels`	The home range level.
`tol`	Numeric scalar, indicating which difference of to stop.
`max.it`	Numeric scalar, indicating the maximum number of acceptable iterations.

Details

This implementation uses a bisection algorithm to the find the smallest value value for the kernel bandwidth within `range` that produces an home-range isopleth at `level` consisting of `n` polygons. Note, no difference is is made between the two dimensions.

Value

`list` with the calculated bandwidth, exit status and the number of iteration.

References

Kie, John G. "A rule-based ad hoc method for selecting a bandwidth in kernel home-range analyses." Animal Biotelemetry 1.1 (2013): 1-12.

hr_overlaps *Methods to calculate home-range overlaps*

Description

Methods to calculate the overlap of two or more home-range estimates.

Usage

```
hr_overlap(x, ...)

## S3 method for class 'hr'
hr_overlap(x, y, type = "hr", conditional = FALSE, ...)

## S3 method for class 'list'
hr_overlap(
  x,
  type = "hr",
  conditional = FALSE,
  which = "consecutive",
  labels = NULL,
  ...
)
```

Arguments

x, y	A home-range estimate
...	Further arguments, none implemented.
type	[character](1) Type of index, should be one of hr, phr, vi, ba, udoi, or hd.
conditional	[logical](1) Whether or not conditional UDs are used. If TRUE levels from that were used to estimate home ranges will be used.
which	[character = "consecutive"] Should only consecutive overlaps be calculated or all combinations?
labels	[character=NULL] Labels for different instances. If NULL (the default) numbers will be used.

Value

data.frame with the isopleth level and area in units of the coordinate reference system.

hr_overlap_feature *Calculate the overlap between a home-range estimate and a polygon*

Description

Sometimes the percentage overlap between a spatial polygon an a home-range is required.

Usage

```
hr_overlap_feature(x, sf, direction = "hr_with_feature", feature_names = NULL)
```

Arguments

x	A home-range estimate.
sf	An object of class sf containing polygons
direction	The direction.
feature_names	optional feature names

Value

A tibble

hr_to_sf *Convert home ranges to* sfc

Description

Convert a list column with many home-range estimates to a tibble with a geometry column (as used by the sf-package).

Usage

```
hr_to_sf(x, ...)

## S3 method for class 'tbl_df'
hr_to_sf(x, col, ...)
```

Arguments

x	A tibble with a list column with individual home ranges.
...	Additional columns that should be transferred to the new tibble.
col	The column where the home

Value

A `data.frame` with a simple feature column (from the `sf`) package.

Examples

```
data("amt_fisher")
hr <- amt_fisher |> nest(data = -id) |>
  mutate(hr = map(data, hr_mcp), n = map_int(data, nrow)) |>
  hr_to_sf(hr, id, n)

hr <- amt_fisher |> nest(data = -id) |>
  mutate(hr = map(data, hr_kde), n = map_int(data, nrow)) |>
  hr_to_sf(hr, id, n)
```

---

hr_ud *Obtain the utilization distribution of a probabilistic home-range estimate*

Description

Obtain the utilization distribution of a probabilistic home-range estimate

Usage

```
hr_ud(x, ...)
```

Arguments

`x`	`[hr_prob]` The home-range estimate
`...`	Further arguments, none implemented.

Value

`SpatRaster`

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 make_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 = 31467)


inspect(x)
inspect(x, cluster = FALSE)
inspect(x, popup = 1:nrow(x), cluster = FALSE)
```

issf_drop_stratum *Create formula without stratum from iSSF*

Description

Creates a formula without stratum variable

Usage

```
issf_drop_stratum(object, l)
```

Arguments

object	[fit_clogit] Fitted iSSF.
l	[list] List returned by prep_test_dat.fit_clogit()

issf_w_form *Create habitat formula from iSSF*

Description

Creates a formula without movement variables

Usage

```
issf_w_form(object, l)
```

Arguments

object	[fit_clogit] Fitted iSSF.
l	[list] List returned by prep_test_dat.fit_clogit()

log_rss *Calculate log-RSS for a fitted model*

Description

Calculate log-RSS(x1, x2) for a fitted RSF or (i)SSF

Usage

```
log_rss(object, ...)

## S3 method for class 'glm'
log_rss(object, x1, x2, ci = NA, ci_level = 0.95, n_boot = 1000, ...)

## S3 method for class 'fit_clogit'
log_rss(object, x1, x2, ci = NA, ci_level = 0.95, n_boot = 1000, ...)
```

Arguments

object	[fit_logit, fit_clogit] A fitted RSF or (i)SSF model.
...	Further arguments, none implemented.
x1	[data.frame] A data.frame representing the habitat values at location x_1. Must contain all fitted covariates as expected by predict().
x2	[data.frame] A 1-row data.frame representing the single hypothetical location of x_2. Must contain all fitted covariates as expected by predict().
ci	[character] Method for estimating confidence intervals around log-RSS. NA skips calculating CIs. Character string "se" uses standard error method and "boot" uses empirical bootstrap method.
ci_level	[numeric] Level for confidence interval. Defaults to 0.95 for a 95% confidence interval.
n_boot	[integer] Number of bootstrap samples to estimate confidence intervals. Ignored if ci != "boot".

Details

This function assumes that the user would like to compare relative selection strengths from at least one proposed location (x1) to exactly one reference location (x2).

The objects object$model, x1, and x2 will be passed to predict(). Therefore, the columns of x1 and x2 must match the terms in the model formula exactly.

Value

Returns a `list` of class `log_rss` with four entries:

- `df`: A `data.frame` with the covariates and the `log_rss`
- `x1`: A `data.frame` with covariate values for `x1`.
- `x2`: A `data.frame` with covariate values for `x2`.
- `formula`: The formula used to fit the model.

Author(s)

Brian J. Smith

References

- Avgar, T., Lele, S.R., Keim, J.L., and Boyce, M.S.. (2017). Relative Selection Strength: Quantifying effect size in habitat- and step-selection inference. *Ecology and Evolution*, 7, 5322–5330.
- Fieberg, J., Signer, J., Smith, B., & Avgar, T. (2021). A "How to" guide for interpreting parameters in habitat-selection analyses. *Journal of Animal Ecology*, 90(5), 1027-1043.

See Also

See Avgar *et al.* 2017 for details about relative selection strength.

Default plotting method available: `plot.log_rss()`

Examples

```
# RSF ----------------------------------------------------
# Fit an RSF, then calculate log-RSS to visualize results.

# Load packages
library(ggplot2)

# Load data
data("amt_fisher")
amt_fisher_covar <- get_amt_fisher_covars()

# Prepare data for RSF
rsf_data <- amt_fisher |>
  filter(name == "Lupe") |>
  make_track(x_, y_, t_) |>
  random_points() |>
  extract_covariates(amt_fisher_covar$elevation) |>
  extract_covariates(amt_fisher_covar$popden) |>
  extract_covariates(amt_fisher_covar$landuse) |>
  mutate(lu = factor(landuse))

# Fit RSF
m1 <- rsf_data |>
```

```
  fit_rsf(case_ ~ lu + elevation + popden)

# Calculate log-RSS
# data.frame of x1s
x1 <- data.frame(lu = factor(50, levels = levels(rsf_data$lu)),
                 elevation = seq(90, 120, length.out = 100),
                 popden = mean(rsf_data$popden))
# data.frame of x2 (note factor levels should be same as model data)
x2 <- data.frame(lu = factor(50, levels = levels(rsf_data$lu)),
                 elevation = mean(rsf_data$elevation),
                 popden = mean(rsf_data$popden))
# Calculate (use se method for confidence interval)
logRSS <- log_rss(object = m1, x1 = x1, x2 = x2, ci = "se")

# Plot
ggplot(logRSS$df, aes(x = elevation_x1, y = log_rss)) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "gray") +
  geom_ribbon(aes(ymin = lwr, ymax = upr), fill = "gray80") +
  geom_line() +
  xlab(expression("Elevation " * (x[1]))) +
  ylab("log-RSS") +
  ggtitle(expression("log-RSS" * (x[1] * ", " * x[2]))) +
  theme_bw()

# SSF -----------------------------------------------------
# Fit an SSF, then calculate log-RSS to visualize results.

# Load data
data(deer)
sh_forest <- get_sh_forest()

# Prepare data for SSF
ssf_data <- deer |>
  steps_by_burst() |>
  random_steps(n = 15) |>
  extract_covariates(sh_forest) |>
  mutate(forest = factor(forest, levels = 1:0,
                   labels = c("forest", "non-forest")),
  cos_ta = cos(ta_),
  log_sl = log(sl_))

# Fit an SSF (note model = TRUE necessary for predict() to work)
m2 <- ssf_data |>
  fit_clogit(case_ ~ forest + strata(step_id_), model = TRUE)

# Calculate log-RSS
# data.frame of x1s
x1 <- data.frame(forest = factor(c("forest", "non-forest")))
# data.frame of x2
x2 <- data.frame(forest = factor("forest", levels = levels(ssf_data$forest)))
# Calculate
logRSS <- log_rss(object = m2, x1 = x1, x2 = x2, ci = "se")
```

```
# Plot
ggplot(logRSS$df, aes(x = forest_x1, y = log_rss)) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "gray") +
  geom_errorbar(aes(ymin = lwr, ymax = upr), width = 0.25) +
  geom_point(size = 3) +
  xlab(expression("Forest Cover " * (x[1]))) +
  ylab("log-RSS") +
  ggtitle(expression("log-RSS" * (x[1] * ", " * x[2]))) +
  theme_bw()
```

`make_issf_model` *Create an* `issf`*-model object from scratch*

Description

In order to simulate from an `issf` a

Usage

```
make_issf_model(
  coefs = c(sl_ = 0),
  sl = make_exp_distr(),
  ta = make_unif_distr()
)
```

Arguments

`coefs`	A named vector with the coefficient values.
`sl`	The tentative step-length distribution.
`ta`	The tentative turn-angle distribution.

Value

An object of `fit_clogit`.

`make_start` *Create an initial step for simulations*

Description

An initial step for simulations. This step can either be created by defining a step from scratch or by using an observed step.

Usage

```
make_start(x, ...)

## S3 method for class 'numeric'
make_start(
  x = c(0, 0),
  ta_ = 0,
  time = Sys.time(),
  dt = hours(1),
  crs = NA,
  ...
)

## S3 method for class 'track_xyt'
make_start(x, ta_ = 0, dt = hours(1), ...)

## S3 method for class 'steps_xyt'
make_start(x, ...)
```

Arguments

x	[steps_xyt,numeric(2)] A step of class steps_xyt or the start coordinates..
...	Further arguments, none implemented.
ta_	[numeric(1)]{0} The initial turn-angle.
time	[POSIXt(1)]{Sys.time()} The time stamp when the simulation starts.
dt	[Period(1)]{hours(1)} The sampling rate of the simulations.
crs	[int(1)]{NA} The coordinate reference system of the start location given as EPSG code.

---

movement_metrics *Movement metrics*

Description

Functions to calculate metrics such as straightness, mean squared 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 make_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).

Value

A numeric vector of length one.

References

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

Examples

```
data(deer)

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

nsd *Net squared displacement (nsd)*

Description

The function `nsd()` calculates the net squared displacement (i.e., the squared distance from the first location of a track) for a track. The function `add_nsd()` add a new column to a track or steps object with the nsd (the column name is `nsd_`).

Usage

```
nsd(x, ...)

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

add_nsd(x, ...)

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

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

Arguments

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

Value

Numeric vector (for `nsd()`) and a tillbe with the original data with a new column for `add_nsd()`.

od *Occurrence Distribution*

Description

`od` is a wrapper around `ctmm::occurrence`. See `help(ctmm::occurrence)` for more details. `rolling_od` estimates occurrence distributions for a subset of a track.

Usage

```
rolling_od(x, ...)

## S3 method for class 'track_xyt'
rolling_od(
  x,
  trast,
  model = fit_ctmm(x, "bm"),
  res.space = 10,
  res.time = 10,
  n.points = 5,
  show.progress = TRUE,
  ...
)

od(x, ...)

## S3 method for class 'track_xyt'
od(x, trast, model = fit_ctmm(x, "bm"), res.space = 10, res.time = 10, ...)
```

Arguments

x	[track_xyt] A track created with make_track that includes time.
...	Further arguments, none implemented.
trast	[SpatRaster] A template raster for the extent and resolution of the result.
model	[An output of fit_ctmm] The autocorrelation model that should be fit to the data. bm corresponds to Brownian motion, ou to an Ornstein-Uhlenbeck process, ouf to an Ornstein-Uhlenbeck forage process.
res.space	[numeric(1)=10] Number of grid point along each axis, relative to the average diffusion (per median timestep) from a stationary point. See also help(ctmm::occurrence).
res.time	[numeric(1)=10] Number of temporal grid points per median timestep.
n.points	[numeric(1)=5] This argument is only relevant for rolling_od and specifies the window size for the od estimation.
show.progress	[logical(1)=TRUE] Indicates if a progress bar is used.

References

Fleming, C. H., Fagan, W. F., Mueller, T., Olson, K. A., Leimgruber, P., & Calabrese, J. M. (2016). Estimating where and how animals travel: an optimal framework for path reconstruction from autocorrelated tracking data. Ecology.

Examples

```
data(deer)
mini_deer <- deer[1:100, ]
trast <- make_trast(mini_deer)
md <- od(mini_deer, trast = trast)
terra::plot(md)

# rolling ud
xx <- rolling_od(mini_deer, trast)
```

`params` *Get parameters from a (fitted) distribution*

Description

Get parameters from a (fitted) distribution

Usage

```
sl_distr_params(x, ...)

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

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

ta_distr_params(x, ...)

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

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

Arguments

`x`	`[amt_distr]` A (fitted) distribution
`...`	None

Value

A list with the parameters of the distribution.

Examples

```
data(deer)
d <- deer |> steps() |> random_steps()
sl_distr_params(d)
ta_distr_params(d)
```

`plot.hr` *Plot a home-range estimate*

Description

Plot a home-range estimate

Usage

```
## S3 method for class 'hr'
plot(x, add.relocations = TRUE, ...)
```

Arguments

`x`	A home-range estimate.
`add.relocations`	`logical(1)` indicates if a relocations should be added to the plot.
`...`	Further arguments, none implemented.

Value

A plot

`plot.log_rss` *Plot a* `log_rss` *object*

Description

Default plot method for an object of class `log_rss`

Usage

```
## S3 method for class 'log_rss'
plot(x, x_var1 = "guess", x_var2 = "guess", ...)
```

Arguments

x	[log_rss] An object returned by the function log_rss().
x_var1	[character] The variable to plot on the x-axis. A string of either "guess" (default – see Details) or the variable name.
x_var2	[character] A second predictor variable to include in the plot. Either "guess" (default – see Details), NA, or the variable name.
...	[any] Additional arguments to be passed to \link{plot}(). *Not currently implemented.*

Details

This function provides defaults for a basic plot, but we encourage the user to carefully consider how to represent the patterns found in their habitat selection model.

The function log_rss() is meant to accept a user-defined input for x1. The structure of x1 likely reflects how the user intended to visualize the results. Therefore, it is possible to "guess" which covariate the user would like to see on the x-axis by choosing the column from x1 with the most unique values. Similarly, if there is a second column with multiple unique values, that could be represented by a color. Note that if the user needs to specify x_var1, then we probably cannot guess x_var2. Therefore, if the user specifies x_var1 != "guess" & x_var2 == "guess", the function will return an error.

This function uses integers to represent colors, and therefore the user can change the default colors by specifying a custom palette() before calling the function.

Value

A plot.

Examples

```
# Load data
data("amt_fisher")
amt_fisher_covar <- get_amt_fisher_covars()

# Prepare data for RSF
rsf_data <- amt_fisher |>
  filter(name == "Leroy") |>
  make_track(x_, y_, t_) |>
  random_points() |>
  extract_covariates(amt_fisher_covar$landuse) |>
  mutate(lu = factor(landuse))

# Fit RSF
m1 <- rsf_data |>
  fit_rsf(case_ ~ lu)

# Calculate log-RSS
# data.frame of x1s
x1 <- data.frame(lu = sort(unique(rsf_data$lu)))
# data.frame of x2 (note factor levels should be same as model data)
```

```
x2 <- data.frame(lu = factor(140,
levels = levels(rsf_data$lu)))
# Calculate
logRSS <- log_rss(object = m1, x1 = x1, x2 = x2)

# Plot
plot(logRSS)
```

`plot.uhc_data` *Plot UHC plots*

Description

Plot an object of class `uhc_data`

Usage

```
## S3 method for class 'uhc_data'
plot(x, ...)
```

Arguments

`x`	`[uhc_data]` An object of class `uhc_data`, as returned by the function `prep_uhc()`.
`...`	Included for consistency with generic `plot()`. Currently ignored.

Details

Makes plots mimicking those in Fieberg et al. (2018), with the bootstrapped distribution in gray, the observed distribution in black, and the available distribution as a dashed red line.

Author(s)

Brian J. Smith

See Also

`prep_uhc()`, `conf_envelope()`, `plot.uhc_envelopes()`

plot.uhc_envelopes *Plot simplified UHC plots*

Description

Plot an object of class uhc_envelopes

Usage

```
## S3 method for class 'uhc_envelopes'
plot(x, ...)
```

Arguments

x	[uhc_envelopes] An object of class uhc_envelopes, as returned by the function conf_envelope().
...	Included for consistency with generic plot(). Currently ignored.

Details

Makes plots mimicking those in Fieberg et al. (2018), with the bootstrapped distribution in gray, the observed distribution in black, and the available distribution as a dashed red line. This differs from plot.uhc_data() in that the bootstrapped distribution (in gray) is drawn as a polygon rather than (many) lines, speeding up plotting performance.

Author(s)

Brian J. Smith

See Also

prep_uhc(), conf_envelope(), plot.uhc_data()

plot_sl *Plot step-length distribution*

Description

Plot step-length distribution

Usage

```
plot_sl(x, ...)

## S3 method for class 'fit_clogit'
plot_sl(x, n = 1000, upper_quantile = 0.99, plot = TRUE, ...)

## S3 method for class 'random_steps'
plot_sl(x, n = 1000, upper_quantile = 0.99, plot = TRUE, ...)
```

Arguments

x	[fit_clogit|random_steps] A fitted step selection or random steps.
...	Further arguments, none implemented.
n	[numeric(1)=1000]{>0} The number of breaks between 0 and upper_quantile.
upper_quantile	[nummeric(1)=0.99]{0-1} The quantile until where the distribution should be plotted. Typically this will be 0.95 or 0.99.
plot	[logical(1)=TRUE] Indicates if a plot should be drawn or not.

Value

A plot of the step-length distribution.

Examples

```
data(deer)

# with random steps
deer[1:100, ] |> steps_by_burst() |> random_steps() |> plot_sl()
deer[1:100, ] |> steps_by_burst() |> random_steps() |> plot_sl(upper_quantile = 0.5)
```

---

prep_uhc *Prepare Data for UHC Plots for a Fitted Model*

Description

Creates data used to make used-habitat calibration plots

Usage

```
prep_uhc(object, test_dat, n_samp = 1000, n_dens = 512, verbose = TRUE)

## S3 method for class 'glm'
prep_uhc(object, test_dat, n_samp = 1000, n_dens = 512, verbose = TRUE)

## S3 method for class 'fit_logit'
prep_uhc(object, test_dat, n_samp = 1000, n_dens = 512, verbose = TRUE)

## S3 method for class 'fit_clogit'
prep_uhc(object, test_dat, n_samp = 1000, n_dens = 512, verbose = TRUE)
```

Arguments

object	[glm, fit_logit, fit_clogit] A fitted RSF or (i)SSF model. Should be fit to *training* dataset separate from the testing data.
test_dat	[data.frame] A data.frame with *testing* data from which to sample test points. Should be separate from the data used to train the model passed to object.
n_samp	[numeric = 1000] A vector of length 1 giving the number of samples to use to characterize the used habitat distribution under the model.
n_dens	[numeric = 512] A numeric vector of length 1 giving the number of equally spaced points at which density (used, available, and sampled) is estimated. Passed to stats::density.default(), which indicates that n should usually be specified as a power of 2.
verbose	[logical] Should messages be displayed (TRUE) or not (FALSE)?

Details

This function performs the heavy lifting of creating UHC plots. It creates the data used later by the plot() method, which actually draws the UHC plots. This function (1) creates density plots of the used and available locations from the *test* data, and (2) resamples the (a) fitted coefficients and (b) test data (weighted by the exponential habitat selection function) to create the distribution of used habitat under the model.

Note that test_dat should contain at least all of the variables that appear in the model object. Any further habitat variables in test_dat will also have UHC plots generated, treating these variables as possible candidate variables that are simply not included in this particular model.

Value

Returns a list of class uhc_data with elements:

- orig: List of data.frames, one per variable (see vars). Each data.frame contains the density plot data (x and y) for the original used (dist == "U") and available (dist == "A") data.
- samp: List of data.frames, one per variable (see vars). Each data.frame contains the density plot data (x and y) for each iteration of bootstrap resampling (iter).

- `vars`: Character vector with names of the habitat variables for which to create UHC plots.
- `type`: Named character vector with the type for each of `vars` (either `"numeric"` or `"factor"`).
- `resp`: Character vector of length 1 with the name of the response variable.

Author(s)

Brian J. Smith

References

Fieberg, J.R., Forester, J.D., Street, G.M., Johnson, D.H., ArchMiller, A.A., and Matthiopoulos, J. 2018. Used-habitat calibration plots: A new procedure for validating species distribution, resource selection, and step-selection models. *Ecography* 41:737–752.

See Also

See Fieberg *et al.* 2018 for details about UHC plots.

Default plotting method available: `plot.uhc_data()`

Coercion to `data.frame`: `as.data.frame.uhc_data()`

Subsetting method: `Extract.uhc_data`

Examples

```
# Load packages
library(amt)
library(dplyr)
library(terra)
library(sf)

# HSF -----------------------------------------------
# Load data
data(uhc_hsf_locs)
data(uhc_hab)
hab <- rast(uhc_hab, type = "xyz", crs = "epsg:32612")
# Convert "cover" layer to factor
levels(hab[[4]]) <- data.frame(id = 1:3,
                               cover = c("grass", "forest", "wetland"))

# Split into train (80%) and test (20%)
set.seed(1)
uhc_hsf_locs$train <- rbinom(n = nrow(uhc_hsf_locs),
                             size = 1, prob = 0.8)
train <- uhc_hsf_locs[uhc_hsf_locs$train == 1, ]
test <- uhc_hsf_locs[uhc_hsf_locs$train == 0, ]

# Available locations
avail_train <- random_points(st_as_sf(st_as_sfc(st_bbox(hab))),
                             n = nrow(train) * 10)
```

```
avail_test <- random_points(st_as_sf(st_as_sfc(st_bbox(hab))),
                            n = nrow(test) * 10)

# Combine with used
train_dat <- train |>
  make_track(x, y, crs = 32612) |>
  mutate(case_ = TRUE) |>
  bind_rows(avail_train) |>
  # Attach covariates
  extract_covariates(hab) |>
  # Assign large weights to available
  mutate(weight = case_when(
    case_ ~ 1,
    !case_ ~ 5000
  ))

test_dat <- test |>
  make_track(x, y, crs = 32612) |>
  mutate(case_ = TRUE) |>
  bind_rows(avail_test) |>
  # Attach covariates
  extract_covariates(hab) |>
  # Assign large weights to available
  mutate(weight = case_when(
    case_ ~ 1,
    !case_ ~ 5000
  ))

# Fit (correct) HSF
hsf1 <- glm(case_ ~ forage + temp + I(temp^2) + pred + cover,
            data = train_dat, family = binomial(), weights = weight)

# Drop weights from 'test_dat'
test_dat$weight <- NULL

# Prep UHC plots
uhc_dat <- prep_uhc(object = hsf1, test_dat = test_dat,
                    n_samp = 500, verbose = TRUE)

# Plot all variables
plot(uhc_dat)

# Plot only first variable
plot(uhc_dat[1])

# Plot only "cover" variable
plot(uhc_dat["cover"])

# Coerce to data.frame
df <- as.data.frame(uhc_dat)

# Simplify sampled lines to confidence envelopes
conf <- conf_envelope(df)
```

```
# Default plot for the envelopes version
plot(conf)
```

`random_numbers` *Sample random numbers*

Description

Sample radom numbers from a distribution created within the `amt` package.

Usage

```
random_numbers(x, n = 100, ...)
```

Arguments

`x`	`[amt_distr]` A distribution object.
`n`	`[integer(1)=100]{>0}` The number of random draws.
`...`	none implemented.

Value

A numermic vector.

`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 'hr'
random_points(x, n = 100, type = "random", presence = NULL, ...)

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

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

Arguments

x	[track_xy, track_xyt] A track created with make_track.
...	[any] None implemented.
n	[integer(1)] The number of random points.
type	[character(1)] Argument passed to sf::st_sample. The default is random.
presence	[track] The presence points, that will be added to the result.
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.

Value

A tibble with the observed and random points and a new column case_ that indicates if a point is observed (case_ = TRUE) or random (case_ TRUE).

Note

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

Examples

```
data(deer)

# track_xyt ------------------------------------------------------------------
# Default settings
rp1 <- random_points(deer)

plot(rp1)

# Ten random points for each observed point
rp <- random_points(deer, n = nrow(deer) * 10)
plot(rp)

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

# 100 random point within the home range
rp <- random_points(hr, n = 100)
plot(rp)

# 100 regular point within the home range
```

```
rp <- random_points(hr, n = 100, type = "regular")
plot(rp)
# 100 hexagonal point within the home range
rp <- random_points(hr, n = 100, type = "hexagonal")
plot(rp)
```

`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 'numeric'
random_steps(
  x,
  n_control = 10,
  angle = 0,
  rand_sl = random_numbers(make_exp_distr(), n = 1e+05),
  rand_ta = random_numbers(make_unif_distr(), n = 1e+05),
  ...
)

## S3 method for class 'steps_xy'
random_steps(
  x,
  n_control = 10,
  sl_distr = fit_distr(x$sl_, "gamma"),
  ta_distr = fit_distr(x$ta_, "vonmises"),
  rand_sl = random_numbers(sl_distr, n = 1e+05),
  rand_ta = random_numbers(ta_distr, n = 1e+05),
  include_observed = TRUE,
  start_id = 1,
  ...
)

## S3 method for class 'bursted_steps_xyt'
random_steps(
  x,
  n_control = 10,
  sl_distr = fit_distr(x$sl_, "gamma"),
```

```
  ta_distr = fit_distr(x$ta_, "vonmises"),
  rand_sl = random_numbers(sl_distr, n = 1e+05),
  rand_ta = random_numbers(ta_distr, n = 1e+05),
  include_observed = TRUE,
  ...
)
```

Arguments

x	Steps.
...	Further arguments, none implemented.
n_control	[integer(1)=10]{>1} The number of control steps paired with each observed step.
angle	[numeric(1) = 0]{-pi < rel_angle < pi} Angle for the first step.
rand_sl	[numeric] Numeric vector with random step lengths an animal can make. This will usually be random numbers drawn from a suitable distribution (e.g., gamma or exponential).
rand_ta	[numeric] Numeric vector with relative turning angles an animal can make. This will usually be random numbers drawn from a suitable distribution (e.g., von Mises or uniform).
sl_distr	[amt_distr] The step-length distribution.
ta_distr	[amt_distr] The turn-angle distribution.
include_observed	[logical(1) = TRUE] Indicates if observed steps are to be included in the result.
start_id	integer Index where the numbering for step ids start.

Value

A tibble of class random_steps.

---

random_steps_simple *Simulate from an ssf model*

Description

Simulate from an ssf model

Usage

```
random_steps_simple(start, sl_model, ta_model, n.control)
```

Arguments

start	First step
sl_model	Step length model to use
ta_model	Turning angle model to use
n.control	How many alternative steps are considered each step

Value

Simulated trajectory

---

range *Geographic range*

Description

The range that in either x, y or both directions, that a track covers.

Usage

```
range_x(x, ...)

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

range_y(x, ...)

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

range_both(x, ...)

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

Arguments

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

Value

Numeric vector with the range.

`redistribution_kernel` *Create a redistribution kernel*

Description

From a fitted integrated step-selection function for a given position a redistribution kernel is calculated (i.e., the product of the movement kernel and the selection function).

Usage

```
redistribution_kernel(
  x = make_issf_model(),
  start = make_start(),
  map,
  fun = function(xy, map) {
     extract_covariates(xy, map, where = "both")
 },
  covars = NULL,
  max.dist = get_max_dist(x),
  n.control = 1e+06,
  n.sample = 1,
  landscape = "continuous",
  compensate.movement = landscape == "discrete",
  normalize = TRUE,
  interpolate = FALSE,
  as.rast = FALSE,
  tolerance.outside = 0
)
```

Arguments

`x`	`[fit_issf]` A fitted integrated step-selection function. Generated either with `fit_issf()` or make `make_issf_model()`.
`start`	`[sim_start]` The start position in space and time. See `make_start()`.
`map`	`[SpatRaster]` A SpatRaster with all covariates.
`fun`	`[function]` A function that is executed on each location of the redistribution kernel. The default function is `extract_covariates()`.
`covars`	`[tibble]` Additional covariates that might be used in the model (e.g., time of day).
`max.dist`	`[numeric(1)]` The maximum distance of the redistribution kernel.

n.control [integer(1)]{1e6}
The number of points of the redistribution kernel (this is only important if landscape = "continuous").

n.sample [integer(1)]{1}
The number of points sampled from the redistribution kernel (this is only important if as.rast = FALSE).

landscape [character(1)]{"continuous"}
If "continuous the redistribution kernel is sampled using a random sample of size n.control. If landscape = "discrete" each cell in the redistribution kernel is used.

compensate.movement
[logical(1)]
Indicates if movement parameters are corrected or not. This only relevant if landscape = 'discrete'.

normalize [logical(1)]{TRUE}
If TRUE the redistribution kernel is normalized to sum to one.

interpolate [logical(1)]{FALSE}
If TRUE a stochastic redistribution kernel is interpolated to return a raster layer. Note, this is just for completeness and is computationally inefficient in most situations.

as.rast [logical(1)]{TRUE}
If TRUE a SpatRaster should be returned.

tolerance.outside
[numeric(1)]{0}
The proportion of the redistribution kernel that is allowed to be outside the map.

---

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 lubridate::Period, indicating the time period to be removed at the beginning of the track.

end A lubridate::Period, indicating the time period to be removed at the end of the track.

Value

A `tibble` without observations that fall within the period of the capture effect.

`remove_incomplete_strata` *Remove strata with missing values for observed steps*

Description

Remove strata with missing values for observed steps

Usage

```
remove_incomplete_strata(x, ...)

## S3 method for class 'random_steps'
remove_incomplete_strata(x, col = "ta_", ...)
```

Arguments

`x`	An object of class `random_steps`.
`...`	Further arguments, none implemented.
`col`	A character with the column name that will be scanned for missing values.

Value

An object of class `random_steps`, where observed steps (`case_ == TRUE`) with missing values (`NA`) in the column `col` are removed (including all random steps).

Examples

```
mini_deer <- deer[1:4, ]

# The first step is removed, because we have `NA` turn angles.
mini_deer |> steps() |> random_steps() |> remove_incomplete_strata() |>
  select(case_, ta_, step_id_)
```

sampling_period *Extract sampling period*

Description

Extracts sampling period from a `track_xyt` object

Usage

```
sampling_period(x, ...)
```

Arguments

`x`	`[track_xyt]` A `track_xyt` object.
`...`	Further arguments, none implemented.

sdr *Calculate SDR for an Object*

Description

Calculates SDR for an object of certain class

Usage

```
sdr(x, time_unit = "secs", append_na = TRUE, ...)

## S3 method for class 'track_xyt'
sdr(x, time_unit = "secs", append_na = TRUE, ...)
```

Arguments

`x`	`[track_xyt]` Object to calculate SDR from. Currently implemented for `track_xyt`.
`time_unit`	`[character]` Character string giving time unit. Should be `"secs"`, `"mins"`, or `"hours"`.
`append_na`	`[logical]` Should `NA` be appended to the end of the vector? Ensures `length(result) == nrow(x)` if appending as a column of `x`.
`...`	Further arguments, none implemented.

Details

`time_unit` defaults to seconds because `calculate_sdr()` returns SDR in m^2/s. We assume the user is also working in a projected CRS with units in meters, thus we expect SDR in m^2/s to be most relevant.

Author(s)

Brian J. Smith and Johannes Signer

See Also

`calculate_sdr()`, `get_displacement()`

---

sh *Relocations of one Red Deer*

Description

1500 GPS fixes 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 from northern Germany.

Usage

```
sh_forest
```

Format

A SpatRast

0 other

1 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.

---

simulate_path *Simulate a movement trajectory.*

Description

Function to simulate a movement trajectory (path) from a redistribution kernel.

Usage

```
simulate_path(x, ...)

## Default S3 method:
simulate_path(x, ...)

## S3 method for class 'redistribution_kernel'
simulate_path(x, n.steps = 100, start = x$args$start, verbose = FALSE, ...)
```

Arguments

x	[redstirubtion_kernel(1)] An object of class redistribution_kernel.
...	Further arguments, none implemented.
n.steps	[integer(1)]{100} The number of simulation steps.
start	[sim_start] The starting point in time and space for the simulations (see make_start()).
verbose	[logical(1)]{FALSE} If TRUE progress of simulations is displayed.

`site_fidelity` *Test for site fidelity of animal movement.*

Description

Calculates two indices (mean squared displacement and linearity) to test for site fidelity. Significance testing is done by permuting step lengths and drawing turning angles from a uniform distribution.

Usage

```
site_fidelity(x, ...)

## S3 method for class 'steps_xy'
site_fidelity(x, n = 100, alpha = 0.05, ...)
```

Arguments

`x`	A track
`...`	None implemented
`n`	Numeric scalar. The number of simulated trajectories.
`alpha`	Numeric scalar. The alpha value used for the bootstrapping.

Value

A list of length 4. `msd_dat` and `li_dat` is the mean square distance and linearity for the real date. `msd_sim` and 'li_sim" are the mean square distances and linearities for the simulated trajectories.

References

Spencer, S. R., Cameron, G. N., & Swihart, R. K. (1990). Operationally defining home range: temporal dependence exhibited by hispid cotton rats. Ecology, 1817-1822.

Examples

```
# real data

data(deer)
ds <- deer |> steps_by_burst()
site_fidelity(ds)
```

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.

ssf_formula *Takes a* clogit *formula and returns a formula without the* strata *and the left-hand side*

Description

Takes a clogit formula and returns a formula without the strata and the left-hand side

Usage

```
ssf_formula(formula)
```

Arguments

formula	A formula object

Examples

```
f1 <- case_ ~ x1 * x2 + strata(step_id_)
ssf_formula(f1)
```

ssf_weights *Given a fitted ssf, and new location the weights for each location is calculated*

Description

Given a fitted ssf, and new location the weights for each location is calculated

Usage

```
ssf_weights(xy, object, compensate.movement = FALSE)
```

Arguments

xy	The new locations.
object	The the fitted (i)SSF.
compensate.movement	Whether or not for the transformation from polar to Cartesian coordinates is corrected.

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,
  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, 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, keep_cols = NULL, ...)

steps(x, ...)

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

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

Arguments

x	[track_xy, track_xyt] A track created with make_track.
...	Further arguments, none implemented
full_circle	[logical(1)=FALSE] If TRUE angles are returned between 0 and $2pi$, otherwise angles are between $-pi$ 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 transferred 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

`dierctions_*()` returns `NA` for 0 step lengths.

`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 `sf::st_distance()`.

Value

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

Examples

```
xy <- tibble(
  x = c(1, 4, 8, 8, 12, 12, 8, 0, 0, 4, 2),
  y = c(0, 0, 0, 8, 12, 12, 12, 12, 8, 4, 2))
trk <- make_track(xy, x, y)

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

# degrees
direction_abs(trk) |> as_degree()

# full circle or not: check
direction_abs(trk, full_circle = TRUE)
direction_abs(trk, full_circle = FALSE)
direction_abs(trk, full_circle = TRUE) |> as_degree()
direction_abs(trk, full_circle = FALSE) |> as_degree()

# 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)
```

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,
  ...
)

summarize_sampling_rate_many(x, ...)

## S3 method for class 'track_xyt'
summarize_sampling_rate_many(x, cols, time_unit = "auto", ...)
```

Arguments

x	A track_xyt.
...	Further arguments, none implemented.
time_unit	[character(1) = "auto"] Which time unit will be used.
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.
cols	Columns used for grouping.

Value

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

Examples

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

```
data(amt_fisher)
# Add the month
amt_fisher |> mutate(yday = lubridate::yday(t_)) |>
summarize_sampling_rate_many(c("id", "yday"))
```

`summarize_sl` *Summarize step lengths*

Description

Summarizes step lengths for a `track_xy*` object

Usage

```
summarize_sl(x, ...)
```

Arguments

`x`	`[track_xy, track_xyt]` A `track_xy*` object.
`...`	Further arguments, none implemented.

`summarize_speed` *Summarize speed*

Description

Summarizes speeds for a `track_xyt` object

Usage

```
summarize_speed(x, ...)
```

Arguments

`x`	`[track_xyt]` A `track_xyt` object.
`...`	Further arguments, none implemented.

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

Description

A convenience wrapper around suncalc::getSunlightTimes to annotate 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, include.crepuscule = FALSE, ...)

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

Arguments

x	[track_xyt,steps_xyt] A track or steps.
...	Further arguments, none implemented.
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?

Value

A tibble with an additional column tod_ that contains the time of the day for each relocation.

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 = NA_crs_,
  order_by_ts = TRUE,
  check_duplicates = FALSE,
  all_cols = FALSE,
  verbose = FALSE
)

make_track(
  tbl,
  .x,
  .y,
  .t,
  ...,
  crs = NA_crs_,
  order_by_ts = TRUE,
  check_duplicates = FALSE,
  all_cols = FALSE,
  verbose = 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	[crs] An optional coordinate reference system of the points. Usually just the epsg code is sufficient.
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.
all_cols	[logical(1)=FALSE] Should all columns be carried over to the track object, default is FALSE.
verbose	[logical(1)=FALSE] Inform when tracks are created.
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.

---

tracked_from_to *Subset to tracking dates*

Description

Subsets a track_xyt object

Usage

```
tracked_from_to(x, from = min(x$t_), to = max(x$t_))
```

Arguments

x	[track_xy, track_xyt] A track_xy* object.
from	[POSIXt] A date and time defining start of subset.
to	[POSIXt] A date and time defining end of subset.

track_align *Selects relocations that fit a new time series*

Description

Functions to only selects relocations that can be aligned with a new time series (within some tolerance).

Usage

```
track_align(x, ...)

## S3 method for class 'track_xyt'
track_align(x, new.times, tolerance, ...)
```

Arguments

x	A track.
...	Further arguments, none implemented.
new.times	The new time trajectory.
tolerance	The tolerance between observed time stamps and new time stamps in seceonds. This should be either a vector of length 1 or length new.times.

Value

A track_xyt.

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.

Value

A resampled track_xyt.

---

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 make_track.
...	Further arguments, none implemented.
crs_to	[crs(1)] Coordinate reference system the data should be transformed to, see sf::st_crs.
crs_from	[crs(1)] Coordinate reference system the data are currently in, see sf::sf_crs. If crs_from is missing, the crs-attribute of the track is used.

Value

A track with transformed coordinates.

See Also

sf::st_transform

Examples

```
data(deer)
get_crs(deer)

# project to geographical coordinates (note the CRS is taken automatically from the object deer).
d1 <- transform_coords(deer, crs_to = 4326)
```

`trast` *Create a template raster layer*

Description

For some home-range estimation methods (e.g., KDE) a template raster is needed. This functions helps to quickly create such a template raster.

Usage

```
make_trast(x, ...)

## S3 method for class 'track_xy'
make_trast(x, factor = 1.5, res = max(c(extent_max(x)/100, 1e-09)), ...)
```

Arguments

`x`	`[track_xy, track_xyt]` A track created with `make_track`.
`...`	Further arguments, none implemented.
`factor`	`[numeric(1)=1.5]{>= 1}` Factor by which the extent of the relocations is extended.
`res`	`[numeric(1)]` Resolution of the output raster.

Value

A `RastLayer` without values.

ua_distr *Summarize distribution of used and available*

Description

Internal function to summarize distribution of numeric or factor variables

Usage

```
ua_distr(name, type, data, lims, resp, n_dens, avail = TRUE)
```

Arguments

name	[character] Name of the column to summarize.
type	[character] Either "numeric" or "factor" as returned by prep_test_dat().
data	[data.frame] The data.frame containing the columns and the response variable.
lims	[numeric(2)] A numeric vector of length 2 containing the range for the density calculation for all variables where type == "numeric" as returned by prep_test_dat(). Will be passed to stats::density.default() arguments from and to.
resp	[character] Name of the response variable.
n_dens	[numeric] A numeric vector of length 1 giving the number of equally spaced points at which density (used, available, and sampled) is estimated. Passed to stats::density.default(), which indicates that n should usually be specified as a power of 2.
avail	[logical] Should distribution be calculated for the available locations? Defaults to TRUE, but should be false when summarizing the bootstrapped "used" samples.

uhc_hab *Simulated habitat rasters for demonstrating UHC plots*

Description

Simulated habitat rasters for demonstrating UHC plots

Usage

```
uhc_hab
```

Format

A `RasterStack` with 1600 cells and 7 variables:

forage Forage biomass in g/m^2^ (resource)

temp mean annual temperature in °C (condition)

pred predator density in predators/100 km^2^ (risk)

cover landcover (forest > grassland > wetland)

dist_to_water distance to the wetland landcover (no effect)

dist_to_cent distance to the centroid of the raster (no effect)

rand random integers (no effect)

`uhc_hsf_locs` *Simulated HSF location data for demonstrating UHC plots*

Description

Simulated HSF location data for demonstrating UHC plots

Usage

```
uhc_hsf_locs
```

Format

A `data.frame` with 2000 rows and 2 variables:

x x-coordinate in UTM Zone 12 (EPSG: 32612)

y Y-coordinate in UTM Zone 12 (EPSG: 32612)

These data were simulated assuming an ordinary habitat selection function (HSF), i.e., all points are independent rather than arising from an underlying movement model.

True parameter values are:

- `forage` = log(5)/500 (resource)
- `temp^2` = -1 * log(2)/36 (condition; quadratic term)
- `temp` = (log(2)/36) * 26 (condition; linear term)
- `pred` = log(0.25)/5 (risk)
- `cover == "forest"` = log(2) (grassland is intercept)
- `cover == "wetland"` = log(1/2) (grassland is intercept)

Note: `temp` is modeled as a quadratic term, with the strongest selection occurring at 13 °C and all other temperatures less selected.

Note: `dist_to_water`, `dist_to_cent`, and `rand` have no real effect on our animal's selection and are included for demonstration purposes.

`uhc_issf_locs` *Simulated iSSF location data for demonstrating UHC plots*

Description

Simulated iSSF location data for demonstrating UHC plots

Usage

```
uhc_issf_locs
```

Format

A `data.frame` with 371 rows and 3 variables:

x x-coordinate in UTM Zone 12 (EPSG: 32612)

y Y-coordinate in UTM Zone 12 (EPSG: 32612)

t timestamp of location (timezone "US/Mountain")

These data were simulated assuming an movement model, i.e., iSSA.

True movement-free habitat selection parameter values are:

- `forage` = log(8)/500 (resource)
- `temp^2` = -1 * log(8)/36 (condition; quadratic term)
- `temp` = (log(8)/36) * 26 (condition; linear term)
- `pred` = log(0.2)/5 (risk)
- `cover == "forest"` = log(2) (grassland is intercept)
- `cover == "wetland"` = log(1/2) (grassland is intercept)
- `dist_to_cent` = -1 * log(10)/500 (keeps trajectory away from boundary)

Note: `temp` is modeled as a quadratic term, with the strongest selection occurring at 13 °C and all other temperatures less selected.

Note: `dist_to_water` and `rand` have no real effect on our animal's selection and are included for demonstration purposes.

True selection-free movement distributions are:

- Step length: `gamma(shape = 3, scale = 25)`
- Turn angle: `vonMises(mu = 0, kappa = 0.5)`

update_distr_man *Manually update* amt_distr

Description

Functions to update amt_distr from iSSF coefficients

Usage

```
update_gamma(dist, beta_sl, beta_log_sl)

update_exp(dist, beta_sl)

update_hnorm(dist, beta_sl_sq)

update_lnorm(dist, beta_log_sl, beta_log_sl_sq)

update_vonmises(dist, beta_cos_ta)
```

Arguments

dist	[amt_distr] The tentative distribution to be updated respective distributions.
beta_sl	[numeric] The estimate of the coefficient of the step length.
beta_log_sl	[numeric] The estimate of the coefficient of the log of the step length.
beta_sl_sq	[character] The name of the coefficient of the square of the step length.
beta_log_sl_sq	[character] The name of the coefficient of the square of log of the step length.
beta_cos_ta	[numeric] The estimate of the coefficient of cosine of the turning angle.

Details

These functions are called internally by update_sl_distr() and update_ta_distr(). However, those simple functions assume that the selection-free step-length and turn-angle distributions are constant (i.e., they do not depend on covariates). In the case of interactions between movement parameters and covariates, the user will want to manually access these functions to update their selection-free movement distributions.

Value

A distribution

Examples

```
#  sh_forest <- get_sh_forest()
#  # Fit an SSF, then update movement parameters.
#
#   #Prepare data for SSF
#  ssf_data <- deer |>
#    steps_by_burst() |>
#    random_steps(n = 15) |>
#    extract_covariates(sh_forest) |>
#    mutate(forest = factor(forest, levels = 1:0,
#                       labels = c("forest", "non-forest")),
#    cos_ta_ = cos(ta_),
#    log_sl_ = log(sl_))
#
#  # Check tentative distributions
#  #    Step length
#  attr(ssf_data, "sl_")
#  #    Turning angle
#  attr(ssf_data, "ta_")
#
#  # Fit an iSSF (note model = TRUE necessary for predict() to work)
#  m1 <- ssf_data |>
#    fit_issf(case_ ~ forest * (sl_ + log_sl_ + cos_ta_) +
#                strata(step_id_), model = TRUE)
#
#  # Update forest step lengths (the reference level)
#  forest_sl <- update_gamma(m1$sl_,
#                            beta_sl = m1$model$coefficients["sl_"],
#                            beta_log_sl = m1$model$coefficients["log_sl_"])
#
#  # Update non-forest step lengths
#  nonforest_sl <- update_gamma(m1$sl_,
#                               beta_sl = m1$model$coefficients["sl_"] +
#                                 m1$model$coefficients["forestnon-forest:sl_"],
#                               beta_log_sl = m1$model$coefficients["log_sl_"] +
#                                 m1$model$coefficients["forestnon-forest:log_sl_"])
#
#  # Update forest turn angles (the reference level)
#  forest_ta <- update_vonmises(m1$ta_,
#                               beta_cos_ta = m1$model$coefficients["cos_ta_"])
#
#  # Update non-forest turn angles
#  nonforest_ta <- update_vonmises(m1$ta_,
#                                  beta_cos_ta = m1$model$coefficients["cos_ta_"] +
#                                    m1$model$coefficients["forestnon-forest:cos_ta_"])
#
```

`update_sl_distr` *Update movement distributions*

Description

Update tentative step length or turning angle distribution from a fitted iSSF.

Usage

```
update_sl_distr(
  object,
  beta_sl = "sl_",
  beta_log_sl = "log_sl_",
  beta_sl_sq = "sl_sq_",
  beta_log_sl_sq = "log_sl_sq_",
  ...
)

update_ta_distr(object, beta_cos_ta = "cos_ta_", ...)
```

Arguments

`object`	`[fit_clogit]` A fitted iSSF model.
`beta_sl`	`[character]` The name of the coefficient of the step length.
`beta_log_sl`	`[character]` The name of the coefficient of the log of the step length.
`beta_sl_sq`	`[character]` The name of the coefficient of the square of the step length.
`beta_log_sl_sq`	`[character]` The name of the coefficient of the square of log of the step length.
`...`	Further arguments, none implemented.
`beta_cos_ta`	`[character]` The name of the coefficient of cosine of the turning angle.

Value

An `amt_distr` object, which consists of a list with the `name` of the distribution and its parameters (saved in `params`).

Author(s)

Brian J. Smith and Johannes Signer

References

Fieberg J, Signer J, Smith BJ, Avgar T (2020). "A "How-to" Guide for Interpreting Parameters in Resource-and Step-Selection Analyses." *bioRxiv*.

See Also

Wrapper to fit a distribution to data `fit_distr()`

Examples

```
# Fit an SSF, then update movement parameters.
data(deer)
mini_deer <- deer[1:100, ]
sh_forest <- get_sh_forest()

# Prepare data for SSF
ssf_data <- mini_deer |>
  steps_by_burst() |>
  random_steps(n = 15) |>
  extract_covariates(sh_forest) |>
  mutate(forest = factor(forest, levels = 1:0,
                    labels = c("forest", "non-forest")),
  cos_ta_ = cos(ta_),
  log_sl_ = log(sl_))

# Check tentative distributions
# Step length
sl_distr_params(ssf_data)
attr(ssf_data, "sl_")
#    Turning angle
ta_distr_params(ssf_data)

# Fit an iSSF
m1 <- ssf_data |>
  fit_issf(case_ ~ forest +
               sl_ + log_sl_ + cos_ta_ +
               strata(step_id_))

# Update step length distribution
new_gamma <- update_sl_distr(m1)

# Update turning angle distribution
new_vm <- update_ta_distr(m1)

# It is also possible to use different step length distributions

# exponential step-length distribution
s2 <- mini_deer |> steps_by_burst()
s2 <- random_steps(s2, sl_distr = fit_distr(s2$sl_, "exp"))
m2 <- s2 |>
  fit_clogit(case_ ~ sl_ + strata(step_id_))
update_sl_distr(m2)

# half normal step-length distribution
s3 <- mini_deer |> steps_by_burst()
s3 <- random_steps(s3, sl_distr = fit_distr(s3$sl_, "hnorm"))
m3 <- s3 |>
  mutate(sl_sq_ = sl_^2) |>
  fit_clogit(case_ ~ sl_sq_ + strata(step_id_))
update_sl_distr(m3)
```

```
# log normal step-length distribution
s4 <- mini_deer |> steps_by_burst()
s4 <- random_steps(s4, sl_distr = fit_distr(s4$sl_, "lnorm"))
m4 <- s4 |>
  mutate(log_sl_ = log(sl_), log_sl_sq_ = log(sl_)^2) |>
  fit_clogit(case_ ~ log_sl_ + log_sl_sq_ + strata(step_id_))
update_sl_distr(m4)
```

Index