# Package 'amt'

April 23, 2020

**Type** Package

**Title** Animal Movement Tools

**Version** 0.1.0

**Description**
Manage and analyze animal movement data. The functionality of 'amt' includes methods to calculate track statistics (e.g. step lengths, speed, or turning angles), prepare data for fitting habitat selection analyses (resource selection functions and step-selection functions <doi:10.1890/04-0953> and integrated step-selection functions <doi:10.1111/2041-210X.12528>), and simulation of space-use from fitted step-selection functions <doi:10.1002/ecs2.1771>.

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

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

**Depends** R (>= 3.5),

**Imports** broom, checkmate, circular, ctmm, dplyr (>= 0.7.0), fitdistrplus, FNN, geosphere, glue, graphics, grDevices, KernSmooth, lazyeval, leaflet, lubridate, magrittr, maptools, methods, purrr, raster, Rcpp (>= 0.12.7), rgeos, rlang, sf, sp, survival, stats, tibble, tidyr (>= 1.0.0), utils

**Suggests** adehabitatLT, ggplot2, bcpa, devtools, moveHMM, move, sessioninfo, spacetime, trajectories, knitr, Rdpack, rmarkdown, tinytest

**LinkingTo** Rcpp

**RoxygenNote** 7.1.0

**RdMacros** Rdpack

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Johannes Signer [aut, cre],
Bjoern Reineking [ctb],
Brian Smith [ctb],
Ulrike Schlaegel [ctb],
Scott LaPoint [dtc]

**Maintainer** Johannes Signer <jsigner@gwdg.de>

**Repository** CRAN

**Date/Publication** 2020-04-23 10:00:02 UTC

# R topics documented:

---

`amt-package` *amt: Animal Movement Tools*

---

## Description

Manage and analyze animal movement data. The functionality of 'amt' includes methods to calculate track statistics (e.g. step lengths, speed, or turning angles), prepare data for fitting habitat selection analyses (resource selection functions and step-selection functions <doi:10.1890/04-0953> and integrated step-selection functions <doi:10.1111/2041-210X.12528>), and simulation of space-use from fitted step-selection functions <doi:10.1002/ecs2.1771>.

## Author(s)

**Maintainer**: Johannes Signer `<jsigner@gwdg.de>`

Other contributors:

- Bjoern Reineking [contributor]
- Brian Smith [contributor]
- Ulrike Schlaegel [contributor]
- Scott LaPoint [data contributor]

## See Also

Useful links:

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

## amt_fisher *GPS tracks from four fishers*

### Description

This file includes spatial data from 4 fisher (Martes 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 usage is permitted for exploratory purposes. For other purposes please get in contact (Scott LaPoint).

### Usage

```
amt_fisher
```

### Format

A `tibble` with 32400 rows and 5 variables:

**id** id of the animal

**x_** the x-coordinate

**y_** the y-coordinate

**t_** the timestamp

**burst_** bursts with 10 min sampling rates

### Source

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

### References

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

## amt_fisher_lu *Landuse for fisher data*

### Description

A Gauss-Random-Field simulation of a landscape.

### Usage

```
amt_fisher_lu
```

### Format

A `RasterLayer`

append_x1 *Append "_x1"*

## Description

Helper function to append "_x1" to variable names

## Usage

```
append_x1(string)
```

## Arguments

| | |
|---|---|
| string | [character] Variable name to possibly append to |

## Details

The function first checks if "_x1" is already appended and adds it if it is not. This is meant for internal use in \link{plot.log_rss}().

---

as_track *Coerce to track*

## Description

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

## Usage

```
as_track(x, ...)

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

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

## Arguments

| | |
|---|---|
| x | [SpatialPoints]<br>Object to be converted to a track. |
| ... | Further arguments, none implemented. |

## Examples

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

## 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"}<br>Should all distributions be returned, or only distributions for turn angles (ta) or step lengths (sl). |
| names_only | [logical(1)=FALSE]<br>Indicates if only the names of distributions should be returned. |
| ... | none implemented. |

## 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, sf = FALSE, ...)

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

### Arguments

| | |
|---|---|
| x | [track_xy,track_xyt]<br>A track created with make_track. |
| ... | Further arguments, none implemented. |
| spatial | [logical(1)=FALSE]<br>Whether or not to return a SpatialPolygons-object or not. |

| | |
|---|---|
| buffer | [numeric(0)=NULL]{NULL, >0}<br>An optional buffer of the bounding box. |
| sf | [logical(1)=FALSE]<br>If TRUE a simple feature polygon is returned. |

## 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)
```

---

# 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]<br>A track created with make_track. |
| ... | Further arguments, none implemented. |
| spatial | [logical(1)=FALSE]<br>Whether or not to return a SpatialPoints-object. |

## Examples

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

coercion *Coerce a track to other formats.*

## Description

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

## Usage

```
as_sp(x, ...)

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

as_move(x, ...)

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

as_ltraj(x, ...)

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

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

as_bcpa(x, ...)

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

as_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]<br>A track created with make_track. |

| | |
|---|---|
| ... | Further arguments, none implemented. |
| end | [logical(1)=TRUE]<br>For steps, should the end or start points be used? |
| id | [numeric,character,factor]<br>Animal id(s). |

## Examples

```
data(deer)
as_move(deer)
as_move(deer, id = "foo")
data(deer)
as_ltraj(deer)
as_ltraj(deer, id = "animal_3")
data(deer)
d <- as_bcpa(deer)
data(deer)
as_telemetry(deer)
# Fit HMM with two states
data(deer)
dm <- as_moveHMM(deer)
```

---

# convert_angles *Converts angles to radians*

---

## Description

Converts angles to radians

## Usage

```
as_rad(x)

as_degree(x)
```

## Arguments

| | |
|---|---|
| x | [numeric]<br>Angles in degrees or rad. |

## Examples

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

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

## crs *Coordinate References System (CRS)*

### Description

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

### Usage

```
get_crs(x, ...)

has_crs(x, ...)
```

### Arguments

| | |
|---|---|
| `x` | `[any]`<br>Object to check. |
| `...` | Further arguments, none implemented. |

### Examples

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

## cum_ud *Calculate a cumulative UD*

### Description

Calculate the cumulative utilization distribution (UD).

### Usage

```
cumulative_ud(x, ...)

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

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

### Arguments

| | |
|---|---|
| x | [RasterLayer]<br>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.

`dispersal_kernel` *Create a dispersal kernel*

## Description

Create a dispersal kernel

## Usage

```
dispersal_kernel(
  formula,
  coefs,
  habitat = NULL,
  other.vars = NULL,
  start,
  max.dist,
  init.dir = amt::as_rad(45),
  standardize = TRUE,
  raster = TRUE,
  stop = 0
)
```

## Arguments

| | |
|---|---|
| `formula` | [formula]<br>The formula for the dispersal kernel. |
| `coefs` | [named numeric]{>1}<br>Coefficients for the terms in the formula. Names of the coefficients must match the name of the terms. |
| `habitat` | [RasterLayer]<br>The habitat matrix / landscape. |
| `other.vars` | [data.frame = NULL]<br>Possible other covariates. |
| `start` | [numeric(2)]<br>Coordinates of the start position. |
| `max.dist` | [numeric(1)]<br>The maximum distance of the dispersal kernel. |
| `init.dir` | [numeric(1)]<br>The initial direction in rad. |
| `standardize` | [logical(1) = TRUE]<br>Should the result be standardized. |
| `raster` | [logical(1) = TRUE]<br>Should a `RasterLayer` be returned. |
| `stop` | [integer(1)=1]{0,1}<br>What happens when the animal steps out of the landscape. |

distributions *Functions to work with distributions as objects*

## Description

make_distributions creates a distribution.

## Usage

```
make_distribution(name, params, ...)

make_exp_distr(rate = 1)

make_unif_distr(min = -pi, max = pi)

make_vonmises_distr(kappa = 1)

make_gamma_distr(shape = 1, scale = 1)

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

## S3 method for class 'vonmises_distr'
random_numbers(x, n = 100, ...)

## S3 method for class 'amt_distr'
random_numbers(x, n = 100, ...)
```

## Arguments

| | |
|---|---|
| name | [char(1)]<br>Short name of distribution. See available_distr() for all currently implemented distributions. |
| params | [list]<br>A named list with parameters of the distribution. |
| ... | none implemented. |
| rate | [double(1)>0]<br>The rate of the exponential distribution. |
| min | [double(1)]<br>The minimum of the uniform distribution. |
| max | [double(1)]<br>The minimum of the uniform distribution. |
| kappa | [double(1)>=0]<br>Concentration parameter of the von Mises distribution. |
| shape, scale | [double(1)>=0]<br>Shape and scale of the Gamma distribution |

| | |
|---|---|
| x | [amt_distr]<br>A distribution object. |
| n | [integer(1)=100]{>0}<br>The number of random draws. |

---

`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. |

---

`dist_cent` *Distance to center*

---

## Description

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

## Usage

```
distance_to_center(x, ...)

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

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

distance_to_centers(x, ...)

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

## Arguments

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

## Value

RasterLayer

## Examples

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

---

# extract_covariates *Extract covariate values*

---

## Description

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

## Usage

```
extract_covariates(x, ...)

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

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

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

extract_covariates_along(x, ...)

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

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]<br>Either a track created with mk_track or track, or steps. |
| ... | Further arguments, none implemented. |
| covariates | [RasterLayer,RasterStack,RasterBrick]<br>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"}<br>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"}<br>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)]<br>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"]<br>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.

## Examples

```
data(deer)
data(sh_forest)
deer %>% extract_covariates(sh_forest)
deer %>% steps %>% extract_covariates(sh_forest)
deer %>% steps %>% extract_covariates(sh_forest, where = "start")
## Not run:
data(deer) # relocation
data("sh_forest") # env covar

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

## End(Not run)

# Simulate some dummy data
# Hourly data for 10 days: 24 * 10
set.seed(123)
path <- data.frame(x = cumsum(rnorm(240)),
              y = cumsum(rnorm(240)),
              t = lubridate::ymd("2018-01-01") + hours(0:239))
trk <- make_track(path, x, y, t)

# dummy env data
rs <- raster::raster(xmn = -50, xmx = 50, ymn = -50, ymx = 50, res = 1)

# create dummy covars for each day
rs <- raster::stack(lapply(1:10, function(i)
  raster::setValues(rs, runif(1e4, i - 1, i))))
```

```
# Env covariates are always taken at noon
rs <- raster::setZ(rs, lubridate::ymd_hm("2018-01-01 12:00") + days(0:9))

# Allow up to 2 hours after
trk %>% extract_covariates_var_time(rs, max_time = hours(2), when = "after") %>%
  print(n = 25)
trk %>% extract_covariates_var_time(rs, max_time = hours(2), when = "before") %>%
  print(n = 25)
trk %>% extract_covariates_var_time(rs, max_time = hours(2), when = "any") %>%
  print(n = 25)

# We can use different time scales
trk %>%
  extract_covariates_var_time(
    rs, max_time = hours(2), when = "any", name_covar = "env_2h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(4), when = "any", name_covar = "env_4h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(6), when = "any", name_covar = "env_6h") %>%
  print(n = 25)

# We can use different time scales: after
trk %>%
  extract_covariates_var_time(
    rs, max_time = hours(2), when = "after", name_covar = "env_2h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(4), when = "after", name_covar = "env_4h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(6), when = "after", name_covar = "env_6h") %>%
  print(n = 25)

# We can use different time scales: before
trk %>%
  extract_covariates_var_time(
    rs, max_time = hours(2), when = "before", name_covar = "env_2h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(4), when = "before", name_covar = "env_4h") %>%
  extract_covariates_var_time(
    rs, max_time = hours(6), when = "before", name_covar = "env_6h") %>%
  print(n = 25)

# The same works also for steps
trk %>%
  steps() %>%
  extract_covariates_var_time(
    rs, max_time = hours(2), when = "before", name_covar = "env_2h") %>%
  print(n = 25)

# also with start and end
trk %>%
  steps() %>%
  extract_covariates_var_time(
    rs, max_time = hours(2), when = "before", name_covar = "env_2h",
```

```
    where = "both") %>%
  print(n = 25)
```

---

`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]`<br>A track created with `make_track`. |
| `...` | Further arguments, none implemented. |
| `min_n` | [numeric(1)=3]<br>Indicating the minimum number of relocations (=fixes per burst). |

---

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

---

## fit_ctmm *Fit a continuous time movement model with* ctmm

---

### Description

Fit a continuous time movement model with ctmm

### Usage

```
fit_ctmm(x, model, ...)
```

### Arguments

| | |
|---|---|
| x | [track_xyt]<br>A track created with make_track that includes time. |
| model | [character(1)="bm"]{"iid", "bm","ou","ouf", "auto"}<br>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. |
| ... | Additional parameters passed to ctmm::ctmm.fit or ctmm::ctmm.select for model = "auto" |

### Value

A ctmm object.

### 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)
m1 <- fit_ctmm(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)]<br>The observed data. |
| `dist_name` | [character(1)]{"exp", "gamma", "unif", "vonmises"}<br>The name of the distribution. |
| `na.rm` | [logical(1)=TRUE]<br>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 piped workflows.

### Usage

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

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

### Arguments

| | |
|---|---|
| `data` | [data.frame]<br>The data used to fit a model. |
| `formula` | [formula]<br>The model formula. |
| `...` | Further arguments passed to `stats::glm`. |

## 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]<br>A track created with make_track. |
| ... | Further arguments, none implemented. |

### Examples

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

---

## habitat_kernel *Simulate UD from fitted SSF*

---

### Description

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

### Usage

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

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

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

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

### Arguments

| | |
|---|---|
| coef | [list]<br>Vector with coeffiecients, not yet implemented. |
| resources | [RasterLayer, RasterStack]<br>The resources. |
| exp | A logical scalar, indicating whether or not the resulting habitat kernel should be exponentiated. This is usually TRUE. |
| scale, shape | [numeric](1)<br>Scale and scale parameter of the gamma distribution of step lengths. |
| template | [RasterLayer,RasterStack]<br>A raster serving as template for the simulations. |
| quant | A numeric scalar, quantile of the step-length distribution that is the maximum movement distance. |
| movement_kernel | [RasterLayer]<br>The movement kernel. |

| | |
|---|---|
| habitat_kernel | [RasterLayer]<br>The habitat kernel. |
| start | [numeric(2)]<br>Starting point of the simulation. |
| n | [integer(1)=1e5]<br>The number of simulation steps. |
| n_rep | [integer(1)=5e3]{>0}<br>The number of times the animal walks of the final position. The mean of all replicates is returned. |

## Details

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

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

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

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

## Value

The habitat kernel, as `RasterLayer`.

## Note

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

## Author(s)

Johannes Signer (jmsigner@gmail.com)

## References

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

helper *Coordinates of a track.*

## Description

Coordinates of a track.

## Usage

```
coords(x, ...)

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

make_trast(x, ...)

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

extent_x(x, ...)

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

extent_y(x, ...)

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

extent_both(x, ...)

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

extent_max(x, ...)

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

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

## Value

[tibble]
The coordinates.

## Examples

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

---

# hr_akde *Home ranges*

---

## Description

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

## Usage

```
hr_akde(x, ...)

## S3 method for class 'track_xyt'
hr_akde(
  x,
  model = fit_ctmm(x, "iid"),
  trast = make_trast(x),
  levels = 0.95,
```

```
  ...
)

hr_area(x, ...)

hr_isopleths(x, ...)

hr_kde(x, ...)

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

hr_kde_ref(x, ...)

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

hr_kde_pi(x, ...)

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

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 = raster(as_sp(x), nrow = 100, ncol = 100)
)

hr_locoh(x, ...)

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

hr_mcp(x, ...)

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

## Arguments

| | |
|---|---|
| x | [track_xy,track_xyt]<br>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. |

| | |
|---|---|
| trast | [RasterLayer]<br>A template raster for kernel density home-ranges. |
| levels | [numeric]<br>The isopleth levels used for calculating home ranges. Should be 0 < level < 1. |
| h | [numeric(2)]<br>The bandwidth for kernel density estimation. |
| rescale | [character(1)]<br>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. |
| range | numeric vector with different candidate h values. |
| which_min | A character indicating if the global or local minimum should be searched for. |
| n | [integer(1)]<br>The number of neighbors used when calculating local convex hulls. |
| type | k, r or a. Type of LoCoH. |
| rand_buffer | [numeric(1)]<br>Random buffer to avoid polygons with area 0 (if coordinates are numerically identical). |

## Details

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

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

The bandwidth, the standardization method and correction.

vector of length two

## References

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).

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

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.

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.

## See Also

KernSmooth::dpik

## Examples

```
# akde
## Not run:
data(deer)
ud1 <- hr_akde(deer) # uses an iid ctmm
ud2 <- hr_akde(deer, model = fit_ctmm(deer, "ou")) # uses an OU ctmm

## End(Not run)
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)

# Local Convex Hull (LoCoH) -----------------------------------------------
locoh1 <- hr_locoh(mini_deer)
hr_area(locoh1)

# calculated Locoh at different levels
locoh <- hr_locoh(mini_deer, levels = seq(0.3, 1, 0.1))
hr_area(locoh)


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

---

# 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 *Different Methods to calculate home-range overlaps*

### Description

Different Methods to calculate home-range overlaps

### Usage

```
hr_overlap(x, ...)

## S3 method for class 'hr'
hr_overlap(x, y, ...)

hr_ba(x, ...)
```

### Arguments

| | |
|---|---|
| x, y | hr A home-range estimate |
| ... | Further arguments, none implemented. |

### Value

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

## hr_to_sf *Convert*

### Description

Convert

### 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 tible. |
| 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)


## Not run:
ggplot(hr) + geom_sf()


## End(Not run)
```

---

# 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]<br>A track created with make_track. |
| ... | Further arguments, none implemented. |
| popup | [character(nrow(x))]<br>Optional labels for popups. |
| cluster | [logical(1)]<br>If TRUE points are clustered at lower zoom levels. |

## Value

An interactive `leaflet` map.

## Note

Important, `x` requires a valid coordinate reference system.

## See Also

`leaflet::leaflet()`

## Examples

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

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

## End(Not run)
```

---

# 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 'fit_logit'
log_rss(object, x1, x2, ...)

## S3 method for class 'fit_clogit'
log_rss(object, x1, x2, ...)
```

## Arguments

| | |
|---|---|
| `object` | [fit_logit, fit_clogit]<br>A fitted RSF or (i)SSF model. |
| `...` | Further arguments, none implemented. |
| `x1` | [data.frame]<br>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().

## 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.

## Author(s)

Brian Smith

## References

Avgar, T., Lele, S.R., Keim, J.L., & Boyce, M.S.. (2017). Relative Selection Strength: Quantifying effect size in habitat- and step-selection inference. *Ecology and Evolution*, 7, 5322–5330.

## See Also

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

Default plotting method available: \link{plot.log_rss}()

## Examples

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

# Load packages
library(ggplot2)

# Load data
data("amt_fisher")
data("amt_fisher_lu")

# Prepare data for RSF
rsf_data <- amt_fisher %>%
  filter(burst_ == 1) %>%
  make_track(x_, y_, t_) %>%
  random_points() %>%
  extract_covariates(amt_fisher_lu) %>%
  mutate(lu = factor(landuse_study_area))

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

```
  fit_rsf(case_ ~ lu)

# Calculate log-RSS
# data.frame of x1s
x1 <- data.frame(lu = sort(unique(rsf_data$lu)))
# data.frame of x2 (note factor levels should be same as model data)
x2 <- data.frame(lu = factor(21, levels = levels(rsf_data$lu)))
# Calculate
logRSS <- log_rss(object = m1, x1 = x1, x2 = x2)

# Plot
ggplot(logRSS$df, aes(x = lu_x1, y = log_rss)) +
  geom_hline(yintercept = 0, linetype = "dashed", color = "gray") +
  geom_point() +
  xlab(expression("Land Use Category " * (x[1]))) +
  ylab("log-RSS") +
  ggtitle(expression("log-RSS" * (x[1] * ", " * x[2]))) +
  theme_bw()

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

 #Prepare data for SSF
ssf_data <- deer %>%
  steps_by_burst() %>%
  random_steps(n = 15) %>%
  extract_covariates(sh_forest) %>%
  mutate(forest = factor(sh.forest, levels = 1:2,
                    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)

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

`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]`<br>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).

## 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)
```

---

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]<br>A track created with make_track that includes time. |
| ... | Further arguments, none implemented. |
| trast | [RasterLayer]<br>A template raster for the extent and resolution of the result. |

| | |
|---|---|
| model | [An output of fit_ctmm]<br>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]<br>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]<br>Number of temporal grid points per median timestep. |
| n.points | [numeric(1)=5]<br>This argument is only relevant for rolling_od and specifies the window size for the od estimation. |
| show.progress | [logical(1)=TRUE]<br>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

```
## Not run:
data(deer)
mini_deer <- deer[1:100, ]
trast <- make_trast(mini_deer)
md <- od(mini_deer, trast = trast)
raster::plot(md)

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

## End(Not run)
```

---

# 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]<br>A (fitted) distribution |
| ... | None |

---

`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 \link{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 \link{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 \link{palette}() before calling the function.

## Examples

```
# Load data
data("amt_fisher")
data("amt_fisher_lu")

# Prepare data for RSF
rsf_data <- amt_fisher %>%
  filter(burst_ == 1) %>%
  make_track(x_, y_, t_) %>%
  random_points() %>%
  extract_covariates(amt_fisher_lu) %>%
  mutate(lu = factor(landuse_study_area))

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

# Calculate log-RSS
# data.frame of x1s
x1 <- data.frame(lu = sort(unique(rsf_data$lu)))
# data.frame of x2 (note factor levels should be same as model data)
x2 <- data.frame(lu = factor(21, levels = levels(rsf_data$lu)))
# Calculate
logRSS <- log_rss(object = m1, x1 = x1, x2 = x2)

# Plot
plot(logRSS)
```

---

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

## Examples

```
data(deer)

# with random steps
deer %>% steps_by_burst %>% random_steps %>% plot_sl
deer %>% steps_by_burst %>% random_steps %>% plot_sl(upper_quantile = 0.5)

# with fitted ssf
deer %>% steps_by_burst %>% random_steps %>%
  fit_ssf(case_ ~ sl_ + strata(step_id_)) %>% plot_sl
```

---

# 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 'SpatialPolygons'
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]<br>A track created with make_track. |
| ... | [any]<br>None implemented. |
| n | [integer(1)]<br>The number of random points. |
| type | [character(1)]<br>Argument passed to sp::spsample type. The default is random. |
| presence | [track]<br>The presence points, that will be added to the result. |
| level | [numeric(1)]<br>Home-range level of the minimum convex polygon, used for generating the background samples. |
| hr | [character(1)]<br>The home range estimator to be used. Currently only MCP is implemented. |

## Note

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

## Examples

```
data(deer)

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

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

```
## End(Not run)

 ## Not run:
trast <- raster(bbox(deer, buffer = 5000), res = 30)
rp3 <- random_points(deer, hr = "kde", trast = trast) # we need a larger template raster

plot(rp3)

## End(Not run)


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

## End(Not run)

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

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

## End(Not run)

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

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

## End(Not run)
```

---

`random_steps` *Generate Random Steps*

---

## Description

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

## Usage

```
random_steps(x, ...)

## S3 method for class '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,
  ...
)
```

## Arguments

| | |
|---|---|
| x | Steps. |
| ... | Further arguments, none implemented. |
| n_control | [integer(1)=10]{>1}<br>The number of control steps paired with each observed step. |
| angle | [numeric(1) = 0]{-pi < rel_angle < pi}<br>Angle for the first step. |
| rand_sl | [numeric]<br>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]<br>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]<br>The step-length distribution. |
| ta_distr | [amt_distr]<br>The turn-angle distribution. |

include_observed
: [logical(1) = TRUE]
Indicates if observed steps are to be included in the result.

---

`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. |

## Examples

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

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

## sh *Relocations of 1 red deer*

### Description

1500 GPS relocations of one red deer from northern Germany.

### Usage

```
sh
```

### Format

A data frame with 1500 rows and 4 variables:

**x_epsg31467** the x-coordinate

**y_epsg31467** the y-coordinate

**day** the day of the relocation

**time** the hour of the relocation

### Source

Verein für Wildtierforschung Dresden und Göttingen e.V.

## sh_forest *Forest cover*

### Description

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

### Usage

```
sh_forest
```

### Format

A `RasterLAyer`

**1** forest

**2** non-forest

### Source

JRC

## References

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

---

`simulate_ud_from_dk` *Simulate a UD from a dispersal kernel*

---

## Description

Simulate a UD from a dispersal kernel

## Usage

```
simulate_ud_from_dk(obj, n = 1000, other.vars = NULL)
```

## Arguments

| | |
|---|---|
| `obj` | A dispersal kernel |
| `n` | Number of time steps |
| `other.vars` | other covariates for each time step. |

---

`simulate_xy` *Simulate a trajectory*

---

## Description

Simulate a trajectory

## Usage

```
simulate_xy(obj, n = 100, other.vars = NULL)
```

## Arguments

| | |
|---|---|
| `obj` | A dispersal kernel. |
| `n` | Number of time steps. |
| `other.vars` | Other covariates (for each time step). |

## 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]<br>Should an NA be appended at the end. |

### Value

[numeric]
The speed in m/s.

## steps *Functions to create and work with steps*

### Description

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

### Usage

```
direction_abs(x, ...)

## S3 method for class 'track_xy'
direction_abs(
  x,
  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]<br>A track created with make_track. |
| ... | Further arguments, none implemented |
| full_circle | [logical(1)=FALSE]<br>If TRUE angles are returned between 0 and $2pi$, otherwise angles are between $-pi$ and $pi$. |
| zero_dir | [character(1)='E']<br>Indicating the zero direction. Must be either N, E, S, or W. |
| clockwise | [logical(1)=FALSE]<br>Should angles be calculated clock or anti-clockwise? |
| append_last | [logical(1)=TRUE]<br>If TRUE an NA is appended at the end of all angles. |
| lonlat | [logical(1)=TRUE]<br>Should geographical or planar coordinates be used? If TRUE geographic distances are calculated. |
| keep_cols | [character(1)=NULL]{'start', 'end', 'both'}<br>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

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

## Value

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

## Examples

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

# How do results compare to other packages
# adehabitatLT
df <- adehabitatLT::as.ltraj(data.frame(x = xy$x, y = xy$y), typeII = FALSE, id = 1)
df[[1]]$abs.angle
amt::direction_abs(trk)

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

# move
m <- move::move(xy$x, xy$y, lubridate::now() + lubridate::hours(1:10),
 proj = sp::CRS("+init=epsg:4326"))
move::angle(m)
direction_abs(trk, lonlat = TRUE, zero_dir = "E") %>% as_degree()

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

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

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

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

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

# moveHMM (only rel. ta)
df <- data.frame(ID = 1, x = xy$x, y = xy$y)
```

```
moveHMM::prepData(df, type = "UTM")

trk

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

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

---

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, ...)
```

## Arguments

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

| | |
|---|---|
| time_unit | A character. The time unit in which the sampling rate is calculated. Acceptable values are sec, min, hour, day, and auto. If auto (the default) is used, the optimal unit is guessed. |
| summarize | A logical. If TRUE a summary is returned, otherwise raw sampling intervals are returned. |
| as_tibble | A logical. Should result be returned as tibble or as table. |
| cols | [character(>= 1)]<br>Indicating columns to be used as grouping variables. |

## 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"))
```

---

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

---

## Description

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

## Usage

```
time_of_day(x, ...)

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

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

## Arguments

| | |
|---|---|
| x | [track_xyt,steps_xyt]<br>A track or steps. |
| ... | Further arguments, none implemented. |

`solar.dep` [numeric(1,n)=6]
The angle of the sun below the horizon in degrees. Passed to `maptools::crepuscule`.

`include.crepuscule`
[logical(1)=TRUE]
Should dawn and dusk be included.

`where` [character(1)="end"]{"start", "end", "both"} For `steps`, should the start, end or both time points be used?

## Examples

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

---

# track *Create a* track_*

---

## Description

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

## Usage

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

make_track(
  tbl,
  .x,
  .y,
  .t,
  ...,
  crs = NULL,
```

```
  order_by_ts = TRUE,
  check_duplicates = FALSE,
  all_cols = FALSE
)

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

## Arguments

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

## Value

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

---

# 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, nt, tol, ...)
```

## Arguments

| | |
|---|---|
| x | A track. |
| ... | Further arguments, none implemented. |
| nt | The new time trajectory. |
| tol | The tolerance. |

---

# track_methods *Track Methods*

---

## Description

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

## Usage

```
velocity(x, ...)

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

nsd(x, ...)

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

diff_x(x, ...)

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

diff_y(x, ...)

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

## Arguments

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

## track_resample *Resample track*

### Description

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

### Usage

```
track_resample(x, ...)

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

### Arguments

| | |
|---|---|
| x | A track_xyt. |
| ... | Further arguments, none implemented. |
| rate | A lubridate Period, that indicates the sampling rate. |
| tolerance | A lubridate Period, that indicates the tolerance of deviations of the sampling rate. |
| start | A integer scalar, that gives the relocation at which the sampling rate starts. |

## transform_coords *Transform CRS*

### Description

Transforms the CRS for a track.

### Usage

```
transform_coords(x, ...)

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

transform_crs(x, ...)
```

## Arguments

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

## See Also

sp::spTransform

## Examples

```
data(deer)
get_crs(deer)

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

# Index