Introduction

Johannes Signer

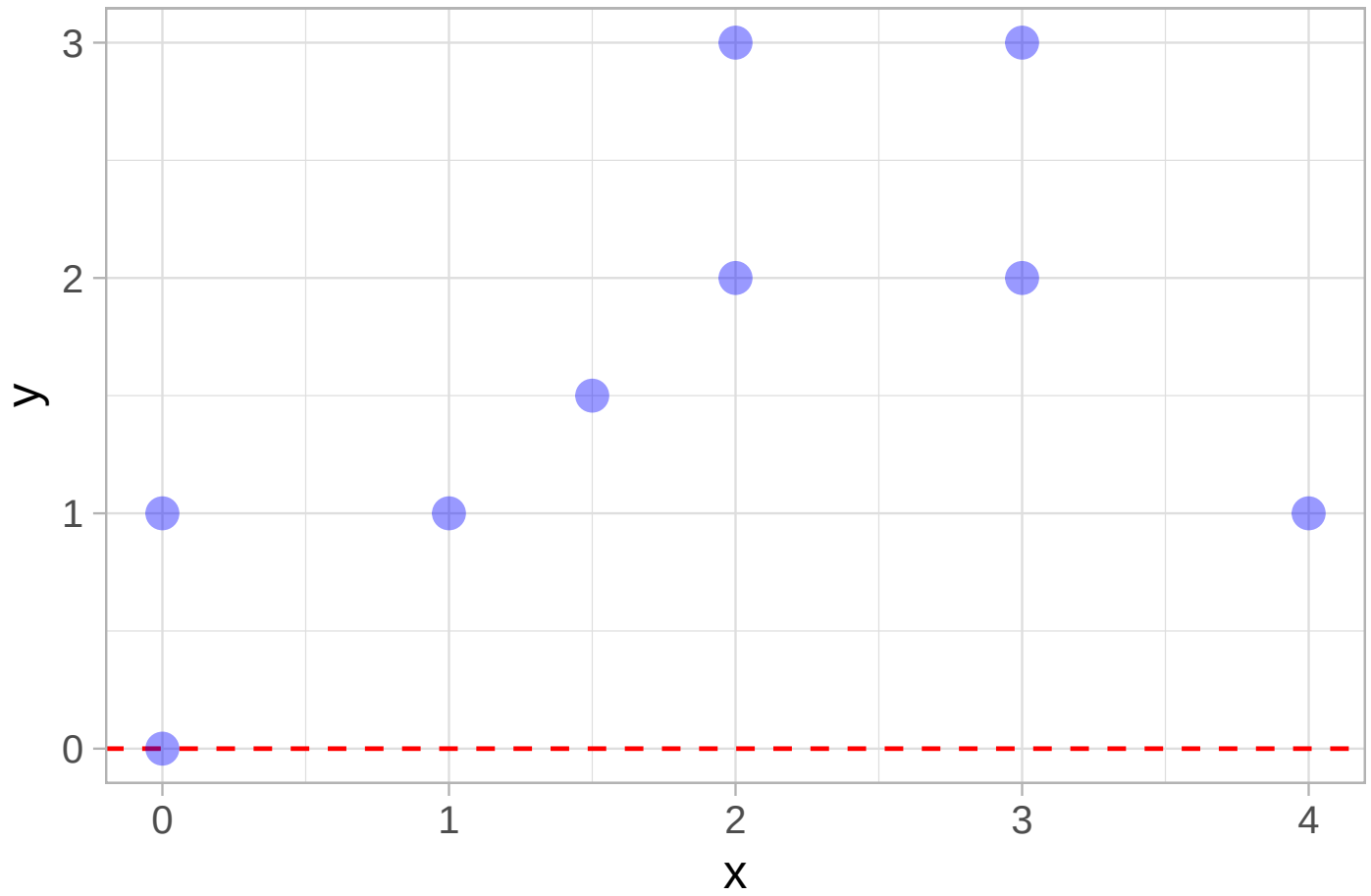
September 2023

Data

Movement data

- Often times the data we receive is just a time series of coordinates (longitude, latitude and time stamp).
- Depending on the sensors we use, other (meta) information may also be stored (this could include temperature, coordinates in a different [projected] CRS, . . .).

```
No;CollarID;UTC_Date;UTC_Time;LMT_Date;LMT_Time;Origin;SCTS_Date;SCTS_Time;ECEF_X [m];ECEF_Y [m];ECEF_Z
13563;9977;30.03.2020;10:01:01;30.03.2020;10:01:01;Collar;11.05.2021;09:15:07;4040415;851181;4845535;49,
13562;9977;30.03.2020;09:01:06;30.03.2020;09:01:06;Collar;11.05.2021;09:15:07;4040398;851190;4845551;49,
13561;9977;30.03.2020;08:02:12;30.03.2020;08:02:12;Collar;11.05.2021;09:15:07;4040451;851194;4845498;49,
13560;9977;30.03.2020;07:01:18;30.03.2020;07:01:18;Collar;11.05.2021;09:15:07;4040383;851193;4845554;49,
13559;9977;30.03.2020;06:01:56;30.03.2020;06:01:56;Collar;11.05.2021;09:15:07;4040403;851166;4845550;49,
13558;9977;30.03.2020;05:01:04;30.03.2020;05:01:04;Collar;11.05.2021;09:15:07;4040452;851209;4845535;49,
13557;9977;30.03.2020;04:00:31;30.03.2020;04:00:31;Collar;11.05.2021;09:15:07;4040388;851194;4845495;49,
13556;9977;30.03.2020;03:01:09;30.03.2020;03:01:09;Collar;11.05.2021;09:15:07;4040401;851191;4845489;49,
13555;9977;30.03.2020;02:01:04;30.03.2020;02:01:04;Collar;11.05.2021;09:15:07;4040394;851198;4845524;49,
13554;9977;30.03.2020;01:01:07;30.03.2020;01:01:07;Collar;11.05.2021;09:15:07;4040417;851185;4845543;49,
13553;9977;30.03.2020;00:01:07;30.03.2020;00:01:07;Collar;11.05.2021;09:15:07;4040472;851176;4845521;49,
```


3
2
1
0
y
0
1
2
3
4
x

Environmental covariates

- Vector layers (e.g., road networks, rivers, protected areas)
- Raster layers (e.g., land use, remotely sensed data such as NDVI, climatic variables)

Tracks and bursts: the basic building block

- For movement data, we usually read a text file into R (`.csv`, `.txt`, ...) as a data frame and then create a analysis-specific object.
- When working with the `amt` package the function `make_track()` takes a sequence of coordinates (with or without timestamps) and creates a track. Note, at this point multiple individuals can be mixed and sampling rates can be heterogeneous.

- **Bursts** can be created from tracks.
- A burst is a sequence of (re)locations from the **same** individual at **equal** time intervals (with some tolerance).
- Options to change from tracks to bursts are:
  1. Use the function `amt::track_resample()`.

Sampling rates, and resampling

- The function `summarize_sampling_rate()` takes an track as input and gives a summary of the sampling rate.
- If there are multiple animals present, there is also the function `summarize_sampling_rate_many()`, which will do the same thing, but for many animals.

- Once a suitable sampling rate is determined, the function `track_resample()` can be used to take a relocation every predefined time interval (e.g., 30 minutes, 2 hours, ...) within a tolerance.
- The result of `track_resample()` is again a track with one additional column called `burst_`.

Take-home messages

- Movement data are often 'just' text files.
- The `amt` package uses tracks as the basic building block.
- A burst is a track (or a part of a track) with a regular sampling rate.
- Use `track_resample()` create tracks with equal sampling rates.

If you have gaps and/or different sampling rates, interpolation with continuouse time movement models may be an option.

Methods in Ecology and Evolution | BRITISH ECOLOGICAL SOCIETY

APPLICATION | Open Access

aniMotum, an R package for animal movement data: Rapid quality control, behavioural estimation and simulation

Ian D. Jonsen, W. James Grecian, Lachlan Phillips, Gemma Carroll, Clive McMahon, Robert G. Harcourt, Mark A. Hindell, Toby A. Patterson

First published: 26 January 2023 | https://doi.org/10.1111/2041-210X.14060

Handling Editor Edward Codling

If you have gaps and/or different sampling rates, interpolation with continuouse time movement models may be an option.

Application | Free Access

ctmm: an R package for analyzing animal relocation data as a continuous-time stochastic process

Justin M. Calabrese, Chris H. Fleming, Eliezer Gurarie

First published: 22 March 2016 | https://doi.org/10.1111/2041-210X.12559 | Citations: 245

Movement characteristics (`sl_`, `ta_`)

Tracks are still *just* the points as they were collected. If we want to get insights, we have can look at different characteristics of steps (i.e., two consecutive relocations).
This include:

- step length
- turn angle
- speed
- Net squared displacement

Note, unless you take care of different instances or bursts, they are ignored.

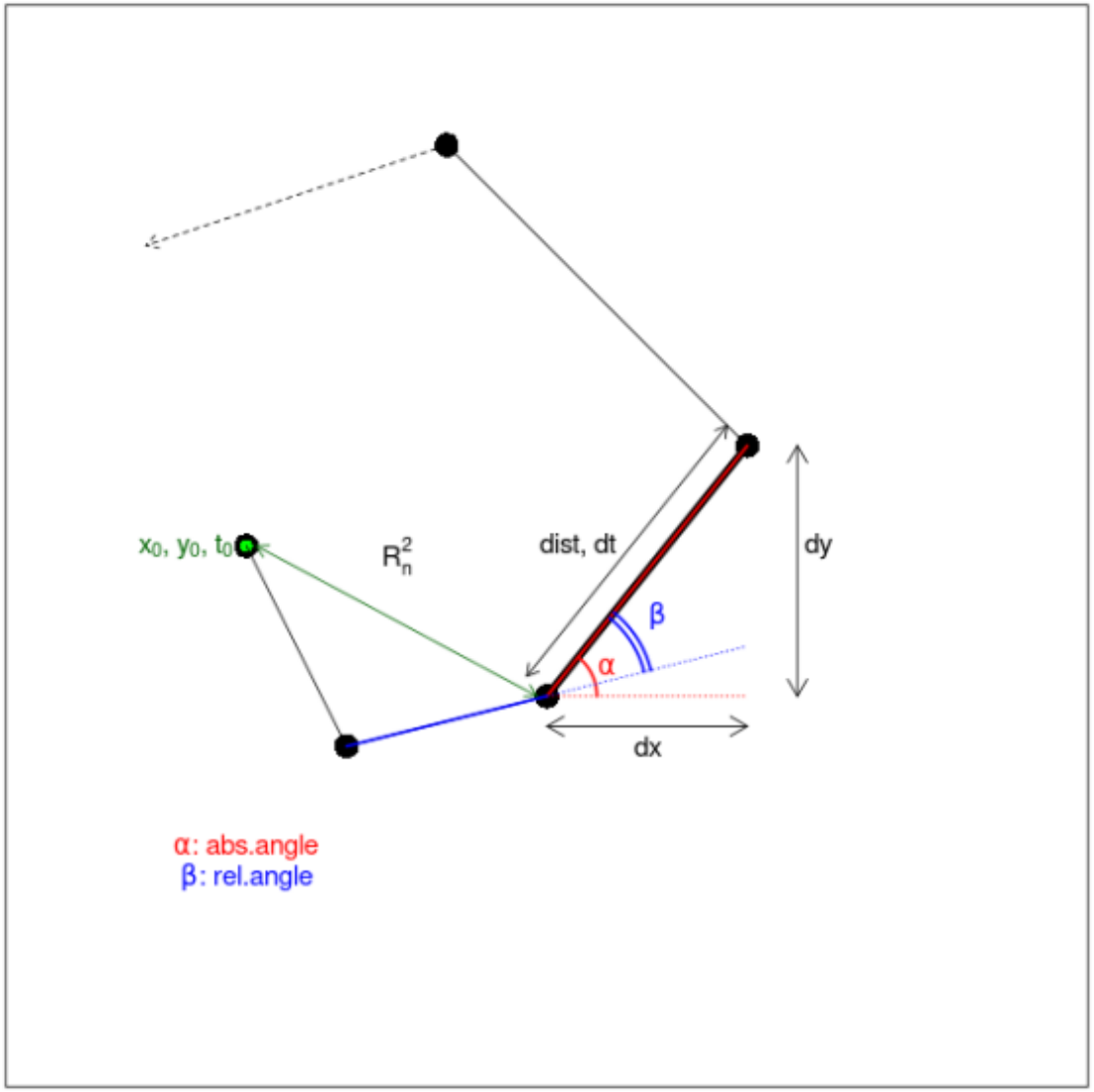


Figure 1: Movement metrics illustrated, from the adehabitatLT package vignette (Calenge 2006)

Net Squared displacement (NSD)

- The NSD is the squared distance between the first relocation of a track and the every relocation that follows.
- Bunnefeld et al. 2011 described different forms of the NSD that resemble different migratory behaviors.
- The different models can fit to the data (e.g., using nonlinear least square with the function `nls()` in R).

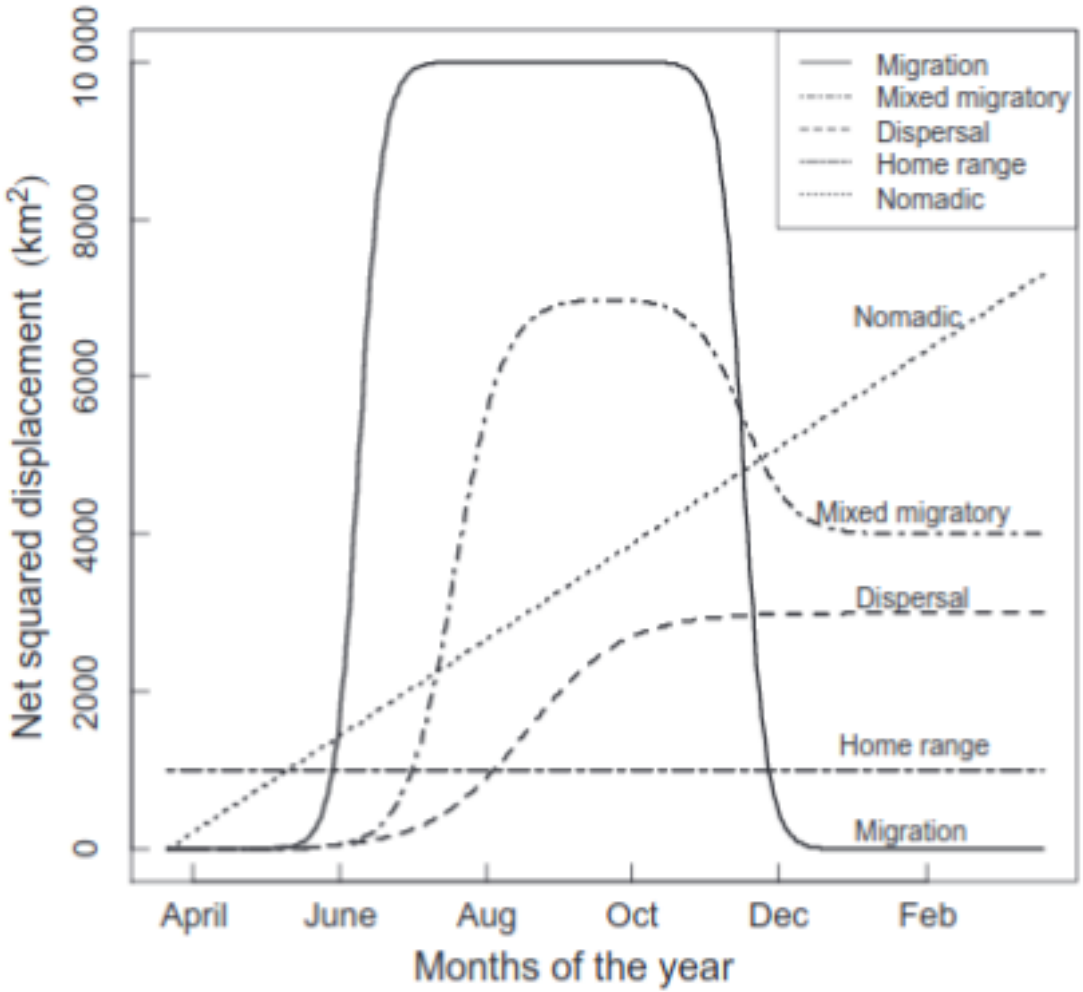


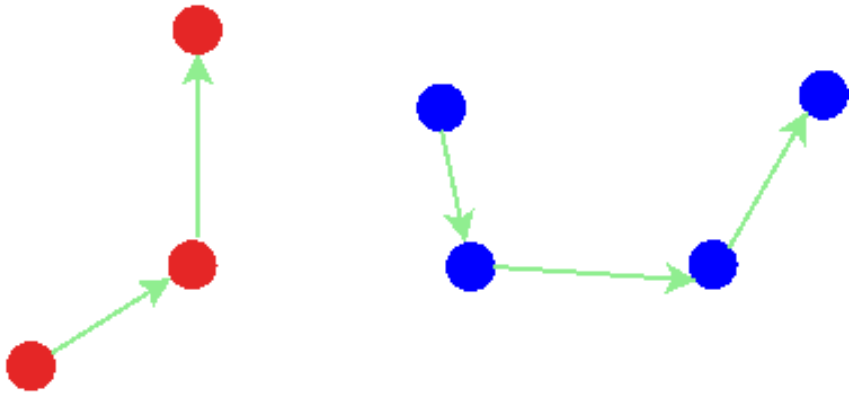
Figure 2: Figure taken from Bunnefeld et al. 2011

Time of the day

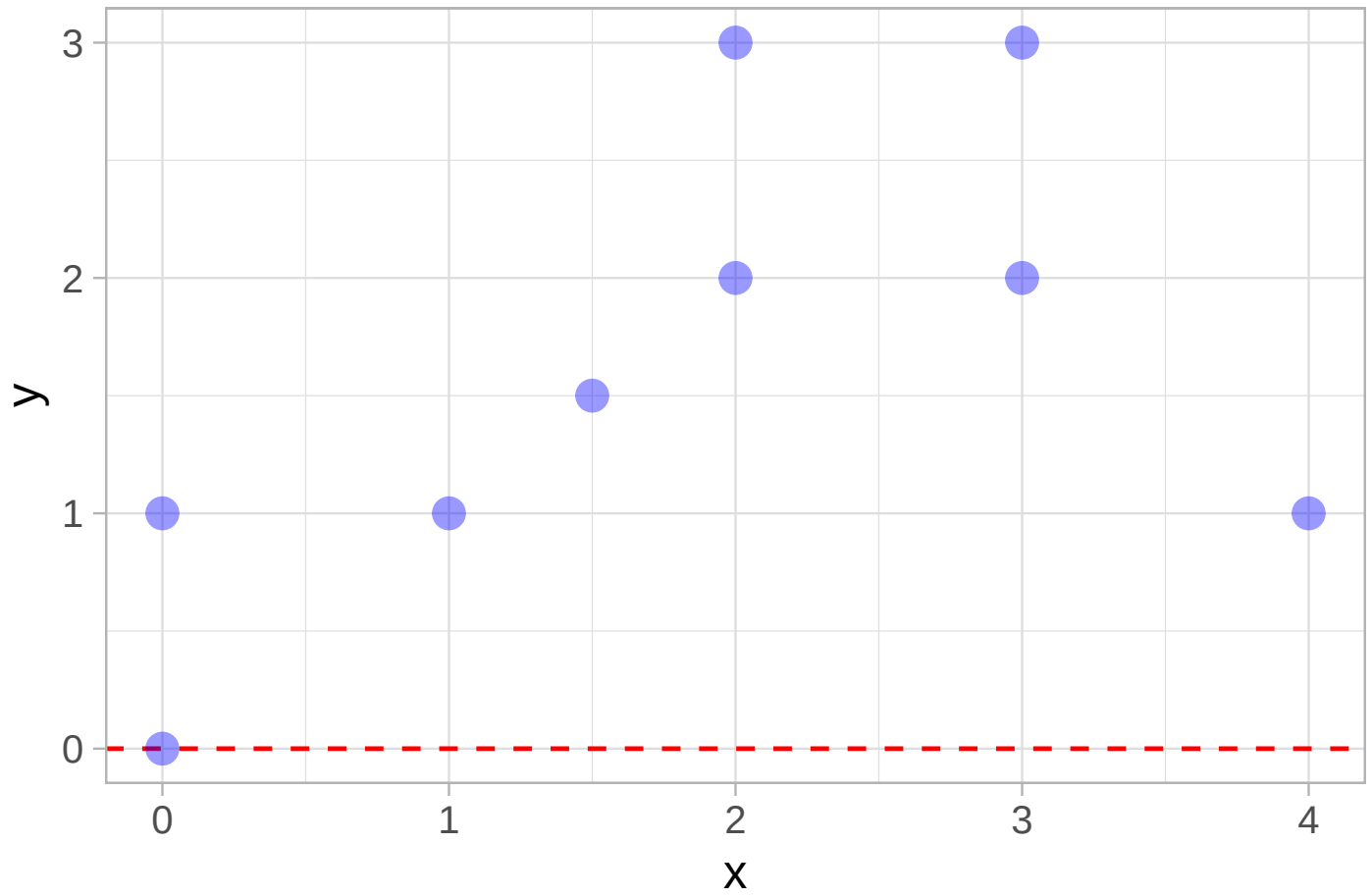
- Time of day can be annotated to steps with the function `amt::time_of_day()`. This will add an additional column to the data frame of steps `tod_end` or `tod_start` depending on the argument `when`.
- If the data is of sufficient temporal resolution, it is also possible to annotate twilight (dawn and dusk).

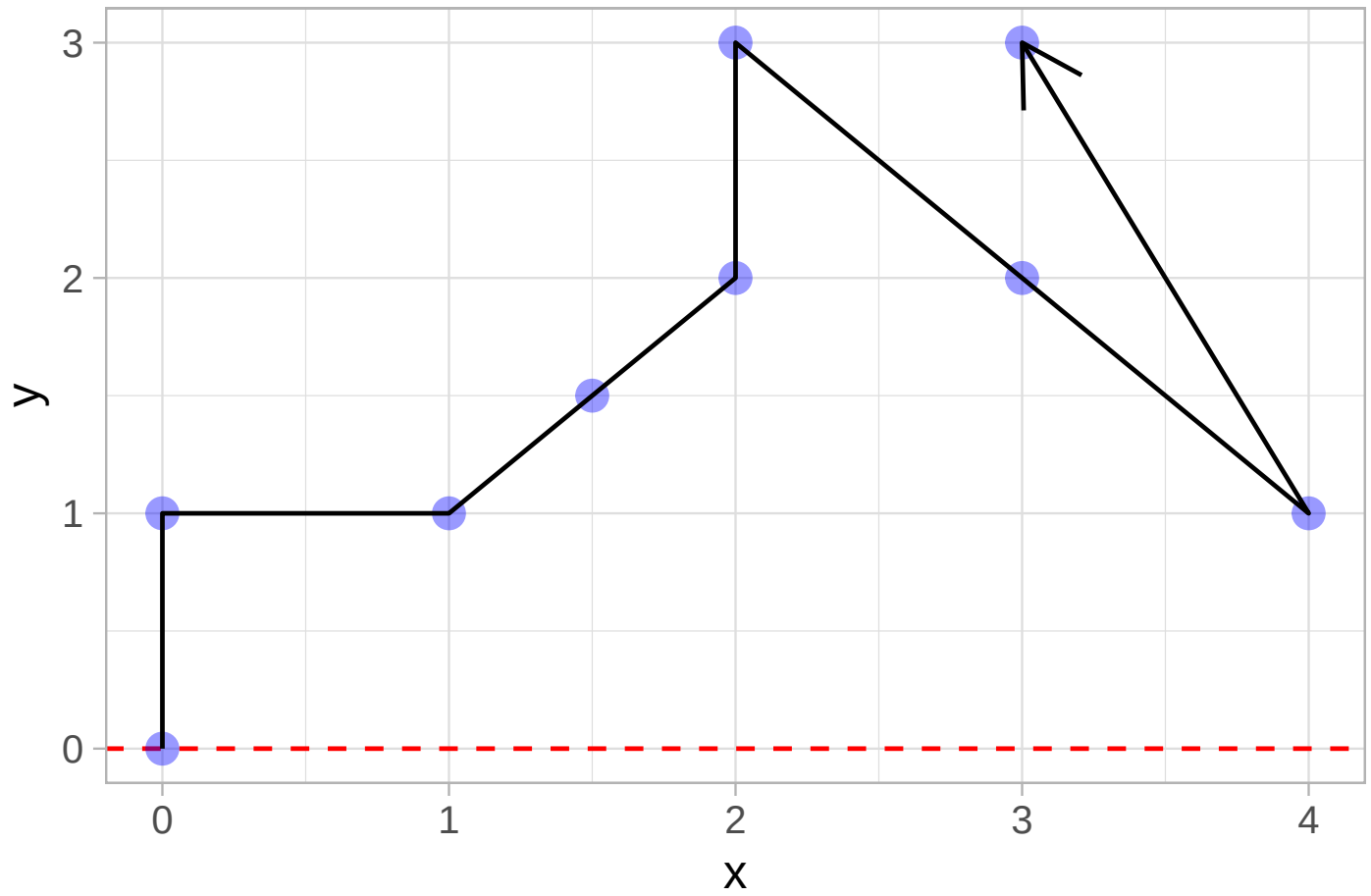
Steps

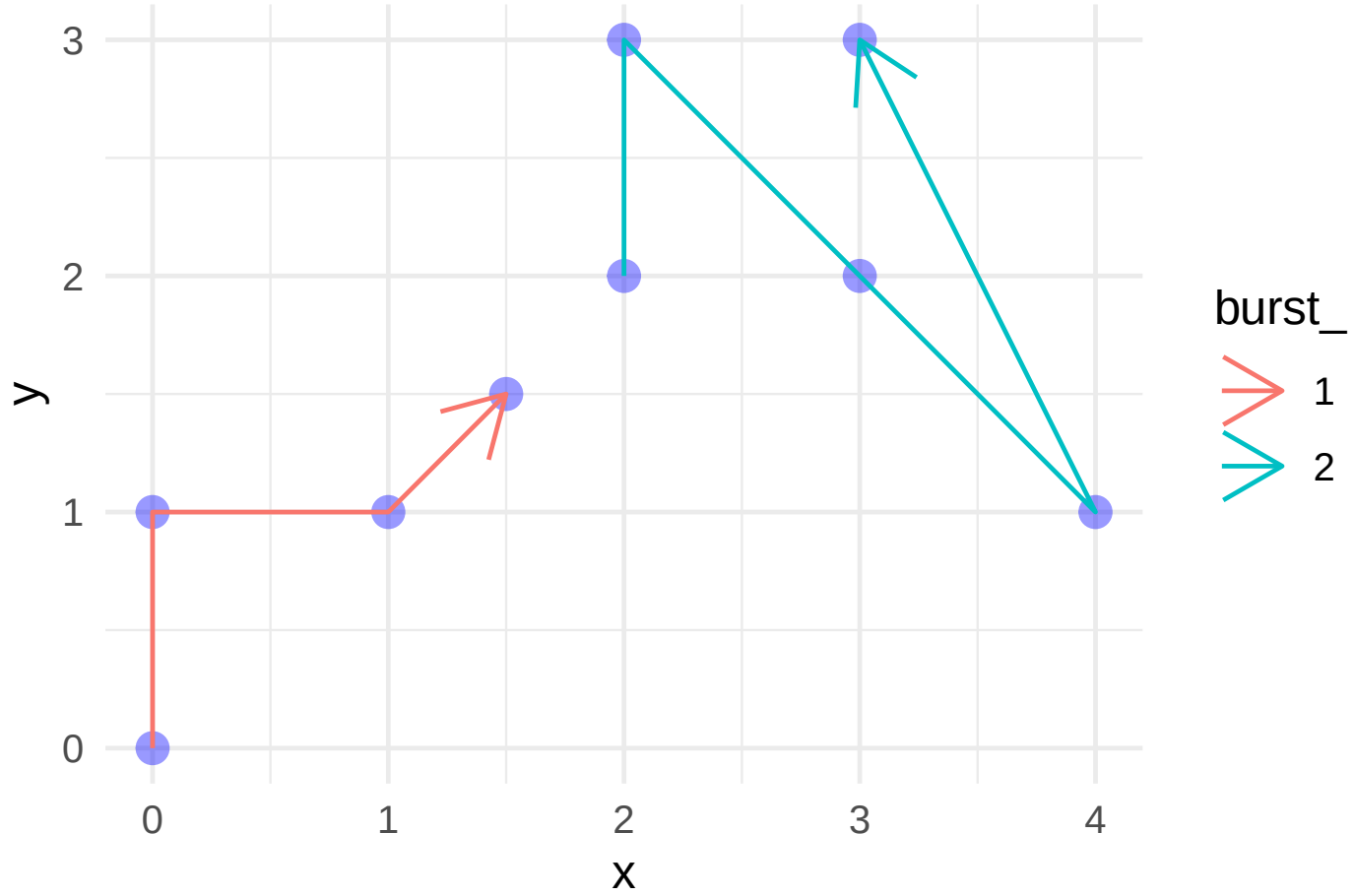
We can start to create a `steps`-representation.



This can be achieved with the function `amt::steps()`. If we resampled the data previously, we can even use `amt::steps_by_burst()`.


3
2
1
0
y
0
1
2
3
4
x


3
2
1
0
y
0
1
2
3
4
x


3
2
1
0
y
0
1
2
3
4
x
burst_
1
2

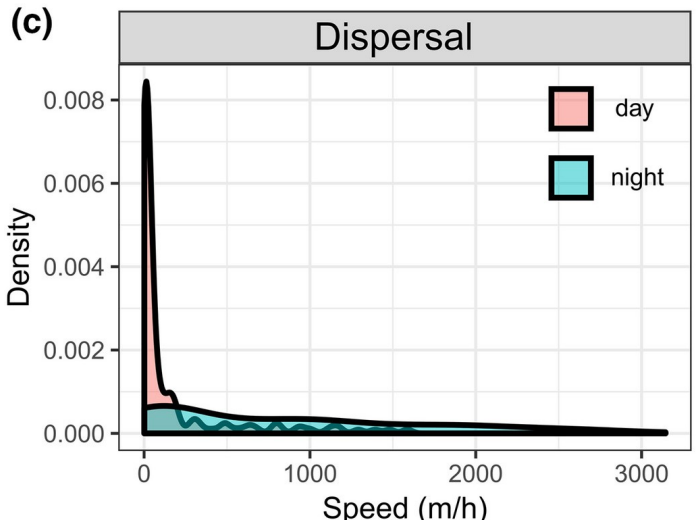
This automatically calculates several step attributes:

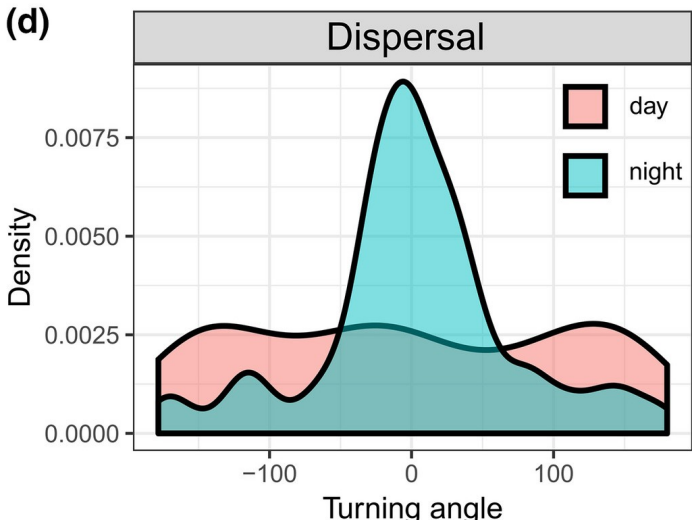
- Start and end point
- Step length
- Absolute and relative turn angles
- Duration

This allows already calculate some step characteristics. It becomes even more informative, if we pair this for example with the whether a step was in the night, day or twilight.

Example

Remington Moll observed a (rare) long distance dispersal for White Tail deer[1] and looked at the turn angle and step distribution for day and night.





[1] Moll et. al Ecology and Evolution; https://onlinelibrary.wiley.com/doi/full/10.1002/ece3.7354.

Take-home message

1. Movement characteristics (e.g., step length or turn angle) are fundamental to many analyses in discrete time.
2. We often compare movement characteristics for different times of day.
3. The Net Squared Displacement (NSD) can be used to infer different migratory modes.

Multiple instances

Motivation

- Most telemetry studies deal with more than one animal.
- Dealing with multiple animals can complicate analyses significantly.
- Some packages provide an infrastructure to deal with multiple animals (e.g. `adehabitat`, `move`).
- `amt` has very limited support for dealing with multiple animals, but relies on the infrastructure provided by the `purrr` package. The hope is, that this adds additional flexibility.

Examples

- Weekly home-range size of bears.
- Does home-range size differ between different sexes, treatments?
- Individual-level habitat selection?
- What is the mean step length for different animals during day and night?

Repeating tasks in R

Three possible ways to deal with multiple instances in R:

1. Copy and paste your code and make slight adaptions (**don't do this**).
2. Use a `for`-loop
3. Use functional programming

Let's consider the following problem: We want to calculate the mean step length of each fisher from the `amt_fisher` data set:

```
library(amt)
data(amt_fisher)
unique(amt_fisher$name)
```

```
[1] "Leroy"   "Ricky T" "Lupe"    "Lucile"
```

For the first animal (here `Leroy`)[2]

```
amt_fisher |> filter(name == "Leroy") |>
  step_lengths() |> mean(na.rm = TRUE)
```

```
[1] 186.9954
```

[2] Note, I am ignoring sampling rates here.

Using a loop:

```
res <- c()
j <- 1
for (i in c("Leroy", "Ricky T", "Lupe", "Lucile")) {
  res[j] <- amt_fisher |> filter(name == i) |>
    step_lengths() |> mean(na.rm = TRUE)
  j <- j + 1
}

res
```

```
[1] 186.99538  41.97390  61.04558  85.95821
```

What is a `tibble`?

- `tibbles` are *modern* `data.frames`.
- A tibble can have list columns.
- A list is an other data structure in R, that can hold any other data structure.

With list columns it is easy to have more complex splits of your data (e.g., animals/seasons/weeks).

What is a list?

Lists are yet an other data structures for R. Lists are can contain any object (even other lists).

```
l <- list("a", b = 1:10, x = list(list(1:10)))
str(l)
```

```
List of 3
$ : chr "a"
$ b: int [1:10] 1 2 3 4 5 6 7 8 9 10
$ x:List of 1
..$ :List of 1
.. ..$ : int [1:10] 1 2 3 4 5 6 7 8 9 10
```

Nest and unnest

An example data set

```
set.seed(12)
dat <- data.frame(
  id = rep(1:10, each = 10),
  x = runif(100),
  y = runif(100)
)

head(dat)
```

```
  id          x         y
1  1 0.06936092 0.4829763
2  1 0.81777520 0.8410553
3  1 0.94262173 0.4551459
4  1 0.26938188 0.8605173
5  1 0.16934812 0.6761024
6  1 0.03389562 0.7275831
```

We can use `nest` and `unnest` to create so called `list`-columns.

```
dat |> nest(data = c(x, y))
```

```
# A tibble: 10 x 2
      id data
   <int> <list>
 1     1 <tibble [10 x 2]>
 2     2 <tibble [10 x 2]>
 3     3 <tibble [10 x 2]>
 4     4 <tibble [10 x 2]>
 5     5 <tibble [10 x 2]>
 6     6 <tibble [10 x 2]>
 7     7 <tibble [10 x 2]>
 8     8 <tibble [10 x 2]>
 9     9 <tibble [10 x 2]>
10    10 <tibble [10 x 2]>
```

```
dat |> nest(data = -id)
```

```
# A tibble: 10 x 2
      id data
   <int> <list>
 1     1 <tibble [10 x 2]>
 2     2 <tibble [10 x 2]>
 3     3 <tibble [10 x 2]>
 4     4 <tibble [10 x 2]>
 5     5 <tibble [10 x 2]>
 6     6 <tibble [10 x 2]>
 7     7 <tibble [10 x 2]>
 8     8 <tibble [10 x 2]>
 9     9 <tibble [10 x 2]>
10    10 <tibble [10 x 2]>
```

We can then work on the nested column(s), using `mutate` in combination with `map_*`:

```
dat |> nest(data = -id) |>
  mutate(centroid.x = map_dbl(data, ~ mean(.x$x)))
```

```
# A tibble: 10 x 3
      id data               centroid.x
   <int> <list>                  <dbl>
 1     1 <tibble [10 x 2]>       0.315
 2     2 <tibble [10 x 2]>       0.444
 3     3 <tibble [10 x 2]>       0.410
 4     4 <tibble [10 x 2]>       0.672
 5     5 <tibble [10 x 2]>       0.459
 6     6 <tibble [10 x 2]>       0.562
 7     7 <tibble [10 x 2]>       0.519
 8     8 <tibble [10 x 2]>       0.575
 9     9 <tibble [10 x 2]>       0.418
10    10 <tibble [10 x 2]>       0.422
```

Lets come back to our example

First, lets use `nest()`:

```
amt_fisher |> nest(data = -name)
```

```
# A tibble: 4 x 2
  name    data
  <chr>   <list>
1 Leroy   <trck_xyt [919 x 5]>
2 Ricky T <trck_xyt [8,958 x 5]>
3 Lupe    <trck_xyt [3,004 x 5]>
4 Lucile  <trck_xyt [1,349 x 5]>
```

Now we can iterate over all animals

```
amt_fisher |> nest(data = -name) |>
  mutate(mean.sl = map_dbl(data, ~ step_lengths(.x) |>
                             mean(na.rm = TRUE)))
```

```
# A tibble: 4 x 3
  name    data                    mean.sl
  <chr>   <list>                    <dbl>
1 Leroy   <trck_xyt [919 x 5]>      187.
2 Ricky T <trck_xyt [8,958 x 5]>     42.0
3 Lupe    <trck_xyt [3,004 x 5]>     61.0
4 Lucile  <trck_xyt [1,349 x 5]>     86.0
```

The approach of list-columns makes it easy to have several grouping instances.

```
amt_fisher |> mutate(wk = lubridate::week(t_)) |>
  nest(data = -c(name, wk)) |>
  mutate(mean.sl = map_dbl(data, ~ step_lengths(.x) |>
                                   mean(na.rm = TRUE)))
```

```
# A tibble: 21 x 4
   name        wk data                    mean.sl
   <chr>    <dbl> <list>                    <dbl>
 1 Leroy        6 <trck_xyt [47 x 5]>       12.6
 2 Leroy        7 <trck_xyt [330 x 5]>     204.
 3 Leroy        8 <trck_xyt [261 x 5]>     239.
 4 Leroy        9 <trck_xyt [281 x 5]>     145.
 5 Ricky T      6 <trck_xyt [232 x 5]>      40.6
 6 Ricky T      7 <trck_xyt [737 x 5]>      40.3
 7 Ricky T      8 <trck_xyt [1,345 x 5]>    46.1
 8 Ricky T      9 <trck_xyt [1,488 x 5]>    47.1
 9 Ricky T     10 <trck_xyt [1,290 x 5]>    42.3
10 Ricky T     11 <trck_xyt [1,280 x 5]>    36.2
# ... with 11 more rows
```

Take-home messages

1. Available data structures can be used to deal with multiple animals.
2. Lists are just “containers” that contain objects.
3. List columns are a great way to organize complex data structures.