CMA - Exercise 4

kinmar01

Table of contents

exercise 4, part of the course cma (mainly based on@Laube2014)

Abstract

1 Input: Segmentation

You've read Laube and Purves (2011) about segmenting trajectories. In the paper, the authors define " *static* " fixes as " * those whose average Euclidean distance to other fixes inside a temporal window v is less than some threshold d * ", as illustrated in **?@fig-laube-purves-2011**

! The figure from Laube and Purves (2011) visualizes steps a) zu d), which will be explained bel

a. Specify a temporal windows v for in which to measure Euclidean distances.
b. Measure the distance from every point to every other point within this temporal window v.
c. Remove "static points":These are points where the average distance is less than a given threshold. This segments the trajectory into subtrajectories.

d. Now remove short subtrajectories:These are trajectories with a short duration (whereas "short" is tbd).

We will ** demonstrate ** implementing this method on the wild boar "Sabi", restricting ourselves to a couple of tracking days. Your task will be to understand this implementation and apply it to your own movement data.

Open a RStudio Project for this week. Next, copy the wild boar data you downloaded last week (*wildschwein_BE_2056.csv*) to your project folder. If you cannot find this dataset on your computer, you can re - download it from moodle. Transform the data into an `sf` object, filter for the wild boar Sabi and a datetime between "2015 - 07 - 01" and "2015 - 07 - 03".

```
pacman::p_load("readr", "sf", "dplyr", "ggplot2")

theme_minimal() |> theme_set()
```

```
wildschwein <- read_delim("data/wildschwein_BE_2056.csv", ",")
```

```
Rows: 51246 Columns: 6
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): TierID, TierName
dbl  (3): CollarID, E, N
dttm (1): DatetimeUTC

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Careful! What Timezone is assumed?
sabi <- wildschwein |>
st_as_sf(coords = c("E", "N"),
crs = 2056,
remove = FALSE) |>
filter(TierName == "Sabi",
DatetimeUTC >= "2015-07-01",
DatetimeUTC < "2015-07-03")

sabi |> summary()
```

```
    TierID            TierName            CollarID
 Length:192         Length:192         Min.   :12275
 Class :character   Class :character   1st Qu.:12275
```

```
 Mode  :character   Mode  :character   Median :12275
                                       Mean   :12275
                                       3rd Qu.:12275
                                       Max.   :12275
  DatetimeUTC                            E                  N
 Min.   :2015-06-30 22:00:13.00   Min.   :2569724   Min.   :1204916
 1st Qu.:2015-07-01 09:56:28.50   1st Qu.:2569791   1st Qu.:1205121
 Median :2015-07-01 21:52:58.50   Median :2570466   Median :1205140
 Mean   :2015-07-01 21:52:50.82   Mean   :2570242   Mean   :1205172
 3rd Qu.:2015-07-02 09:49:05.75   3rd Qu.:2570475   3rd Qu.:1205180
 Max.   :2015-07-02 21:45:16.00   Max.   :2570927   Max.   :1205957
          geometry
 POINT        :192
 epsg:2056    :  0
 +proj=some...:  0
```

```
sabi |> str()
```

```
sf [192 x 7] (S3: sf/spec_tbl_df/tbl_df/tbl/data.frame)
 $ TierID     : chr [1:192] "002A" "002A" "002A" "002A" ...
 $ TierName   : chr [1:192] "Sabi" "Sabi" "Sabi" "Sabi" ...
 $ CollarID   : num [1:192] 12275 12275 12275 12275 12275 ...
 $ DatetimeUTC: POSIXct[1:192], format: "2015-06-30 22:00:13" "2015-06-30 22:16:06" ...
 $ E          : num [1:192] 2569972 2569975 2570266 2570208 2570247 ...
 $ N          : num [1:192] 1205366 1205637 1205857 1205913 1205731 ...
 $ geometry   :sfc_POINT of length 192; first list element:  'XY' num [1:2] 2569972 1205366
 - attr(*, "spec")=
  .. cols(
  ..   TierID = col_character(),
  ..   TierName = col_character(),
  ..   CollarID = col_double(),
  ..   DatetimeUTC = col_datetime(format = ""),
  ..   E = col_double(),
  ..   N = col_double()
  .. )
 - attr(*, "problems")=<externalptr>
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA
  ..- attr(*, "names")= chr [1:6] "TierID" "TierName" "CollarID" "DatetimeUTC" ...
```

```r
sabi |>
ggplot(aes(E, N)) +
geom_point() +
geom_path() +
theme_minimal()
```

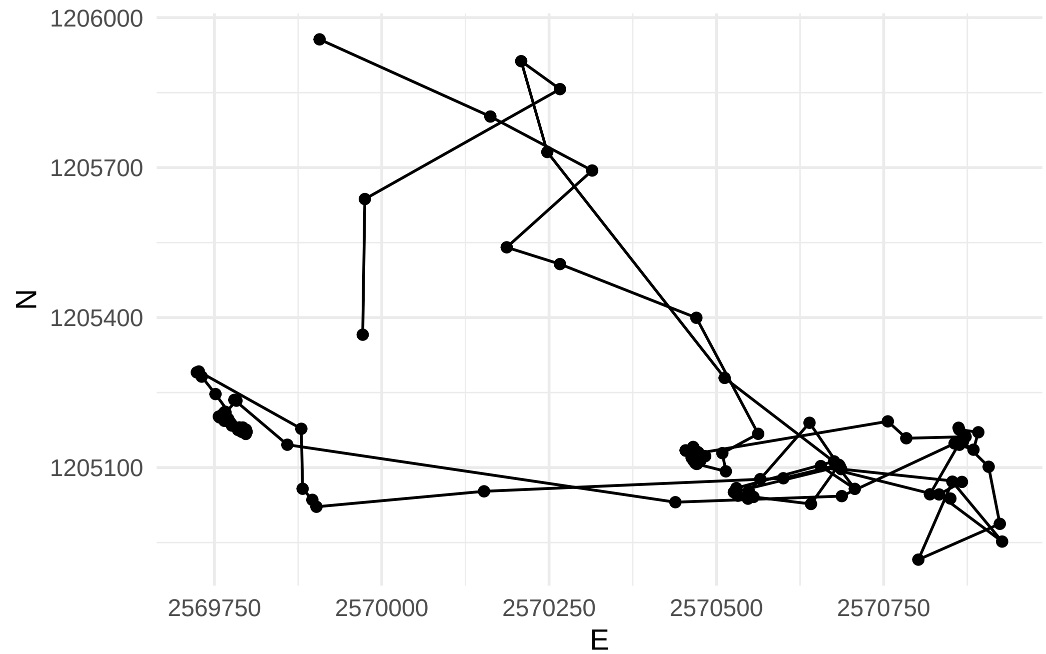


Figure 1: Movement of the wild boar 'Sabi' in the timespan 01 - 02.07.2015. The cluster of dots / fixes are possible 'static' points

Step a): Specify a temporal window v

In the above dataset, the sampling interval is 15 minutes. If we take a temporal window of 60 minutes, that would mean including 4 fixes. We need to calculate the following Euclidean distances (pos representing single location):

1. `pos[n-2]` to `pos[n]`
2. `pos[n-1]` to `pos[n]`
3. `pos[n]` to `pos[n+1]`
4. `pos[n]` to `pos[n+2]`

Step b): Measure the distance to every point within v

We can use the function distance_by_element from week 2 in combination with `lead()` and `lag()` to calculate the Euclidean distance. For example, to create the necessary offset of n-2, we use `lag(x, 2)`. For each offset, we create one individual column.

```r
distance_by_element <- function(later, now) {
  as.numeric(
    st_distance(later, now, by_element = TRUE)
  )
}



sabi <- sabi |>
  mutate(
    nMinus2 = distance_by_element(lag(geometry,2),geometry),
    nMinus1 = distance_by_element(lag(geometry,1),geometry),
    nPlus1 = distance_by_element(geometry,lead(geometry,1)),
    nPlus2 = distance_by_element(geometry,lead(geometry,2))
  )
```

Now we want to calculate the mean distance of `nMinus2`, `nMinus1`, `nPlus1`, `nPlus2` for each row. Since we want the mean value *per Row*, we have to explicitly specify this before `mutate()` with the function `rowwise()`. To remove this rowwise-grouping, we end the operation with `ungroup()`.

Note that for the first two positions, we cannot calculate a `stepMean` since there is no Position `n-2` for these positions. This is also true for the last to positions (lacking a position `n+2`).

```r
sabi <- sabi |>
  rowwise() |>
  mutate(
    stepMean = mean(c(nMinus2, nMinus1, nPlus1, nPlus2))
  ) |>
  ungroup()
```

Step c): Remove "static points"

We can now determine if an animal is moving or not by specifying a threshold distance on `stepMean`. In our example, we use the mean value as a threshold: Positions with distances below this value are considered static.

```r
sabi <- sabi |>
  mutate(static = stepMean < mean(stepMean, na.rm = TRUE))
```

```
sabi_moving <- sabi |>
  filter(!static)

sabi_static <- sabi |>
  filter(static)
```

```
sabi_moving |>
  ggplot(aes(E, N)) +
  geom_point(data = sabi_static, col = "red") +
  geom_path() +
  geom_point() +
  coord_fixed() +
  theme(legend.position = "bottom")
```

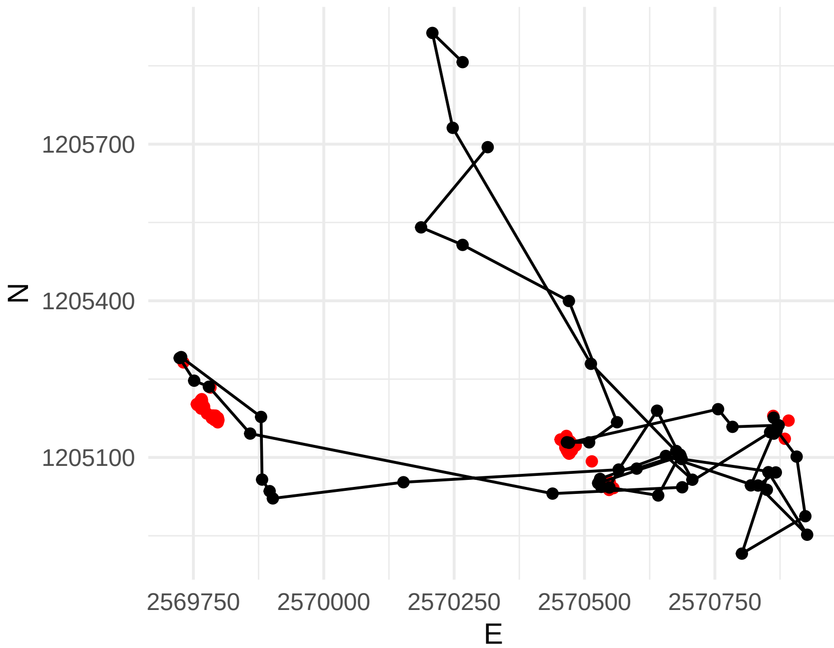


Figure 2: The trajectory of sabi. Red dots are static points, the black dots signify moving points

2 Exercise A: Segmentation

With the skills from Input: Segmentation you can now implement the segmentation algorithm described in Laube and Purves (2011) to either your own movement data or to a different wild boar using different sampling intervals.

2.1 Task 1: Calculate distances

Now, you can Step a): Specify a temporal window v and Step b): Measure the distance to every point within v, which you had used with sabi, on on your own movement data or to a different wild boar using different sampling intervals.

```
df_tannenhaeher <- read_delim("tannenhaeher.csv") |>
  st_as_sf(coords = c("x", "y"), crs = 2056, remove = FALSE)
```

```
Rows: 8721 Columns: 17
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (5): tag_tech_s, sensor_typ, individual, ind_ident, study_name
dbl  (9): long, lat, external_t, hdop, satellite_, height, tag_ident, x, y
lgl  (2): date, time
dttm (1): timestamp

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
df_tannenhaeher_K125864 <- df_tannenhaeher |>
  filter(ind_ident=="K125864")

df_tannenhaeher_K121752 <- df_tannenhaeher |>
  filter(ind_ident=="K121752")
```

```
df_tannenhaeher_K125864 |>
  ggplot(aes(x,y)) +
  geom_point() +
  geom_path(alpha=0.4)
```

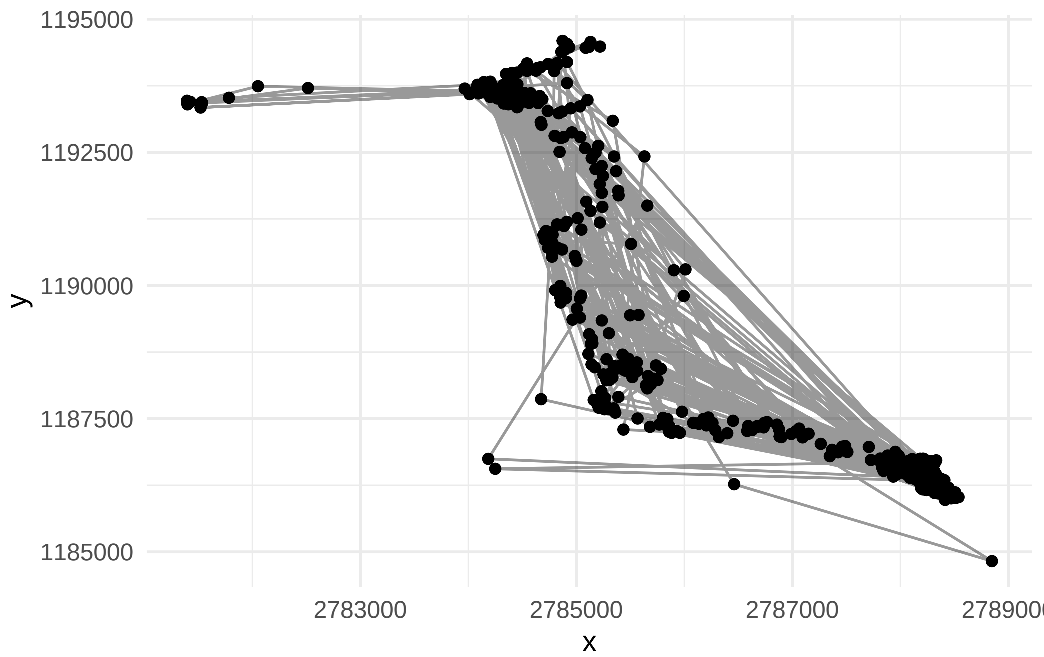


Figure 3: Movement of the Spotted Nutcracker K125864. The cluster of dots / fixes are possible ‘static’ points

> **move or stop**
>
> Ask chatGPT for a reasonable threshold beside mean, median and Q1

```
steps <- function(df) {
  df_updated <- df |>
    mutate(
      nMinus2 = distance_by_element(lag(geometry,2),geometry),
      nMinus1 = distance_by_element(lag(geometry,1),geometry),
      nPlus1 = distance_by_element(geometry,lead(geometry,1)),
      nPlus2 = distance_by_element(geometry,lead(geometry,2))
    ) |>
    rowwise() |>
    mutate(
      stepMean = mean(c(nMinus2, nMinus1, nPlus1, nPlus2))
    ) |>
    ungroup()|>
    mutate(
      mean = mean(stepMean, na.rm = TRUE),
```

```
      median = median(stepMean, na.rm = TRUE),
      Q1 = quantile(stepMean, 0.25, na.rm = TRUE),
      static = stepMean < Q1
    )

  return (df_updated)
}


df_tannenhaeher_K125864 <- df_tannenhaeher_K125864 |> steps()
df_tannenhaeher_K121752 <- df_tannenhaeher_K121752 |> steps()
```

2.2 Task 2: Specify and apply threshold *d*

After calculating the Euclidean distances to positions within the temporal window v in task 1, you can explore these values (we stored them in the column `stepMean`) using summary statistics (histograms, boxplot, `summary()`): This way we can define a reasonable threshold value to differentiate between *stops* and *moves*. There is no "correct" way of doing this, specifying a threshold always depends on data as well as the question that needs to be answered. In this exercise, use the mean of all `stepMean` values.

Store the new information (boolean to differentiate between stops (`TRUE`) and moves (`FALSE`)) in a new column named `static`.

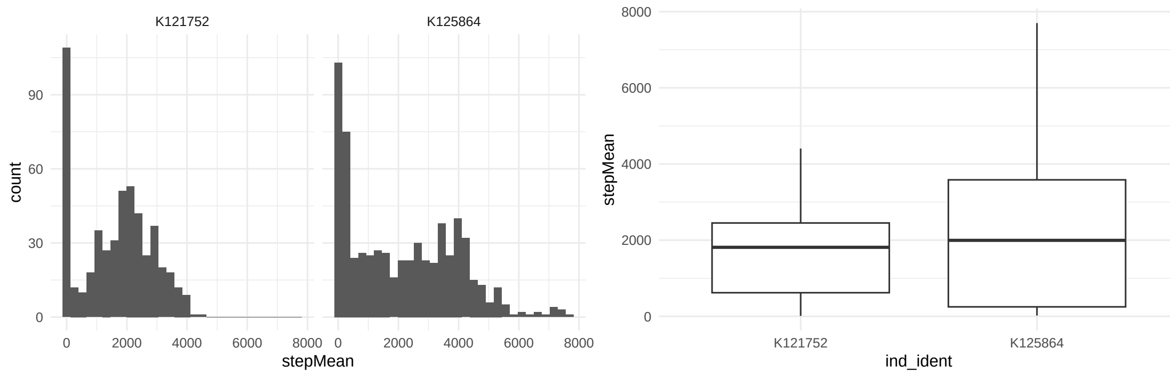
```
df_spotted_nutcracker <- union(
  df_tannenhaeher_K121752,
  df_tannenhaeher_K125864
)

df_spotted_nutcracker|>
  ggplot(aes(stepMean)) +
  geom_histogram() +
  facet_wrap(.~ind_ident)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
Warning: Removed 8 rows containing non-finite outside the scale range
(`stat_bin()`).
```

```
df_spotted_nutcracker |>
  ggplot(aes(ind_ident,stepMean))+
  geom_boxplot()
```

```
Warning: Removed 8 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```



(a) Histogram

(b) Boxplot

Figure 4: Summary statistics for both spotted Nutcracker

2.3 Task 3: Visualize segmented trajectories

Now visualize the segmented trajectory spatially. Just like last week, you can use ggplot with `geom_path()`, `geom_point()` and `coord_equal()`. Assign `colour = static` within `aes()` to distinguish between segments *with* “movement” and *without*.

```
df_spotted_nutcracker|>
  filter(!static) |>
  ggplot(aes(x, y)) +
  geom_path(alpha=0.3) +
  geom_point() +
  geom_point(data = df_spotted_nutcracker |> filter(static), col = "red") +
  coord_equal()+
  facet_wrap(.~ind_ident)
```

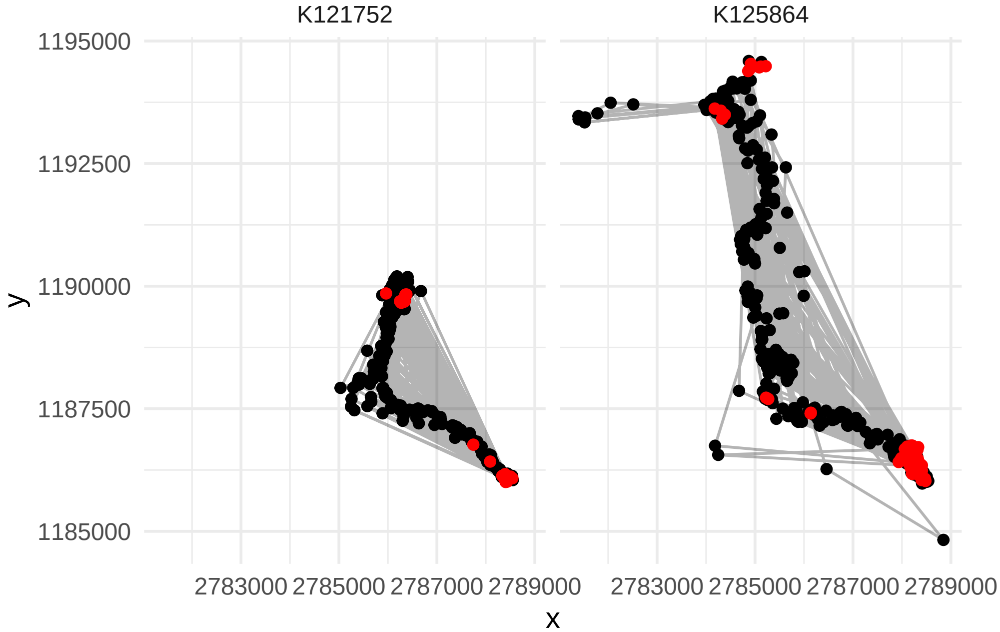


Figure 5: The trajectory of Spotted Nutcracker K125864 & K121752.. Red dots are static points, the black dots signify moving points

2.4 Task 4: Segment-based analysis

In applying Laube and Purves (2011), we've come as far as step b) in **?@fig-laube-purves-2011**. In order to complete the last steps (c and d), we need a *unique* ID for each segment that we can use as a grouping variable. The following function does just that (it assigns unique IDs based on the column `static` which you created in Task 2). You will learn about functions next week. For now, just copy the following code chunk into your script and run it.

3 References

Laube, Patrick, and Ross S. Purves. 2011. "How Fast Is a Cow? Cross-Scale Analysis of Movement Data." *Transactions in GIS* 15 (3): 401–18. https://doi.org/https://doi.org/10.1111/j.1467-9671.2011.01256.x.