deepSSF Data Prep

Scott Forrest

2025-02-13

Here we prepare the data for fitting a deepSSF model. We load the GPS data and tidy it, create a trajectory object containing a series of steps and read in the environmental layers. For each observed step we crop out local subsets of the environmental layers, centred on the step's location. For every step we will then have the surrounding environmental information stored as a stack of small rasters, with the 'target' of what we're trying to predict (the actual location of the next step) as a raster layer as well, with all cells being 0 except the observed next location, which is 1. These, along with temporal covariates and the previous bearing, will be the input data for the deepSSF model.

Table of contents

Loading packages

```
library(tidyverse)
packages <- c("amt", "sf", "terra", "beepr", "tictoc")
walk(packages, require, character.only = T)
```

Import data

```
buffalo <- read_csv("data/buffalo.csv")
```

```
New names:
Rows: 133161 Columns: 11
-- Column specification
-------------------------------------------------------- Delimiter: "," chr
(2): node, dates dbl (7): ...1, lat, lon, height, accuracy, heading, speed dttm
(2): timestamp, DateTime
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.
* `` -> `...1`
```

```
head(buffalo)
```

```
# A tibble: 6 x 11
   ...1 timestamp             lat   lon height accuracy heading speed node
  <dbl> <dttm>              <dbl> <dbl>  <dbl>    <dbl>   <dbl> <dbl> <chr>
1   169 2018-07-25 00:04:02 -12.3  134.   91.3      977     0    0     2005
2   170 2018-07-25 01:04:23 -12.3  134.   92.1      704    77.0 0.034 2005
3   171 2018-07-25 02:04:39 -12.3  134.   95.2     1105   129.  0.008 2005
4   172 2018-07-25 03:04:17 -12.3  134.  104.       865     0   0.013 2005
5   173 2018-07-25 04:04:39 -12.3  134.   85.5      711   231.  0.01  2005
6   174 2018-07-25 05:04:27 -12.3  134.   95.8      780   207.  0.012 2005
# i 2 more variables: dates <chr>, DateTime <dttm>
```

Tidy data

```
# remove individuals that have poor data quality or less than about 3 months of data.
# The "2014.GPS_COMPACT copy.csv" string is a duplicate of ID 2024, so we also exclude it
buffalo <- buffalo %>% filter(!node %in% c("2014.GPS_COMPACT copy.csv",
                                           # 2005, 2014, 2018, 2021, 2022, 2024,
```

```
                                          2029, 2043, 2265, 2284, 2346, 2354))

# arrange by time and exclude any duplicate timestamps (within the same individual's data)
buffalo <- buffalo %>%
  group_by(node) %>%
  arrange(DateTime, .by_group = T) %>%
  distinct(DateTime, .keep_all = T) %>%
  arrange(node) %>%
  mutate(ID = node)

# keep only the relevant columns
buffalo_clean <- buffalo[, c(12, 2, 4, 3)]

# rename the columns
colnames(buffalo_clean) <- c("id", "time", "lon", "lat")

# convert the time to the correct timezone
attr(buffalo_clean$time, "tzone") <- "Australia/Queensland"
head(buffalo_clean)
```

```
# A tibble: 6 x 4
  id    time                 lon   lat
  <chr> <dttm>             <dbl> <dbl>
1 2005  2018-07-25 10:04:02  134. -12.3
2 2005  2018-07-25 11:04:23  134. -12.3
3 2005  2018-07-25 12:04:39  134. -12.3
4 2005  2018-07-25 13:04:17  134. -12.3
5 2005  2018-07-25 14:04:39  134. -12.3
6 2005  2018-07-25 15:04:27  134. -12.3
```

```
# check timezone
tz(buffalo_clean$time)
```

```
[1] "Australia/Queensland"
```

```
# create a vector of the unique ids for indexing later
buffalo_ids <- unique(buffalo_clean$id)
```

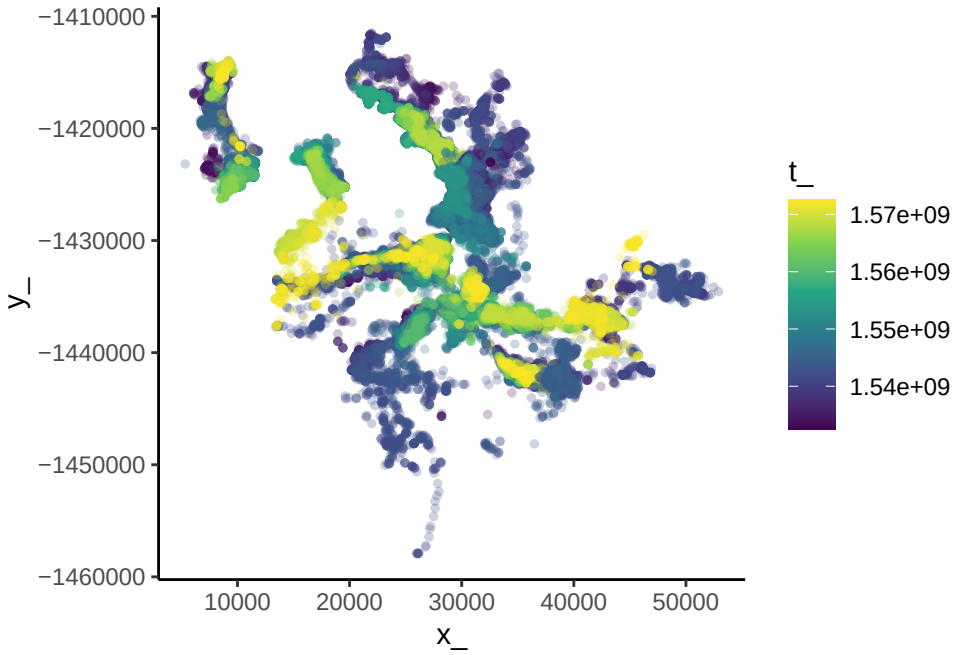
Setup trajectory

Use the `amt` package to create a trajectory object from the cleaned data.

```r
buffalo_all <- buffalo_clean %>% mk_track(id = id,
                                          lon,
                                          lat,
                                          time,
                                          all_cols = T,
                                          crs = 4326) %>%
  # Transformation to GDA94 / Geoscience Australia Lambert (https://epsg.io/3112)
  transform_coords(crs_to = 3112, crs_from = 4326) %>% arrange(id)
```

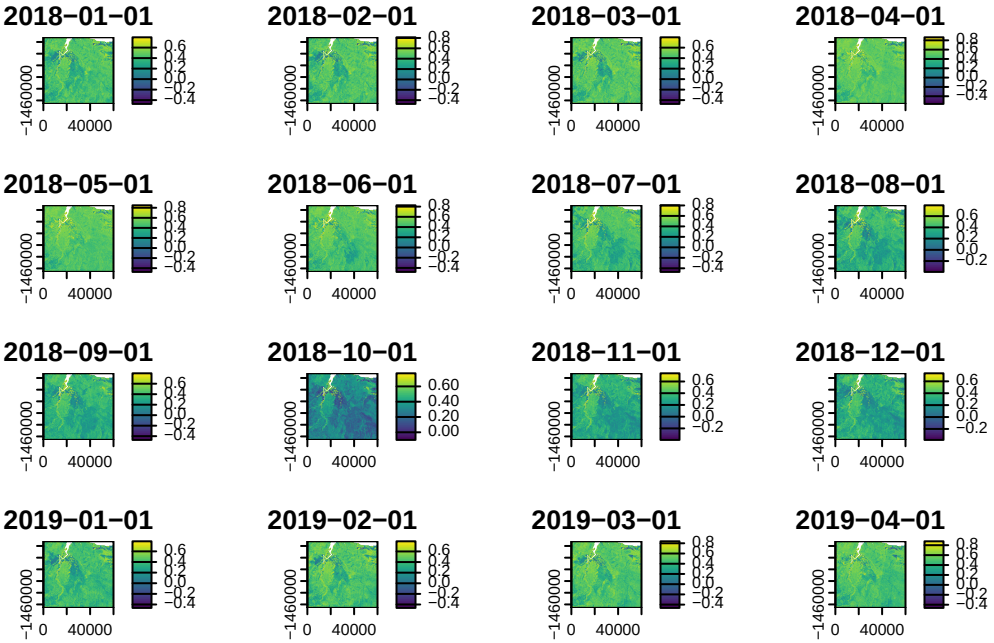
Plot the data coloured by time

```r
buffalo_all %>%
  ggplot(aes(x = x_, y = y_, colour = t_)) +
  geom_point(alpha = 0.25, size = 1) +
  coord_fixed() +
  scale_colour_viridis_c() +
  theme_classic()
```

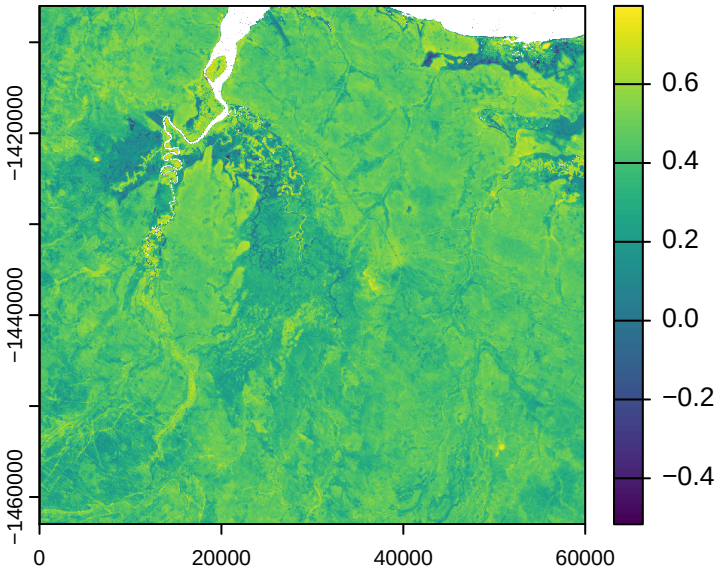


Read in the environmental covariates

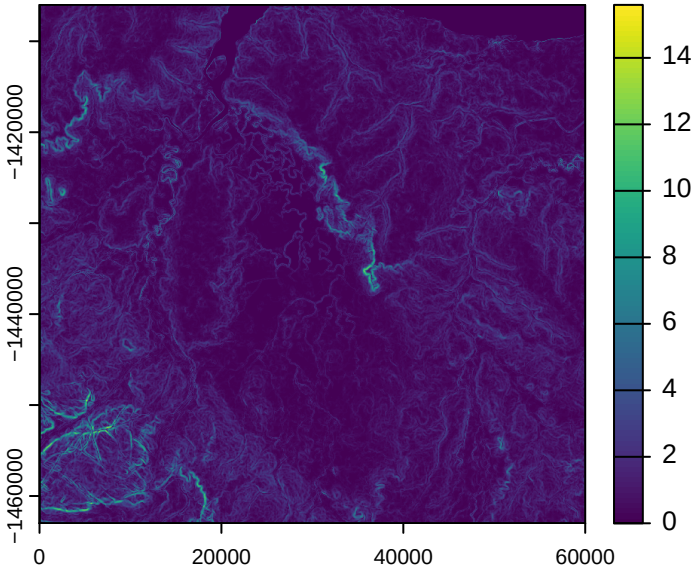
```r
ndvi_projected <- rast("mapping/cropped rasters/ndvi_GEE_projected_watermask20230207.tif")
terra::time(ndvi_projected) <- as.POSIXct(lubridate::ymd("2018-01-01") + months(0:23))
slope <- rast("mapping/cropped rasters/slope_raster.tif")
veg_herby <- rast("mapping/cropped rasters/veg_herby.tif")
canopy_cover <- rast("mapping/cropped rasters/canopy_cover.tif")

# change the names (these will become the column names when extracting
# covariate values at the used and random steps)
names(ndvi_projected) <- rep("ndvi", terra::nlyr(ndvi_projected))
names(slope) <- "slope"
names(veg_herby) <- "veg_herby"
names(canopy_cover) <- "canopy_cover"

# to plot the rasters
plot(ndvi_projected)
```



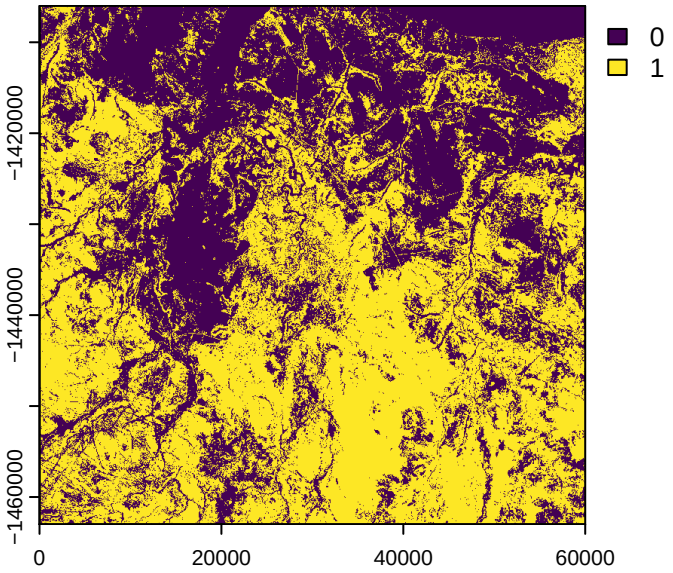

```r
plot(ndvi_projected[[1]])
```



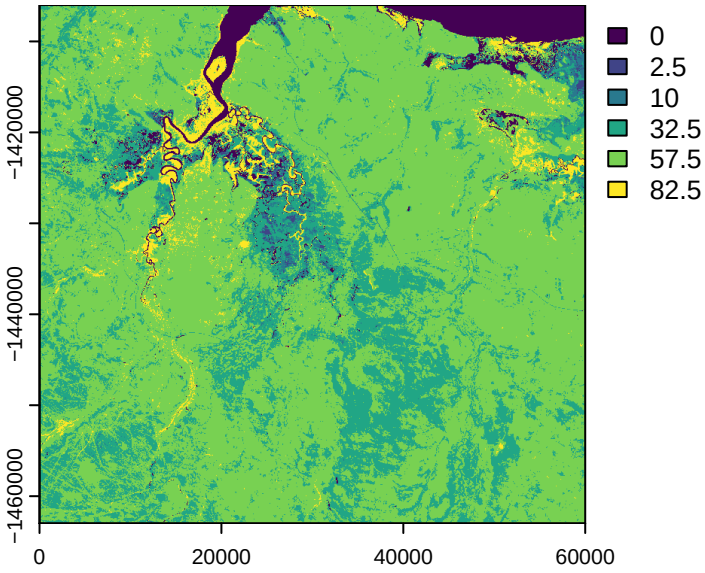

```
plot(slope)
```




```
plot(veg_herby)
```




```
plot(canopy_cover)
```




```
# get the resolution from the covariates
res <- terra::res(ndvi_projected)[1]
```

Generating the data to fit a deepSSF model

Create a steps object to check the step lengths

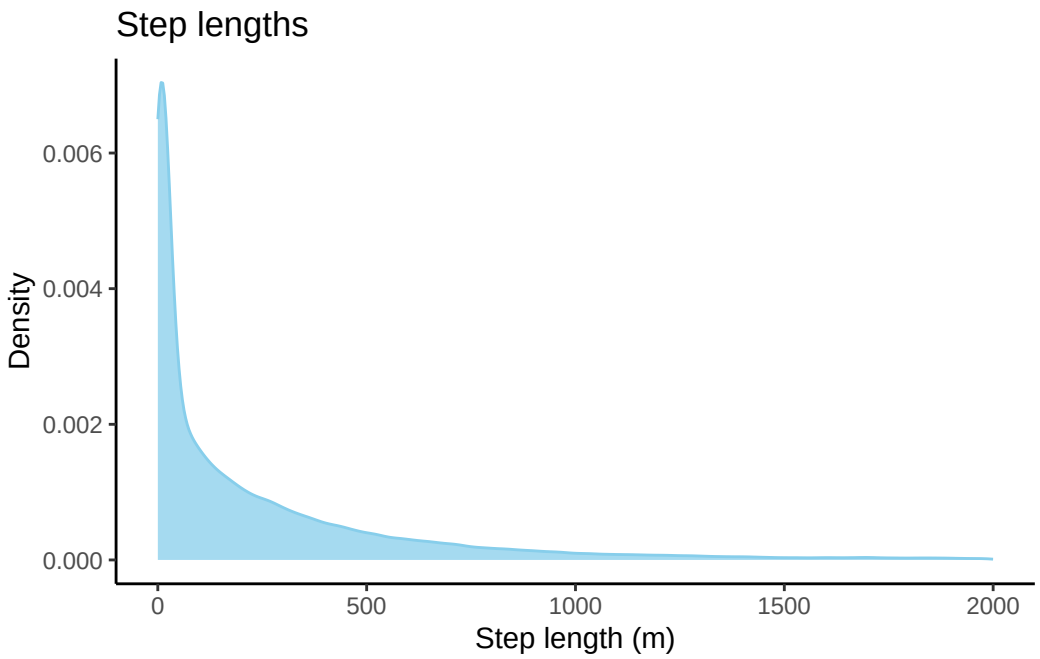
```
# nest the data by individual
buffalo_all_nested <- buffalo_all %>% arrange(id) %>% nest(data = -"id")

buffalo_all_nested_steps <- buffalo_all_nested %>%
  mutate(steps = map(data, function(x)
    x %>%
      # track_resample(rate = hours(1), tolerance = minutes(10)) %>%
      steps()))

# unnest the data after creating 'steps' objects
buffalo_all_steps <- buffalo_all_nested_steps %>%
  amt::select(id, steps) %>%
  amt::unnest(cols = steps)

head(buffalo_all_steps)
```

```
# A tibble: 6 x 11
  id       x1_    x2_       y1_       y2_   sl_ direction_p     ta_
* <chr>  <dbl>  <dbl>     <dbl>     <dbl> <dbl>       <dbl>   <dbl>
1 2005  41941. 41969. -1435875. -1435671. 206.        1.43  NA
2 2005  41969. 41922. -1435671. -1435654.  50.7       2.80   1.37
3 2005  41922. 41779. -1435654. -1435601. 152.        2.78  -0.0214
4 2005  41779. 41841. -1435601. -1435635.  70.7      -0.507  2.99
5 2005  41841. 41655. -1435635. -1435604. 188.        2.98  -2.80
6 2005  41655. 41619. -1435604. -1435608.  37.1      -3.02   0.285
# i 3 more variables: t1_ <dttm>, t2_ <dttm>, dt_ <drtn>
```

Plot step lengths

```
# plot step lengths
buffalo_all_steps %>%
  filter(sl_ < 2000) %>%
  ggplot() +
  geom_density(aes(x = sl_), #, bins = 50
                 fill = "skyblue", colour = "skyblue", alpha = 0.75) +
  scale_x_continuous("Step length (m)") + #, limits = c(-25, 1250)
  scale_y_continuous("Density") +
```

```
  ggtitle("Step lengths") +
  theme_classic()
```




```
# proportion of steps less than the buffer distance
# the distance should be a multiple of the resolution
distance <- 1250

# check if multiple of the resolution?
distance %% res == 0
```

```
[1] TRUE
```

```
# buffer distance (add the resolution of the cell/2 as we want a centre cell)
buffer <- 1250 + (res/2)

# calculate proportion
prop_steps <- buffalo_all_steps %>% filter(sl_ < buffer) %>% nrow() / buffalo_all_steps %>%
print(paste0("Proportion of steps less than ", buffer, "m: ", round(prop_steps, 4)))
```

```
[1] "Proportion of steps less than 1262.5m: 0.9573"
```

```
# distance to the corner
corner_dist <- sqrt(buffer^2 + buffer^2)
print(paste0("Distance to the corner: ", round(corner_dist, 2)))
```

```
[1] "Distance to the corner: 1785.44"
```

In our case 95% of the steps were less than a distance of 1262.5m, which would mean that including 101 x 101 cells (which would be 2525 m x 2525 m) would include at least 95% of the steps.

Note: as 1262.5 m is the shortest distance to the boundary, and it is further to a corner (which is 1786 m away), which means we will actually be including more than 95% of the steps.

Set up the spatial extent of the local covariates

We calculate the number of cells in each axis, and set the lag between locations. Typically this will just be 1, as we want the next-step to be the target. However, it may be possible to improve model training by using different lags and including the time difference between locations as a covariate. I expect that this would help to predict across different time scales and not be restricted to the time scale that the data was collected at.

```
# calculate the number of cells in each axis
nxn_cells <- buffer*2/res
print(paste0("Number of cells in each axis: ", nxn_cells))
```

```
[1] "Number of cells in each axis: 101"
```

```
# hourly lag - to set larger time differences between locations
hourly_lag <- 1
```

Loop over each individual and save the local rasters

This is the main function of the script. It loops over each individual and calculates step level information, such as the location and time of the next step (x2, y2, t2), step length, turning angle, and different time components.

It also determines where to crop the local layers by subtracting and adding the buffer distance from the location of each step. It then crops out the local layers for each step and saves them as a list entry into the data frame.

It then creates a raster stack (with the number of layers equal to the number of steps) for each covariate and saves it as a tif.

For each individual, this will therefore create a csv containing the step level information, and a tif for each covariate containing the local layers for each step.

It takes quite a while to run, around 1 hour per individual in our case, so we will illustrate the function with 10 steps per individual. Uncomment the line specified in the loop to use the full dataset.

```
for(i in 1:length(buffalo_ids)) {

  buffalo_data <- buffalo_all %>% filter(id == buffalo_ids[i])
  # all data for that individual
  buffalo_data <- buffalo_data %>% arrange(t_)

  # to use a subset of the data for that individual for testing
  # COMMENT THIS LINE OUT TO USE THE FULL DATASET
  buffalo_data <- buffalo_data %>% arrange(t_) |>  slice(1:10)

  n_samples <- nrow(buffalo_data)

  tic()

  buffalo_data_covs <- buffalo_data %>% mutate(

    x1_ = x_,
    y1_ = y_,
    x2_ = lead(x1_, n = hourly_lag, default = NA),
    y2_ = lead(y1_, n = hourly_lag, default = NA),
    x2_cent = x2_ - x1_,
    y2_cent = y2_ - y1_,
    t2_ = lead(t_, n = hourly_lag, default = NA),
    t_diff = round(difftime(t2_, t_, units = "hours"),0),
    hour_t1 = lubridate::hour(t_),
    yday_t1 = lubridate::yday(t_),
    hour_t2 = lubridate::hour(t2_),
    hour_t2_sin = sin(2*pi*hour_t2/24),
    hour_t2_cos = cos(2*pi*hour_t2/24),
    yday_t2 = lubridate::yday(t2_),
    yday_t2_sin = sin(2*pi*yday_t2/365.25),
    yday_t2_cos = cos(2*pi*yday_t2/365.25),

    sl = c(sqrt(diff(y_)^2 + diff(x_)^2), NA),
    log_sl = log(sl),
    bearing = c(atan2(diff(y_), diff(x_)), NA),
    bearing_sin = sin(bearing),
    bearing_cos = cos(bearing),
    ta = c(NA, ifelse(
      diff(bearing) > pi, diff(bearing)-(2*pi), ifelse(
        diff(bearing) < -pi, diff(bearing)+(2*pi), diff(bearing)))),
```

```
  cos_ta = cos(ta),

  # extent for cropping the spatial covariates
  x_min = x_ - buffer,
  x_max = x_ + buffer,
  y_min = y_ - buffer,
  y_max = y_ + buffer,


  # crop out and store the local covariates centered on the animal's location
  # with an extent set in the previous chunk
) %>% rowwise() %>% mutate(

  extent_00centre = list(ext(x_min - x_, x_max - x_, y_min - y_, y_max - y_)),

  # NDVI
  ndvi_index = which.min(abs(difftime(t_, terra::time(ndvi_projected)))),
  ndvi_cent = list({
    ndvi_cent = crop(ndvi_projected[[ndvi_index]], ext(x_min, x_max, y_min, y_max))
    ext(ndvi_cent) <- extent_00centre
    ndvi_cent
    }),

  # herbaceous vegetation
  veg_herby_cent = list({
    veg_herby_cent = crop(veg_herby, ext(x_min, x_max, y_min, y_max))
    ext(veg_herby_cent) <- extent_00centre
    veg_herby_cent
    }),

  # canopy cover
  canopy_cover_cent = list({
    canopy_cover_cent = crop(canopy_cover, ext(x_min, x_max, y_min, y_max))
    ext(canopy_cover_cent) <- extent_00centre
    canopy_cover_cent
    }),

  # slope
  slope_cent = list({
    slope_cent <- crop(slope, ext(x_min, x_max, y_min, y_max))
    ext(slope_cent) <- extent_00centre
    slope_cent
    }),
```

```
  # rasterised location of the next step - centred on (0,0)
  points_vect_cent = list(terra::vect(cbind(x2_ - x_, y2_ - y_), type = "points", crs = "E
  pres_cent = list(rasterize(points_vect_cent, ndvi_cent, background=0))

) %>% ungroup() # to remove the 'rowwise' class


toc()

# remove steps that fall outside of the local spatial extent
buffalo_data_covs <- buffalo_data_covs %>%
  filter(x2_cent > -buffer & x2_cent < buffer & y2_cent > -buffer & y2_cent < buffer) %>%
  drop_na(ta)

# remove the columns that are no longer needed
buffalo_data_df <- buffalo_data_covs %>%
dplyr::select(-extent_00centre,
              -ndvi_cent,
              -veg_herby_cent,
              -canopy_cover_cent,
              -slope_cent,
              -points_vect_cent,
              -pres_cent
              )

# save the data
write_csv(buffalo_data_df, paste0("buffalo_local_data_id/website_temp/buffalo_", buffalo_i
                                  "_data_df_lag_", hourly_lag, "hr_n", n_samples, ".csv"))

# saving the raster objects
rast(buffalo_data_covs$ndvi_cent) %>%
  writeRaster(paste0("buffalo_local_layers_id/buffalo_", buffalo_ids[i], "_ndvi_cent",
                     nxn_cells, "x", nxn_cells, "_lag_", hourly_lag, "hr_n", n_samples, ".
              overwrite = T)

rast(buffalo_data_covs$veg_herby_cent) %>%
  writeRaster(paste0("buffalo_local_layers_id/buffalo_", buffalo_ids[i], "_herby_cent",
                     nxn_cells, "x", nxn_cells, "_lag_", hourly_lag, "hr_n", n_samples, ".
              overwrite = T)

rast(buffalo_data_covs$canopy_cover_cent) %>%
  writeRaster(paste0("buffalo_local_layers_id/buffalo_", buffalo_ids[i], "_canopy_cent",
                     nxn_cells, "x", nxn_cells, "_lag_", hourly_lag, "hr_n", n_samples, ".
              overwrite = T)
```

```
  rast(buffalo_data_covs$slope_cent) %>%
    writeRaster(paste0("buffalo_local_layers_id/buffalo_", buffalo_ids[i], "_slope_cent",
                       nxn_cells, "x", nxn_cells, "_lag_", hourly_lag, "hr_n", n_samples, ".
                overwrite = T)

  rast(buffalo_data_covs$pres_cent) %>%
    writeRaster(paste0("buffalo_local_layers_id/buffalo_", buffalo_ids[i], "_pres_cent",
                       nxn_cells, "x", nxn_cells, "_lag_", hourly_lag, "hr_n", n_samples, ".
                overwrite = T)

}
```

```
1.77 sec elapsed
1.28 sec elapsed
1.34 sec elapsed
1.31 sec elapsed
1.53 sec elapsed
1.22 sec elapsed
1.36 sec elapsed
1.48 sec elapsed
1.23 sec elapsed
1.31 sec elapsed
1.77 sec elapsed
1.26 sec elapsed
1.42 sec elapsed
```

Check the outputs

```
head(buffalo_data_covs)
```

```
# A tibble: 6 x 39
      x_      y_ t_                  id       x1_     y1_    x2_     y2_ x2_cent
   <dbl>   <dbl> <dttm>               <chr>  <dbl>   <dbl>  <dbl>   <dbl>   <dbl>
1 32550. -1.42e6 2018-07-25 12:11:31 2393  32550. -1.42e6 32551. -1.42e6   0.323
2 32551. -1.42e6 2018-07-25 13:10:28 2393  32551. -1.42e6 32550. -1.42e6  -1.13
3 32550. -1.42e6 2018-07-25 14:11:04 2393  32550. -1.42e6 32552. -1.42e6   2.57
4 32552. -1.42e6 2018-07-25 15:12:01 2393  32552. -1.42e6 32556. -1.42e6   3.67
5 32556. -1.42e6 2018-07-25 16:10:49 2393  32556. -1.42e6 32551. -1.42e6  -4.85
6 32551. -1.42e6 2018-07-25 17:12:20 2393  32551. -1.42e6 32547. -1.42e6  -4.18
# i 30 more variables: y2_cent <dbl>, t2_ <dttm>, t_diff <drtn>, hour_t1 <int>,
#   yday_t1 <dbl>, hour_t2 <int>, hour_t2_sin <dbl>, hour_t2_cos <dbl>,
```

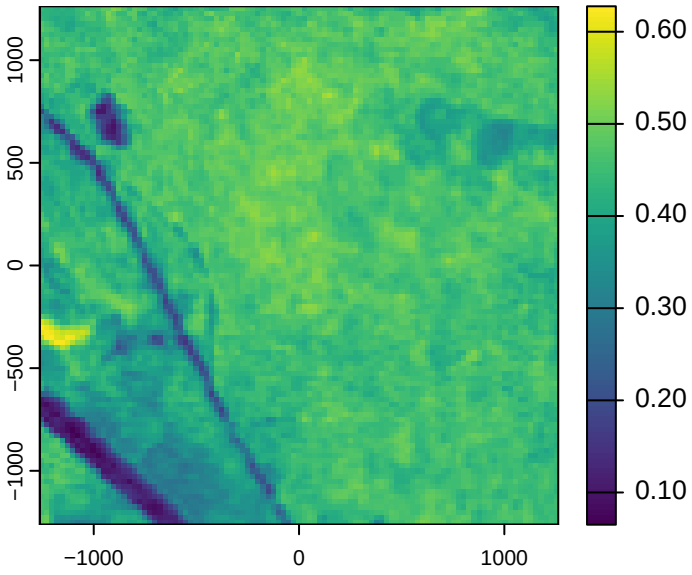
```
#   yday_t2 <dbl>, yday_t2_sin <dbl>, yday_t2_cos <dbl>, sl <dbl>,
#   log_sl <dbl>, bearing <dbl>, bearing_sin <dbl>, bearing_cos <dbl>,
#   ta <dbl>, cos_ta <dbl>, x_min <dbl>, x_max <dbl>, y_min <dbl>, y_max <dbl>,
#   extent_00centre <list>, ndvi_index <int>, ndvi_cent <list>,
#   veg_herby_cent <list>, canopy_cover_cent <list>, slope_cent <list>, ...
```

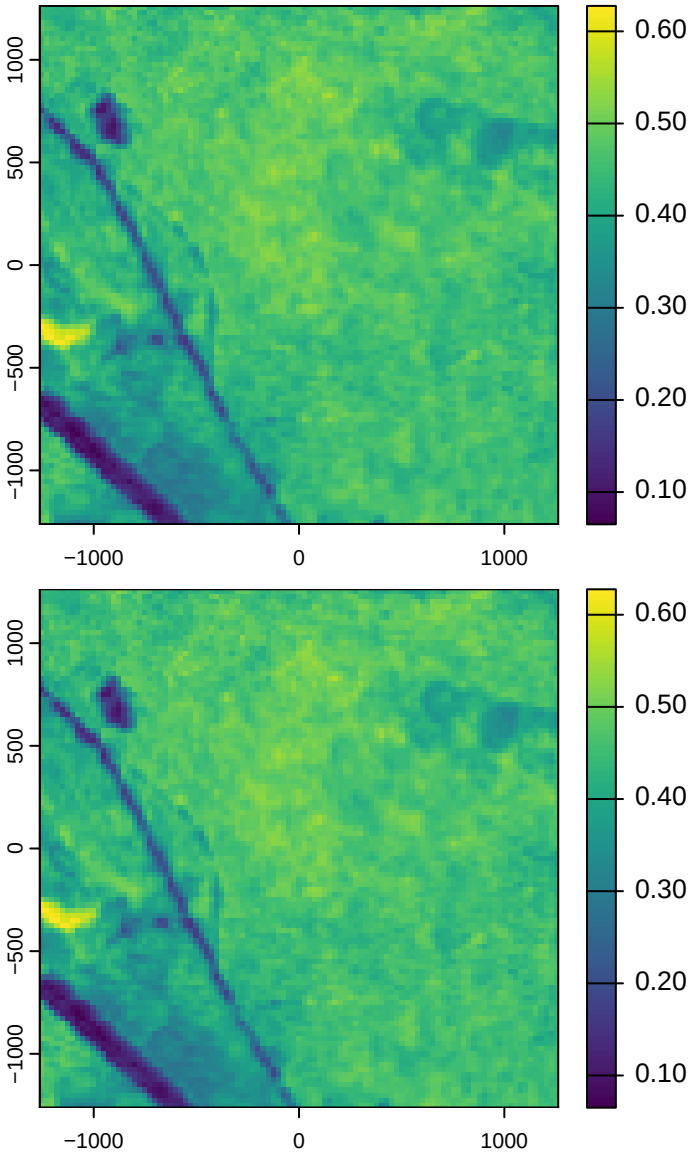
```
which(is.na(buffalo_data_covs$ta))
```

```
integer(0)
```

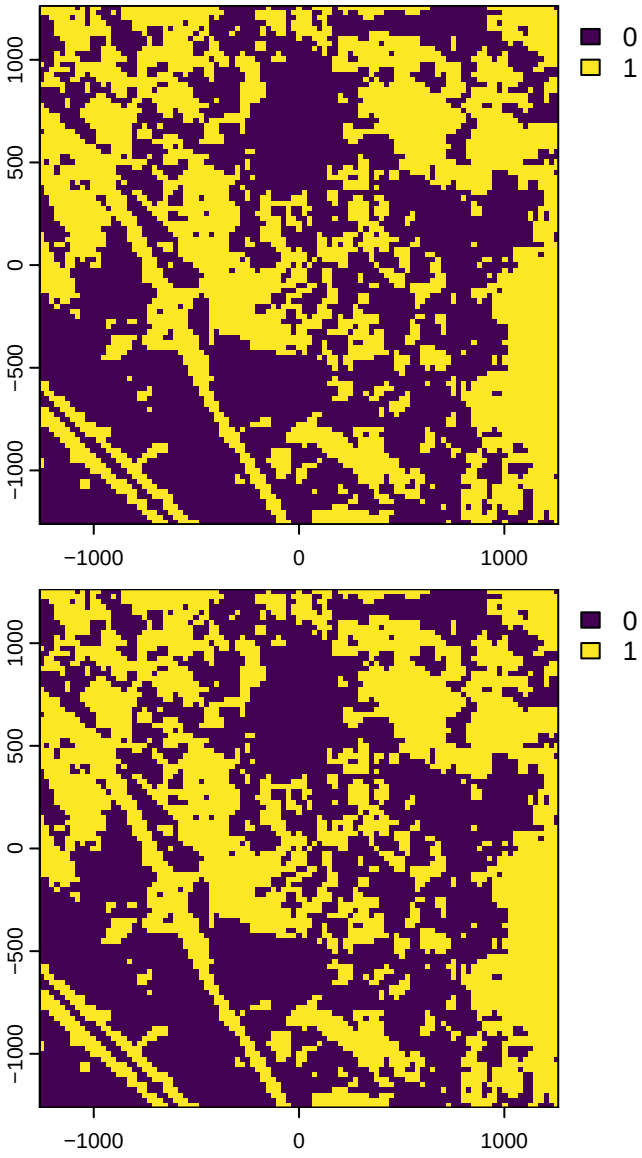
Plot a subset of the local covariates

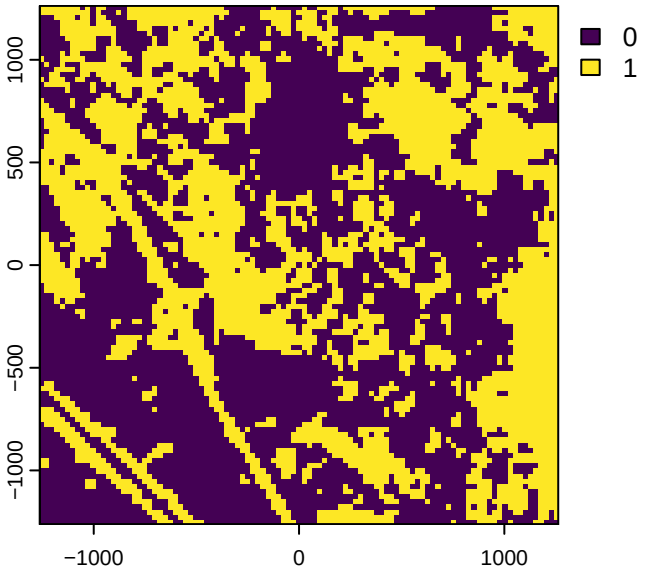
```
n_plots <- 3

# NDVI
walk(buffalo_data_covs$ndvi_cent[1:n_plots], terra::plot)
```



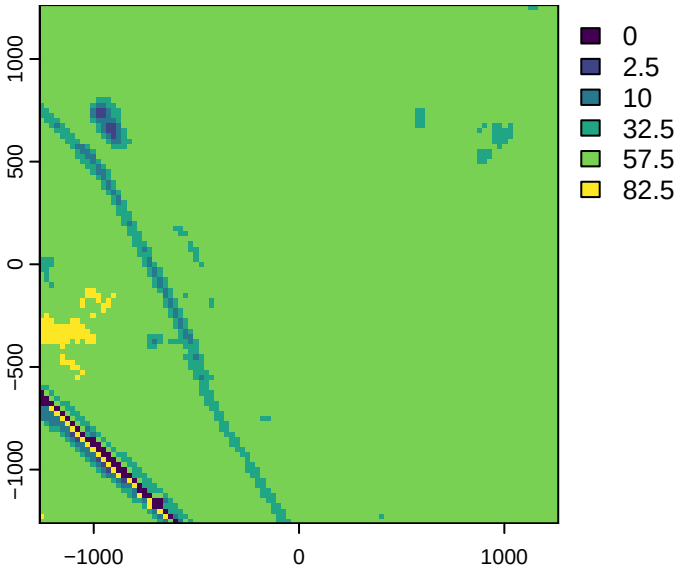


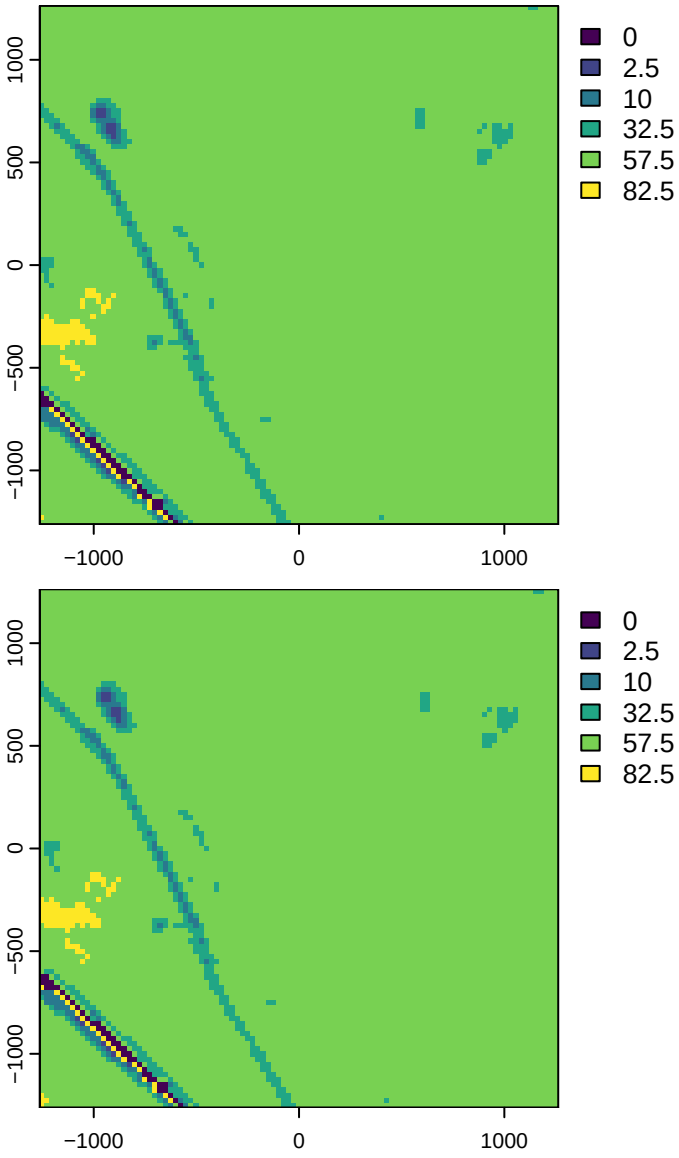

```
# Herbaceous vegetation
walk(buffalo_data_covs$veg_herby_cent[1:n_plots], terra::plot)
```


1000
500
0
−500
−1000
−1000
0
1000
0
1
1000
500
0
−500
−1000
−1000
0
1000
0
1

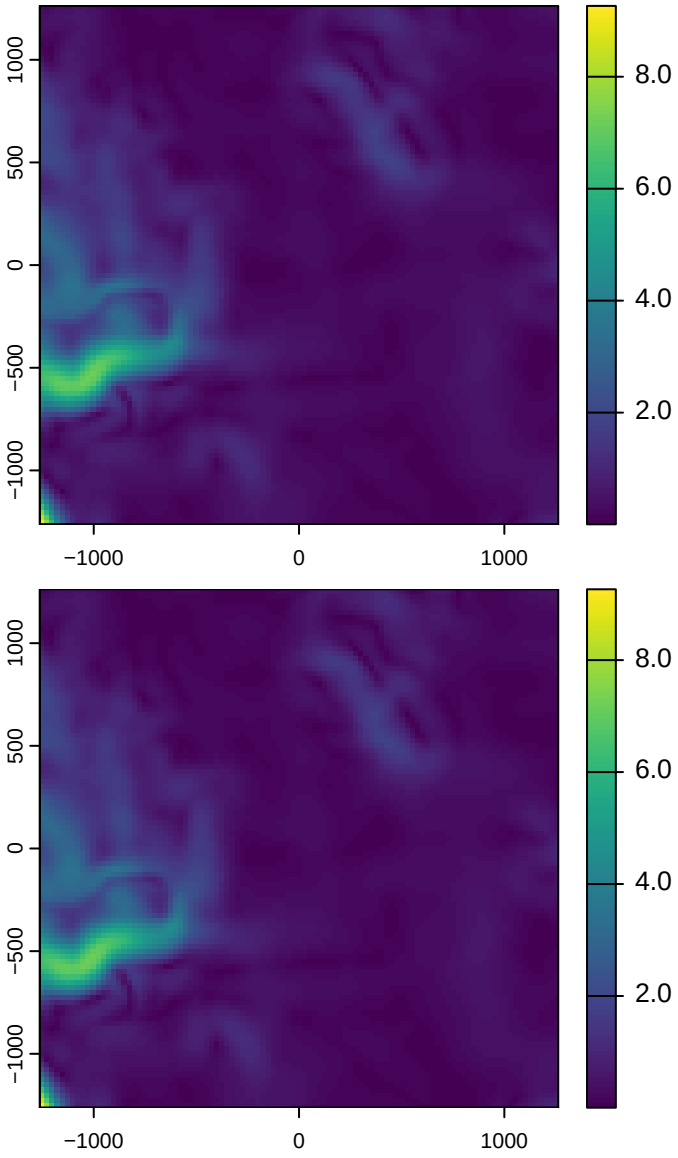


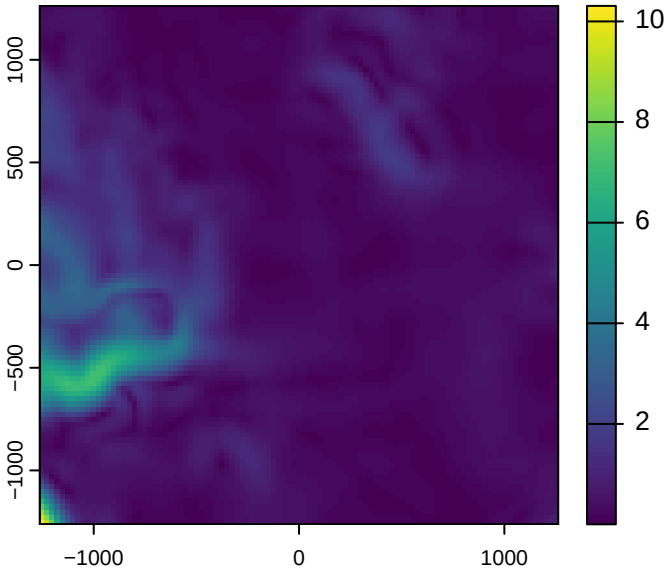

```
# Canopy cover
walk(buffalo_data_covs$canopy_cover_cent[1:n_plots], terra::plot)
```



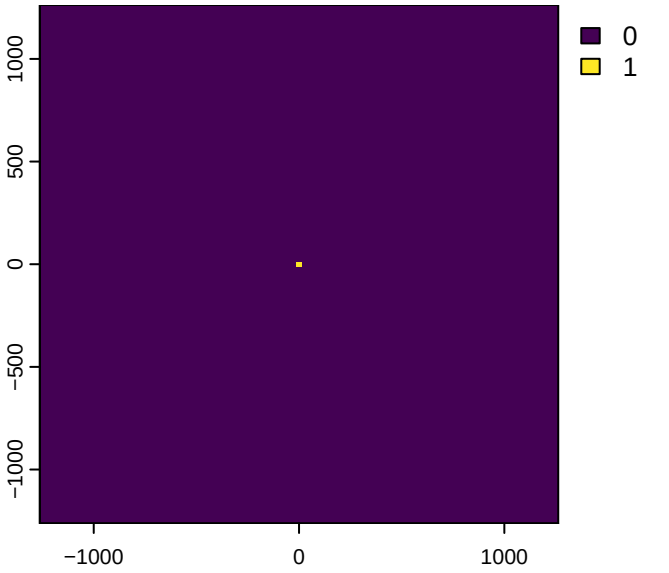


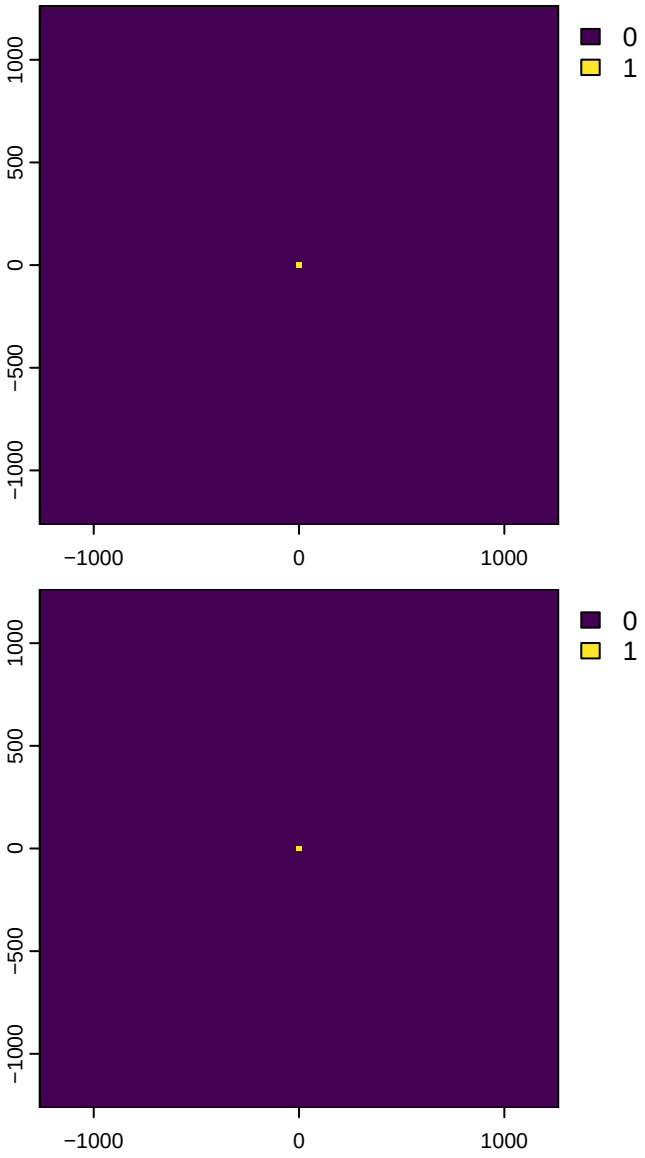

```
# Slope
walk(buffalo_data_covs$slope_cent[1:n_plots], terra::plot)
```


1000
500
0
−500
−1000
−1000
0
1000
8.0
6.0
4.0
2.0
1000
500
0
−500
−1000
−1000
0
1000
8.0
6.0
4.0
2.0




```
# Target (location of the next step)
walk(buffalo_data_covs$pres_cent[1:n_plots], terra::plot)
```





To save the object with all of the local covariates

```
# saveRDS(buffalo_data_covs, paste0("buffalo_data_id_covs_", Sys.Date(), ".rds"))
```

To save some plots

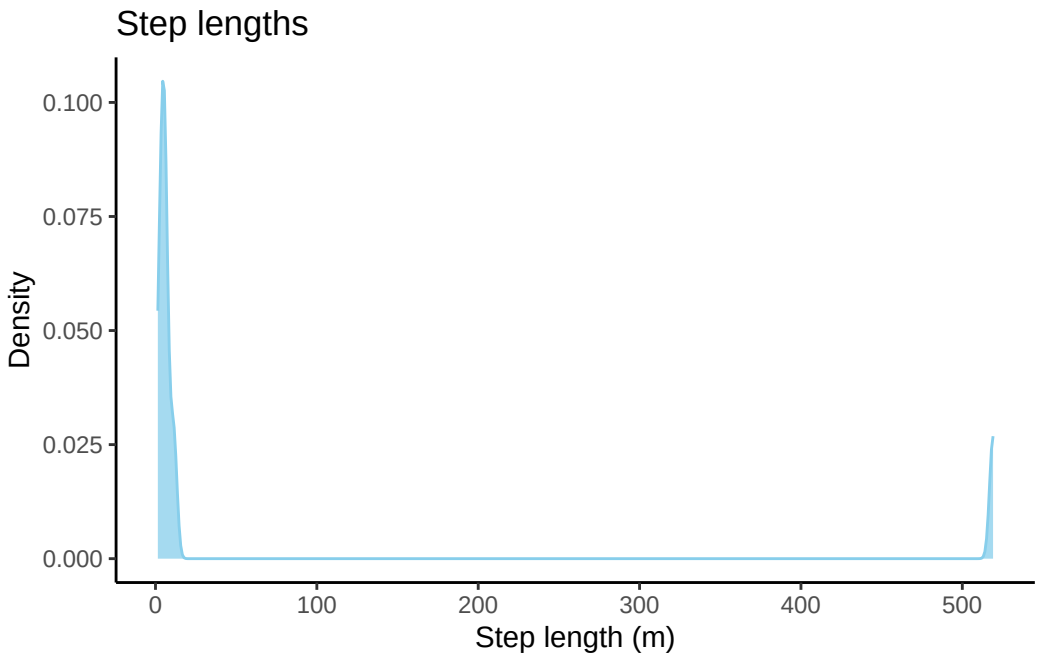
```
# for(i in 1:n_plots) {
#   png(filename = paste0("ndvi_cent_", i, ".png"),
```

```
#       width = 150, height = 150, units = "mm", res = 600)
#   terra::plot(buffalo_data_covs$ndvi_cent[[i]])
#   dev.off()
# }
#
# # for the pres layers
# for(i in 1:n_plots) {
#
#   # change the 0 values to NA
#   layer <- buffalo_data_covs$pres_cent[[i]]
#   layer[layer == 0] <- 0.5
#
#   ndvi_layer <- buffalo_data_covs$ndvi_cent[[i]]
#   new_layer <- layer*ndvi_layer
#
#   # save the plot
#   png(filename = paste0("pres_cent_", i, ".png"),
#       width = 150, height = 150, units = "mm", res = 600)
#   terra::plot(new_layer)
#   dev.off()
# }
#
#
# # other plots
#
# n_plots <- 1
#
# for(i in 1:n_plots) {
#   png(filename = paste0("veg_herby_cent_", i, ".png"),
#       width = 150, height = 150, units = "mm", res = 600)
#   terra::plot(buffalo_data_covs$veg_herby_cent[[i]])
#   dev.off()
# }
#
# for(i in 1:n_plots) {
#   png(filename = paste0("canopy_cover_cent_", i, ".png"),
#       width = 150, height = 150, units = "mm", res = 600)
#   terra::plot(buffalo_data_covs$canopy_cover_cent[[i]])
#   dev.off()
# }
#
# for(i in 1:n_plots) {
#   png(filename = paste0("slope_cent_", i, ".png"),
#       width = 150, height = 150, units = "mm", res = 600)
```

```
#   terra::plot(buffalo_data_covs$slope_cent[[i]])
#   dev.off()
# }
```

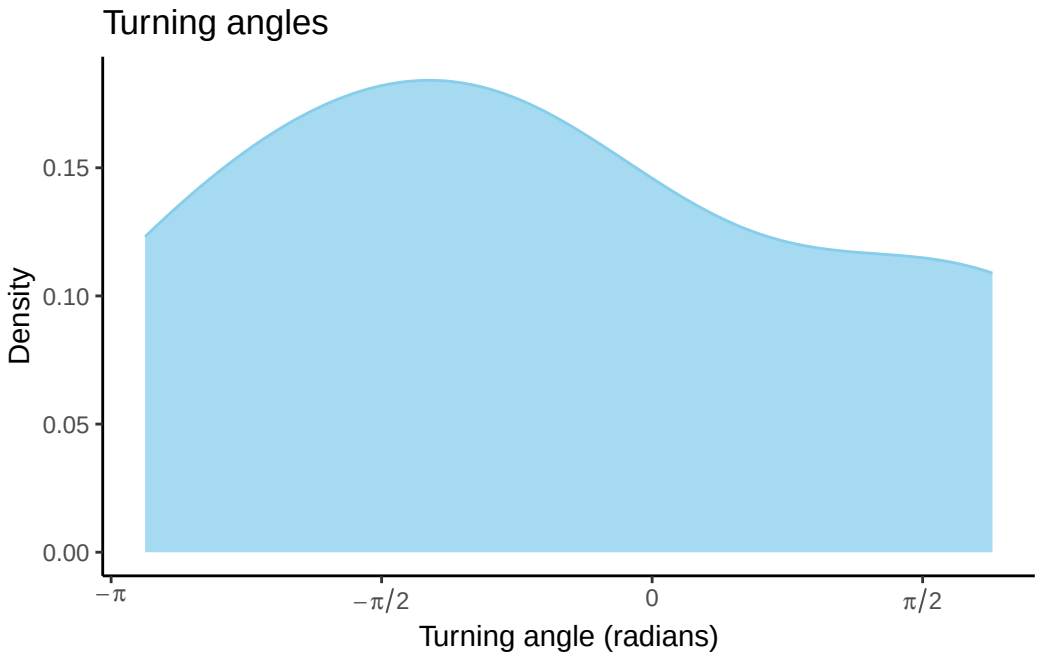
Plot the step lengths and turning angles

```
# plot step lengths
buffalo_data_covs %>% filter(sl < 2000) %>% ggplot() +
  geom_density(aes(x = sl),
                fill = "skyblue", colour = "skyblue", alpha = 0.75) +
  scale_x_continuous("Step length (m)") + #, limits = c(-25, 1250)
  scale_y_continuous("Density") +
  ggtitle("Step lengths") +
  theme_classic()
```




```
# ggsave("outputs/step_length.png", height = 60, width = 120, units = "mm", dpi = 600)


# plot turning angles
buffalo_data_covs %>% ggplot() +
  geom_density(aes(x = ta),
                fill = "skyblue", colour = "skyblue", alpha = 0.75) +
  scale_x_continuous("Turning angle (radians)",
```

```
    breaks = c(-pi, -pi/2, 0, pi/2, pi),
    labels = c(expression(-pi), expression(-pi/2), "0", expression(pi/2), expression(pi))
  ) +
  scale_y_continuous("Density") +
  ggtitle("Turning angles") +
  theme_classic()
```




```
# ggsave("outputs/turning_angle.png", height = 60, width = 120, units = "mm", dpi = 600)
```