Step selection simulations

Generating simulations from a fitted (temporally dynamic) step selection function

Scott Forrest

2024-02-29

In this script we will simulate trajectories from a temporally dynamic step selection function (SSF) fitted to GPS data. The model fitting is done in a separate script, as well as the construction of the temporally varying coefficients from the harmonic terms.

Load packages

```
options(scipen=999)

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

Load a table of temporally varying coefficients (reconstructed from the harmonic terms) from the model fitting script. This has coefficients at 0.1 hour increments, but we are simulating steps at 1 hourly intervals, so we will subset to retain coefficients at 1 hourly intervals.

We are using the coefficients from the model with three pairs of harmonics. To use coefficients from a different model, the only change in the script should be the table of coefficients.

These coefficients should be on the *natural* scale, as they will be multiplied by the habitat covariates in their natural scales. For more info on this see the model fitting script.

We will plot the temporally varying coefficients to check whether everything looks as it should.

```
#Load coefficient estimates from the model fitting script
hourly_coefs <- read_csv("outputs/TwoStep_3pDaily_coefs_dry_2024-02-20.csv")
```

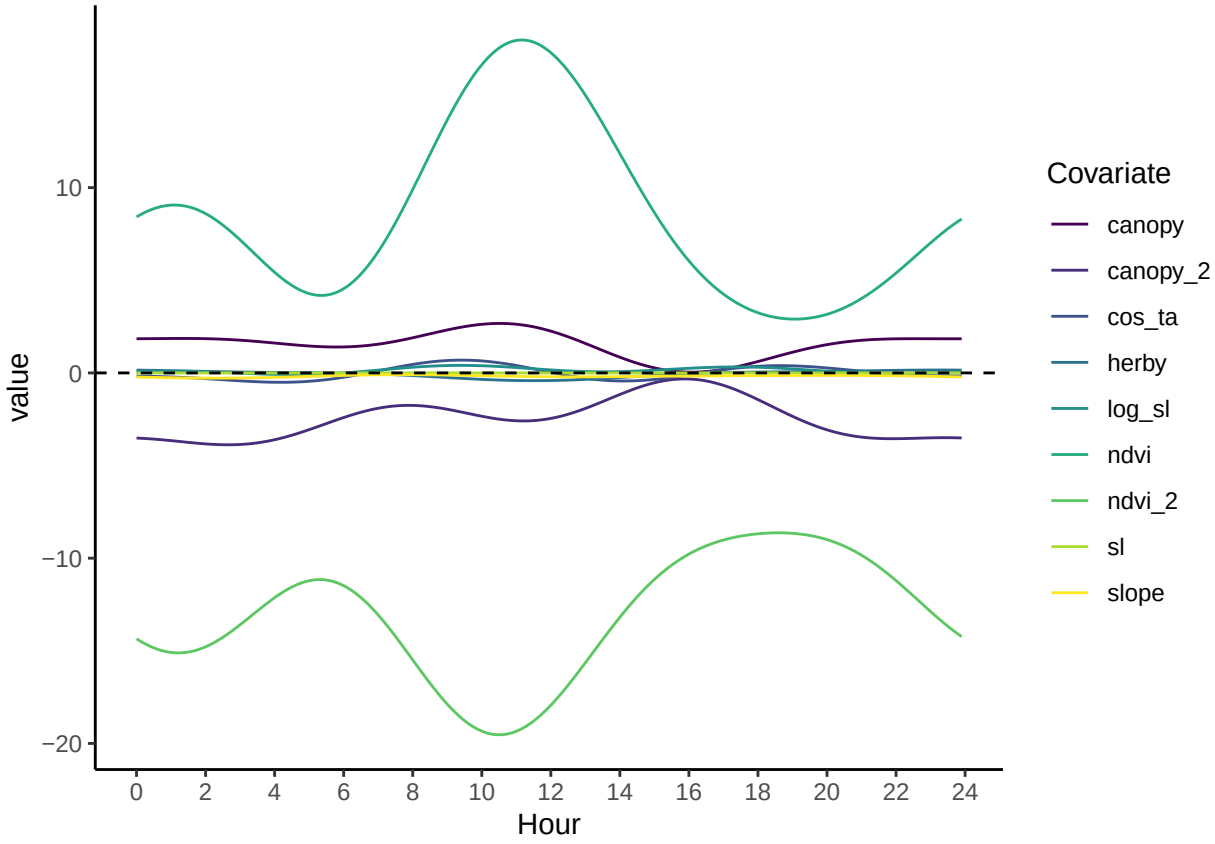
```
## Rows: 240 Columns: 15
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (15): hour, ndvi, ndvi_2, canopy, canopy_2, slope, herby, memory, memory_2, sl, log_sl, cos_ta, s
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(hourly_coefs)
```

```
## # A tibble: 6 x 15
##    hour  ndvi ndvi_2 canopy canopy_2  slope herby    memory memory_2       sl log_sl cos_ta shape sca
##   <dbl> <dbl>  <dbl>  <dbl>    <dbl>  <dbl> <dbl>     <dbl>    <dbl>    <dbl>  <dbl>  <dbl> <dbl> <dbl
## 1   0    8.42  -14.4   1.84    -3.52 -0.223 0.149 21297718. -3.12e13 -0.00169 0.0632 -0.173 0.501  28
## 2   0.1  8.53  -14.5   1.85    -3.53 -0.227 0.147 20905425. -3.00e13 -0.00178 0.0680 -0.176 0.506  27
## 3   0.2  8.62  -14.6   1.85    -3.54 -0.231 0.145 20514573. -2.88e13 -0.00188 0.0725 -0.179 0.511  26
## 4   0.3  8.71  -14.7   1.85    -3.55 -0.235 0.143 20125640. -2.76e13 -0.00198 0.0766 -0.183 0.515  26
```

```
## 5   0.4  8.79  -14.8   1.85    -3.56 -0.239 0.140 19739083. -2.65e13 -0.00207 0.0802 -0.187 0.518  25
## 6   0.5  8.86  -14.8   1.85    -3.57 -0.242 0.137 19355341. -2.53e13 -0.00216 0.0834 -0.191 0.522  24
```

```r
# lengthen data frame for ggplot
hourly_coefs_long <- hourly_coefs %>% pivot_longer(cols = !hour)

# uncomment variables to also plot them, but some of these (particularly memory)
# are on very different scales (as the previous space use density is tiny)
hourly_coefs_long %>%
  filter(!name %in% c("shape", "scale", "kappa", "memory", "memory_2")) %>%
  ggplot(aes(x = hour, y = value, colour = factor(name))) +
  geom_line() +
  scale_colour_viridis_d("Covariate") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  theme_classic()
```



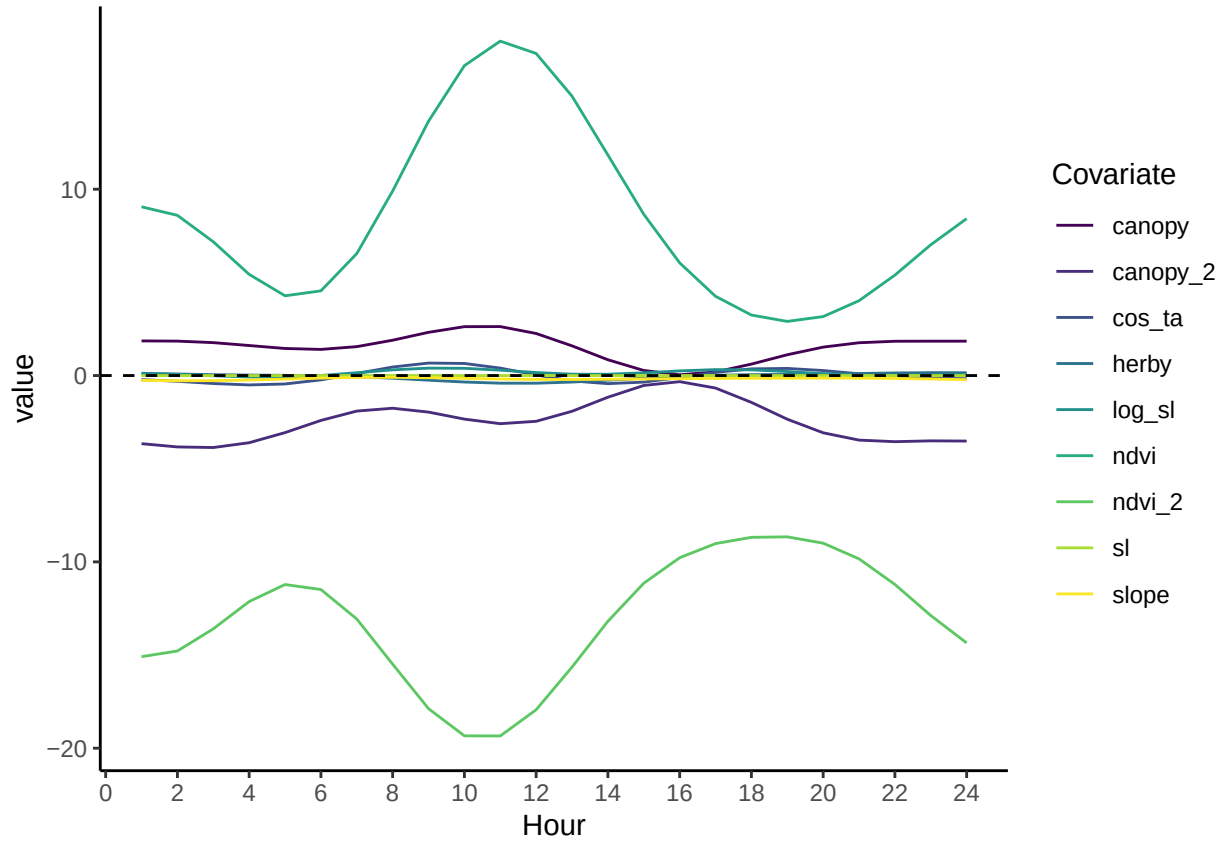
Subset to retain coefficients at 1 hourly intervals. We'll also set hour 0 to be hour 24 (as they are the same thing) so we can index from 1.

Again we will plot the temporally varying coefficients to check whether the subsetting worked correctly.

```r
# keep only the integer hours using the modulo operator, which returns the remainder of the division of
hourly_coefs <- hourly_coefs %>% filter(hourly_coefs$hour %% 1 == 0) %>%
  mutate(hour = ifelse(hour == 0, 24, hour))

head(hourly_coefs)
```

```
## # A tibble: 6 x 15
##    hour  ndvi ndvi_2 canopy canopy_2  slope  herby    memory memory_2       sl  log_sl cos_ta shape s
##   <dbl> <dbl>  <dbl>  <dbl>    <dbl>  <dbl>  <dbl>     <dbl>    <dbl>    <dbl>   <dbl>  <dbl> <dbl> <
## 1    24  8.42  -14.4   1.84    -3.52 -0.223 0.149  21297718. -3.12e13 -0.00169  0.0632 -0.173 0.501
## 2     1  9.06  -15.1   1.86    -3.66 -0.260 0.121  17493049. -2.00e13 -0.00260  0.0910 -0.218 0.529
## 3     2  8.60  -14.8   1.85    -3.83 -0.281 0.0821 14195313. -1.16e13 -0.00321  0.0613 -0.309 0.499
## 4     3  7.18  -13.6   1.77    -3.86 -0.275 0.0475 11769173. -7.39e12 -0.00334 -0.0115 -0.428 0.427
## 5     4  5.43  -12.1   1.61    -3.61 -0.240 0.0252 10666304. -7.72e12 -0.00289 -0.0779 -0.504 0.360
## 6     5  4.28  -11.2   1.45    -3.06 -0.188 0.0103 11431084. -1.28e13 -0.00193 -0.0826 -0.453 0.356
```

```r
# lengthen data frame for ggplot
hourly_coefs_long <- hourly_coefs %>% pivot_longer(cols = !hour)

# uncomment variables to also plot them, but some of these (particularly memory)
# are on very different scales (as the previous space use density is tiny)
hourly_coefs_long %>%
  filter(!name %in% c("shape", "scale", "kappa", "memory", "memory_2")) %>%
  ggplot(aes(x = hour, y = value, colour = factor(name))) +
  geom_line() +
  scale_colour_viridis_d("Covariate") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  scale_x_continuous("Hour", breaks = seq(0,24,2)) +
  theme_classic()
```



Load the memory parameters (to determine previous space use density) from the `Memory_parameter_estimation_` scripts. For the simulations in the paper we used the `Memory_parameter_estimation_loc-subset.Rmd` script. The file that we are reading in contains the **mean** of the KDE bandwidth, $\bar{\sigma}$ (Gaussian kernel

standard deviation), and the temporal exponential decay parameter for the optimisation over **all** individuals, γ_{pop}.

The `Memory_parameter_estimation_` scripts also outputs a csv that has individual-level KDE bandwidths, σ_i, and temporal exponential decay parameters, γ_i where i is an individual buffalo.

The parameters that relate to the model that was fitted should be used - if the individual-level memory parameters were used to estimate the previous space use density and then individual SSFs were fitted, then the individual-level memory parameters should be used in the simulations. If the population-level memory parameters were used to estimate the previous space use density and then a population-level SSF was fitted, then the population-level memory parameters should be used in the simulations.

```
memory_params_ALLoptim <- read_csv("outputs/memory_params_ALLoptim_2024-02-05.csv")
```

```
## New names:
## Rows: 1 Columns: 3
## -- Column specification
## ---------------------------------------------------------------------------------------------------- Deli
## (3): ...1, exp_gamma_all_optim, mean_kde_sd
## i Use `spec()` to retrieve the full column specification for this data. i Specify the column types o
## `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```
memory_params_ALLoptim
```

```
## # A tibble: 1 x 3
##    ...1 exp_gamma_all_optim mean_kde_sd
##   <dbl>               <dbl>       <dbl>
## 1     1             0.00737        572.
```

Import the buffalo starting locations (to start the simulations from). This is simple enough to generate by importing data and selecting the first location, although we import the file here so we don't have to import a large dataset.

```
start_buffalo_df <- read_csv("outputs/buffalo_starting_locations_13inds_2024-02-06.csv")
```

```
## Rows: 13 Columns: 3
## -- Column specification --------------------------------------------------------------------------------
## Delimiter: ","
## dbl (3): id, x, y
##
## 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.
```

```
head(start_buffalo_df)
```

```
## # A tibble: 6 x 3
##      id      x         y
##   <dbl>  <dbl>     <dbl>
## 1  2005 41968. -1435673.
## 2  2014 33615. -1440916.
## 3  2018 30093. -1423073.
## 4  2021 25964. -1437033.
## 5  2022 27928. -1421152.
## 6  2024  8227. -1416224.
```

Import environmental raster layers that the SSF model was fitted with.

```
# unscaled rasters
ndvi_stack <- rast("mapping/cropped rasters/ndvi_GEE_projected_watermask20230207.tif")
canopy <- rast("mapping/cropped rasters/canopy_cover.tif")
herby <- rast("mapping/cropped rasters/veg_herby.tif")
slope <- rast("mapping/cropped rasters/slope_raster.tif")
```

Predictions

The external selection of covariates is denoted by

$$\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha})) = \exp(\beta_1(\tau; \boldsymbol{\alpha}_1)X_1(s_t) + \cdots + \beta_n(\tau; \boldsymbol{\alpha}_n)X_n(s_t)),$$

where $\mathbf{X}(s_t)$ is a vector of covariate values at location s_t, $\boldsymbol{\beta}(\tau; \boldsymbol{\alpha})$ is the vector of temporally varying coefficients, which is a function of time τ, and $\boldsymbol{\alpha}$, which is the vector of parameters defined by the harmonic terms.

To aid the computation of hte simulations, we can precompute $\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha}))$ prior to running the simulations.

In the dataframe of temporally varying coefficients, for each covariate we have reconstructed $\beta_i(\tau; \boldsymbol{\alpha}_i)$ and discretised for each hour of the day, resulting in $\beta_{i,\tau}$ for $i = 1, \ldots, n$ where n is the number of covariates and $\tau = 0, \ldots, 23$.

Given these, we can solve $\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha}))$ for every hour of the day. This will result in a spatial map of the selection of covariates for each hour of the day.

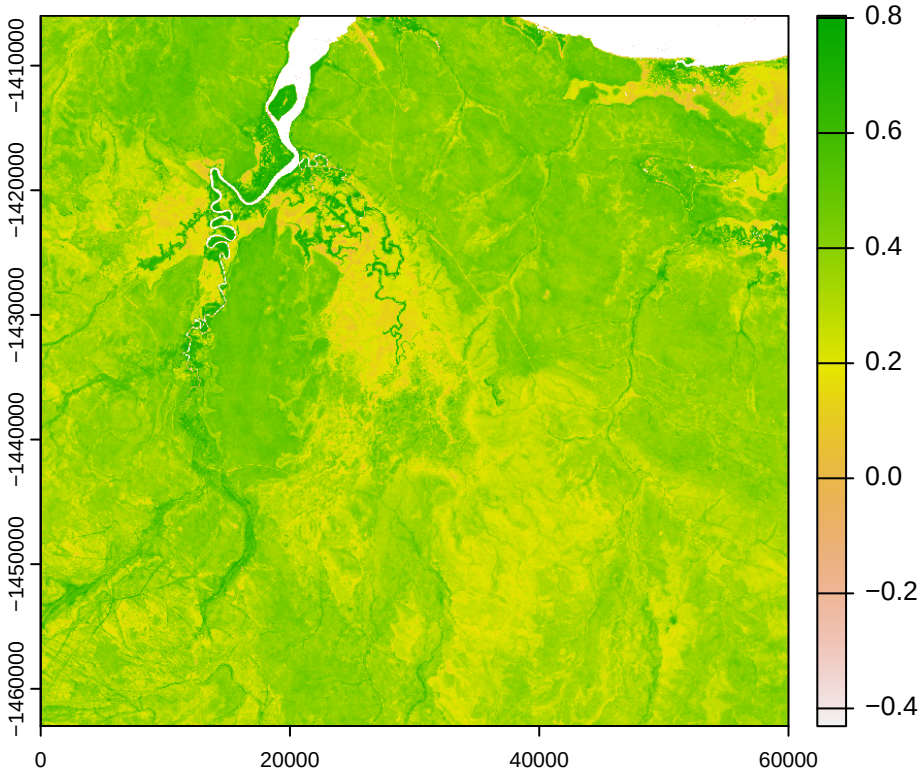
Then, when we do our step selection simulations, we can just subset these maps by the current hour of the day, and extract the values of $\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha}))$ for each proposed step location, rather than solving $\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha}))$ for every step location.

Calculating $\omega(\mathbf{X}(s_t); \boldsymbol{\beta}(\tau; \boldsymbol{\alpha}))$ has been called the 'naive' prediction approach by Signer, Fieberg, and Avgar (2017) as it doesn't account for the movement dynamics, which we will incorporate through the simulations.

Naive prediction approach

As we fitted either wet or dry season models to data that covered 2018 and 2019, we will use the mean of the NDVI for the parts of the 2018 and 2019 dry season that the data covered. We will also square NDVI and canopy cover now, and then it's just a multiplication with the temporally varying coefficient within the function.

```
ndvi_2018_dry <- ndvi_stack[[8:10]]
ndvi_2019_dry <- ndvi_stack[[19:21]]
ndvi <- terra::mean(c(ndvi_2018_dry, ndvi_2019_dry))
names(ndvi) <- "NDVI"
plot(ndvi)
```

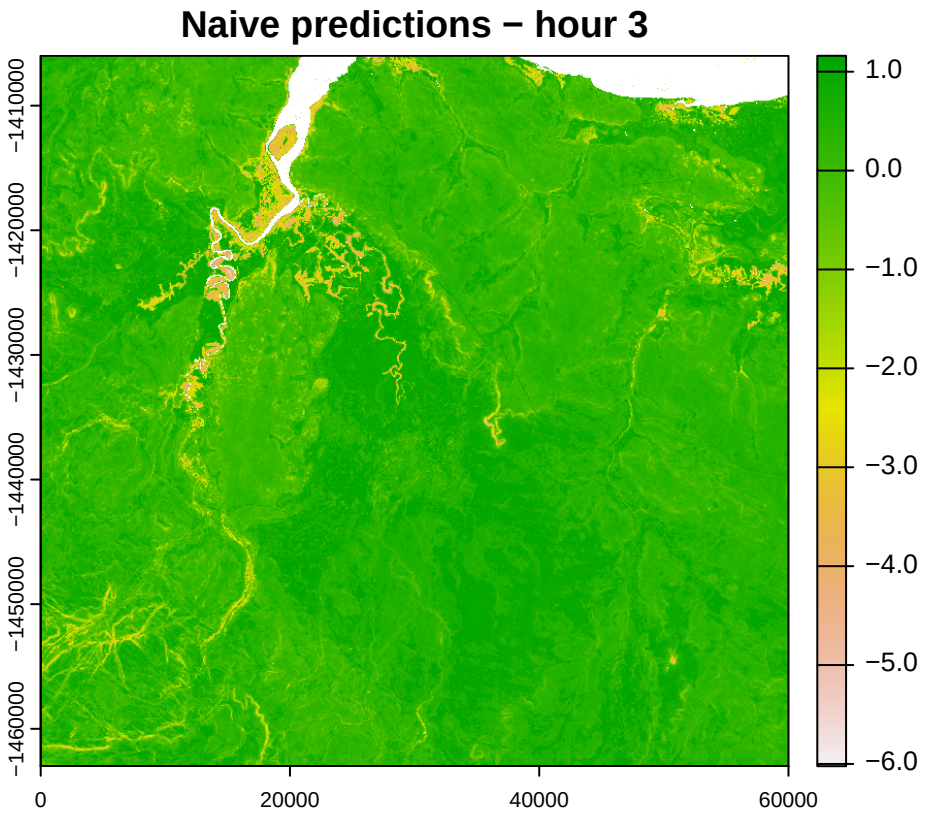



```
# for plotting with ggplot after generating simulations
ndvi_df <- as.data.frame(ndvi, xy = TRUE)
ndvi_df$NDVI_discrete <- cut(ndvi_df$NDVI, breaks=9, dig.lab = 2)

# NDVI squared
ndvi_sq <- ndvi ^ 2

# rescale canopy cover to 0 - 1 (as this was what the model was fitted to)
canopy01 <- canopy/100
# canopy cover squared
canopy01_sq <- canopy01 ^ 2
```

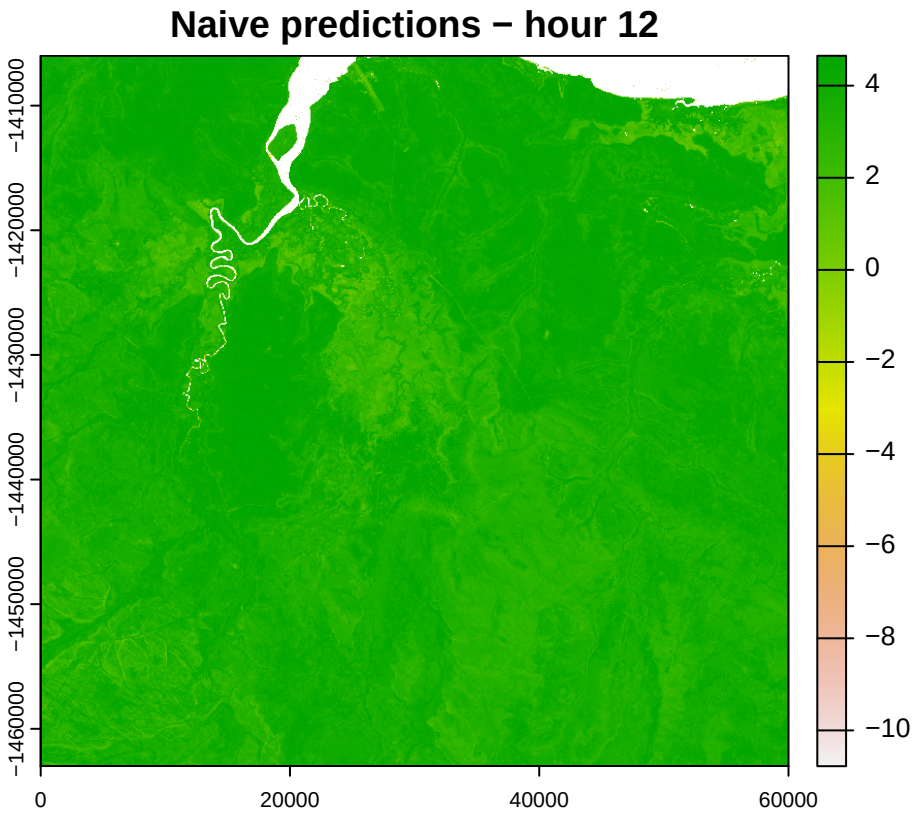
Generating the 'naive' habitat selection layers for stochastic simulations

```
# creaty empty objects to store the layers
naive_pred_stack <- c()

tic("Naive predictions")

# for each hour of the day
for(hour_no in 1:24) {

  # create a raster stack of the covariates (including the squared terms)
  resources <- c(ndvi,
                 ndvi_sq,
                 canopy01,
                 canopy01_sq,
```

```
                slope,
                herby
                )
  # ndvi
  # using the linear term
  ndvi_lin <- resources[[1]] *
    hourly_coefs$ndvi[[which(hourly_coefs$hour == hour_no)]]
  # using the quadratic term
  ndvi_quad <- resources[[2]] *
    hourly_coefs$ndvi_2[[which(hourly_coefs$hour == hour_no)]]
  # combining
  ndvi_pred <- ndvi_lin + ndvi_quad

  # canopy cover
  # using the linear term
  canopy_lin <- resources[[3]] *
    hourly_coefs$canopy[[which(hourly_coefs$hour == hour_no)]]
  # using the quadratic term
  canopy_quad <- resources[[4]] *
    hourly_coefs$canopy_2[[which(hourly_coefs$hour == hour_no)]]
  # combining
  canopy_pred <- canopy_lin + canopy_quad

  # herby
  slope_lin <- resources[[5]]
  slope_pred <- slope_lin *
    hourly_coefs$slope[[which(hourly_coefs$hour == hour_no)]]

  # herby
  herby_lin <- resources[[6]]
  herby_pred <- herby_lin *
    hourly_coefs$herby[[which(hourly_coefs$hour == hour_no)]]

  # combining all covariates (but not exponentiating yet)
  naive_pred <- ndvi_pred + canopy_pred + slope_pred + herby_pred

  # adding to the list of rasters for each hour
  naive_pred_stack <- c(naive_pred_stack, naive_pred)

}

# plot some example naive predictions
plot(naive_pred_stack[[3]], main = "Naive predictions - hour 3")
```




```
plot(naive_pred_stack[[12]], main = "Naive predictions - hour 12")
```




```
# if we want to save the stack to reimport later so we don't have to recompute
filename <- paste0("mapping/CLR_TS_2pDailyMemALL_naive_unnorm_hourly_",
                   Sys.Date(), ".tif")
# filename
terra::writeRaster(rast(naive_pred_stack), filename, overwrite = TRUE)

toc()
```

```
## Naive predictions: 111.49 sec elapsed
```

```
beep(sound = 2)
```

Cropping the predictions and setting the origin

If using a 'wrapped' or 'torus' boundary, then the origin should be set to (0,0) as we'll use the modulo operator (which is `%%` in R) to do the wrapping. We will set it back to it's original coordinates after running the simulations. To be consistent we will just always set the origin to (0,0).

We can also crop layers at this point if we want a smaller extent. Just change the `xmin`, `xmax`, `ymin`, `ymax` values. Here we won't crop, so we just pull the extent values from the layer itself, but we do use these values (actually just `xmin` and `ymin`) to set the origin.

```
# convert the list of naive prediction rasters to a raster stack
naive_predictions <- rast(naive_pred_stack)
# set the time of the rasters - here we are only interested in the hour,
# so we just set an arbitrary date. Although if predicting over seasons for
# instance, then set the date with the correct increment
terra::time(naive_predictions) <- ymd_hm("2018-08-01 00:00",
```

```
                                        tz = "Australia/Queensland") + hours(1:24)

terra::time(naive_predictions)
```

```
##  [1] "2018-08-01 01:00:00 AEST" "2018-08-01 02:00:00 AEST" "2018-08-01 03:00:00 AEST" "2018-08-01 04
##  [5] "2018-08-01 05:00:00 AEST" "2018-08-01 06:00:00 AEST" "2018-08-01 07:00:00 AEST" "2018-08-01 08
##  [9] "2018-08-01 09:00:00 AEST" "2018-08-01 10:00:00 AEST" "2018-08-01 11:00:00 AEST" "2018-08-01 12
## [13] "2018-08-01 13:00:00 AEST" "2018-08-01 14:00:00 AEST" "2018-08-01 15:00:00 AEST" "2018-08-01 16
## [17] "2018-08-01 17:00:00 AEST" "2018-08-01 18:00:00 AEST" "2018-08-01 19:00:00 AEST" "2018-08-01 20
## [21] "2018-08-01 21:00:00 AEST" "2018-08-01 22:00:00 AEST" "2018-08-01 23:00:00 AEST" "2018-08-02 00
```

```
hour(time(naive_predictions))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23  0
```

```
# full extent
xmin <- ext(naive_predictions)[1]
xmax <- ext(naive_predictions)[2]
ymin <- ext(naive_predictions)[3]
ymax <- ext(naive_predictions)[4]
crop_extent <- ext(xmin, xmax, ymin, ymax)

# if we wanted to set the extent just around the buffalo data (with a buffer)
# buffer = 5000 # in metres
# xmin <- round(min(buffalo_data$x2), digits = -2) - buffer
# xmax <- round(max(buffalo_data$x2), digits = -2) + buffer
# ymin <- round(min(buffalo_data$y2), digits = -2) - buffer
# ymax <- round(max(buffalo_data$y2), digits = -2) + buffer
# crop_extent <- ext(xmin, xmax, ymin, ymax)

# to create a smaller extent centred around the centre point
# x_centrepoint <- (xmin+xmax)/2
# y_centrepoint <- (ymin+ymax)/2
# cell_size <- 25
# width <- 500 # number of cells
# height <- 500 # number of cells
# xmin <- x_centrepoint - (cell_size * width/2)
# xmax <- x_centrepoint + (cell_size * width/2)
# ymin <- y_centrepoint - (cell_size * height/2)
# ymax <- y_centrepoint + (cell_size * height/2)
# crop_extent <- ext(xmin, xmax, ymin, ymax)

# crop the naive predictions
naive_predictions_cropped <- terra::crop(naive_predictions, crop_extent)
# set the origin at (0,0)
ext(naive_predictions_cropped) <- c(xmin - xmin, xmax - xmin,
                                    ymin - ymin, ymax - ymin)
```

Setting up the simulation function

There are several parts to the memory process as it needs to 'warm-up' so there are enough locations to start using as the memory. This part of the trajectory is then discarded.

```r
simulate_ssf_memory <- function(
    n_steps, # final number of steps in the trajectory
    n_ch, # number of proposed steps
    coef, # table of temporally varying coefficient values
    xy0, # starting location
    starting_time, # time of the first step in POSIXct format
    step_time_increment, # time increment between steps in minutes
    resc, # resources - naive predictions
    # boundary = "wrapped", # boundary type (either "wrapped" or "reflective")
    memory_period, # number of locations to be included in the temporal decay
    memory_delay, # number of locations to exclude prior to the current step
    spatial_sd, # KDE bandwidth parameter for estimating previous space use
    exp_gamma # temporal decay parameter for estimating previous space use
    ) {

  tic()

  # vectorisation of the random sampling leads to iterating over hourly coef values
  sl <- rgamma(n_steps * n_ch,
               shape = rep(coef$shape, each = n_ch),
               scale = rep(coef$scale, each = n_ch))

  # create a vector of the turning angle coefficients (over the full trajectory)
  ta_coefs <- rep(coef$kappa, each = n_ch, length.out = n_steps)

  # if positive, mean = 0 (which is rvonmises(mu = pi) - pi),
  # indicating persistent steps in a similar direction,
  # otherwise mean = pi (which is rvonmises(mu = 0) - pi),
  # indicating reversal steps
  ta_coefs_mu <- ifelse(ta_coefs > 0, pi, 0)

  # sample turning angles with the different kappa and mean values
  ta <- as.vector(mapply(Rfast::rvonmises,
                         n = n_ch,
                         m = ta_coefs_mu,
                         k = abs(ta_coefs))) - pi

  # setup the simulation
  steps <- rep(1:n_steps, each = n_ch)
  step_time <- starting_time + minutes((0:(n_steps-1))*step_time_increment)

  # set the starting location and (random) angle
  x_0 <- xy0[[1]]
  y_0 <- xy0[[2]]
  angle_0 <- runif(1, min = -pi, max = pi)

  # create vectors to store the results
  x <- rep(NA, n_steps)
  y <- rep(NA, n_steps)
  hour <- rep(NA, n_steps)
  step_length <- rep(NA, n_steps)
  angle <- rep(NA, n_steps)
  bearing <- rep(NA, n_steps)
```

```r
hab_p <- rep(NA, n_steps)
mem_p <- rep(NA, n_steps)

# set initial values
x[1] <- x_0
y[1] <- y_0
hour[1] <- 1
step_length[1] <- 0
angle[1] <- angle_0
bearing[1] <- 0
hab_p[1] <- 0
mem_p[1] <- 0

for (i in 2:n_steps) {

  # if you want the option for the boundary to be wrapped or reflective,
  # uncomment the following lines, and then specify in the function arguments
  # if(boundary == "wrapped") {

  # wrapped boundary
  # adding the modulo operator %% ensures a wrapped/toroid landscape,
  # but the origin must be at (0,0)
  # bearing <- atan2(y[i - 2] - y[i - 1], x[i - 2] - x[i - 1])
  #
  # x_prop <- (x[i - 1] + sl[steps == i] * cos(bearing[i - 1] +
  #                                              ta[steps == i])) %% ext(resc)[2]
  #
  # y_prop <- (y[i - 1] + sl[steps == i] * sin(bearing[i - 1] +
  #                                              ta[steps == i])) %% ext(resc)[4]
  #
  # angle_prop <- ta[steps == i]
  # bearing_prop <- atan2(y_prop - y[i - 1], x_prop - x[i - 1])

  # } else if(boundary == "reflective") {
  #
  # # reflective boundary

  # proposed step in the x-direction
  x_prop <- (x[i - 1] + sl[steps == i] * cos(bearing[i - 1] + ta[steps == i]))
  # proposed step in the y-direction
  y_prop <- (y[i - 1] + sl[steps == i] * sin(bearing[i - 1] + ta[steps == i]))
  # proposed step length
  sl_prop <- sl[steps == i]
  # proposed turning angle
  angle_prop <- ta[steps == i]
  # proposed bearing (required for imposing correlated movement,
  # as the turning angle of the following step is relative to the bearing)
  bearing_prop <- atan2(y_prop - y[i - 1], x_prop - x[i - 1])
  # hour of the proposed step, which could be day, month, etc for other time scales
  hour_prop <- lubridate::hour(step_time[i])

  #
  # } else {
```

```
    #   print("Specify either 'reflective' (hard boundary) or
    #   'wrapped' (toroidal) for the boundary")
    # }

    # sampling environmental suitability from the naive prediction rasters
    # pull out the relevant raster based on the time and extract values from it
    # these values will be exponentiated after added to the log-memory probability
    p <- terra::extract(
      resc[[which(lubridate::hour(terra::time(naive_predictions)) == hour_prop)]],
      cbind(x_prop, y_prop))[,1]


    ### memory process ###

    # the coefficients (including memory process) for hour span from 1 to 24
    # rather than 0 to 23, so we'll update the hour_prop index to 24 if it's 0,
    # as hour 0 and hour 24 are the same
    hour_prop_mem <- ifelse(hour_prop == 0, 24, hour_prop)

    # if the memory period is less than 2x the memory delay,
    # then all steps are included and there is no memory delay
    # (e.g. excluding the past 24 hours)
    if(sum(!is.na(x)) < 2*memory_delay) {

      memory_density <- mapply(function(x_prop, y_prop) {

        # subset all previous steps
        recent_x <- x[1:(i-1)]
        recent_y <- y[1:(i-1)]

        # calculate the distance and time between the current and previous steps
        diff_x <- x_prop - recent_x
        diff_y <- y_prop - recent_y
        diff_time <- length(recent_x):1

        # calculate the memory density
        # the logSumExp function is used to avoid numerical underflow and overflow
        # we calculate all components (i.e. the x and y spatial and temporal components)
        # in the log-scale
        # we normalise by the temporal decay component
        return(logSumExp(dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
                           dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
                           (-exp_gamma*diff_time), na.rm = TRUE) -
                 logSumExp(-exp_gamma*diff_time, na.rm = TRUE))

      }, x_prop, y_prop) # pass current locations in here down

      # calculate the memory probability by multiplying the density
      # (and it's square) by the coefficients
# for calculations it is
# better to use the log-scale above,
# so we exponentiate here to get it back to the natural scale (which was the
# scale of our covariate when fitting the model)
```

```
# this should not be confused with exponentiating the get the external
# selection probability, which we do below after adding the habitat suitability
# from the naive predictions
memory_p <- coef$memory[hour_prop_mem] * exp(memory_density) +
  coef$memory_2[hour_prop_mem] * exp(memory_density)^2


# if the memory period is greater than 2x the memory delay,
# but less than the memory period, then all steps are considered,
# but there is a memory delay (e.g. excluding the past 24 hours)

} else if(sum(!is.na(x)) <= memory_period) {

  memory_density <- mapply(function(x_prop, y_prop) {

    # subset all previous steps, but exclude the memory delay
    recent_x <- x[1:(i-memory_delay)]
    recent_y <- y[1:(i-memory_delay)]

    # calculate the distance and time between the current and previous steps
    diff_x <- x_prop - recent_x
    diff_y <- y_prop - recent_y
    diff_time <- length(recent_x):1

    # calculate the memory density
    # the logSumExp function is used to avoid numerical underflow and overflow
    # we calculate all components (i.e. the x and y spatial and temporal components)
    # in the log-scale
    # we normalise by the temporal decay component
    return(logSumExp(dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
                       dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
                       (-exp_gamma*diff_time)
                     , na.rm = TRUE) -
             logSumExp(-exp_gamma*diff_time, na.rm = TRUE))

  }, x_prop, y_prop) # pass current locations in here down

  # calculate the memory probability by multiplying the density
  # (and it's square) by the coefficients
  # for calculations it is
  # better to use the log-scale above,
  # so we exponentiate here to get it back to the natural scale (which was the
  # scale of our covariate when fitting the model)
  # this should not be confused with exponentiating the get the external
  # selection probability, which we do below after adding the habitat suitability
  # from the naive predictions
  memory_p <- coef$memory[hour_prop_mem] * exp(memory_density) +
    coef$memory_2[hour_prop_mem] * exp(memory_density)^2

  # if the memory period is greater than the memory period,
  #then only the memory period steps are considered (e.g. excluding the past 24 hours)
} else if(sum(!is.na(x)) > memory_period) {
```

```r
  memory_density <- mapply(function(x_prop, y_prop) {

    # now we subset from the start of the memory period to the memory delay,
    # i.e. a maximum of memory_period - memory_delay steps
    recent_x <- x[(i-memory_period):(i-memory_delay)]
    recent_y <- y[(i-memory_period):(i-memory_delay)]

    # calculate the distance and time between the current and previous steps
    diff_x <- x_prop - recent_x
    diff_y <- y_prop - recent_y
    diff_time <- length(recent_x):1

    # calculate the memory density
    # the logSumExp function is used to avoid numerical underflow and overflow
    # we calculate all components (i.e. the x and y spatial and temporal components)
    # in the log-scale
    # we normalise by the temporal decay component
    return(logSumExp(dnorm(diff_x, mean = 0, sd = spatial_sd, log = TRUE) +
                       dnorm(diff_y, mean = 0, sd = spatial_sd, log = TRUE) +
                       (-exp_gamma*diff_time)
                     , na.rm = TRUE) -
             logSumExp(-exp_gamma*diff_time, na.rm = TRUE))

  }, x_prop, y_prop) # pass current locations in here down

  # calculate the memory probability by multiplying the density
  # (and it's square) by the coefficients
  # for calculations it is
  # better to use the log-scale above,
  # so we exponentiate here to get it back to the natural scale (which was the
  # scale of our covariate when fitting the model)
  # this should not be confused with exponentiating the get the external
  # selection probability, which we do below after adding the habitat suitability
  # from the naive predictions
  memory_p <- coef$memory[hour_prop_mem] * exp(memory_density) +
    coef$memory_2[hour_prop_mem] * exp(memory_density)^2

} else {
  print("Out of specified memory conditions - check memory arguments")
}

# if the probability of either the habitat suitability probability or
# memory probability is NA (i.e. if falls outside the extent when using
# the reflective boundary, or in non-habitat such as water),
# then set it to a very low value (prior to exponenitating, e.g. -50)
p[is.na(p)] <- -50
memory_p[is.na(memory_p)] <- -50

# calculate the total probability of the proposed step and
# sample one of the locations based on this probability
w <- sample(n_ch, 1, prob = exp(memory_p + p))

# store all the information for the proposed step
```

```
x[i] <- x_prop[w]
y[i] <- y_prop[w]
hour[i] <- hour_prop
step_length[i] <- sl_prop[w]
angle[i] <- angle_prop[w]
bearing[i] <- bearing_prop[w]
hab_p[i] <- p[w]
mem_p[i] <- memory_p[w]

}
(toc())
return(data_frame(x = x,
                  y = y,
                  t = step_time,
                  hour = hour,
                  sl = step_length,
                  angle = angle,
                  bearing = bearing,
                  hab_p = hab_p,
                  mem_p = mem_p))
}
```

Setup starting locations

```
# for the starting points of each individual buffalo
start_buffalo <- matrix(c(start_buffalo_df$x - xmin,
                          start_buffalo_df$y - ymin),
                        ncol = 2, nrow = nrow(start_buffalo_df))

# for random starting locations
start_unif <- cbind(runif(1e3,
                          min = ext(naive_predictions_cropped[[1]])[1],
                          max = ext(naive_predictions_cropped[[1]])[2]),
                    runif(1e3,
                          min = ext(naive_predictions_cropped[[1]])[3],
                          max = ext(naive_predictions_cropped[[1]])[4]))
```

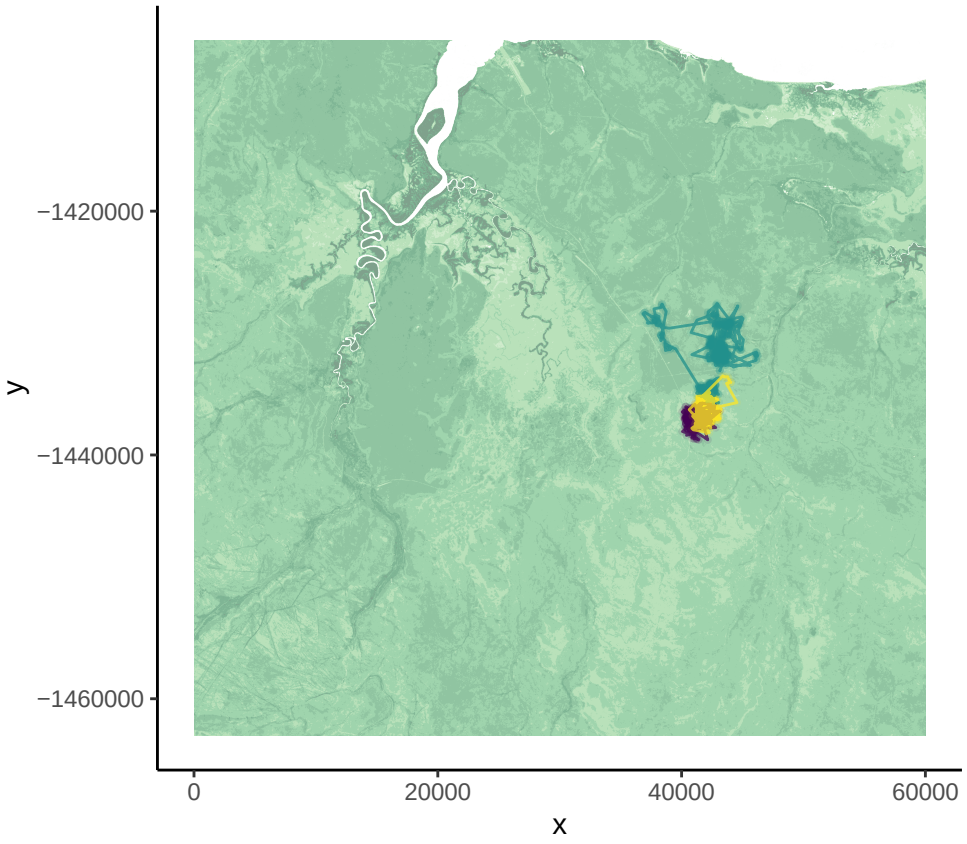
Simulating trajectories

```
n_indvs <- 3 # number of individual trajectories per starting location
n_steps <- 1000 # number of steps in the trajectory
n_ch <- 50 # number of proposed steps to be selected from
coef <- hourly_coefs  # hourly coefficients
resc <- naive_predictions_cropped  # resource selection (naive) raster stack
# most of the memory has declined to 0 by 500 locations
memory_period <- 500
memory_delay <- 24
spatial_sd <- memory_params_ALLoptim$mean_kde_sd
exp_gamma <- memory_params_ALLoptim$exp_gamma_all_optim

for(k in 1:length(start_buffalo_df$id)) {

  stps_mem_list <- vector(mode = "list", length = n_indvs)
```

```
tic()

for(j in 1:n_indvs) {

  stps_mem_list[[j]] <- simulate_ssf_memory(
    n_steps = n_steps,
    n_ch = n_ch,
    coef = hourly_coefs,
    xy0 = start_buffalo[k,],
    starting_time = ymd_hm("2018-07-01 01:00", tz = "Australia/Queensland"),
    step_time_increment = 60,
    resc = resc,
    memory_period = memory_period,
    memory_delay = memory_delay,
    # the KDE bandwidth (spatial sd) and temporal decay (exp_gamma) parameters
    # here are the same for all individuals, but they could be indexed over
    # a table of individual-specific parameters
    spatial_sd = spatial_sd,
    exp_gamma = exp_gamma)
}

toc()

animals_mem <- data_frame(
  id = paste0("id", start_buffalo_df$id[k], 1:n_indvs),
  track = map(stps_mem_list, ~ amt::track(
    x = .$x,
    y = .$y,
    # we set time column to start at - which should allow enough time to
    # exclude the warm-up period (e.g. 500 locations)
    t = ymd_hm("2018-07-01 00:00") + hours(1:nrow(.)),
    hour = .$hour,
    angle = .$angle,
    bearing = .$bearing,
    hab_p = .$hab_p,
    mem_p = .$mem_p
  )))

sim_data_mem <- unnest(animals_mem, cols = track)

#check that the time lines up to the hour
head(sim_data_mem)

# return the locations to the original scale
#(i.e. add the xmin and ymin back to the x and y coordinates)
sim_data_mem_truexy <- sim_data_mem %>% mutate(x_ = x_ + xmin, y_ = y_ + ymin)

print(ggplot() +
        geom_raster(data = ndvi_df,
                    aes(x = x, y = y, fill = NDVI_discrete),
                    alpha = 0.5) +
        geom_point(data = sim_data_mem_truexy,
                   aes(x = x_, y = y_, colour = id),
```

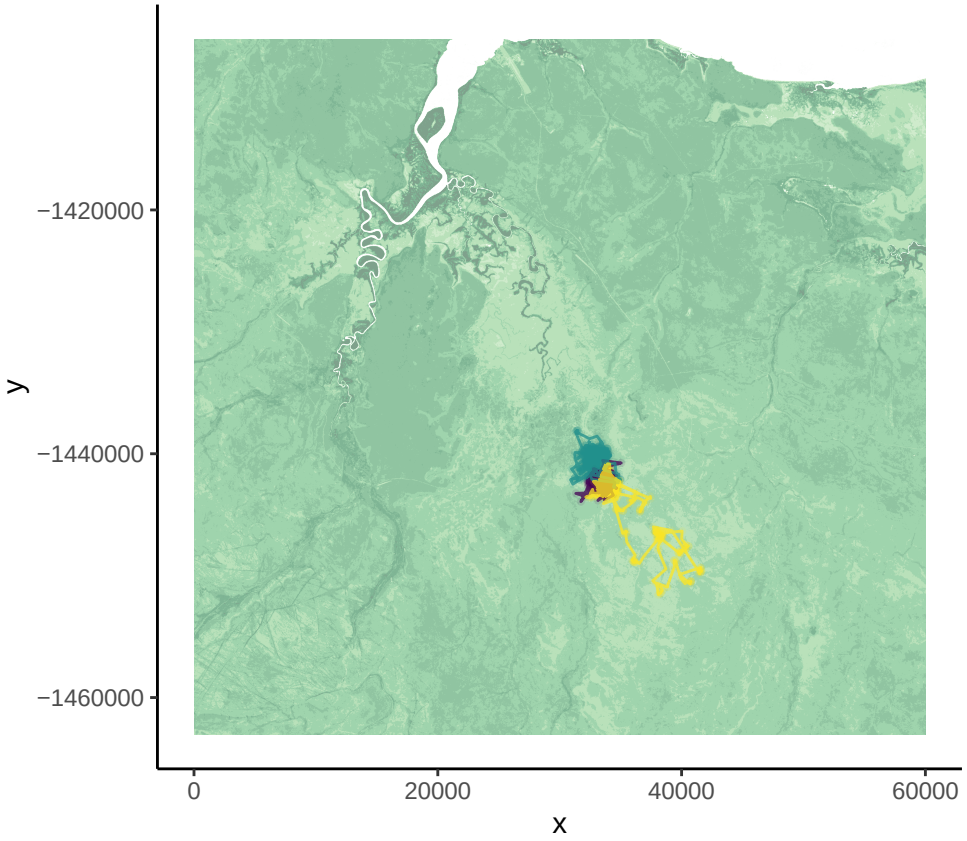
```
                    size = 0.75, alpha = 0.1) +
          geom_path(data = sim_data_mem_truexy,
                    aes(x = x_, y = y_, colour = id),
                    alpha = 0.75) +
          scale_fill_brewer(palette = "Greens") +
          scale_color_viridis_d("Sim ID") +
          coord_equal() +
          theme_classic() +
          theme(legend.position = "none"))

  filename <- paste0("outputs/simulated trajectories/CLR TS 2pDaily GvM MemNat/id_",
                     start_buffalo_df$id[k],"_GvM_mem", memory_period, "_", memory_delay,
                     "_", n_indvs, "ind_", n_ch, "ch_", n_steps, "stp_",
                     Sys.Date(), ".csv")
  # print(filename)
  write_csv(sim_data_mem_truexy, filename)

}
```

```
## 107.87 sec elapsed
## 101.18 sec elapsed
## 99.72 sec elapsed
## 309.11 sec elapsed
```

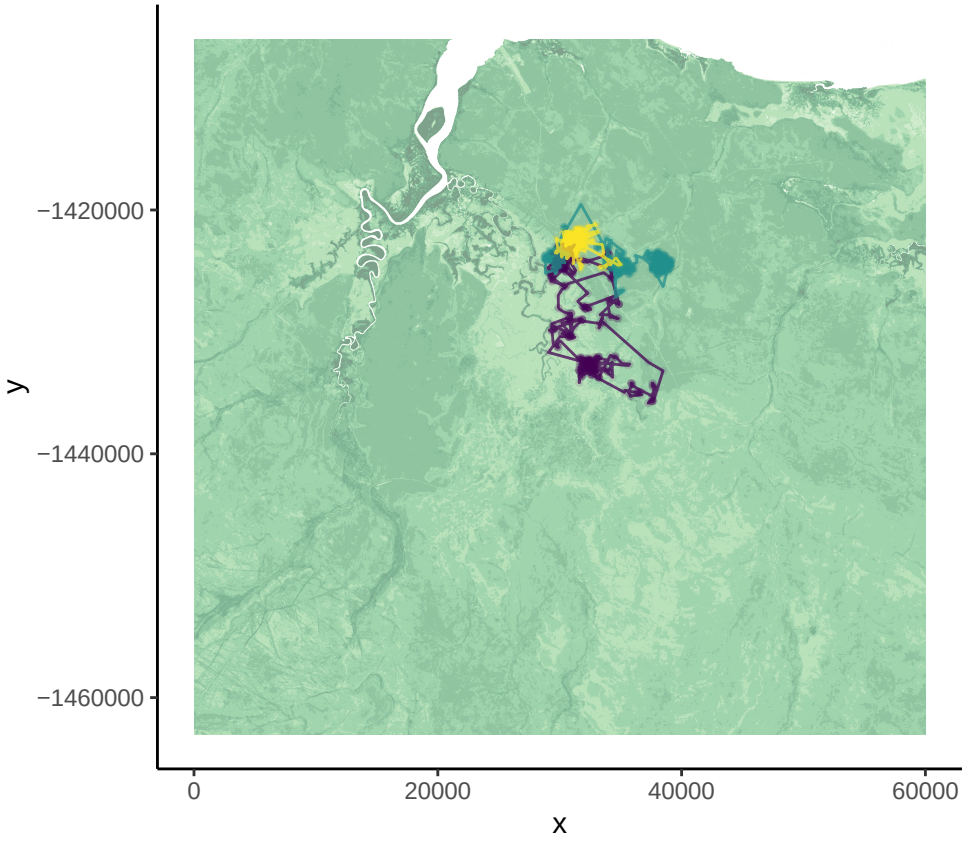



```
## 103.49 sec elapsed
## 100.05 sec elapsed
```

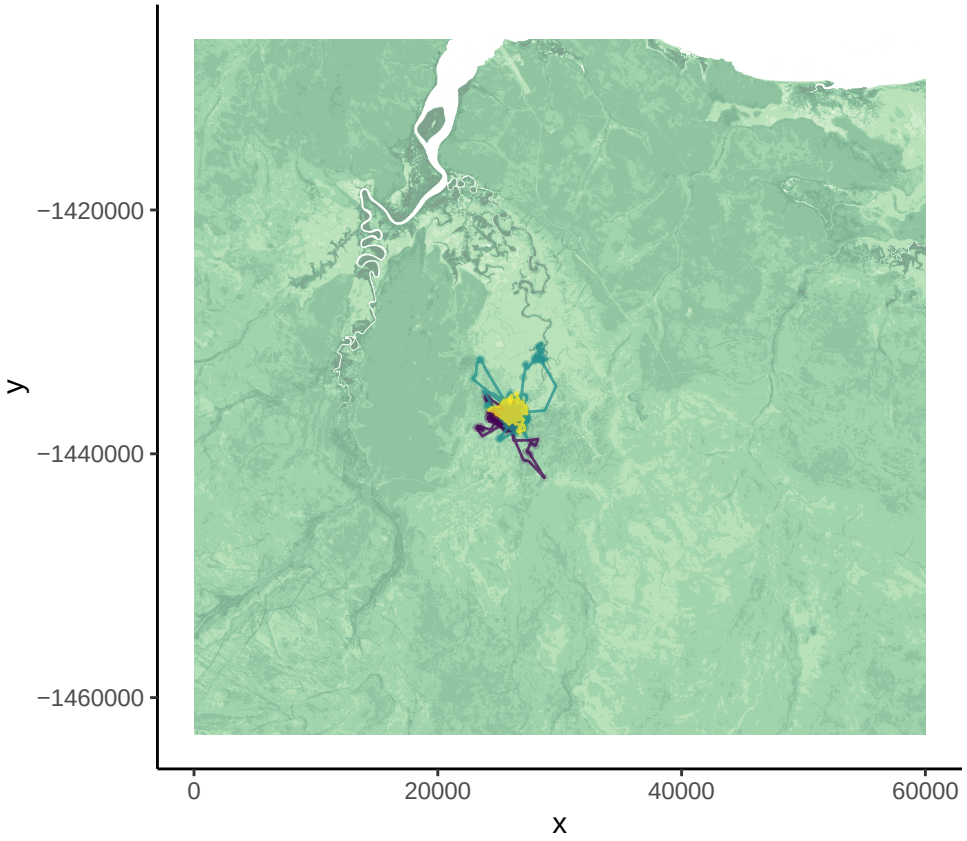
```
## 100.63 sec elapsed
## 304.22 sec elapsed
```



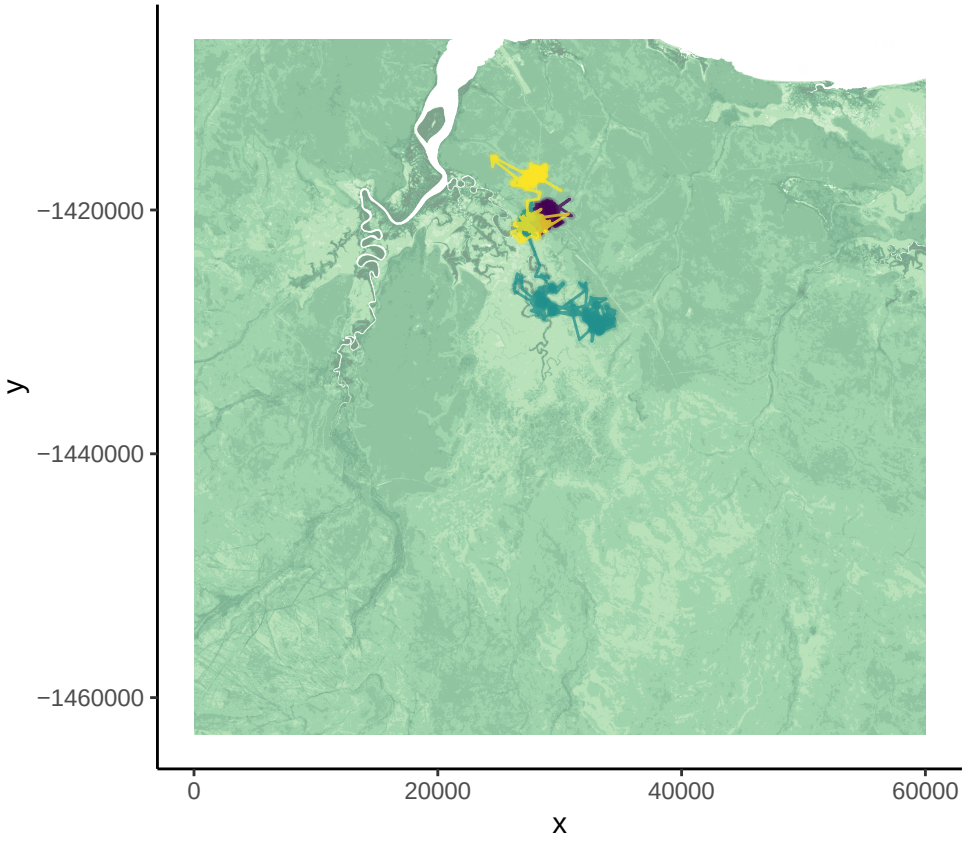

```
## 111.02 sec elapsed
## 100.61 sec elapsed
## 98.75 sec elapsed
## 310.61 sec elapsed
```



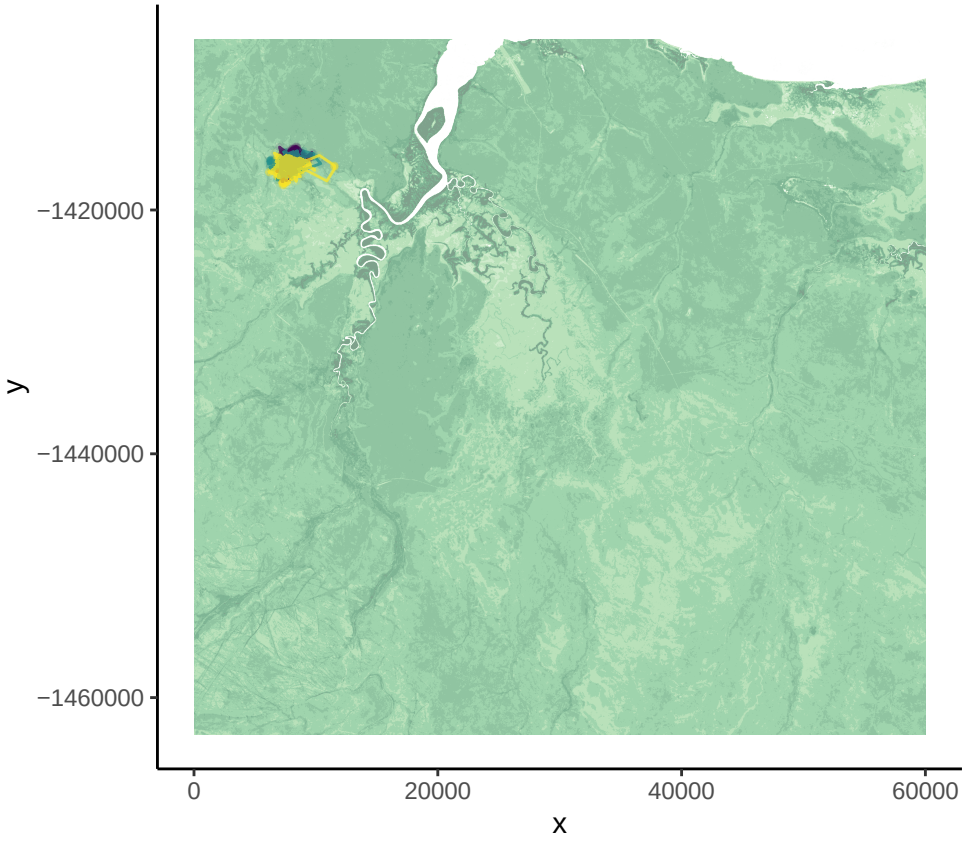

```
## 119.97 sec elapsed
## 98.44 sec elapsed
## 99.38 sec elapsed
## 318.46 sec elapsed
```



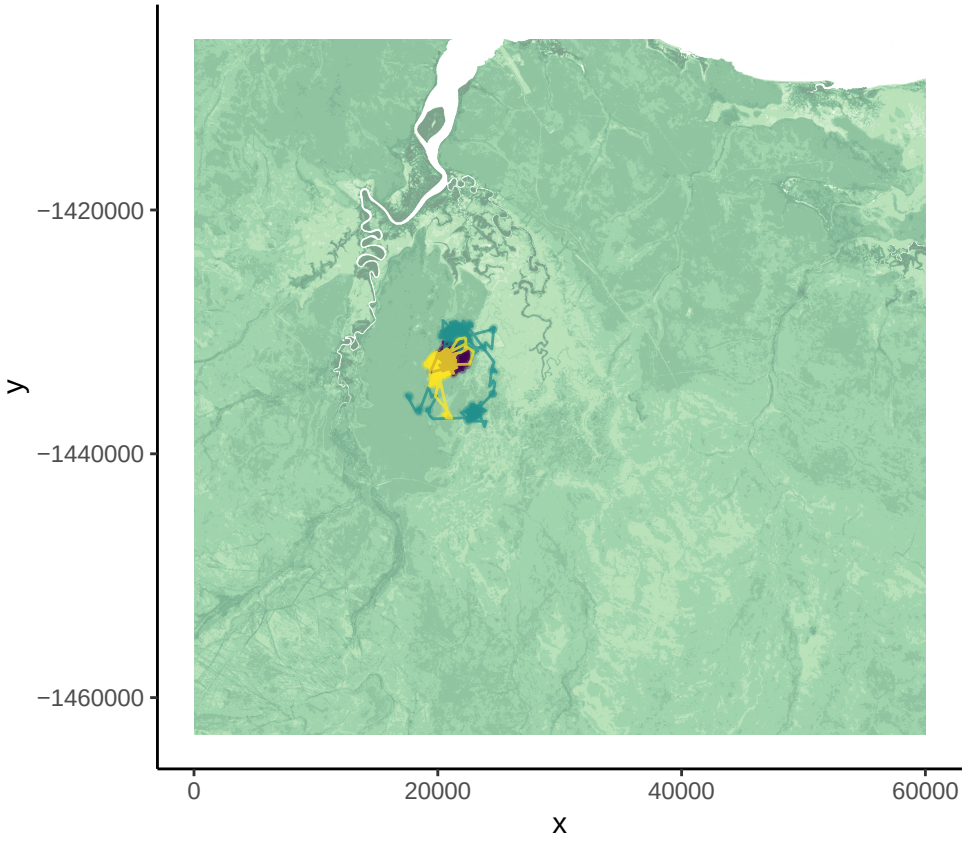

```
## 107.7 sec elapsed
## 102.64 sec elapsed
## 97.37 sec elapsed
## 307.99 sec elapsed
```



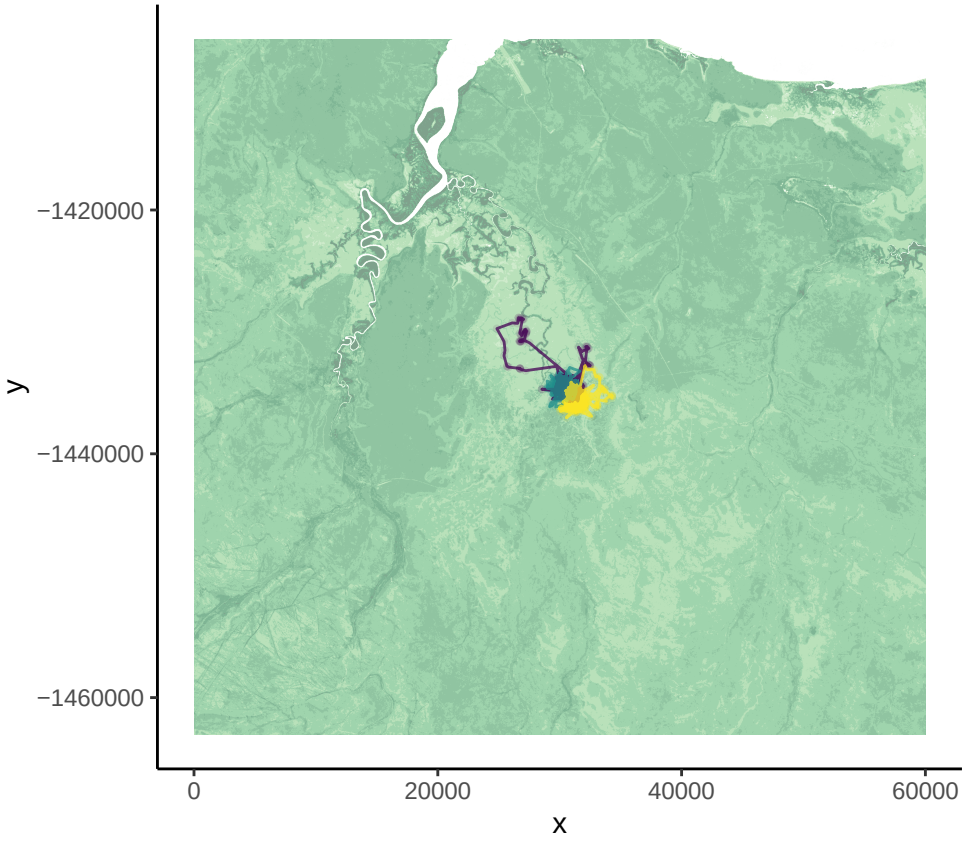

```
## 109.63 sec elapsed
## 100.07 sec elapsed
## 100.58 sec elapsed
## 310.83 sec elapsed
```



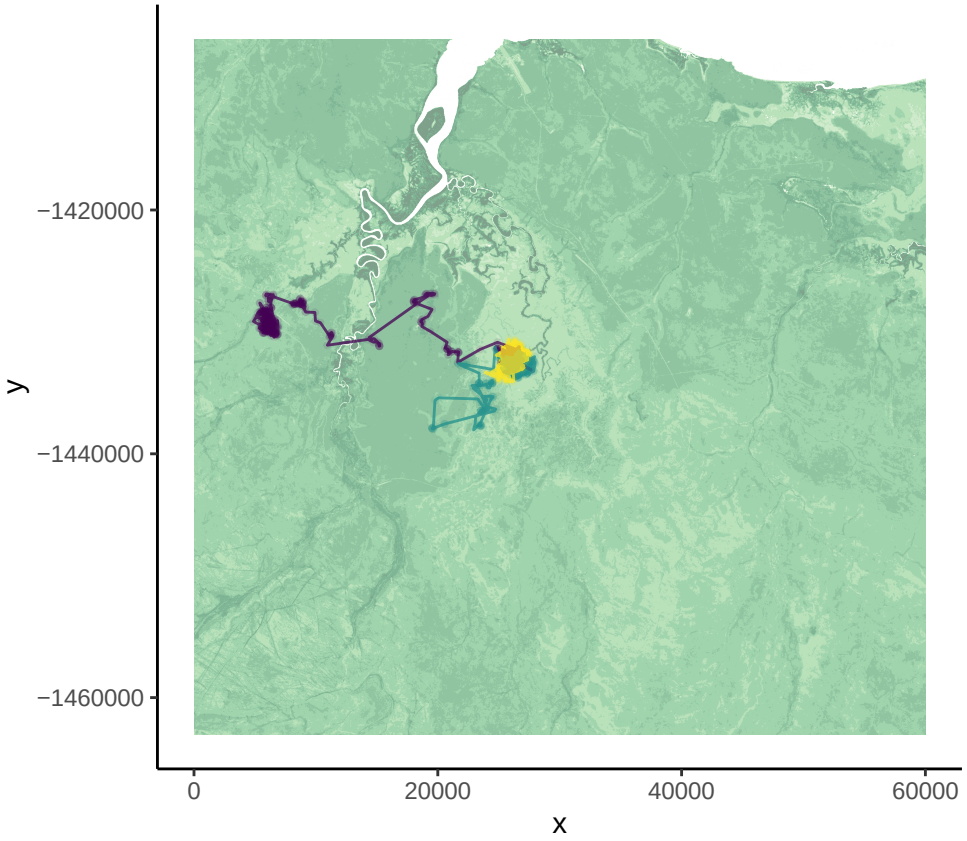

```
## 108.86 sec elapsed
## 101.88 sec elapsed
## 100.33 sec elapsed
## 311.27 sec elapsed
```



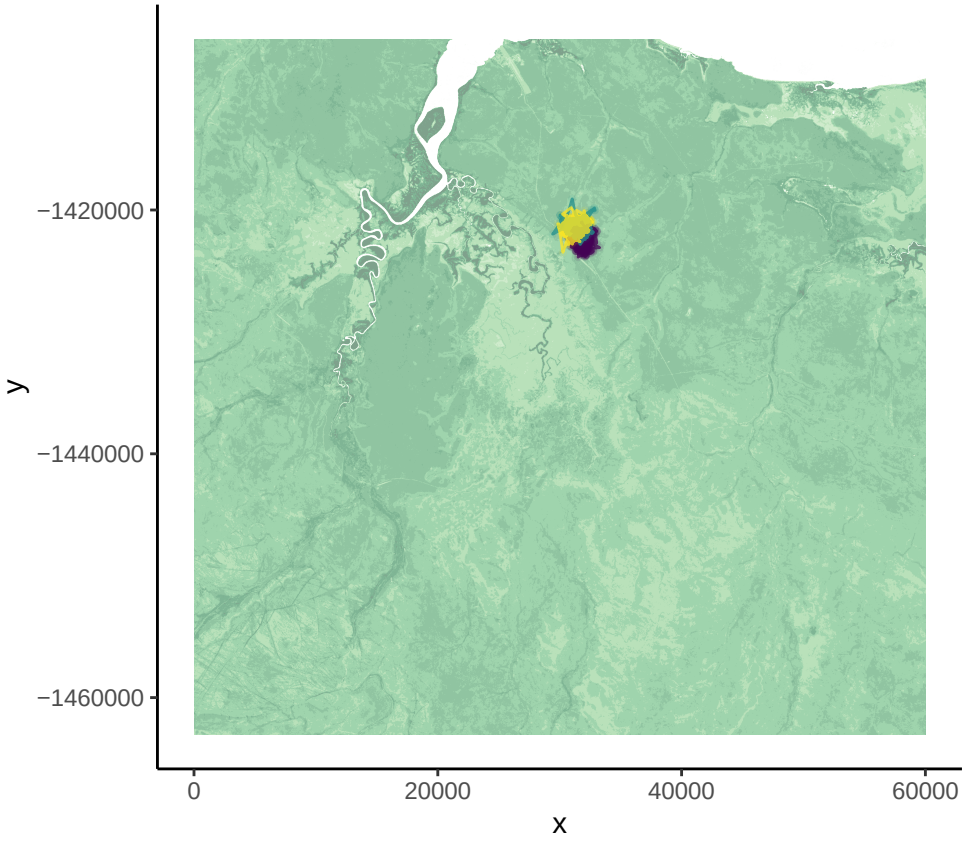

```
## 168.26 sec elapsed
## 96.79 sec elapsed
## 97.87 sec elapsed
## 363.03 sec elapsed
```



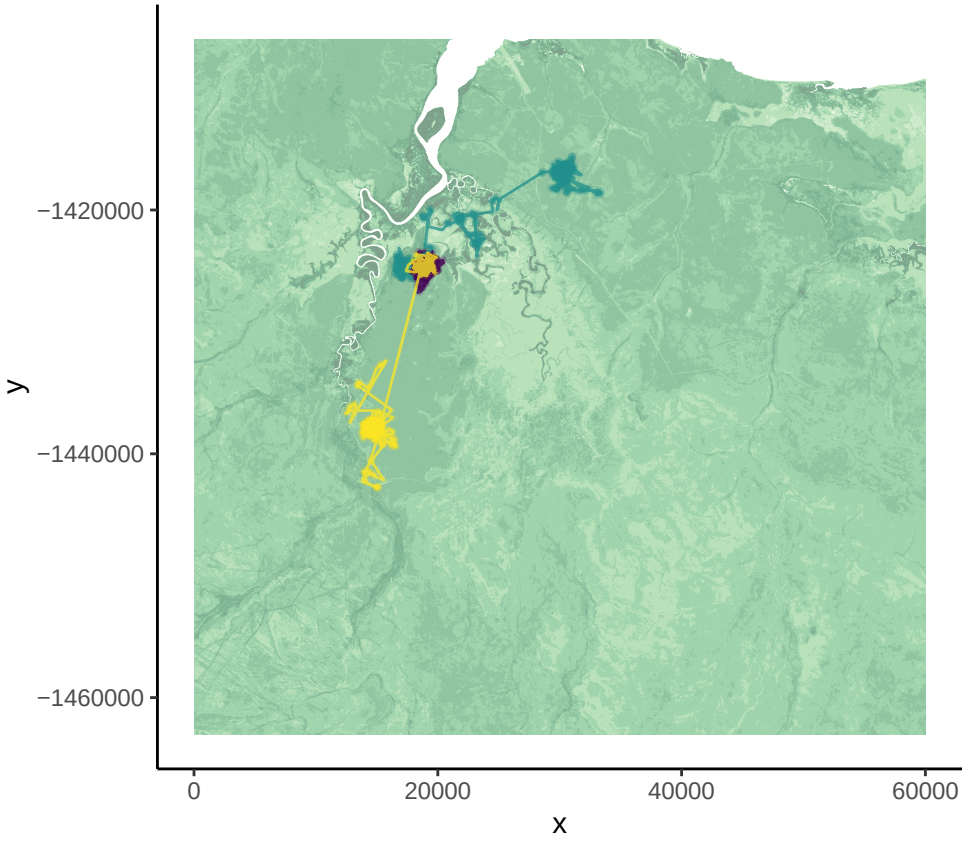

```
## 107.3 sec elapsed
## 96.47 sec elapsed
## 97.97 sec elapsed
## 301.97 sec elapsed
```



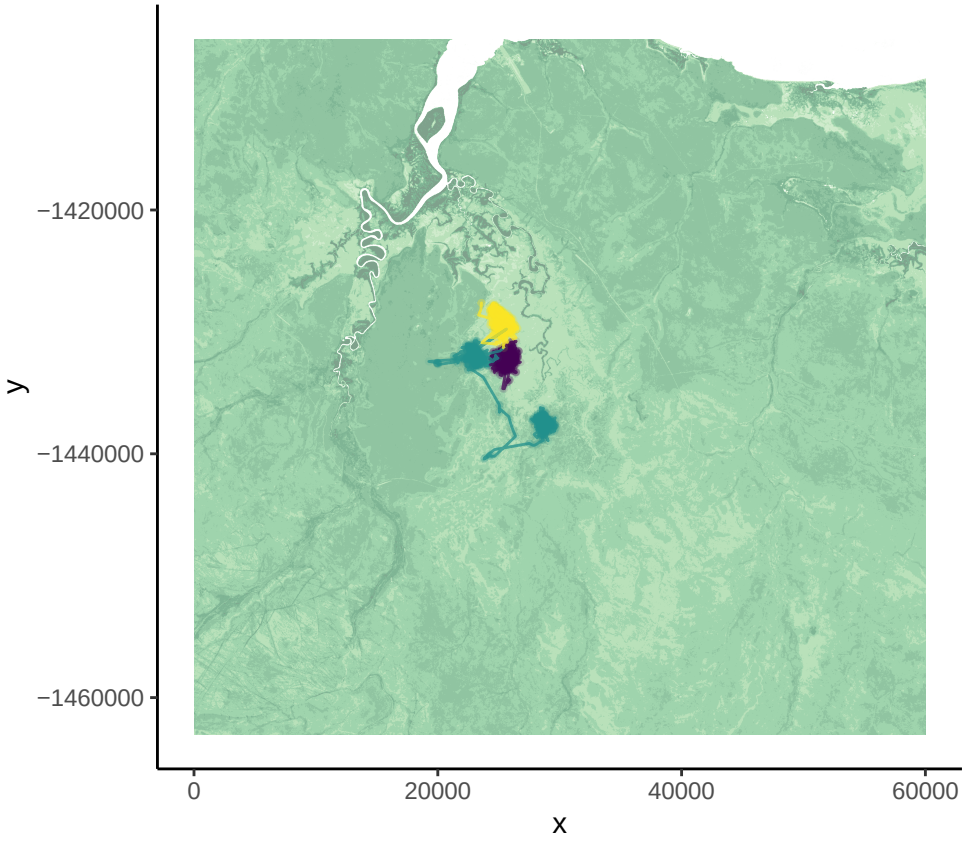

```
## 99.01 sec elapsed
## 96.62 sec elapsed
## 99.53 sec elapsed
## 295.33 sec elapsed
```



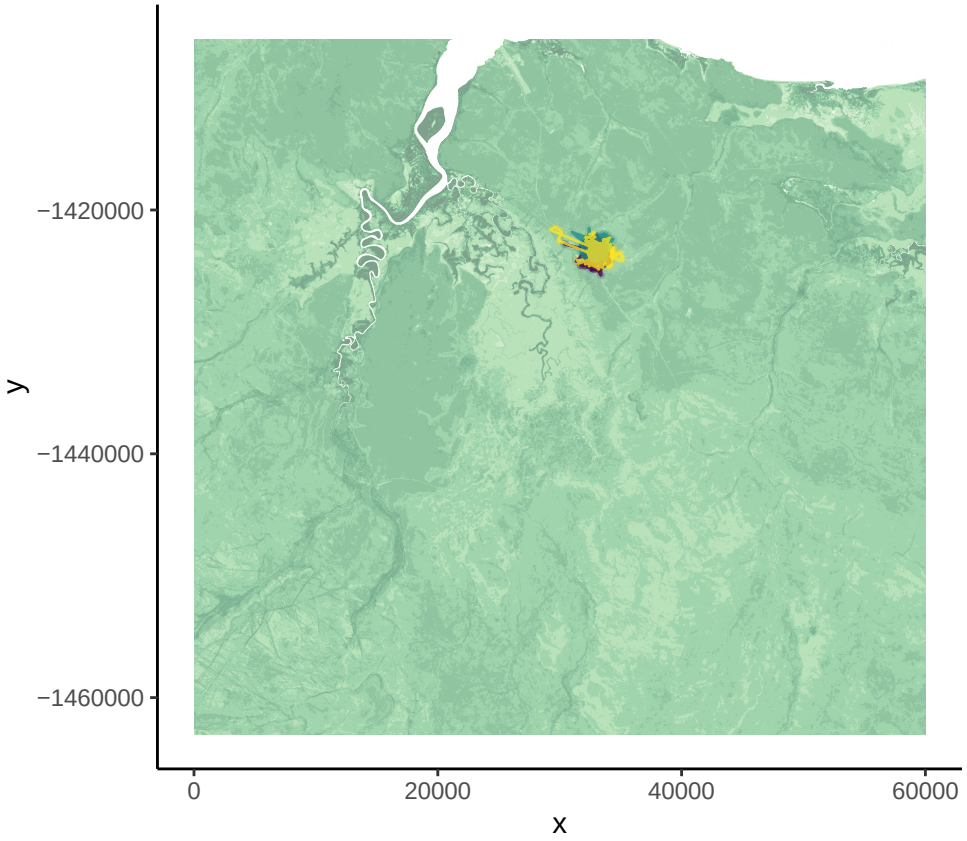

```
## 129.18 sec elapsed
## 99.46 sec elapsed
## 98.85 sec elapsed
## 327.75 sec elapsed
```




```
## 100.12 sec elapsed
## 97.82 sec elapsed
## 96.11 sec elapsed
## 294.29 sec elapsed
```




```
## 99.25 sec elapsed
## 98.34 sec elapsed
## 95.56 sec elapsed
## 293.24 sec elapsed
```



References

Signer, Johannes, John Fieberg, and Tal Avgar. 2017. "Estimating Utilization Distributions from Fitted Step-selection Functions." *Ecosphere* 8 (4): e01771. https://doi.org/10.1002/ecs2.1771.

Session info

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_New Zealand.utf8  LC_CTYPE=English_New Zealand.utf8    LC_MONETARY=English_Ne
## [4] LC_NUMERIC=C                         LC_TIME=English_New Zealand.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] Rfast_2.0.7         RcppZiggurat_0.1.6  formatR_1.14        scales_1.2.1        glmmTMB_1.1.8
##  [6] clogitL1_1.5        Rcpp_1.0.10         ecospat_3.5         TwoStepCLogit_1.2.5 survival_3.5-5
## [11] viridis_0.6.2       viridisLite_0.4.2   matrixStats_1.0.0   patchwork_1.1.2     ggpubr_0.6.0
```

```
## [16] adehabitatHR_0.4.21 adehabitatLT_0.3.27 CircStats_0.2-6     boot_1.3-28.1       MASS_7.3-59
## [21] adehabitatMA_0.3.16 ade4_1.7-22          sp_1.6-0            ks_1.14.0           beepr_1.3
## [26] tictoc_1.2          terra_1.7-23         amt_0.2.1.0         lubridate_1.9.2     forcats_1.0.0
## [31] stringr_1.5.0       dplyr_1.1.2          purrr_1.0.1         readr_2.1.4         tidyr_1.3.0
## [36] tibble_3.2.1        ggplot2_3.4.2        tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1        Hmisc_5.0-1            systemfonts_1.0.4      plyr_1.8.8             TM
##   [6] splines_4.2.1          TH.data_1.1-2          digest_0.6.31          foreach_1.5.2          htm
##  [11] earth_5.3.2            fansi_1.0.4            magrittr_2.0.3         checkmate_2.1.0        clu
##  [16] tzdb_0.3.0             vroom_1.6.1            sandwich_3.0-2         timechange_0.2.0       col
##  [21] textshaping_0.3.6      rbibutils_2.2.13       xfun_0.39              crayon_1.5.2           jso
##  [26] lme4_1.1-32            zoo_1.8-12             iterators_1.0.14       ape_5.7-1              glu
##  [31] PresenceAbsence_1.1.11 gtable_0.3.3           emmeans_1.8.5          car_3.1-2              abi
##  [36] mvtnorm_1.1-3          DBI_1.1.3              rstatix_0.7.2          isoband_0.2.7          ple
##  [41] xtable_1.8-4           htmlTable_2.4.1        units_0.8-1            foreign_0.8-84         bit
##  [46] proxy_0.4-27           mclust_6.0.0           Formula_1.2-5          htmlwidgets_1.6.2      RCo
##  [51] nabor_0.5.0            pkgconfig_2.0.3        reshape_0.8.9          farver_2.1.1           nnc
##  [56] sass_0.4.5             utf8_1.2.3             tidyselect_1.2.0       labeling_0.4.2         rla
##  [61] reshape2_1.4.4         munsell_0.5.0          TeachingDemos_2.12     tools_4.2.1            cac
##  [66] cli_3.6.1              generics_0.1.3         audio_0.1-10           broom_1.0.4            eva
##  [71] fastmap_1.1.1          yaml_2.3.7             ragg_1.2.5             maxnet_0.1.4           kni
##  [76] bit64_4.0.5            fitdistrplus_1.1-8     randomForest_4.7-1.1   nlme_3.1-162           pra
##  [81] biomod2_4.2-2          compiler_4.2.1         rstudioapi_0.14        e1071_1.7-13           ggs
##  [86] bslib_0.4.2            stringi_1.7.12         plotmo_3.6.2           highr_0.10             lat
##  [91] poibin_1.5             circular_0.4-95        Matrix_1.6-5           nloptr_2.0.3           cla
##  [96] permute_0.9-7          vegan_2.6-4            gbm_2.1.8.1            vctrs_0.6.2            pil
## [101] lifecycle_1.0.3        Rdpack_2.4             jquerylib_0.1.4        estimability_1.4.1     cow
## [106] data.table_1.14.8      raster_3.6-20          R6_2.5.1               KernSmooth_2.23-20     gri
## [111] codetools_0.2-19       gtools_3.9.4           withr_2.5.0            multcomp_1.4-23        mgc
## [116] parallel_4.2.1         hms_1.1.3              grid_4.2.1             rpart_4.1.19           coc
## [121] minqa_1.2.5            class_7.3-21           rmarkdown_2.21         carData_3.0-5          mda
## [126] sf_1.0-12              pROC_1.18.0            numDeriv_2016.8-1.1    base64enc_0.1-3        tin
```