University of Göttingen

amtGUI

Developing a graphical user interface with Shiny for the analysis of telemetry data

Authors:

Andreas Buchmüller, Dragan Höhn, Oliver Dippel

supervised by

Stanislaus Stadlmann,

Dr. Johannes Signer

Contents

1. Motivation

The improvement of tracking technology has increased the number and extent to which data can be collected for animals. The newly acquired wealth of data contains important information regarding the behaviour of animals and movement patterns in their environment. There are several ways to study the data. For example, if you want to make statements about particular behaviour patterns. One way is by applying Step Selection Functions (SSF). SSFs are mostly used to combine environmental covariates with the GPS data of the animals [1]. The empirical data, more precisely the steps chosen by the animals, are compared with random steps. SSFs are largely based on Resource Selection Functions (RSF) and Resource Selection Probability Functions (RSPF). RSFs can be described as exponential functions of used and available areas: $w(x) = exp(\beta x)$. In other words, it describes the ratio of used area (f_u) to available area (f_a) for a given x. Where the entries in x correspond to discrete units in space (such as a map pixel or a habitat patch)[1]. The core feature of SSFs compared to RSFs is the linking of sequential data points to so-called steps. Each step at time t is linked to a fixed number of random steps. The starting point is the same. The random steps are drawn from distributions of step length and turning angles. The model of a SSF is the same as before for a RSF only that this time the ratio of (f_u) to (f_a) is dependent. More precise the used units are dependent on the previous ones and the available step drawn from a distribution of step lengths and turning angles. SSFs therefore enable the assignment of parameters from motion rules to landscape characteristics [2].

The `R` package from [7] Animal Tracking Tool (`amt`) provides tools for the analysis described above. This is done in four steps. In the first step, the data is prepared for further analysis. Then the exploratory data analysis and descriptive analyses are performed. The next step is modeling. The fourth and last point is the simulation. However, the use of the `amt` package requires a certain knowledge of the open source programming language `R`. Since not every forester or similar target group of the `amt` package is familiar with `R` or has a statistical/mathematical background we developed a Shiny web app

based on the `amt` package to fit a model in a few steps either based on RSF or Integrated Step Selection Function (iSSF). In order to get a more detailed insight into our Shiny web app, in section two the analysis steps of the `amt` package will be discussed. Hence, the underlying models will be clarified and an example for the interpretation is given. In section three the exact handling of the Shiny app is explained by a case study. Every single step from uploading data to visualizing the data is clearly explained. Section four describes the implementation in more detail. This takes place in six steps and is thus based on the structure of the app. Finally a conclusion takes place.

2. Analyzing Animal Telemetry Data

The `amt` package has the aim to contribute in two ways to the movement ecology of animals. First the package will help users to manage and analyze their data. Additionally, access to iSSF will be facilitated. Our `Shiny` app can be seen as an extension of this. The user should be able to read, analyze and visualize his data with just a few clicks. For this reason, the relevant steps, for our `Shiny` app, of the `amt` package are described in more detail below.

2.1. Resampling

The `amt` package provides functions to quantify the variability of sampling rates over time and individuals. Periods at the beginning and end of the tracks are removed to exclude possible capture effects and to restructure the data into regular bursts (i.e. to divide the track into groups of observations with regular sampling rates within a tolerance limit). Burts are clusters of several observations but at least three (Figure 6). This is important to draw conclusions about the step length and the turning angles. In our Shiny app this is done manually. The user can determine the resampling rate and the tolerance himself. He can decide to what extent he wants to trim the original

data set. The information about the remaining data is then given to the user according to his settings.

2.2. Modeling

In `amtGUI` two functions are offered for modeling. On the one hand the Resource Selection Function (RSF) and on the other hand the integrated Step Selection Function (iSSF).

2.2.1. Resource Selection Function

The RSF provides a prediction which is proportional to the selection probability. More specifically, it assesses which habitat characteristics are important for a particular population. This involves statements about the probability that an animal will choose a particular resource in proportion to the available space. For the estimation a binomial regression is used with logistic link to fit the RSF.

Aim of the binomial regression analysis is to model and estimate the effect of the covariates on the conditional probability

$$\pi_i = P(y_i = 1|x_{i1}, \ldots, x_{ik})$$

for the occurrence of $y_i = 1$, given x_{i1} to x_{ik}. Usually, the probability π_i is linked to the linear predictor η_i by a relation of the form

$$\pi_i = h(\eta_i) = h(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_k x_{ik}). \tag{1}$$

Here h is a strictly monotonic increasing distribution function so that $h(\eta) \in [0, 1]$ is always valid and the relation (1) can be written in the form $\eta_i = g(\pi_i)$ with the help of the inverse function $g = h^{-1}$. Whereas h is the response function and g is the link function. In this case a logistic response function

$$\pi = h(\eta) = \frac{exp(\eta)}{1 + exp(\eta)}$$

and a logit link function

$$g(\pi) = log\Big(\frac{\pi}{1-\pi}\Big) = \eta.$$

The log-odds are then given by: $log(\pi/(1-\pi))$. Taking $exp()$ of the log-odds leads to $P(y_i = 1|x_i/P(y_i = 0|x_i))$. The magnitude of the β_k can then be interpreted. For instance x_{i1} increases to $x_{i1}+1$ the ratio of the chances are:

$$\frac{P(y_i = 1|x_{i1}, \dots)}{P(y_i = 0|x_{i1}, \dots)} / \frac{P(y_i = 1|x_{i1}+1, \dots)}{P(y_i = 0|x_{i1}+1, \dots)} = exp(\beta_1). \tag{2}$$

For the odds means that if $\beta_1 > 0$ ($\beta_1 < 0$) the odds increase (decrease). For $\beta_1 = 0$ the chance $P(y_i = 1)/P(y_i = 0)$ stays the same [4]. Even if this is the underlying model and an interpretation of the coefficients as log-odds is common [6] claims that this is incorrect for animal telemetry data.

Avgar [2] suggests that in the context of habitat selection an interpretation of the coefficients as Relative Selection Strength (RSS) is correct. Generally, the relative risk is the ratio of a probability that an event will occur in a treatment group to the probability of the same event occurring in a control group. Assume that x_1 and x_2 are spatial coordinates of two different locations then the RSS is the ratio of the corresponding RSFs

$$RSS(x_1, x_2) = \frac{w(x_1)}{w(x_2)}$$

where

$$RSF = w(x) = exp\ (\beta_1 x_1 +, \dots, +\beta_n x_n). \tag{3}$$

Thus, the RSS is only dependent on the differences in habitat conditions of two different places. Suppose there are two different locations with $w(x_1) = 0.90$ and $w(x_2) = 0.18$. Then $RSS(x_1, x_2) = 5$. This means that it is five times more likely that an animal will choose location x_1 versus location x_2. Another important point is how the change in one of the habitat covariates influences the probability of selection. If x_1 is increased to x_{1+1} and all other values are left unchanged from equation (3). Then it follows equation (2).

Thus, $exp(\beta_1)$ is the RSS of habitat covariate x_1 given all other covariates kept the same. However, this is a conditional interpretation of the covariate x_1 and not the effect of the covariate x_1 without reference to other covariates [2]. The estimated value of β and its exponential value as interpretation as RSS therefore depends on the respective model and the covariates used. Another factor are correlation structures. The interpretation should therefore be used with caution.

2.2.2. Integrated Step Selection Function

Since a RSF is a model of the likelihood that an available unit will be used by an animal, given its resource value, it is not clear what is exactly available and what is not. Step selection analysis deals with that issue. Simultaneous estimation of the movement and habitat selection leads to the Integrated Step Selection Analysis (iSSA). The iSSA approach makes it possible to distinguish the effects of environmental varibals on the movement structure and the selection process of the animal. This makes hypotheses testing accessible to test if an animal moves faster through a certain area or if it avoids certain regions. This information can then be used to project behavior patterns on landscape maps and to translate individual-level observations to population-level utilization distributions across space and time [1].

To generate Integrated Step Selection Functions several random steps are assigned to each measured location of the animal. The steps are randomly drawn. These random steps are generated by fitting a parametric distribution to the observed step lengths and turning angles. The gamma distribution is fitted to the step lengths and the von-Mises distribution to the turn angles by using maximum likelihood. The fitted distributions are then used for generating random steps for each observed step. Where each random step is generated by a turn angle and a step length pair, drawn from the respective fitted distributions [7]. Then the covariates are extracted at the end of each observed and each random step to fit a conditional logit model. The estimates can then be interpreted as RSS again. Note that in case of an iSSA, the RSS only depends on the difference between two steps that share

the same starting point but ending in x_1 and x_2. Where x_1 and x_2 denote spatial coordinates of two different locations [2].
The interpretation depends on which covariates you have chosen and in which combination. It depends on which question you want to answer. It is also important whether covariate values are extracted at the beginning or at the end of a step. It is also possible to extract the covariate values at the beginning and at the end. If the covariates are extracted at the end of a step, then you investigate the habitat selection process. The covariates are included in the model as main effects to answer questions like: where the animal moves? Are covariates extracted at the beginning of the step they are typically included in the model as an interaction with motion properties, i.e. stride length, stride length protocol or cosine of rotation angle, to test hypotheses whether animals move faster or more directed when they start in a particular habitat. In the case that covariates are integrated into the model, which are extracted at the beginning and end of a step, questions can be asked such as: Do animals tend to stay in a particular habitat when they are already in the same habitat? [7]

The iSSA approach can also be applied to the derivation of directional persistence and external bias. Assuming that the angular deviations from the selected preferred directions, which depend either on the previous course, the target course or both, are von-Mises distributed. Then the cosine of these angular deviations can be included as covariates in a Step Selection Analysis to obtain Maximum Likelihood estimates of the corresponding von-Mises concentration parameters [1]. These estimates in iSSA affiliated with directional deviations and step lengths are therefore directly interpretable as the parameters of distributions that determine the underlying Biased Correlated Random Walk provided by [3].

3. Case Study

To demonstrate `amtGUI`'s abilities we will be using the Fisher (*Pekania pennanti*) data set containing GPS data of multiple animals over a time span of

over two years. [5] For a data set to be suitable it has to contain telemetry data, i.e. GPS coordinates of one or more animals and timestamps for each observation. In addition there can be an ID column where it is recorded which observation belongs to which animal. If we take a look at the fisher data set the following variables are of interest:

1. **location-long:** The geographic longitude of a location along an animal's track as estimated by the processed sensor data. Positive values are east of the Greenwich Meridian, negative values are west of it
2. **location-lat:** The geographic latitude of a location along an animal's track as estimated by the processed sensor data. Positive values above the equator, negative values are below of it
3. **timestamp:** The date and time a sensor measurement was taken
4. **individual-local-identifier:** An individual identifier for the animal

After a successful data upload (or in our case after selecting the example data that comes with the app) a data frame with our data should load immediately in the bottom half of the tab. All coordinates in this data set are given in the WGS 84 reference system which is equivalent to EPSG code 4326, thus we select "4326" in the data upload tab. Generally, the user will have to check beforehand, which reference system was used for the coordinates in the uploaded data set and then assign the corresponding EPSG code.

After assigning the correct EPSG code we can continue in the next tab and upload our map (i.e. land use raster). We upload our map to the app in the same way we upload data. Here, `amtGUI` will automatically detect any EPSG codes found in the meta data of the image and accordingly make suggestions for the EPSG code. We load the example map, which is a land use raster of the area our animal resides in and select "5071" as our EPSG code for the map. If both uploads are done and the correct EPSG codes assigned we proceed to create a track.

To analyze animal movement by fitting iSSF's and RSF's [7] we need to create a track or path of our animals in question. We go to the "Configure

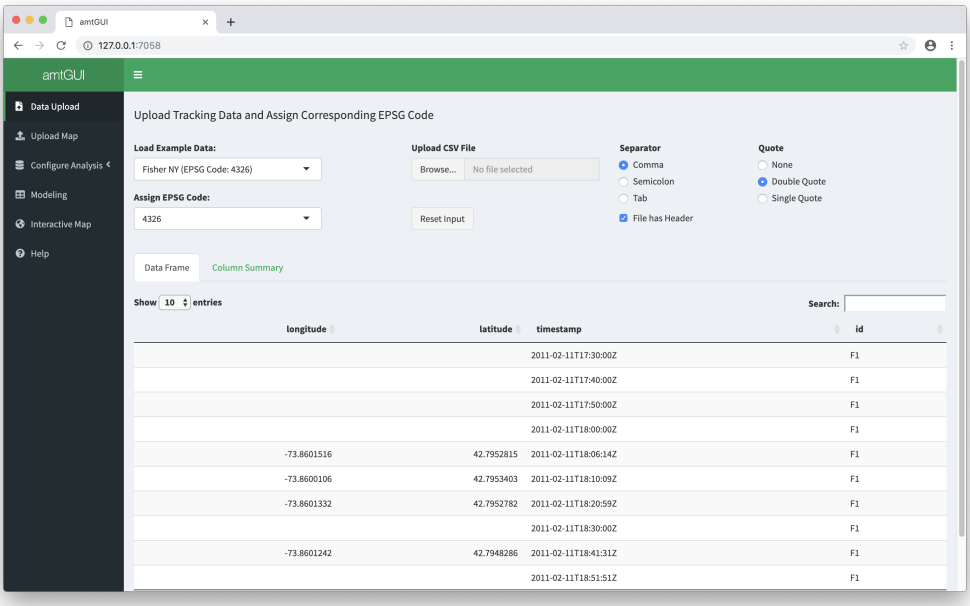

Figure 1: Data upload tool

Analysis" tab and from there into the "Create Track" subtab. Here we can fill out multiple input boxes to create our path. First we need to tell `amtGUI` which of the columns of our data set hold coordinates, timestamps and individual IDs. We select the corresponding columns of our data set from the drop down menus that open when we click any of the input boxes with a pointed down arrow.

Our showcase data set only has the four variables mentioned earlier, all of which are named appropriately making this step easy. Now we need to set a resampling rate for the track to be created. The resampling rate should be as small as possible given the observed sampling rate. Therefore, you may compare it to the median. Note that setting it too small can often times result in dropping too many observations from the original data set. Below the input boxes we can view a summary of the track's observed sampling rate (of each ID if we have multiple animals) by clicking the "Summary of Sampling Rate" tab. This is a quick way to check what resampling rate may be appropriate. As an example, we drop all animals except IDs M4 and M5

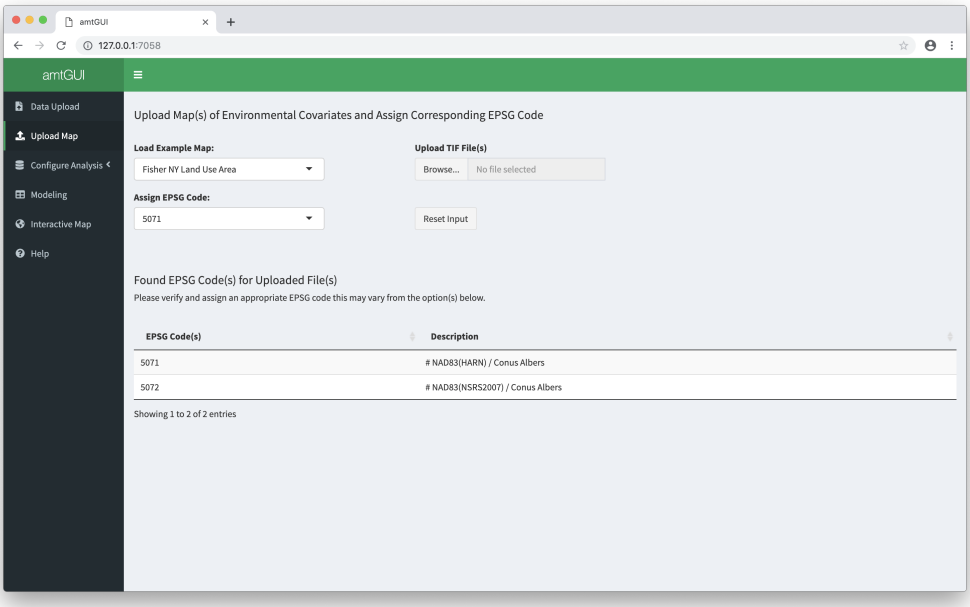

Figure 2: Map upload tool

for our analysis. We may choose a resampling rate of 10 minutes with a tolerance of 2 minutes, so the steps in our track are all 8 - 12 minutes apart. For these two IDs we could have chosen a smaller rate given a median of about 2 minutes. However, iSSF inference is scale dependent, i.e. if we choose to include all IDs later on where some of them exhibit a median of about 10 minutes we need to adjust to that value to obtain comparable results. We retain 5071 as the pre-selected EPSG code for our track, as it should match the code from our map. Additionally, we have the ability to select a date range for our track, dropping all observations not within the specified range. If we omit an ID value the default date range will be refreshed accordingly. We continue without modifying the time frame and go to the next page of the configuration tab.

In the "Add Covariates" section of the app we can configure our analysis even further by setting the minimum number of relocations allowed per burst and transforming environmental covariates from our land use raster (map) into categorical variables. We can also set to include the time of day as

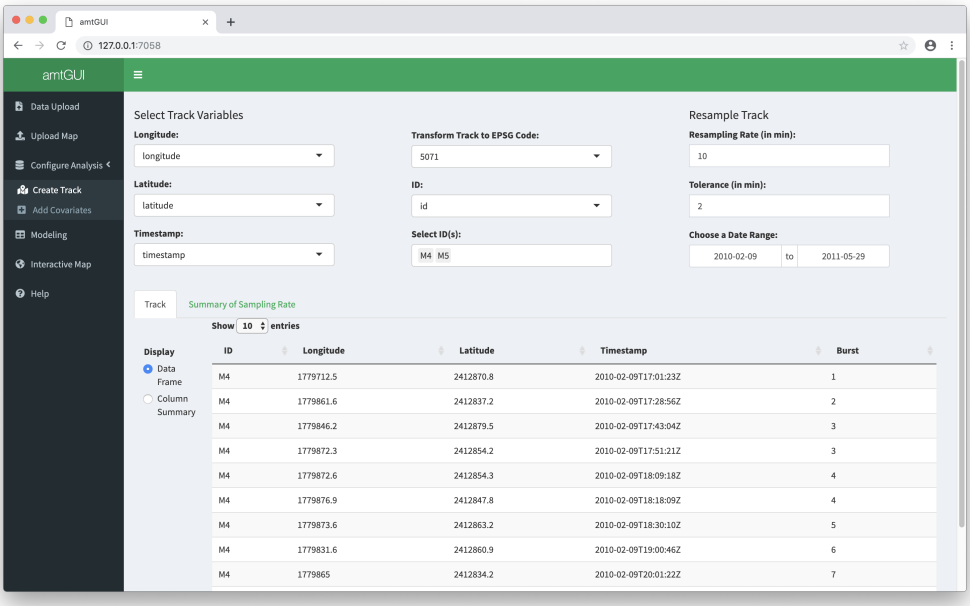

Figure 3: Track creation

a categorical variable into our analysis. `amtGUI` will create an appropriate factor variable to be included in the model. We retain the default setting for the minimum number of relocations per burst (3 because we need at least three relocations to calculate a turn angle) and can see a table below the input box showing us the number of bursts remaining for each ID as well as the corresponding number of observations (n) at our current setting. Besides, we add the time of day covariate with 4 levels. Now we are done configuring our analysis and can continue with model fitting in the next tab.

In the modeling tab of `amtGUI` we can choose between fitting a RSF and an iSSF. For our case study, we chose to fit an iSSF and create three random steps for each observed step. Next, we need to select variables and interaction terms used in model fitting. Here we choose the variables "land_use_end", (the "end" part indicates that the environmental covariates were extracted at the end of each step, similarly for "land_use_start" the covariates were extracted at the beginning), "_sl" (the step length) and "cos_ta_" (cosine of the turn angles) and set the interaction term slider to "2". For our first

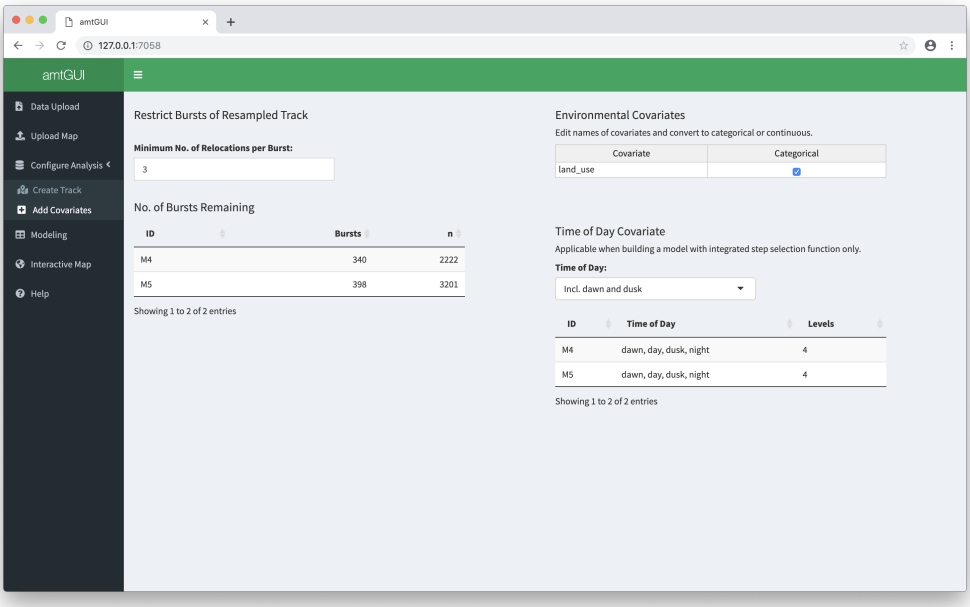

Figure 4: Adding covariates

interaction term we select "sl_" and "tod_end_" (time of day at end of each step) as we suspect the step length might differ by daytime and for our second term we choose "cos_ta_" and "tod_end_" because we suspect the same for the turning angles. By clicking the "Fit Model" button we initiate the fitting process. A progress bar will appear at the bottom right of the screen. When the calculation is finished, a data frame appears below the buttons yielding our parameter estimates for each animal ID. Addtionally, the data frame contains standard errors, value of the t-statistic and corresponding p-value and 95% confidence intervals. From here we either start interpreting our results or tweak our model if dissatisfied with our results. If we choose to rebuild our model, we can simply clear the modeling tab by clicking "Clear" and fit a new model. If we are satisfied with our result and we chose to retain our model we can save our results by clicking "Download Estimates" and download our estimates in a CSV file. Additionally, we can download a user report by clicking "Download Report", which creates an HTML file containing all user inputs which may be used by other researchers in order

to replicate our results or we can continue to work on our model later on, without losing all inputs from a previous session. [1]

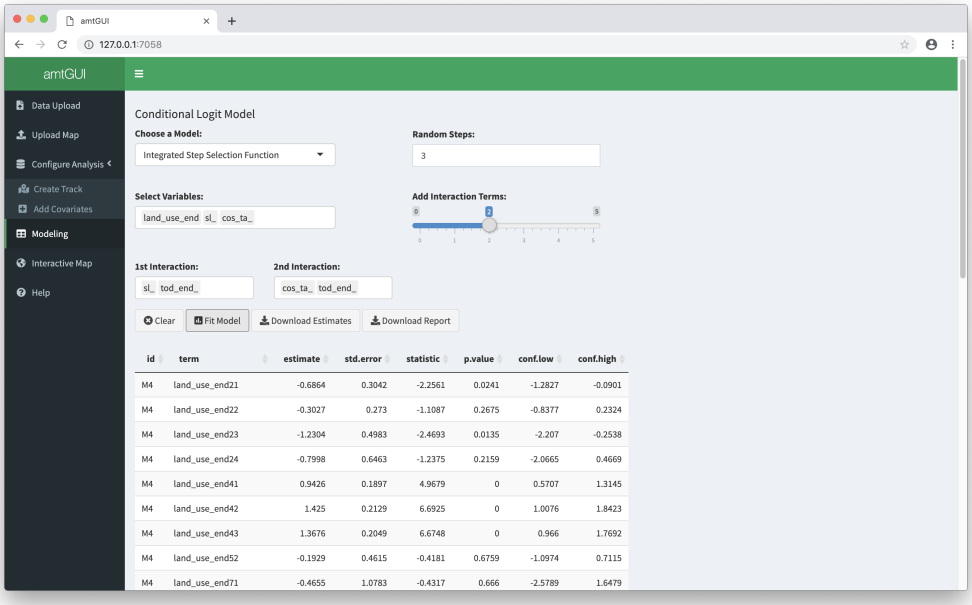

Figure 5: Fitting Models

The interpretation of our results depends on the choices we made earlier and on which question we want to answer. It also depends on whether covariate values are extracted at the beginning or at the end of a step, since it is possible to extract the covariate values at both the beginning or the end. The covariates are extracted at the end of a step in our case study, meaning we investigate the habitat selection process. With the covariates included in the model as main effects we can answer questions like: where does the animal move? The coefficients are interpretable in terms of the RSS, i.e. by applying the exponential function to our estimates we get the RSS of an animal moving to a certain environment compared to a different environment which acts as a base for calculating the RSS. For example "land_use_end21" could be forested wetlands in our land use raster and is defined by the user when creating the map file with a geographic information system (GIS) externally beforehand.

[1] The user report of our case study can be found in the appendix

2

4. Application Structure & Guide

4.1. Data Upload

The user can either load example data to get acquainted with the app or upload a CSV file containing the tracking data of interest. The example data is accessible by a drop down menu (R function `shiny::selectizeInput()`).

The upload button allows the user to browse through folders and select a single file (implemented by `shiny::fileInput()`). Additionally, a reset button is implemented in case the user wants to revert his choice provided by `shiny::actionButton()`. This is followed by format options concerning the uploaded data set. Starting with a check box to tick off whether the data set has a header or not (`shiny::checkboxInput()`). Complemented by radio buttons (`shiny::radioButtons()`) to choose separator and quote accordingly. The uploaded tracking data will be shown in an on page sub tab named "Data Frame". Summary statistics concerning the data sets columns are provided with the second on page sub tab "Column Summary". Besides, the user is obligated to assign an EPSG code to the tracking data. The options are extracted from `rgdal::make_EPSG()` and provided by a drop-down menu.

The server section corresponding to the data upload tab starts with an adjustment of the allowed maximum upload size. The default value of 5 MB is raised to 30 MB (`base::options()` with argument `shiny.maxRequestSize`). This step affects the global options of the user. Subsequently, we define dynamic values to monitor the current upload state (`shiny::reactiveValues()` and `shiny::observeEvent()`). The handler expression in both cases assigns states "uploaded" and "reset" respectively. A reactive expression that accesses the data input evaluates the assigned upload state within `if else`

[2]For a more in-depth explanation on how to interpret estimates from RSF/iSSF see Chapter 2

statements (`shiny::reactive()`). The data set will be read in using `readr::read_csv()` and its variants for semicolon and tab separated values. These functions preserve the classes of the input columns which in our case is necessary to process the timestamp column. Next, the data input will be converted to a data frame by `DT::datatable()`. The UI will be refreshed by `DT::renderDataTable()` in case of the data frame and `shiny::renderPrint()` in case of the column summary which will be generated by `base::summary()`.

4.2. Map Upload

The Upload Map tab features basically the same functionality as the previous tab but is restricted to the upload of TIF files. Analogous to the CSV upload, the user may select an example map or upload one or multiple maps containing environmental covariates. The images are converted to raster layer objects (`raster::raster()`). In case of multiple maps the single raster layers are stacked utilizing `raster::stack()`. Each map will be renamed to replace the generic names by the actual file names. These names will be displayed as default in the "Add Covariates" tab.

Again, the user is obligated to assign an EPSG code. When the raster object features a coordinate reference system (CRS) that information will be extracted as a string by `raster::projection()` and we left join the data frame containing the EPSG codes from above to provide the user with a list of matching EPSG code(s) (`dplyr::left_join()`). Since there are multiple possible matches in some cases, the user still must verify and assign a proper EPSG code manually. It is required that all files feature the same CRS, i.e., a single EPSG code will be assigned to all raster layers.

4.3. Track Creation

At this point the user can start to build a track using the previously uploaded CSV file. Therefore, the longitude (x-coordinate), latitude (y-coordinate)

and timestamp fields need to be assigned using the drop-down menus. If this step is completed a reactive expression that subsets the CSV file and omits NA's will be triggered (`shiny::validate()`).

Once again, an EPSG code needs to be assigned, however, this one defaults to the EPSG code assigned to the map in the previous tab. Typically, the tracking data is three-dimensional as opposed to the map of environmental covariates which is two-dimensional. For this very reason we need to transform the tracking data into a two-dimensional representation to match the map.

Following this, the subseted data frame will be passed to another reactive expression which eventually constructs a track (the basic building block of the `amt` package) using `amt::make_track()`. In the course of this, the CRS will be set using the EPSG code entered in the data upload tab (`sp::CRS()`). In a next step the CRS will be transformed accordingly using the map's EPSG code (`amt::transform_coords()`). In case the tracking data does not need to be transformed as this step was done externally beforehand the entered EPSG code in the track tab simply needs to be equal to the one assigned in the data upload tab. Therefore, the transformation will be skipped. The resulting data frame will be displayed within the first on page sub tab. Radio buttons allow to switch to column statistics of the tracking data frame.

The second on page sub tab features summary statistics of the track's sampling rate, i.e., the time intervals in minutes between successive locations of the tracked animals. Therefore, a different reactive expression utilizes `amt::summarize_sampling_rate()`.

Up to this point the tracking data was not grouped, i.e., we cannot distinguish individual animals or say differences caused by seasons. Therefore, the ID drop down allows to assign a column that will be used to group the data. The columns already assigned are removed from the options as it wouldn't make any sense to include them but also to prevent the app from freezing as this happens if one accidently assigned one of these columns. If an ID column is assigned a second drop-down menu pops up below which allows to select

multiple values of a given ID by default all values are chosen. The track data frame will be updated accordingly. As well as the date range input, its minimum and maximum values are adjusted dynamically dependent on the chosen ID values (`shiny::dateRangeInput()`).

Consequently, a structure starting with the track creating reactive expression is introduced which in principle recurs for most of the subsequent reactive expressions. This structure is characterized by an `if else` statement that evaluates whether the data is grouped by an ID and if that is the case a distinction between multiple chosen IDs and a single one is made.

We use a different data structure when grouping by an ID and including multiple ID values in contrast to including a single ID value or creating a track not grouped by an ID. Instead of a common data frame we create data frames with list-columns as introduced to `R` through the `tidyverse` package-family [9]. A list-column is a regular data frame column where each cell stores a list. Therefore, each row of the data frame stores the data associated with an ID in a list. Analogous, the created track and model output will be added in separate list columns. The structure is implemented using the `purr::nest()` command where we nest the columns relevant to create a track excluding the ID column so this column basically serves as unique row names to the list-column's cells where the number of rows equals the number of distinct ID values for e.g. the number of individual animals by default the list column is named "data". This time, we vectorize to create a track using `base::lapply()` combined with an anonymous function. The track is added to the data frame as a second list-column named "track" (`dplyr::mutate()`). The CRS transformations will be done within this step if applicable.

The applied data structure is supposed to increase efficiency, however, is not easy to interpret for the user. For this very reason, the data frame displayed to the user in case of multiple ID values selected is the unnested version of the data frame which will not be used for any other purpose (`purrr::unnest()`). The summary of the track's sampling rate located within the second on page sub tab needs to be unnested as well, as it is based on the previously created

tracking data frame with its list columns.

The final step concerning the track creation tab focuses on resampling the track. Therefore, two numeric inputs allow the user to enter a resampling rate and a corresponding tolerance both in minutes (`shiny::numericInput()`). The sampling rate statistics serve as a guidance to find a balance considering the time intervals. The resampled track will have the same distance in minutes between successive locations of individuals. To mitigate the loss of observations it is recommended to add a sufficiently large tolerance. Again, we vectorize in case of multiple ID values chosen and apply `amt::track_resample()` to resample the track. As a result, a new column will be added to the displayed track data frame called "Burst". The column indicates how many successive observations (subsets of the track) exist given the selected resampling rate including tolerance. A burst ends if the resampling rate cannot be met for an consecutive observation.

4.4. Add Covariates

Given the user has created a track and resampled it he can proceed with the Add Covariates tab, if he did not resample the track a message informs him to complete that step first. Before adding additional covariates, the user can restrict the minimum number of relocations per burst following the resampling in the previous tab. The default is set to three because in case of building a model using an integrated step selection function (iSSF) at least three observations per burst are required to calculate a turn angle. However, some IDs possibly cannot meet this requirement. Naturally, increasing the value makes this more likely. To inform the user about the effect on ID(s) or the whole track in case of none ID assigned a table is displayed below showing the current number of bursts remaining per ID and the corresponding number of observations "n". Holding the minimum number of relocations per burst constant the number of bursts is also dependent on the resampling rate and its tolerance. Therefore, the user may go back to the previous tab and adjust the resampling rate instead or additionally to the restriction of the bursts.

The underlying logic applied that is embedded in a reactive expression consists of a for-loop that iterates through the ID(s). As the number of IDs is expected to be rather small vectorizing doesn't improve the performance significantly but would worsen readability. We iterate through the list-column called "track" introduced above where in each cell a list containing the resampled tracking data associated with an ID is stored. We are interested in the observations per burst. Therefore, we use `base::table()` on the track's burst column. We proceed by checking whether a burst exhibits a number of observations that is greater or equal to the restriction entered by the user. Then we simply determine the number of bursts remaining and aggregate the number of observations over all bursts.

If zero bursts remain, the user will be informed via a notification and provided with instructions how he may proceed in addition to the result shown in the table. In case of multiple IDs it's a warning type message in case of a single ID or none it's an error type message as model fitting is not possible (`shiny::showNotification()`).

Subsequently, the user may edit the environmental covariate(s) he uploaded in the second tab. The displayed handsontable differs from the rest of the tables as it allows the user to modify the content (`rhandsontable::rhandsontable()` and wrapper function refreshing the UI `rhandsontable::renderRHandsontable()`). Therefore, the functionality is quite similar to an excel sheet and the user may copy and paste from and excel document. By default, all covariates are converted to categorical, however, the user can tick off the corresponding checkbox of the column called "Categorical" to treat the variable as continuous. How the conversion is implemented will be outlined below.

Initially, on the server side a data frame wrapped by a reactive expression will be created and filled with a column containing the raster layer name(s) provided by the previously uploaded map(s) as well as the checkbox-column containing the same logical (`TRUE`) repeatedly. An `if else` statement evaluates whether handsontable input exists. If it does the input entered by the user replaces the initial data frame. As the handsontable is based on JavaScript (JS)

we need to convert the input to an R data set (`rhandsontable::hot_to_r()`). Another reactive expression which is used to pass the actual raster layer(s) to subsequent code sections updates changes to the covariate(s) name(s) induced by the handsontable accordingly.

After editing the environmental covariate(s) the user may proceed adding a time of day covariate. This is applicable if one intends to fit an iSSF only. Two options are provided "Excl. dawn and dusk" and "Incl. dawn and dusk". The former divides a day into the categories "day" and "night" the latter performs a division into the categories "dawn", "day", "dusk" and "night". If one of the categories is selected a table will be shown right below informing the user if all factor levels exist for the select ID(s) excluding ID(s) that do not meet the restrictions on bursts as those will be removed before model building. This is particularly relevant as the time of day variable is supposed to be used to create interaction terms for e.g. in combination with the variable "step length" (sl_). However, interaction terms can be applied only to factors with 2 or more levels. Hence, the user receives an error type notification and needs to decide if he wants to remove an affected ID or alternatively adjust the restrictions on bursts and/ or resample the track differently as those specifications can impact the number of factor levels of the time of day covariate.

4.5. Model Building

4.5.1. Model Preparation

Before fitting a model, we first need to implement the properties specified in the previous tab. Therefore, we use a reactive expression consisting of nested `if else` statements. The main structure follows the pattern introduced above, i.e., checking whether multiple IDs are selected as opposed to a single ID or none assigned at all where the latter two cases are treated as one. In each case `if else` statements check for the conditional logit function type the user needs to choose as a first step in the modeling tab the options are "Resource Selection Function" (RSF) and "Integrated Step Selection Func-

tion" (iSSF). If an input was chosen and a resampled track exists, the reactive expression will be triggered.

If multiple IDs are selected, we start by removing those IDs that do not meet the minimum number of relocations per burst from the resampled track which corresponds to a single row per ID as the track is stored as list. If the RSF is selected, the inputs for variable selection and interaction terms will be displayed as well as an input for the number of random points by default set to a value of 100. A constraint (`shiny::validate()`) makes sure this input is provided before starting model preparation. As the random number generator will be applied in this section we set a seed to ensure reproducibility. This applies to all other subsections of different branches, i.e., the order of user inputs does not impact the results.

The model preparation is added to the data frame containing the resampled track as a new list-column called "points" referring to the random points function (`amt::random_points()`) applied within this step. Here, we only retain bursts that meet the minimum number of relocations per burst for the remaining IDs (`amt::filter_min_n_burst()`). Next, the random points within an animal's home range are generated eventually. Following this, we extract the environmental covariate values at relocations (`amt::extract_covariates()`). Those steps are combined using the pipe operator and vectorized in combination with an anonymous function as mentioned above.

Now that we added the environmental covariates, we can convert them to categorical variables if applicable or convert them back into continuous variables. The underlying logic consists of a nested for-loop with two levels. Again, the efficiency advancement of vectorizing this step is quite low as we iterate through IDs in the outer loop and iterate through environmental covariates in the inner loop which are even expected to be clearly single-digit for most applications. Therefore, we rather choose to preserve the readability. Within the nested loops we evaluate by an `if else if` statement whether the handsontable's checkbox-column "Categorical" is `TRUE` for a covariate and if it is stored as numeric in the "points" list-column's cell. Given both

requirements are met we access the column and convert it to a factor. If that does not apply, we check whether the checkbox is not ticked off and the covariate is stored as a factor. Given that is accurate, the covariate will be converted to a numeric variable. To increase performance, we do not convert the single values but the factor levels to numeric.

The model preparation for the iSSF given multiple ID values chosen follows the same structure. However, we add a new list-column to the data frame containing the resampled track named "steps" as we convert from a point to a step representation of the tracks (`amt::steps_by_burst()`) and generate random steps for each observed step (`amt::random_steps()`) where the default is set to three but the user can adjust that number using the numeric input. Additionally, we add two transformed variables the logarithm of step length (log_sl_) and the cosine of the turn angle (cos_ta_).

Another difference is, that the environmental covariates are extracted at the beginning and at the end of a step, i.e., two columns per covariate are added to a track. Therefore, the conversion into categorical or continuous needs to be duplicated within the nested loops. By default, the two columns are named in the pattern "covariate_start" and "covariate_end". Hence, to access these columns we need to add the "_start" and "_end" part (`base::paste0()`) to a given covariate name, i.e., the corresponding raster layer's name we obtain from the respective reactive expression introduced earlier.

Since the time of day covariate can be added to an iSSF model a second branch includes the same steps as before but is extended by `amt::time_of_day()` to add the categorical variable with its two options as mentioned above.

The branch of the outer `if else` statement for a single ID or none is constructed analogously, however, slightly simplified. Therefore, at this stage the model preparation is finished. During the process the user will be informed by a progress bar notification.

Before fitting a model, we need to pass some of this information to the inputs on the modeling tab. The select variables drop down receives the list

of variables by a reactive expression that extracts the column names from a list-column's cell either the column "points" (RSF) or "steps" (iSSF) in case of multiple IDs and an ordinary data frame for a single ID or none (`base::colnames()`). Since those data frames contain quite a few variables generated that are not of interest for the model building we exclude those in a different reactive expression before passing the variable names to the drop-down menu's input as well as the interaction terms inputs. Concerning the latter, a maximum of 2 items can be chosen.

The user's inputs need to be combined to an actual mathematical formula that can be passed to the model fitting functions. Therefore, a reactive expression concatenates the inputs. The expression will be triggered when the user has entered variable(s). These will be collapsed by a plus sign. Each interaction term is build buy testing with an `ifelse()` statement whether the input is of length two, i.e., whether it contains two variables. If that applies the variables will be concatenated to the characteristic form "variable_1:variable_2" preceded by a plus sign. If that does not apply an empty string is returned. In the end all strings are concatenated to a single one.

4.5.2. Model Fitting

Finally, we can proceed to fit a model by simply clicking on a button named accordingly. A progress bar indicates that the fitting procedure has started. The model output will be rendered in a table that shows the estimates, their standard error, t-statistic, p-value and in case of the iSSF also the 95% confidence intervals grouped by ID if applicable.

This process is implemented as follows, the fit-button is monitored by two instances. The first one triggers the model fitting reactive expression (`shiny::eventReactive()`). The second one overwrites a reactive value that stores the current model state, similar to the reactive values used for the upload tabs. It changes the state from its initial value (`NULL`) to "fit". At this stage the latter wouldn't be necessary. If we intend to fit a new model, we can press

the clear-button. That resets all inputs to their default. However, it only affects native Shiny inputs, i.e., excludes action buttons, shiny widgets as well as rendered tables. On the UI side we are obligated to apply the function `shinyjs::useShinyjs()` to enable the following steps. We create a wrapper around all resettable inputs (`tags$div()`) and define an ID for its content. On the server side we can access this section of code using `shinyjs::reset()` and the ID as argument wrapped by `shiny::observeEvent()` which observes the clear-button. Since this cannot clear the table rendering the model output we assign the model state "clear" to the reactive values above. The model state is validated within the model's render table function which will be triggered only if it matches "fit" otherwise it does not return any output, i.e., the latest output vanishes.

The actual fitting part is wrapped in a reactive expression that will be triggered if a model type is selected and the model preparation equivalent is loaded. The `if else` structure corresponds to the model preparation without the distinction concerning the time of day covariate.

The RSF for multiple IDs adds a new list-column called "fit". The input is once again the "points" list-column. Within the `dplyr::mutate()` we apply `purrr::map()` this time instead of `base::lapply()` to vectorize to make use of its operators to keep the model fitting code in a compact form. The fitting procedure is implemented by `amt::fit_rsf()` which is a wrapper around `stats::glm()` for piped workflows. As input we paste the selected variables combined to a mathematical formula, preceded by the hardcoded dependent variable "case" it is one for observed steps and zero for random steps. This term is wrapped by `stats::as.formula()` to turn the string in an object of class `formula` that can be evaluated. As a final step, the relevant part of the model output (`broom::tidy()`) will be added to the data frame as a list-column called "coef" which in turn will be unnested so the user obtains a clean data frame grouped by ID (`purrr::unnest()`).

The iSSF part follows the same pattern. Analogously, the list-column "steps" will be accessed and we apply `amt::fit_issf()` which is a wrapper around `survival::clogit()` to fit the model, however, we extend the hard coded

part of the formula by adding `strata(step_id_)` on the right hand side (`survival::strata()`). A stratum includes an observed step and its matched random counterparts where the number of the latter can be adjusted by the user. The column "step_id_" which identifies different strata is automatically added during the model preparation process by `amt::random_steps()`.

4.5.3. Saving Results & Inputs

`amtGUI` offers ways to retain the results of an analysis as well as the inputs for later reproduction. Saving results is achieved by implementing a simple download handler (`shiny::downloadHandler()` into the modeling tab, which pulls the data frame from the bottom of the page and writes a CSV file named "model_estimates.csv" using the built-in `write.csv()` function, which then can be saved to the users computer. User reports are also created by a download handler in combination with a parameterized R Markdown file called `report.Rmd`. The (empty) parameterized R Markdown file is in the apps directory. Once a user presses the "Download Report" button of the app, the download handler will create a copy of `report.Rmd` in a temporary directory (ensuring the download works even if the user lacks writing permission for the working directory) and all relevant input parameters of the user are pulled from the app and saved (in a new global environment) to a list called `params` before being passed to the Markdown file, which then will immediately start knitting and return an HTML file named "report.html", which can be then saved to the users computer. Generating user reports works at any time from within the app, even if the user hasn't completed his model.

4.6. Interactive Map

The interactive map of the `amtGUI` app is based on the R package `Leaflet` in a broader sense. The spatial data is combined with the world map. Thus it is possible to classify the data geographically and to visualize the respective places where the measuring points were recorded. The `amt` package has

a configured function which is based on `Leaflet` called `inspect()`. Note that unlike `leaflet()`, `inspect()` needs a data frame which was created by `make_track()` and contains information about the EPSG code. If you visualize your data with leaflet you only need longitude and latitude, which can be added by `addMarker()`. To look at each animal individually select the ID in the section "Create Track" and go back to "Interactive Map". It is also possible to view all animals at once (without information about the respective IDs). This includes the sphere of influence of the total population or the whole sample. This can be done by not assigning an ID column in the "Create Track" tab.

5. Conclusion

This term paper lays out the foundations of animal telemetry data analysis. We introduced resource selection and (integrated) step selection functions and demonstrated how our app can be used to fit RSF's and iSSF's.

With `amtGUI` users without extensive backgrounds in statistics or `R` can analyze animal movement data by sampling animal tracks and fitting RSF's or the recently introduced iSSF's respectively. The ability to choose between including environmental covariates at the beginning or end of each step makes `amtGUI` viable for movement or habitat analysis respectively. By letting the user control the whole track creation process, analysis with `amtGUI` is highly customizable. We hope that this will facilitate the use of iSSF's in animal movement analysis and make it more accessible to researchers, foresters and ecologists worldwide.

However, even though `amtGUI` makes it simple to fit models, the user needs to have at least some theoretical knowledge to prevent misfitting and wrong conclusions about environmental covariates. `amtGUI` also relies heavily on `amt`. Changes in `amt`'s codebase likely will affect `amtGUI`.

Moreover, `amtGUI` can be developed further. If the app evolves and becomes even more customizable in the future, it could benefit from another

upload/download tool, where users can, in addition to creating a user report, download their results in a file which can be re-uploaded for later use to further ease the reproduction process.

References

Tal Avgar, Jonathan R Potts, Mark A Lewis, and Mark S Boyce. Integrated step selection analysis: bridging the gap between resource selection and animal movement. *Methods in Ecology and Evolution*, 7(5):619–630, 2016.

Tal Avgar, Subhash R Lele, Jonah L Keim, and Mark S Boyce. Relative selection strength: Quantifying effect size in habitat-and step-selection inference. *Ecology and evolution*, 7(14):5322–5330, 2017.

Thierry Duchesne, Daniel Fortin, and Louis-Paul Rivest. Equivalence between step selection functions and biased correlated random walks for statistical inference on animal movement. *PloS one*, 10(4):e0122947, 2015.

Ludwig Fahrmeir, Thomas Kneib, Stefan Lang, and Brian Marx. *Regression: models, methods and applications*. Springer Science & Business Media, 2013.

Scott LaPoint, Paul Gallery, Martin Wikelski, and Roland Kays. Animal behavior, cost-based corridor models, and real corridors. *Landscape Ecology*, 28(8):1615–1630, Oct 2013. ISSN 1572-9761. doi: 10.1007/s10980-013-9910-0. URL `https://doi.org/10.1007/s10980-013-9910-0`.

Subhash R Lele, Evelyn H Merrill, Jonah Keim, and Mark S Boyce. Selection, use, choice and occupancy: clarifying concepts in resource selection studies. *Journal of Animal Ecology*, 82(6):1183–1191, 2013.

Johannes Signer, John Fieberg, and Tal Avgar. Animal movement tools (amt): R package for managing tracking data and conducting habitat selection analyses. *Ecology and Evolution*, 9(2):880–890, 2019.

Henrik Thurfjell, Simone Ciuti, and Mark S Boyce. Applications of step-selection functions in ecology and conservation. *Movement ecology*, 2(1):4, 2014.

Hadley Wickham. Tidyverse: Easily install and load'tidyverse'packages. *R package version*, 1(1), 2017.

A. Appendix

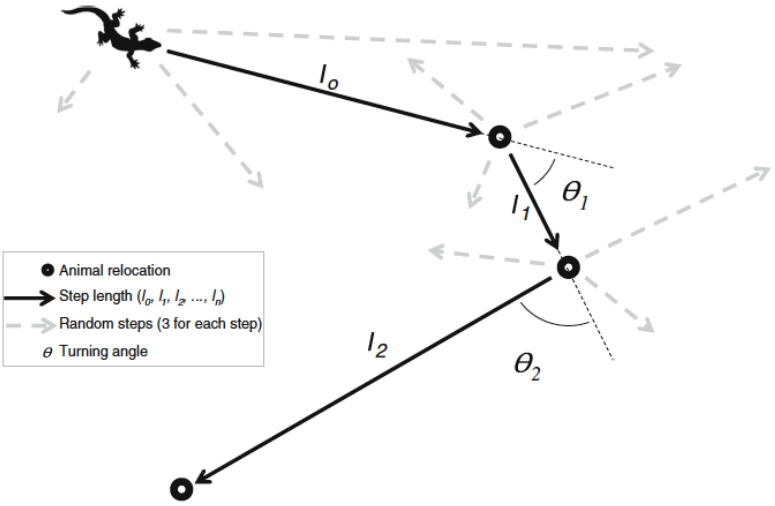

Figure 6: Example of how a movement pathway can be simplified into linear step lengths and turning angles occurring between successive locations in any type of animal tracked visually or using VHF or GPS devices. In this example, 3 random steps have been matched with actual steps walked by the lizard. Graphic from [8]

`amtGUI` User Report

This user report serves as a simple way to reproduce the results of previous `amtGUI` sessions. Simply save or print this file and reproduce your previous analysis using the input values below.

Uploaded dataset

```
## NULL
```

Uploaded environmental data

```
## NULL
```

EPSG code assigned to csv

```
## [1] "4326"
```

EPSG code assigned to map

```
## [1] "5071"
```

EPSG code used in track creation

```
## [1] "5071"
```

Selected input for x-coordinates (longitude)

```
## [1] "longitude"
```

Selected input for y-coordinates (latitude)

```
## [1] "latitude"
```

Selected input for timestamp

```
## [1] "timestamp"
```

Selected input for ID

```
## [1] "id"
```

Selected ID's for track creation

```
## [1] "M4" "M5"
```

Resampling rate

```
## [1] 10
```

Tolerance

```
## [1] 2
```

Chosen date range

```
## [1] "2010-02-09" "2011-05-29"
```

Number of bursts set

```
## [1] 3
```

Environmental covariates

```
##       Name Categorical
## 1 land_use        TRUE
```

Time of day included?

```
## [1] "Incl. dawn and dusk"
```

Model type

```
## [1] "Integrated Step Selection Function"
```

Number of random steps used (iSSF)

```
## [1] 3
```

Number of random points used (RSF)

```
## NULL
```

Model covariates

```
## [1] "land_use_end" "sl_"          "cos_ta_"
```

Number of interaction terms

```
## [1] 2
```

Interaction terms used

```
## [[1]]
## [1] "sl_"      "tod_end_"
##
## [[2]]
## [1] "cos_ta_"  "tod_end_"
```