A tutorial for Step Selection Function

P. Antkowiak[*] H. Tripke[†] C. Wilhelm[‡]

November 25, 2014

Contents

[*] M.Sc. programme "GIS und Umweltmodellierung" at University of Freiburg
[†] M.Sc. programme "Wildlife, Biodiversity and Vegetation" at University of Freiburg
[‡] M.Sc. programme "Wildlife, Biodiversity and Vegetation" at University of Freiburg

1 Introduction

In addition to Resources Selection Functions (RSF) a more detailed anlysis for telemetry data can be conducted by using Step Selection Function (SSF). The use of latter is providing answers to th actual selection of animals on their habitat rather than analysing the use of a habitat only. (DO WE WANT TO CITE, HERE FOR EXAMPLE Simone and friends). So far most of the SSF were done in GIS to use spatial data together with mathematics equations. However, more and more packages are provided in R and thus becomes a valuable alternative. This tutorial provides an overview on how to implement SSF with R. The main package we will use in the following is the package **adehabitatLT**. All single steps that need to be taken care of are summarized in (Figure 1). In our tutorial we use telemetry data from Cougars/Mountain Lion (*Latin name*) collected by Simone Ciuti[1]. As spatial parameters x tables for ruggedness, slope, canopy cover etc. are available. Describe study area...

$$w(x) = exp(\beta x1 + \beta x + ... + \beta)$$

Why did we not use the data (Wildboar) prepared for the adehabitatLT package? - No information on details, metadata provided, it is hard to understand when to use which dataset and why.

2 Preparations

Before you can actually start using the tutorial for conducting SSF you need to load a bunch of packages in R. Some of them require others so that you have to add all these to your library:

2.1 Packages - what we need

```
# installing packages -----------------------------------------------------
## for implementing SSF
# install.packages("adehabitat") # outdated version, not needed for this tutorial
install.packages("adehabitatHR")
install.packages("adehabitatHS")
install.packages("adehabitatLT")
install.packages("adehabitatMA")
install.packages("tkrplot")
install.packages("hab", repos = "http://ase-research.org/R/") # regular
install.packages("hab", repos = "http://ase-research.org/R/", type = "source") # for self-c

# for handling ratser data
install.packages("move")
install.packages("raster")
install.packages("rgdal")
#install.packages("")

# loading the packages
# require(adehabitat) # keep fingers off this package. It is outdated.
require(hab)
require(adehabitatMA)
require(adehabitatHR)
require(adehabitatHS)
```

[1]we might have to specify that and name and thank the institute.../collected/containing

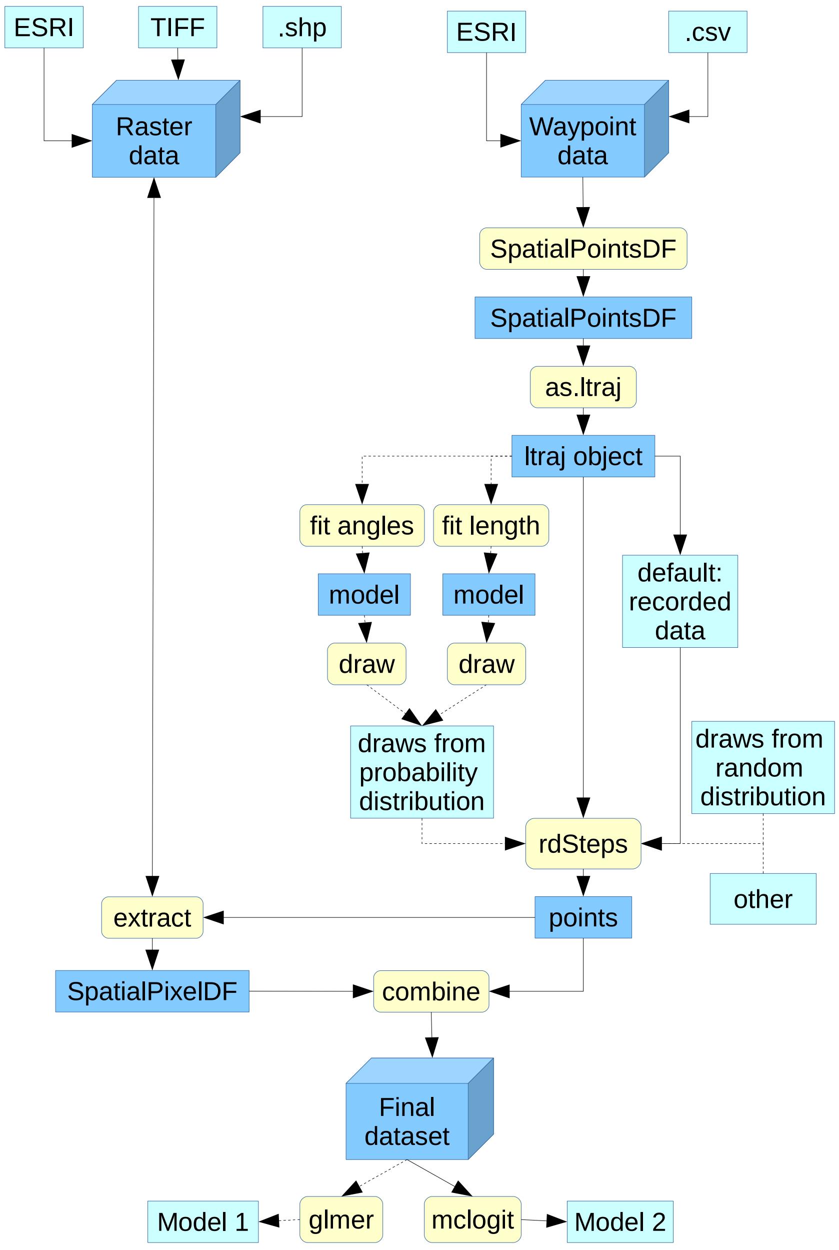

Figure 1: *Conducting a SSF using existing R-packages:* this figures provides an overview of the steps necessary to conduct a SSF. The steps are separated in subsections, which the turtorial will guide you through.

```
require(adehabitatLT)


## for i dont know

#require(move)
#require(raster)
#require(rgdal)
#require(tkrplot)
#require(raster)
#require(sp)
```

3 Loading telemetry data (*.csv, ESRI)

The data for the analysis should be saved in a simple *.csv file format. The table should have column headings in the first line and for each observation include at least the following values:

1. x-coordinate (easting)
2. y-coordinate (northing)
3. date and/or time
4. animal ID

Note that the coordinates need to be provided in the same coordinate system and spatial projection as the raster data.

Depending on your analysis you can include further values such as:

1. ID for each record
2. GPS precision
3. other recording parameters such as season
4. temperature / elevation at the moment of record
5. other values that might be of interest in the further analysis

Use the following commands to set your working directory and read the data:

```
setwd("/your/working/directory")
xmpl = read.csv("xmpl.csv", head=T)
```

You can execute `head(xmpl)` and `str(xmpl)` to check, whether the data were successfully read.

4 Creating a Spatial Points Data Frame

The functions used along the rest of the toolchain can only process data that are stored as objects of class ”SpatialPointsDataFrame”. This object class stores the coordinates separately and can be created using the according function from package ”sp”.

```r
require(sp)

xmpl.spdf = SpatialPointsDataFrame(coords = xmpl[,c("easting","northing")], data = xmpl)

names(xmpl)
```

In the "coords" argument of the function you should specify the coordinate vectors for your dataset. In our example, we simply assign the two coordinate columns of the example dataset, but you can read the coordinates from a separate file if you want.

5 Creating an ltraj object

After storing the data in a Spatial Points Data Frame, you now need to connect the single points and turn them into a set of trajectories. This operation is carried out by the function `as.ltraj` from the "hab" package and produces objects of class "ltraj". The function `as.ltraj` requires at least three arguments to work:

1. "xy" (x- and y- coordinates for each point)
2. "date" (timestamp for each point, given as POSIXct class)
3. "id" (the animal id)

Both coordinates and animal id can easily be adopted from the Spatial Points Data Frame. The timestamp however, is not stored in the required format yet and you therefore need to convert it first:

```r
date <- as.POSIXct(strptime(paste(xmpl.spdf$LMT_DATE, xmpl.spdf$LMT_TIME), "%d/%m/%Y  %H:%M
```

It is necessary to combine date and time in one POSIXct value. If your dataset already features a POSIXct timestamp, you can skip this step.

Now you can proceed and actually create the ltraj object by executing the following command:

```r
xmpl.ltr <- hab:::as.ltraj(xy = xmpl.spdf@coords, date = date, id = xmpl.spdf$cat)
```

Two comments to the function as used: By typing `hab:::as.ltraj` you tell R to use the `as.ltraj` function from the "hab" package which is speed optimized against its `adehabitatLT` sibling. Unlike the "xmpl.spdf@coords" promt which works for any SPDF object, the `xmpl.spdf$cat` prompt is specific to your dataset. In the example dataset, animal ID's are stored as an integer vector called "cat". If this differs in your dataset and you should change the prompt accordingly.

You now may want to have a closer look at the created ltraj object. Display its structure with by executing `str(xmpl.ltr)`.

```r
str(xmpl.ltr)
```

When scrolling through the output you will first notice that it consists of 7 elements - the number of individuals in the example dataset. This is because `as.ltraj()` automatically splits the dataset into subsets one for each individual. We will refine these subsets in the next chapter. Furthermore, the ltraj contains information on the distances and turning angles between consecutive locations. To get a visual impression of your data you can plot the trajectory for all or for one particular animal:

```
plot(xmpl.ltr)
unique(xmpl.spdf$cat) # prompts a list of all cat ID's. Choose one that you are interested
plot(xmpl.ltr, id=10289)
```

6 Creating bursts

7 Compute random steps

The function **rdSteps** removes the first and the last data point. That's what you want.

8 Spatial covariates

This section explains the handling of spatial parameters that will be tested for selection by the target species. You should store these data in raster files (ESRI *.adf or georeferenced *.tif). These should have the same coordinate system as your telemetry data and should (for time reasons) already be clipped to your study area and. For instructions how to do this in R, please read the GIS instructions from the other group ;)

8.1 Load raster data (ESRI, *.tif, (*.shp))

With a simple function stored in the package **raster** you are able to upload any raster file into R. Examplarily we use raster data on the following parameters for the study area:

1. ruggedness of the terrain
2. land cover
3. canopy cover
4. distance to the nearest highway
5. distance to the nearest road

For reading the raster data, three packages are required:

```
require(raster)
require(rgdal)
require(sp)
```

The source files for the raster data can be stored in the working directory or loaded by specifying the exact path. The `raster()` function is a universal and very powerful tool for loading all kinds of raster data. For reading shapefiles, use the `readOGR()` function. Below is an example of how to read a set of raster layers.

```
# First, make sure that your working directory is still the one specified earlier:
getwd()

# Now read the layers:
ruggedness <- raster("ruggedness.adf")
landcover <- raster("landcover.adf")
canopycover <- raster("canopycover.adf")
disthighway <- raster("disthighway.adf")
distroad <- raster("distroad.adf")
# In the ESRI directory system, raster layers are usually stored in files called "w001001.a
```

You can plot the data for a first overview. As this can take a while with large datasets, outcomment the following chunk if you please!

```r
plot(ruggedness)
plot(landcover)
plot(canopycover)
plot(disthighway)
plot(distroad)
```

8.2 Raster extraction

Now that you have generated the random steps and loaded the raster data, you can take the next step and actually connect the trajectories with the spatial covariates. There are different functions that can do this. When choosing one, you need to consider that raster files are large and juggling with them occupies lots of memory and computing power. For this reason we suggest using the `extract()` function that allows for querying single pixel values without loading the whole source file into working memory. The code for compiling the final dataset involves three steps: Converting the `xmpl.steps` data frame into a Spatial Points Data frame, extracting the raster values and combining them to the final dataset. Converting `xmpl.steps` into a SpatialPointsDataFrame:

```r
xmpl.steps.spdf <- SpatialPointsDataFrame(coords = xmpl.steps[,c("new_x","new_y")], data = x
```

Extracting the values from each raster layer:

```r
ruggedness.extr <- extract(ruggedness, xmpl.steps.spdf, method='simple', sp=F, df=T)
canopycover.extr <- extract(canopycover, xmpl.steps.spdf, method='simple', sp=F, df=T)
disthighway.extr <- extract(disthighway, xmpl.steps.spdf, method='simple', sp=F, df=T)
distroad.extr <- extract(distroad, xmpl.steps.spdf, method='simple', sp=F, df=T)
landcover.extr <- extract(landcover, xmpl.steps.spdf, method='simple', sp=F, df=T)
```

The extraction is done separately for each layer. The option `method = 'simple'` extracts value from nearest cell whereas `method = 'bilinear'` interpolates from the four nearest cells. You can adjust this option according to the resolution of your dataset and ecological considerations. `df=T` returns the result as a data frame and `sp=F` ensures that the output is not added to the original dataset right away.

Automatically adding the extracted values to the original dataset sounds like a handy option. For two reasons we do not use it here: Firstly, we want to set the column names manually for not ending up with several columns called "w001001". Secondly, our data include a categorial covariate (landcover) that we want to reclassify and flag as a factor.

This is the code for compiling the final dataset:

```r
xmpl.steps.spdf$ruggedness <- ruggedness.extr[,2]
xmpl.steps.spdf$canopycover <- canopycover.extr[,2]
xmpl.steps.spdf$disthighway <- disthighway.extr[,2]
xmpl.steps.spdf$distroad <- distroad.extr[,2]

# The landcover covariate comes coded in integers between 0 an 10 and is by default (mis)in
unique(landcover.extr[,2])
# Re-classifying landcover:
xmpl.steps.spdf$landcover <- as.factor(
    ifelse(landcover.extr[,2] == 0,"NA",
    ifelse(landcover.extr[,2] < 5, "forest",
    ifelse(landcover.extr[,2] < 8, "open","NA"))))
```

Now your final dataset should be ready for analysis. Examine it:

```
head(xmpl.steps.spdf)
```

8.3 Extract coordinates for comparison of used and random points

Peter is successfully doing this step!!

9 Final SSF model

10 Acknowledgements

Don't forget to thank TeX and R and other opensource communities if you use their products! The correct way to cite R is shown when typing "`citation()`", and "`citation("mgcv")`" for packages.

11 Appendix / Literature

Session Info:

```
## R version 3.1.2 (2014-10-31)
## Platform: x86_64-redhat-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=de_DE.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=de_DE.UTF-8
##  [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=de_DE.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.7
##
## loaded via a namespace (and not attached):
## [1] evaluate_0.5.5 formatR_1.0    highr_0.4      stringr_0.6.2  tools_3.1.2
```