A tutorial for Step Selection Function

P. Antkowiak* H. Tripke† C. Wilhelm‡

November 19, 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).

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 ------------------------------------------------------

# install.packages("adehabitat")

# install.packages("hab")
install.packages("hab", repos = "http://ase-research.org/R/", type = "source")
# install.packages("hab", repos = "http://ase-research.org/R/")

install.packages("adehabitatMA")
install.packages("adehabitatHR")
install.packages("adehabitatHS")
install.packages("adehabitatLT") # will be installed when insatlling adehabitatHR
install.packages("tkrplot")


require(hab)
require(adehabitatMA)
require(adehabitatLT)  # includes "ade4"
require(adehabitatHS)
require(tkrplot)

# require(adehabitat)  # not necessary to load
```

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

4 Extract coordinates for comparison of used and random points

Peter is successfully doing this step!!

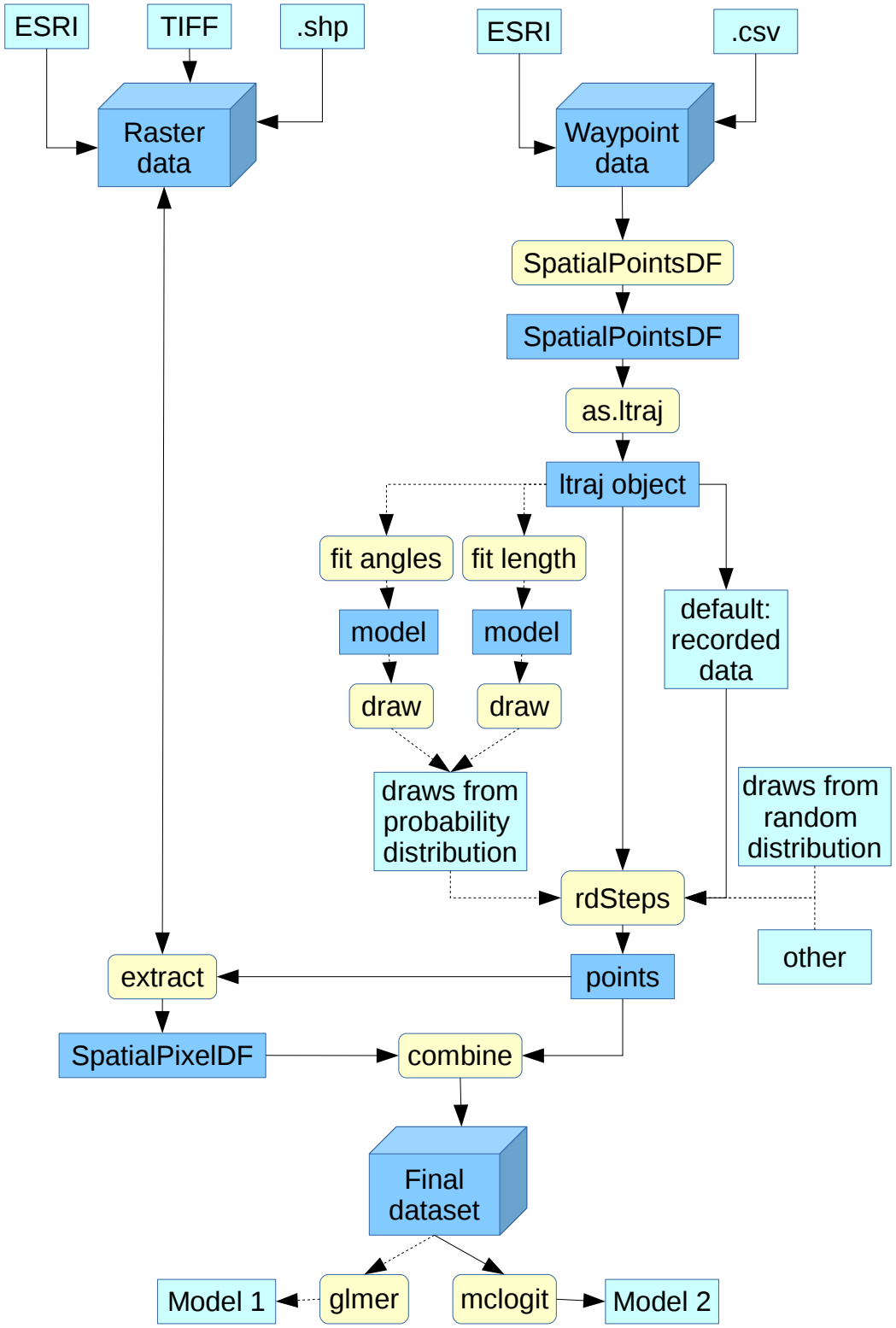


Figure 1: *Conducting a SSF using existing R-packages:* this figures provides an overview of the steps necessary to conduct a SSF. The steps are separated in subsections, which the turtorial will guide you through.

5 Load telemetry data (*.csv, ESRI)

The data for the analysis is should be safed in a simple *.csv format. Depending on your analysis you have to include

1. coordinates
2. ID
3. date and/or time

6 Create a Spatial Points Data Frame

7 Create a ltraj object

8 Compute random steps

9 Final SSF model

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.

10 still useful code from Carsten

I left this in there cause we might need the help when adding our code...

```
# This is how I could document code:
runif(100)
# but it only contains the code and can also run over the line, as shown here ...
# convenient for short, but not for long pieces of code
# any symbol is plotted "as is", even if it is \LaTeX: *+#&ß\/
```

any symbol is plotted "as is", even if it is LaTeX: *+ß

Alternatively, in Sweave, I can actually use R-code and have it evaluated by R and results returned and pasted into the LaTeX-document! Like so:

```
runif(10)
```

```
##  [1] 0.9829789 0.5495859 0.5235186 0.2654087 0.1903815 0.6912136 0.8983265 0.253218
##  [9] 0.5575985 0.7742271
```

Or I can only have the code returned, but **not** evaluated (useful if it takes a long time, or if results are produced externally beforehand):

```
runif(10)
```

Similarly, I can have the code evaluated but the call not returned:

```
##  [1] 0.56906532 0.37986997 0.73346162 0.06215843 0.64431340 0.90812612 0.45775895
##  [8] 0.51576533 0.43007653 0.14327208
```

To combine both, simply add both options, separated by a ",".

You can also put an R-expression into the text. For example, the mean of 10 random number between 0 and 1 is (in this simulation) 0.63, rounded to 3 digits. This value will change every time you compile this document.

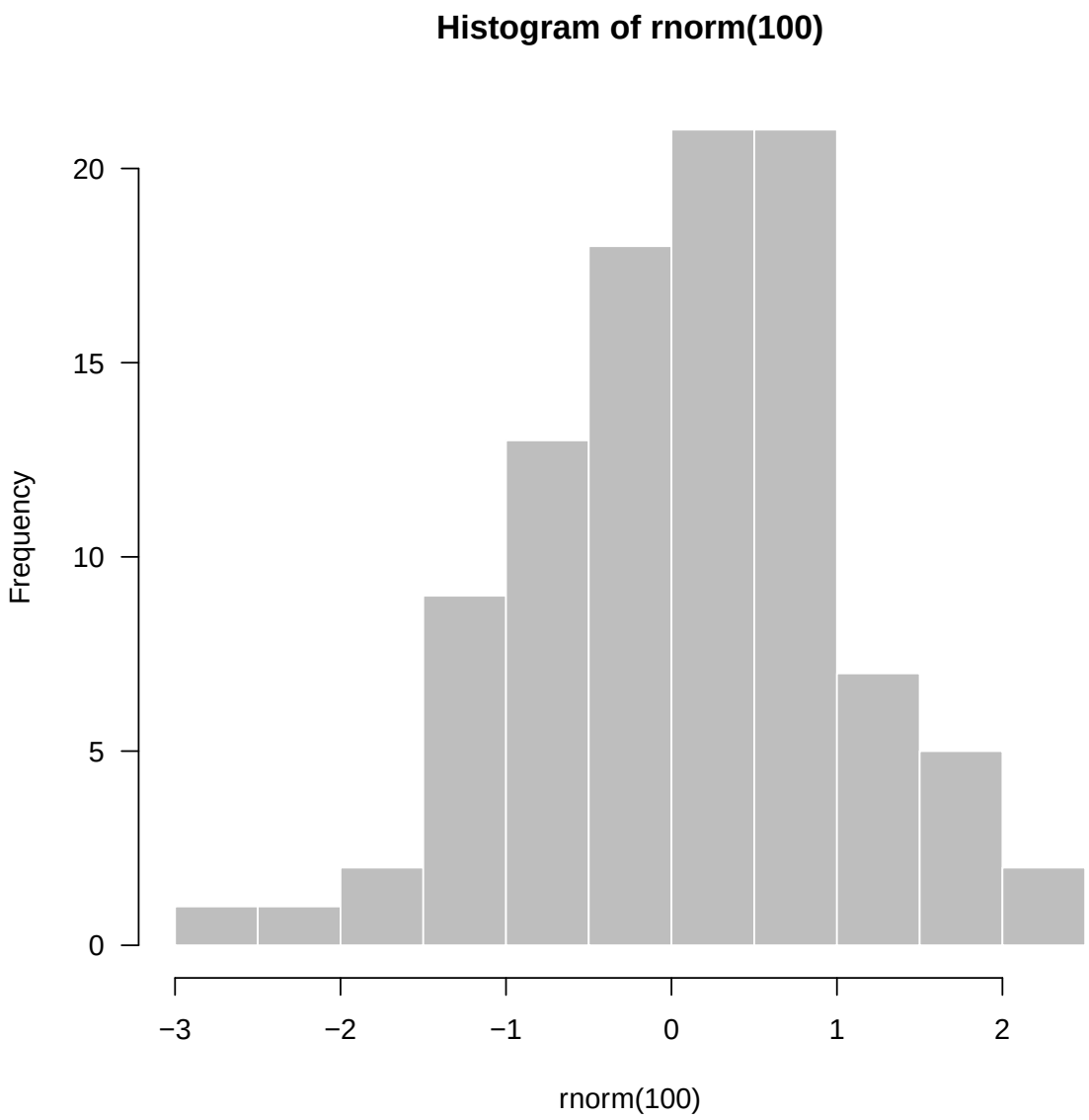
Finally, you can first do the computations, store them (in R objects) and then refer to them in the text.

```
##
##  Pearson's product-moment correlation
##
## data:  X and Y
## t = 1.9196, df = 8, p-value = 0.09117
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1052939  0.8800316
## sample estimates:
##       cor
## 0.5615616
```

So, the correlation between X and Y is not significant, and the mean of Y is 1.

So that's all cool, and the only thing we also could wish for is to produce figures and insert them into the text automatically:

```
hist(rnorm(100), col="grey", border="white", las=1)
```



We may want to "hide" the figure generating text and show the result in a float-environment with a proper caption. To do so, we only have to wrap the above code in a figure environment, like so:

```
\begin{figure}
\centering
#<<label=fig2plot, echo=FALSE, fig=TRUE>>=
hist(rnorm(100), col="grey", border="white", las=1)
#@
\caption{This is an example figure, computed and immediately returned.}
\end{figure}
```

Notice that I only put the `#` in to prevent R from evaluating the code. In a real document, you would NOT do that.

Now one minor issue is that you cannot readily change the size of this figure. The best way to do this is to split the figure-generating process up into two part: one generates the figure and

Figure 2: This is an example figure, exported as PDF and then, on loading, scaled to half the text's width.

saves it as PDF, then we use standard LaTeXto load this figure. Like so:[1]

```
#<<anexamplehistogram, echo=FALSE, dev="pdf">>=
hist(rnorm(100), col="cornflowerblue", border="white", las=1)
#@

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{anexamplehistogram}
\caption{This is an example figure, exported as PDF and then, on loading,
  scaled to half the text's width.}
\end{figure}
```

I leave it to you to do the same thing for tables, using R's xtable command, as illustrated in the Sweave demo of Friedrich Leisch (`users.stat.umn.edu/~geyer/Sweave/foo.pdf`).

All this was Sweave. More recently, knitr made its entrance, and has increased the flexibility of Sweave. It is VERY similar and you have to tell RStudio in the options, whether you are sweaving or knitting. For minimal examples and comparisons, also using markdown rather than LaTeXand PDF or html as output, see here: `http://yihui.name/knitr/demo/minimal/`.

[1] Obviously in the document you would have to remove the #s.