Chapter 10
From Data Management to Advanced Analytical Approaches: Connecting R to the Database

Bram Van Moorter

Abstract The previous chapters explored the wide set of tools that PostgreSQL and PostGIS offer to manage tracking data. In this chapter, you will expand database functionalities with those provided by dedicated software for statistical computing and graphics: the R programming language and software environment. Specifically, you will use the advanced graphics available through R and its libraries (especially '*adehabitat*') to perform exploratory analysis of animal tracking data. This data exploration is followed with two short ecological analyses. These ecological analyses are discussed in the context of two central concepts in animal space use: geographic versus environmental space, and the spatiotemporal scale of the research question. In the first analysis, you investigate the animal's home range, which is its use of geographic space. In the second, to explore an animal's use of environmental space we introduce briefly the study of both use and selection of environmental features by animals. For both demonstrations we consider explicitly the temporal scale of the study through the seasonal changes introduced by seasonal migration.

Introduction: From Data Management to Data Analysis

In previous chapters, you explored the wide set of tools that PostgreSQL and PostGIS offer to process and analyse tracking data. Nevertheless, a database is not specifically designed to perform advanced statistical analysis or to implement complex analytical algorithms, which are key elements to extract scientific knowledge from the data for both fundamental and applied research. In fact, these functionalities must be part of an information system that aims at a proper handling of wildlife tracking data. The possibility of a tighter integration of analytical functions with the database is

B. Van Moorter (✉)
Norwegian Institute for Nature Research (NINA), Høgskoleringen 9, 7034 Trondheim, Norway
e-mail: Bram.van.moorter@gmail.com

particularly interesting because the availability of large amounts of information from the new generation sensors blurs the boundary between data analysis and data management. Tasks like outlier filtering, real-time detection of specific events (e.g. virtual fencing), or meta-analysis (analysis of results of a first analytical step, e.g. variation in home range size in the different months of a year) are clearly in the overlapping area between data analysis and management.

Background for the Analysis of Animal Space Use Data

Two questions need to be answered to move from an ecological question on animal space use to the analysis of those data: first, which is the relevant space (geographic versus environmental space), and second, which is the relevant spatio-temporal scale? Animals occupy a position in space at a given time *t*, which is called geographic space, and many ecological questions are related to this space: e.g. 'How large of an area is used by an animal during a year'? or 'How fast can an animal travel?'. Notably, the question on the area traversed by the animal has received much research interest, and this area is often called a 'home range'. The home range has been defined by Burt (1943) as 'the area traversed by the individual in its normal activities of food gathering, mating, and caring for young'. Many statistical approaches have been developed to use sets of location 'points' to estimate a home range area, from convex polygons to kernel density estimators (and many variants; for a review, see Kie et al. 2010).

On the other hand, by being in a certain geographic location, the animal encounters a set of environmental conditions, which are called environmental space, and questions related to the animal's ecological relationships are to be answered in this space: e.g. 'Which environmental characteristics does the animal prefer'? or 'How does human land use affect the animal's space use?'. These questions are the main topic of habitat selection studies. In general, in these studies, one compares the environmental conditions used by the animal to those available to the animal. An important challenge is to decide 'What environmental conditions were available to the animal'? This issue is tightly linked to the next question to be answered about scale.

The second important question for animal space use studies is about the relevant scale for the analysis. Small-scale studies can focus on the spatial behaviour of animals within a day or even an hour, whereas large-scale studies can look at space use over a year or even an animal's lifetime. The required scale of the study leads to certain demands on the data as well: a small-scale study requires high-resolution collection of precise locations, whereas a large-scale study will require a sufficient tracking duration to allow inferences over this long period. It seems obvious that any description of the area traversed by an individual must specify the time period over which the traversing occurred. Only for stable home ranges is it so that after a certain amount of time the size of the range no longer increases and the area becomes no longer time-dependent. Although often assumed, the stability

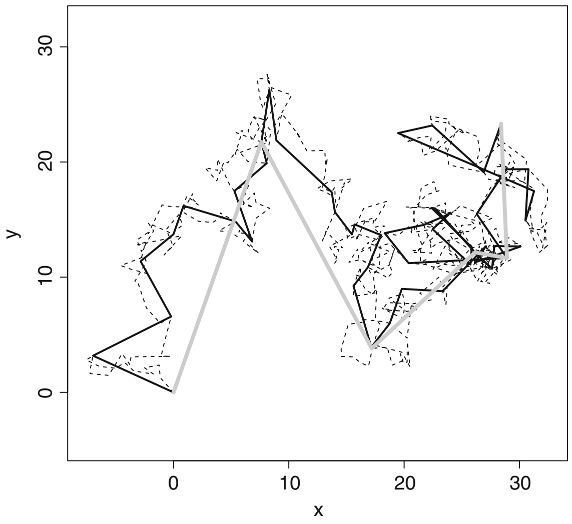


Fig. 10.1 This simulated trajectory clearly illustrates that reducing the number of acquired fixes within a time period can alter substantially the properties of the trajectory. The *dashed line* is the original trajectory, sampling with a ten times coarser resolution leads to a shorter trajectory in *black*, and a further reduction results in an even shorter trajectory in *gray*. Even though the total length of the trajectory decreases with a reduction in resolution, the distance between the consecutive points increases. It is thus very important for the researcher to be aware of such effects of sampling scale on trajectory characteristics

of a home range should be tested as often this is not the case. Also the distance travelled is very sensitive to the scale of the study, see Fig. 10.1.

For habitat selection studies—i.e. studies in environmental space—scale not only affects the sampling of the animal's space use, but even more important also the sampling of the environmental conditions available to the animal to choose from. Areas available to an animal in the course of several days may not be reachable within the time span of a few hours. This behavioural limitation has led several studies to consider availability specific to each used location (e.g. step-selection functions, Fortin et al. (2005)), instead of considering the same choice set available for all locations of a single individual (or even population). Thus, how the researcher defines the available choice set should be informed by the scale of the study.

The Tools: R and Adehabitat

R is an open source programming language and environment for statistical computing and graphics (http://www.r-project.org/). It is a popular choice for data analysis in academics, with its popularity for ecological research increasing rapidly. This popularity is not only the result of R being available for free, but also due to its flexibility and extendibility. The large user base of R has developed an extensive suite of libraries to extend the basic functionalities provided in the R-base package. Before going into some of these really fantastic features, it is good to point out that R's flexibility comes at a cost: a rather steep-learning curve. R is not very tolerant to small human mistakes and demands of the first-time user some initial investment to understand its sometimes cryptic error messages. Fortunately, there are many resources out there to help the novice on her way (look on the R website for more options): online tutorials (e.g. 'R for Beginners' or 'Introduction to R'), books (a popular choice is Crawley's (2005) 'Statistics: An Introduction using R'), extensive searchable online help (e.g. http://www.rseek.org/), and the many statistics courses that include an introduction to R (search the Internet for a course nearby; some may even be offered online).

One of the main strengths of R is the availability of a large range of packages or libraries for specific applications; by 2013, more than 4,000 libraries were published on CRAN. There are libraries for advanced statistical analysis (e.g. '*lme4*' for mixed-effects models or '*mgcv*' for general additive models), advanced graphics (e.g. '*ggplot2*'), spatial analysis (e.g. '*sp*'), or database connections (e.g. '*RPostgreSQL*'). Many packages have been developed to allow the use of R as a general interface to interact with databases or GIS. Other packages have gone even further and increase the performance of R in fields for which it was not originally developed such as the handling of very large data sets or GIS. Hence, many different R users have come into existence: from users relying on specific software for specific tasks who use R exclusively for their statistical analysis and use different files to push data through their workflow, to the other extreme of users who use R to control a workflow in which R sometimes calls on external software to get the job done, but more often with the help of designated libraries, R gets the job done itself. Instead, the approach advocated in this book is not centred around software, but places the data in the centre of the workflow. The data are stored in a spatial database, and all software used interacts with this database. You have seen several examples in this book using different software (e.g. pgadmin or QGIS); in this chapter, you will use R as another option to perform some additional specific tasks with the data stored in the database.

For the analysis of animal tracking data, we often use functions from adehabitat (Calenge 2006), which today consists of '*adehabitatLT*' (for the analysis of trajectories), '*adehabitatHR*' (for home range estimation), '*adehabitatHS*' (for habitat-selection analysis), and '*adehabitatMA*' (for the management of raster maps). For the general management of spatial data, we rely on the '*sp*'- and '*rgdal*'-libraries, for the advanced management of raster data; the '*raster*'-library is becoming the standard. We recommend as a general introduction to the use of spatial data in R the book by Bivand et al. (2008). To install a package with a library in R, you use the *install.packages* command, as illustrated here for the *'adehabitat'*-libraries:

```
# comments in R start with a '#', and what follows will be ignored by R
# to install the adehabitat packages:
install.packages("adehabitatLT")
install.packages("adehabitatHR")
install.packages("adehabitatHS")
install.packages("adehabitatMA")
```

The different adehabitat packages come with extensive tutorials, which are accessible in R through

```
vignette("adehabitatLT")
vignette("adehabitatHR")
vignette("adehabitatHS")
vignette("adehabitatMA")
```

Before using the database to analyse your animal tracking data with R, you can use adehabitat to replicate Fig. 10.1:

```
library(adehabitatLT)
set.seed(0)  #this allows you to replicate
# the exact same 'random' numbers as we did for the figure
simdat <- simm.crw(c(1:501))[[1]]
plot(simdat$x, simdat$y, type = "l", lty = "dashed", xlab = "x", ylab = "y",
     asp = T)
lines(simdat$x[seq(1, 501, by = 10)], simdat$y[seq(1, 501, by = 10)], lwd = 2)
lines(simdat$x[seq(1, 501, by = 100)], simdat$y[seq(1, 501, by = 100)], lwd = 4,
      col = "grey")
```

The *simm.crw* function simulates a random walk. A random walk is a movement where the direction and the distance of each consecutive location are randomised; it is therefore also referred to as a ‘drunkard’s walk’. It is beyond the scope of this chapter to give an introduction to random walks (see e.g. Turchin 1998 for more on random walks to model movement of organisms).

You can find more information on a function with a ‘?’ in front of the function; this will access its associated help pages with explanation and working examples of the function’s uses:

```
`?`(simm.crw)
```

You can see in these help pages that there are several parameters that can be altered to make the random walk behave differently.

Connecting R to the Database

To use R for the analysis of the data stored and managed within the database, there are two approaches: first, connect from R to the database, and second, connect from the database to R with the Pl/R-interface. We will start by demonstrating the first approach and using it for some exercises. In the next chapter, you will see how this approach can be extended to connect from within the database to R with the PostgreSQL procedural language Pl/R (www.joeconway.com/plr/doc/).

First, to connect from R to the database, we make use of the '*RPostgreSQL*' library. It is possible with '*rgdal*' to read spatial features from PostGIS directly in R into '*sp*''s spatial classes. However, using '*rgdal*', it is no longer possible to perform SQL operations (such as SELECT) on these data. One solution could be to create in the database a temporary table with the selected spatial features and then use '*rgdal*' to read the spatial features into R. However, the performance of '*rgdal*' is considerably lower—it can be 100 times slower for certain operations—than for the database libraries, such as '*RPostgreSQL*'. Unfortunately, to date, there is no straightforward way for Windows users to read spatial data into R using SQL statements from a PostgreSQL-database. Thus, when you want to include an SQL statement, you will have to convert the data to non-spatial classes and then subsequently convert them back to spatial features in R. In the next chapter, we will discuss the pros and cons of the use of R within the database through Pl/R.

To connect to a PostgreSQL-database, we use the '*RPostgreSQL*' library. The driver is '*PostgreSQL*' for a PostgreSQL-database as yours. The connection requires information on the driver, database name, host, port, user, and password. Except from the driver, all other parameters may have to be adjusted for your own specific case. If you have the database on your own machine, then the host and port will likely be 'localhost' and 5432, respectively, as shown here. You can see the tables in the database with the *dbListTables* command:

```
library(RPostgreSQL)
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="gps_tracking_db", host="localhost",
                 port="5432", user="postgres", password="********")
dbListTables(con)
## Loading required package:  DBI

##  [1] "lu_species"             "lu_age_class"
##  [3] "gps_data_animals"       "gps_sensors"
##  [5] "gps_data"               "gps_sensors_animals"
##  [7] "spatial_ref_sys"        "meteo_stations"
##  [9] "study_area"             "roads"
## [11] "adm_boundaries"         "srtm_dem"
## [13] "corine_land_cover"      "corine_land_cover_legend"
## [15] "lu_gps_validity"        "ndvi_modis"
## [17] "trajectories"           "animals"
## [19] "home_ranges_mcp"        "test_randompoints"
## [21] "activity_sensors_animals" "activity_data"
## [23] "activity_sensors"       "activity_data_animals"
```

The following code retrieves the first five lines from the gps_data_animals table:

```
fetch(dbSendQuery(con, "SELECT * FROM main.gps_data_animals LIMIT 5;"), -1)

##   gps_data_animals_id gps_sensors_id animals_id    acquisition_time
## 1               28250              6          6 2005-04-08 10:01:24
## 2               28344              6          6 2005-04-18 14:00:47
## 3               28396              6          6 2005-04-27 06:01:54
## 4               26109              1          2 2005-08-01 21:00:30
## 5               26684              1          2 2005-10-15 20:32:17
##   longitude latitude    insert_timestamp    update_timestamp
## 1     10.95    45.97 2013-09-12 18:45:17 2013-10-10 16:08:23
## 2     11.04    46.06 2013-09-12 18:45:17 2013-10-10 16:08:44
## 3     11.10    46.07 2013-09-12 18:45:17 2013-10-10 16:08:44
## 4     11.03    46.03 2013-09-12 09:56:38 2013-10-10 16:08:44
## 5     11.02    46.02 2013-09-12 09:56:38 2013-10-10 16:08:44
##                                                geom pro_com
## 1 0101000020E6100000C054D8B1B6E62540169C6626BDFB4640      NA
## 2 0101000020E6100000D9EBDD1FEF15264043812D65CF074740   22205
## 3 0101000020E610000099BDC7F4DF3226409FD9BFFC5F094740   22205
## 4 0101000020E6100000CA6E66F4A30D2640CA0E3B9D75034740   22101
## 5 0101000020E61000000C186E0A750A2640AF04F7A864024740   22101
##   corine_land_cover_code altitude_srtm station_id roads_dist ndvi_modis
## 1                     21           411          1        119         NA
## 2                     25           884          3       2167         NA
## 3                     25           357          3       1243         NA
## 4                     24          1681          5        372         NA
## 5                     24          1666          5       1173         NA
##   gps_validity_code
## 1                12
## 2                 2
## 3                 2
## 4                 2
## 5                 2
```

You can see that R did not understand the *geom* column correctly.

Now, you want to retrieve all the necessary information for your analyses of roe deer space use. You first send a query to the database:

```
rs <- dbSendQuery(con, "SELECT animals_id, acquisition_time,longitude, latitude,
                  ST_X(ST_Transform(geom, 32632)) as x32,
                  ST_Y(ST_Transform(geom, 32632)) as y32, roads_dist,
                  ndvi_modis, corine_land_cover_code, altitude_srtm
                  FROM main.gps_data_animals where gps_validity_code = 1;"  )
locs <- fetch(rs,-1)
dbClearResult(rs)
```

You then need to fetch those data (with the -1, you indicate that you want all data), and then ‘clear’ the result set. Virtually all spatial operations (such as projection) could also be done in R; however, it is faster and easier to have the database project the data to UTM32 (which has the SRID code 32632).

```
head(locs)

##   animals_id     acquisition_time longitude latitude    x32     y32
## 1          2 2005-03-21 01:03:06     11.07    45.99 660151 5095008
## 2          2 2005-03-21 13:02:19     11.07    46.00 660004 5095733
## 3          2 2005-03-21 21:01:49     11.07    46.00 659954 5095673
## 4          2 2005-03-21 05:01:45     11.07    45.99 660104 5095155
## 5          6 2005-04-05 20:02:48     11.06    46.07 659592 5103359
## 6          6 2005-04-06 04:01:46     11.06    46.07 659631 5103352
##   roads_dist ndvi_modis corine_land_cover_code altitude_srtm
## 1        682       5048                     23          1085
## 2       1139       4314                     25          1378
## 3       1214       4797                     25          1401
## 4        834       5048                     23          1201
## 5       1205       5793                     19           740
## 6       1168       5793                     19           740
```

The *head* function allows us to inspect the first lines (by default six) of a dataframe. You see that you have successfully imported your data into R.

For dates, you should always carefully inspect their time zone. Due to the different time zones in the world, it is easy to get errors and make mistakes in the treatment of dates:

```
head(locs$acquisition_time)

## [1] "2005-03-21 01:03:06 CET"  "2005-03-21 13:02:19 CET"
## [3] "2005-03-21 21:01:49 CET"  "2005-03-21 05:01:45 CET"
## [5] "2005-04-05 20:02:48 CEST" "2005-04-06 04:01:46 CEST"
```

The time zone is said to be CEST and CET (i.e. Central European Summer Time and Central European Winter Time), note that the time zone will depend upon the local settings of your computer. However, we know that the actual time zone of these data is UTC (i.e. Universal Time or Greenwich Mean Time).

Let us then inspect whether the issue is an automatic transformation of the time zone, or whether the time zone was not correctly imported. With the library ‘*lubridate*’, you can easily access the hour or month of a *POSIXct*-object:

```
library(lubridate)
table(month(locs$acquisition_time), hour(locs$acquisition_time))

## 
##        0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
##   1    0 152   0   0  36 118   0  35   0 126  33   0   0 149   0   0  35
##   2    0 121   0   0  33  97   0  31   0  95  34   0   0 116   0   0  42
##   3    0 129  21   0   0 122  23   0   0 127  24   0   0 114  20   0   0
##   4    9   0 168   0  10   0 171   0  11   0 164   0  11   0 156   0  11
##   5    5   1 158   1   6   1 147   1   6   1 122   0   5   1 149   0   5
##   6    0   0 125   0   0   0 107   0   0   0 100   0   0   0 107   0   0
##   7    2   2 144   2   2   2 114   1   1   2 128   2   2   2 110   2   2
##   8   10   8 137   7   8   8 131   7   9   8 126   7   8   8 112   9   9
##   9   10  10 136   7  10   9 142   8   8   7 112   8   6   7  99   4   7
##   10   8  16 161   8   8  23 154   8   8  20 132   8   8  18 128   7   7
##   11   0 181   0   0   0 182   0   0   0 167   0   0   0 166   0   0   0
##   12   4 196   6   6   6 193   5   5   6 178   5   6   5 183   6   5   6
## 
##       17  18  19  20  21  22  23
##   1  118   0  33   0 118  35   0
##   2   95   0  31   0 103  33   0
##   3  130  27   0   0 123  30   0
##   4    0 175   0  12   0 180   0
##   5    1 128   1   5   1 151   1
##   6    0 102   0   0   0 125   0
##   7    2 103   2   0   0 131   1
##   8    7 104   6   8   8 139   9
##   9   10 117   8   9   7 145  10
##   10  17 138   8   7  20 161   8
##   11 179   0   0   0 184   0   0
##   12 197   5   5   6 198   6   5
```

The table shows us that there is a clear change in the frequency of the daily hours between March–April and October–November, which indicates the presence of daylight saving time. You can therefore safely assume that the UTC time in the database was converted to CEST/CET time.

To prevent mistakes due to daylight saving time, it is much easier to work with UTC time (UTC does not have daylight saving). Thus, you have to convert the dates back to UTC time. With the aforementioned *'lubridate'* library, you can do this easily: The function *with_tz* allows you to convert the local back to the UTC zone:

```
locs$UTC_time <- with_tz(locs$acquisition_time, tz = "UTC")
head(locs$acquisition_time)

## [1] "2005-03-21 01:03:06 CET"  "2005-03-21 13:02:19 CET" 
## [3] "2005-03-21 21:01:49 CET"  "2005-03-21 05:01:45 CET" 
## [5] "2005-04-05 20:02:48 CEST" "2005-04-06 04:01:46 CEST"

head(locs$UTC_time)

## [1] "2005-03-21 00:03:06 UTC" "2005-03-21 12:02:19 UTC"
## [3] "2005-03-21 20:01:49 UTC" "2005-03-21 04:01:45 UTC"
## [5] "2005-04-05 18:02:48 UTC" "2005-04-06 02:01:46 UTC"
```

Indeed, the *UTC_time*-column now contains the time zone: 'UTC'. You can run the command '*table(month(locs$UTC_time), hour(locs$UTC_time))*' to verify that no obvious shift in sampling occurred in the data. From personal experience, we know that many mistakes happen with time zones and daylight saving time, and we therefore recommend that you use UTC and carefully inspect your dates and ensure that they were correctly imported into R.

Data Inspection and Exploration

Before you dive into the analysis to answer your ecological question, it is crucial to perform a preliminary inspection of the data to verify data properties and ensure the quality of your data. Several of the following functionalities that are implemented in R can also easily (and more quickly) be implemented into the database itself. The main strength of R, however, lies in its visualisation capabilities. The visualisation of different aspects of the data is one of the major tasks during an exploratory analysis.

The basic trajectory format in adehabitat is *ltraj*, which is a list used to store trajectories from different animals. For more details on the *ltraj*-format, you refer to the vignettes (remember: *vignette(adehabitatLT)*) and the help pages for the '*adehabitatLT*'-library. An *ltraj*-object requires projected coordinates, a date for each location, and an animal identifier:

```
library(adehabitatLT)
ltrj <- as.ltraj(locs[, c("x32", "y32")], locs$UTC_time, locs$animals_id)
class(ltrj)

## [1] "ltraj" "list"

class(ltrj[[1]])

## [1] "data.frame"
```

The class-function shows us that *ltraj* is an object from the class *ltraj* and *list*. Each element of an *ltraj*-object is a *data.frame* with the trajectory information for each burst of each animal. A burst is a more or less intense monitoring of the animal followed by a gap in the data. For instance, animals that are only tracked during the day and not during the night will have for each day period a burst of data. The automatic schedule used for the GPS tracking of the roe deer in your database did not contain any intentional gaps; we therefore consider all data from an animal as belonging to a single burst.

```
head(ltrj[[1]])

##           x       y                date      dx      dy   dist     dt
## 1549 658249 5097296 2005-10-18 20:00:54   29.89   92.04  96.77  14429
## 1720 658279 5097388 2005-10-19 00:01:23   57.87 -426.16 430.07  14459
## 1025 658337 5096962 2005-10-19 04:02:22   46.07 -213.36 218.27  14446
## 1402 658383 5096749 2005-10-19 08:03:08 -229.96  454.70 509.55 129465
## 1298 658153 5097204 2005-10-20 20:00:53  -88.91   38.42  96.86  14395
## 1839 658064 5097242 2005-10-21 00:00:48  -63.56  -27.02  69.06  14405
##         R2n abs.angle rel.angle
## 1549      0     1.257        NA
## 1720   9365    -1.436   -2.6927
## 1025 119334    -1.358    0.0777
## 1402 317634     2.039   -2.8860
## 1298  17847     2.734    0.6947
## 1839  37195    -2.740    0.8099
```

The *head* function shows us that the *data.frames* within an *ltraj* object have ten columns. The first three columns define the location of the animal: the x and y coordinate and its date. The following columns describe the step (or change in location) toward the next location: the change in the x and y coordinates, the distance, the time interval between both locations, the direction of the movement, and the change in movement direction. The *R2n* is the squared displacement from the start point (or the net squared displacement [NSD]); we will discuss this metric in more detail later (see Calenge et al. 2009, and Fig. 10.2 for more explanation on these movement metrics).

Note that the animal's identifier is not in the table. As all locations belong to the same animal, there is no need to provide this information here. To obtain the identifiers of all the *data.frames* in an *ltraj*, you use the id function:

```
id(ltrj)

## [1] "1" "2" "3" "4" "5" "6"
```

Or, to obtain the id of only one animal,

```
id(ltrj[1])

## [1] "1"
```

The summary function gives some basic information on the *ltraj*-object:

```
(sumy <- summary(ltrj))

##   id burst nb.reloc NAs          date.begin            date.end
## 1  1     1     1647   0 2005-10-18 20:00:54 2006-10-29 12:00:49
## 2  2     2     2194   0 2005-03-20 16:03:14 2006-05-27 16:02:25
## 3  3     3     1826   0 2005-10-23 20:00:53 2006-10-28 12:01:18
## 4  4     4     2641   0 2005-10-21 20:00:47 2007-02-09 08:11:24
## 5  5     5     2695   0 2006-11-13 00:02:24 2008-03-15 08:01:37
## 6  6     6      278   0 2005-04-04 06:01:41 2005-05-05 23:01:47
```

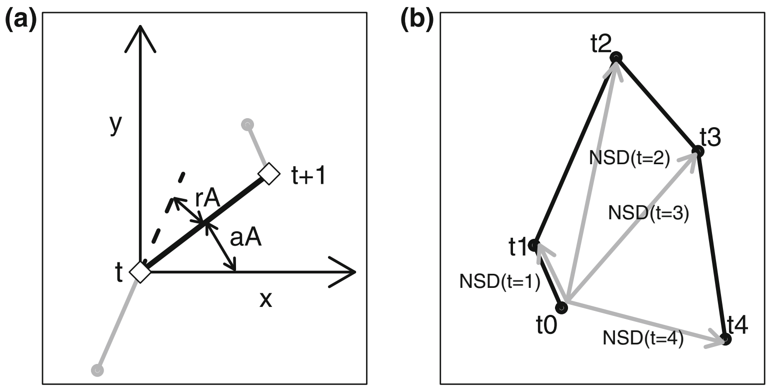


Fig. 10.2 The common trajectory characteristics stored in an *ltraj*. Panel **a** Shows the properties of one step from *t* to *t* + 1, and panel **b** the NSD of a series of locations (*t* = 1–5). The relative (*rA*) and absolute (*aA*) angles are also called the turning angle and direction of a step, respectively

Note that it marks 0 for missing values (i.e. NAs). However, this is not correct; you will see below how to tell R that there are missing observations in the data.

You see also that the total tracking duration is highly variable among individuals; notably Animal 6 was tracked for a much shorter time than the other animals. To ensure homogeneous data for the following analyses, you will only keep animals that have a complete year of data (i.e. number of days ≥365):

```
(duration <- difftime(sumy$date.end, sumy$date.begin, units = "days"))

## Time differences in days
## [1] 375.67 433.00 369.67 475.51 488.33  31.71
## attr(,"tzone")
## [1] "UTC"

ltrj <- ltrj[duration >= 365]
```

Moreover, for animals that were tracked for a longer period than one year, you remove locations in excess. Of course, if your ecological question is addressing space use during another period (e.g. spring) than you would want to keep all animals that provide this information, and Animal 6 may be retained for analysis, while removing all locations that are not required for this analysis.

```
ltrj <- cutltraj(ltrj,"difftime(date, date[1], units='days')>365")

summary (ltrj)

##    id  burst  nb.relocNAs          date.begin          date.end
## 1   1   1.01     1603    0 2005-10-18 20:00:54 2006-10-19 00:00:49
## 2   2  2.001     1894    0 2005-03-20 16:03:14 2006-03-20 20:02:06
## 3   3   3.01     1804    0 2005-10-23 20:00:53 2006-10-23 20:00:54
## 4   4  4.001     2009    0 2005-10-21 20:00:47 2006-10-21 20:00:54
## 5   5  5.001     1990    0 2006-11-13 00:02:24 2007-11-13 04:00:56
```

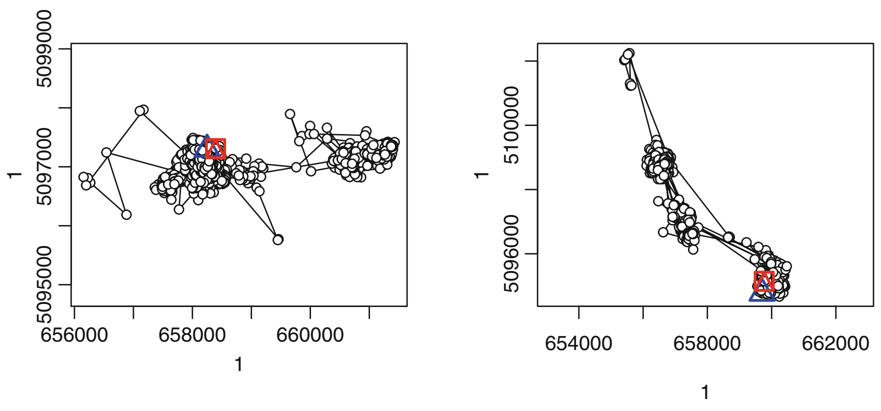


Fig. 10.3 Plot of the trajectories for the two first animals. You could have plotted all animals by simply running *plot(ltrj)*

Now, all animals are tracked for a whole year.

With the *plot* function, you can show the different trajectories (see the result in Fig. 10.3):

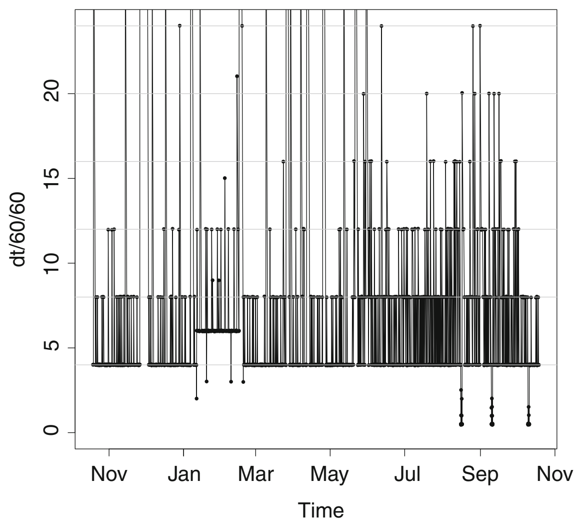
```
par(mfrow = c(1, 2))
plot(ltrj[1])
plot(ltrj[2])
```

The *plotltr* function allows us to show other characteristics than the spatial representation of the trajectory. For instance, it is very useful to get an overview of the sampling interval of the data. We discussed before how the sampling interval has a large effect on the patterns that you can observe in the data.

```
plotltr(ltrj[1], which = "dt/60/60", ylim = c(0, 24))
abline(h = seq(4, 24, by = 4), col = "grey")
```

Figure 10.4 shows that the time gap (dt) is not always constant between consecutive locations. Most often there is a gap of 4 h; however, there are several times that data are missing, and the gap is larger. Surprisingly, there are also locations where the gap is smaller than 4 h. We wrote a function to remove locations that are not part of a predefined sampling regime:

Fig. 10.4 The time interval between locations for animal 1



```
removeOutside <- function(x, date.ref, dt_hours = 1, tol_mins = 5) {
    require(adehabitatLT)
    x <- ld(x)
    tmp <- x$date + tol_mins * 60
    tmp_h <- as.POSIXlt(tmp)$hour
    tmp_m <- as.POSIXlt(tmp)$min
    hrs <- as.POSIXlt(date.ref)$hour
    hrs <- seq(hrs, 24, by = dt_hours)
    x <- x[tmp_h %in% hrs & tmp_m < (2 * tol_mins), ]
    x <- dl(x)
    return(x)
}
```

You now use this function on the *ltraj*; you specify a reference date at midnight, and the expected time lags in hours (dt_hours) is 4, and you set a tolerance of ±3 min (tol_mins):

```
ltrj <- removeOutside(ltrj, dt_hours = 4, tol_mins = 3,
                      date.ref=as.POSIXct("2005-01-01 00:00:00", tz="UTC"))
```

You can now inspect the time lag for each animal again:

```
plotltr(ltrj[1], which = "dt/60/60", ylim = c(0, 24))
abline(h = seq(4, 24, by = 4), col = "grey")
```

You see in Fig. 10.5 that there are no longer observations that deviate from the 4-hour schedule we programmed in our GPS sensors, i.e. all time lags between consecutive locations are a multiple of four. You still see gaps in the data, i.e. some

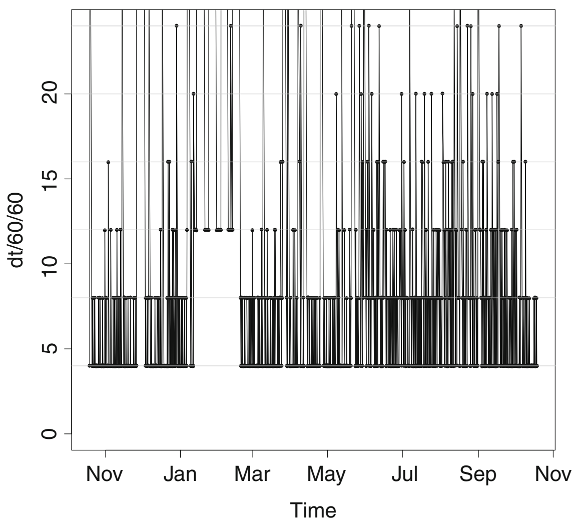


Fig. 10.5 The time interval between locations for animal 1 after the removal of locations outside the base sampling interval

gaps are larger than 4 h. Thus, there are missing values. The summary did not show the presence of missing data in the trajectory; you therefore have to specify the occurrence of missing locations.

The *setNA* function allows us to place the missing values into the trajectory at places where the GPS was expected to obtain a fix but failed to do so. You have to indicate a GPS schedule, which is in your case 4 h:

```
  ltrj <- setNA(ltrj, as.POSIXct("2004-01-01 00:00:00", tz="UTC"),
                dt=4*60*60)
  summary(ltrj)

##   id burst nb.reloc NAs          date.begin            date.end
## 1  1  1.01     2192 942 2005-10-18 20:00:54 2006-10-19 00:00:49
## 2  2 2.001     2189 758 2005-03-21 04:01:45 2006-03-20 20:02:06
## 3  3  3.01     2191 726 2005-10-23 20:00:53 2006-10-23 20:00:54
## 4  4 4.001     2191 492 2005-10-21 20:00:47 2006-10-21 20:00:54
## 5  5 5.001     2192 325 2006-11-13 00:02:24 2007-11-13 04:00:56
```

Indeed, now you see that the trajectories do contain a fair number of missing locations.

If locations are missing randomly, it will not bias the results of an analysis. However, when missing values occur in runs, this may affect your results. In adehabitat, there are two figures to inspect patterns in the missing data. The function *plotNAltraj* shows for a trajectory where the missing values occur and can be very instructive to show important gaps in the data:

```
da <- ltrj[[1]]$date
# the vector with dates allows us to zoom in on a range of dates in the plot
plotNAltraj(ltrj[1], xlim = c(da[1], da[500]))
```

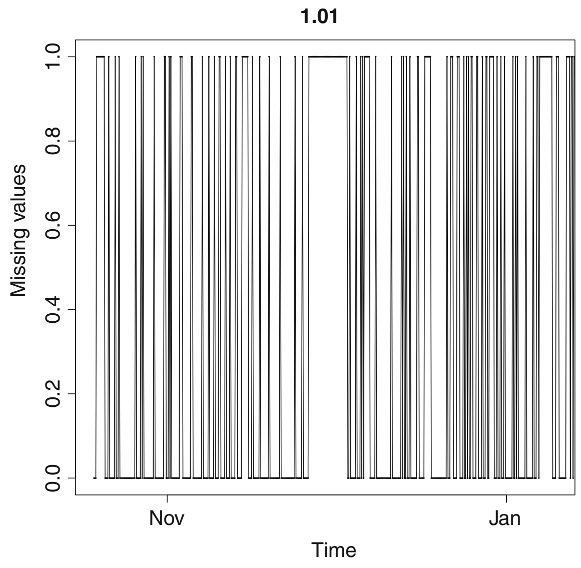


Fig. 10.6 The occurrence of missing locations over time for the first 500 locations: missing values are 1 and successful fixes are 0

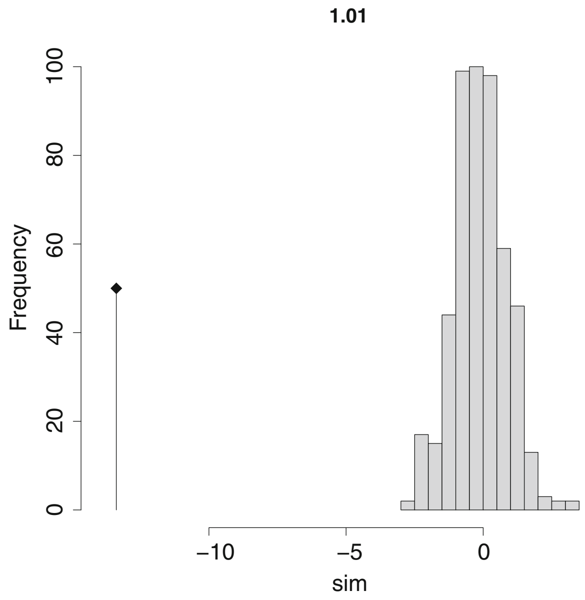


Fig. 10.7 Testing the temporal independence among missing locations. The *histogram* represents the expected distribution of missing values if they occurred independently. The pin on the *left* shows the observed distribution, which shows that missing values did not occur independently from each other

Figure 10.6 reveals that the missing values in November and December are not likely to occur independent of each other (you can verify this yourself for other periods by changing the limits of the x-axis with the xlim argument). You can test

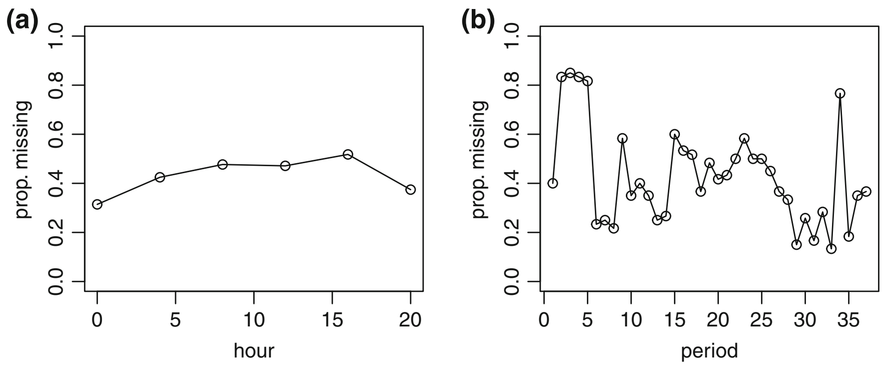


Fig. 10.8 Plot with the proportion missing locations for each hour of the day (panel **a**) and for each period of 10 days (panel **b**)

with the *runsNAltraj* function whether there is statistical significant clustering of the missing values:

```
runsNAltraj(ltrj[1])  #indeed, as the figures showed, it is not random
```

Indeed, Fig. 10.7 shows that for this trajectory, there is significant clustering of the missing fixes (you can test yourself whether this is also the case for the other animals). Thus, when you have one missing location, it is more likely that the next location will be missing too. Such temporal dependence in the probability to obtain fixes is not surprising, because the conditions affecting this probability are likely temporally autocorrelated. For instance, it is known that for a GPS receiver, it is more difficult to contact the satellites within dense forests, and so when an animal is in such a forest at time t, it is more likely to be still in this forest at time $t + 1$ than at time $t + 2$, thus causing temporal dependence in the fix-acquisition probability. Unfortunately, as said, this temporal dependence in the 'missingness' of locations holds the risk of introducing biases in the results of your analysis.

Visual inspection of figures like Fig. 10.6 can help the assessment of whether the temporal dependence in missing locations will have large effects on the analysis. Other figures can also help this assessment. For instance, you can plot the number of missing locations for each hour of the day, or for periods of the year (e.g. each week or month):

```
par(mfrow=c(1,2))
plot(c(0, 4, 8, 12, 16, 20),
     tapply(ltrj[[1]]$x, as.POSIXlt(ltrj[[1]]$date)$hour,
     function(x)mean(is.na(x))),
     xlab="hour", ylab="prop. missing", type="o", ylim=c(0,1), main="a")
periods <- trunc(as.POSIXlt(ltrj[[1]]$date)$yday/10)
plot(tapply(ltrj[[1]]$x, periods, function(x)mean(is.na(x))),
     xlab="period", ylab="prop. missing", type="o", ylim=c(0,1), main="b")
```

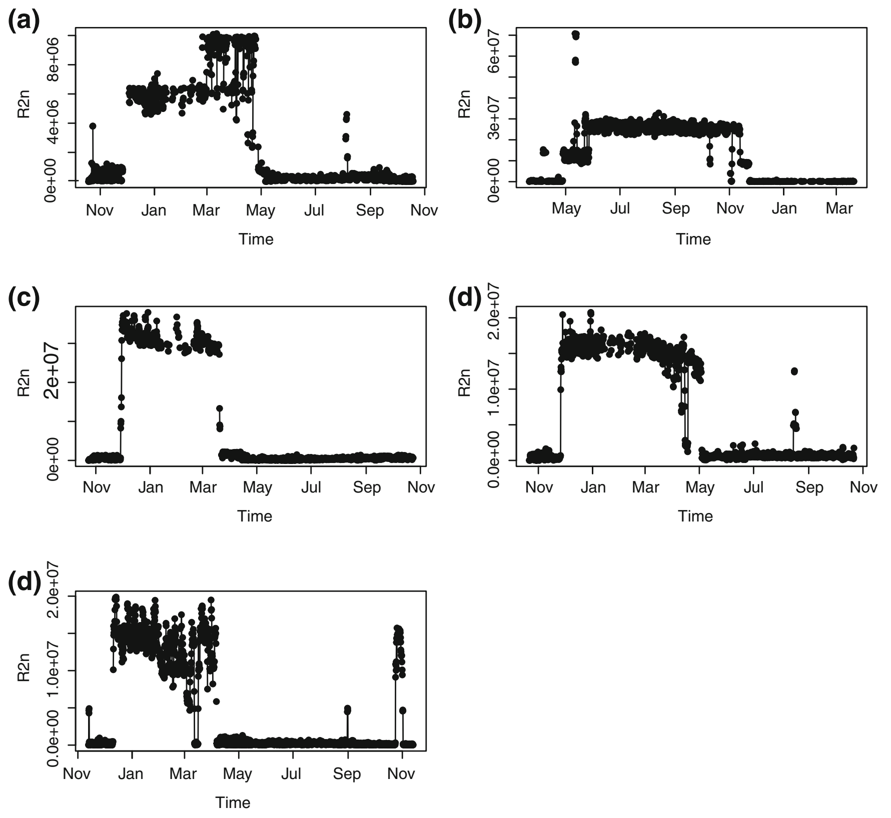


Fig. 10.9 The NSD for each individual

Figure 10.8 shows that there is no strong bias in the time of the day; you can therefore be fairly confident that additional results will not be biased regarding the diurnal cycle. However, there are four consecutive blocks of 10 days with low fix-acquisition rate, which could be an issue. Fortunately, the other periods during the winter are providing us with enough data. You therefore expect bias on your results to be minimal. In cases when there are longer periods with missing data, it can be necessary to balance the data. It is obviously not straightforward to create new data; however, you can remove random locations in periods when you have 'too many' observations. In our demonstration, we will proceed without further balancing the data.

The NSD is a commonly used metric for animal space use; it is the squared straight-line distance between each point and the first point of the trajectory (see Fig. 10.2b). It is a very useful metric to assess, for instance, the occurrence of seasonal migration patterns. An animal that migrates between summer and winter ranges will often show a characteristic hump shape in the NSD, as exemplified clearly in Fig. 10.9:

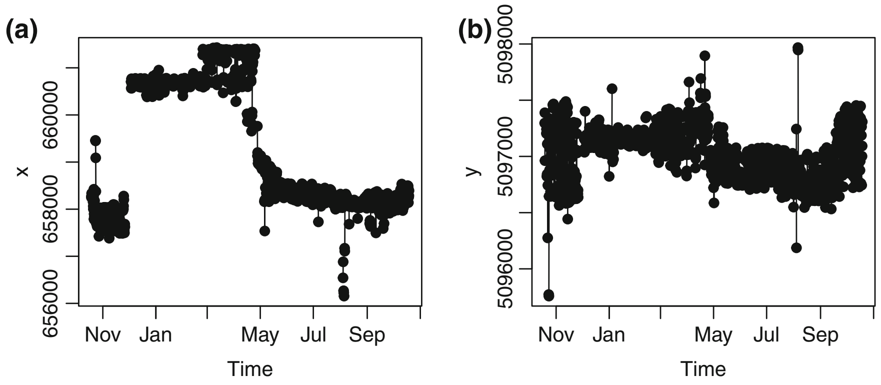


Fig. 10.10 The x- and y-axis against time in panel **a** and **b**, respectively, for the first individual

```
plotltr(ltrj, which = "R2n")
```

The NSD profiles in Fig. 10.9 strongly suggest the occurrence of seasonal migration in all five animals. During the summer (from May till November), the animals seem to be in one area and during the winter (from December till April) in another. The NSD is a one-dimensional representation of the animal's space use, which facilitates the inspection of it against the second dimension of time. On the other hand, it removes information present in the two-dimensional locations provided by the GPS. Relying exclusively on the NSD can in certain situations give rise to wrong inferences; we therefore highly recommend also inspecting the locations in two dimensions. One of the disadvantages of the NSD is that the starting point is often somewhat arbitrary. It can help to use a biological criterion such as the fawning period to start the year.

As an alternative for (or in addition to) the NSD, you can plot both spatial dimensions against time as in Fig. 10.10:

```
par(mfrow=c(1,2))
plot(ltrj[[1]]$date, ltrj[[1]]$x, pch=19, type="o", xlab="Time",
     ylab="x", main="a")
plot(ltrj[[1]]$date, ltrj[[1]]$y, pch=19, type="o", xlab="Time",
     ylab="y", main="b")
```

To avoid the intrinsic reduction of information by collapsing two dimensions into one single dimension, you also plot both spatial dimensions and use colour to depict the temporal dimension:

```
ltrj2 <- na.omit.ltraj(ltrj)
par(mfrow=c(3,2))
for (i in c(1:5))
{plot(ltrj2[[i]][c("x","y")], col=rainbow(12)[as.POSIXlt(ltrj2[[i]]$date)$mon+1],
      pch=19, asp=T)
 segments(ltrj2[[i]]$x[-nrow(ltrj[[i]])], ltrj2[[i]]$y[-nrow(ltrj2[[i]])],
         ltrj2[[i]]$x[-1],ltrj2[[i]]$y[-1],
         col=rainbow(12)[as.POSIXlt(ltrj2[[i]]$date[-1])$mon+1])
 title(c("a", "b", "c", "d", "e")[i])
 }
plot(c(0:100),c(0:100), type="n", xaxt="n", yaxt="n", xlab="", ylab="", bty="n")
mon <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
        "Oct", "Nov", "Dec")
legend(x=20, y=90, legend=mon[c(0:5)+1], pch=19, col=rainbow(12)[1:6], bty="n")
legend(x=60, y=90, legend=mon[c(6:11)+1], pch=19, col=rainbow(12)[7:12], bty="n")
```

Figure 10.11 confirms our interpretation of Fig. 10.9. All five animals have at least two seasonal centres of activity: winter versus summer. The movement between these centres occurs around November and around April. The comparison of Figs. 10.9 and 10.11 reveals easily the respective strengths of both figures. It is easier to read from Fig. 10.9 the timing of events, but it is easier to read from Fig. 10.11 the geographic position of these events. This demonstrates the importance of making several figures to explore the data.

Now that you have familiarised yourselves with the structure of the data and have ensured that your data are appropriate for the analysis, you can proceed with answering your ecological questions in the following sections.

Home Range Estimation

A home range is the area in which an animal lives. In addition to Burt's (1943) aforementioned definition of the home range as 'the area an animal utilizes in its normal activities', Cooper (1978) pointed out that a central characteristic of the home range is that it is temporally stable. Our previous exploration of the data has shown that the space use of our roe deer is not stable. Instead, it seems to consist of a migration between two seasonal home ranges. Figures 10.9 and 10.10 suggest that space use within these seasonal ranges is fairly stable. It is thus clear that the concept of a home range is inherently tied to a time frame over which space use was fairly stable, in our case two seasons.

An animal's home range has been quantified by the concept of the 'utilization distribution (UD)'. Van Winkle (1975) used the term UD to refer to 'the relative frequency distribution for the points of location of an animal over a period of time'. The most common estimator for the UD is the kernel density estimator. In Fig. 10.12, we remind the reader of the general principle underlying such analysis. Several methods for home range computation are implemented in the

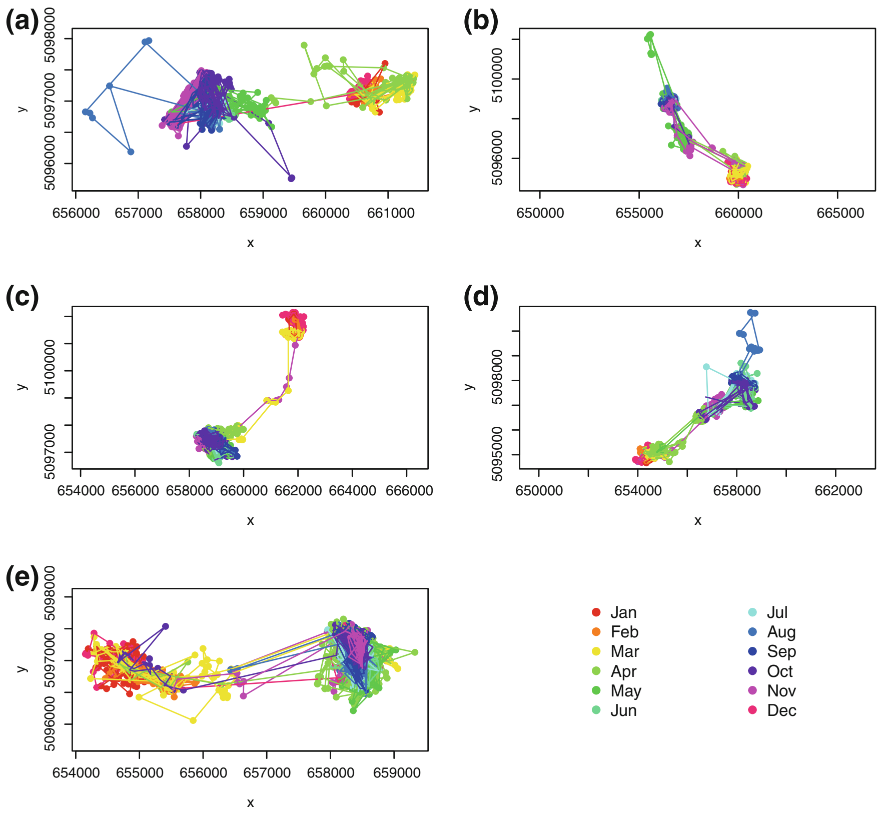


Fig. 10.11 Coloured trajectories for all individuals, locations from each month are coloured differently

'*adehabitatHR*' library; the *kernelUD* function calculates the kernel utilization density from 2D locations:

```
library(adehabitatHR)
library(sp)
trj <- ld(ltrj)
trj <- trj[!is.na(trj$x),]
(kUD <- kernelUD(SpatialPointsDataFrame(trj[c("x","y")],
                 data=data.frame(id=trj$id)),h=100,grid=200,kern="epa"))
image(kUD[[1]])
```

The function *kernelUD* requires a *SpatialPoints* object or a *SpatialPointsDataFrame*, and it returns a *SpatialPixel* object, adehabitat relies on the spatial classes from the '*sp*' library. A familiarity with the spatial classes from '*sp*' will

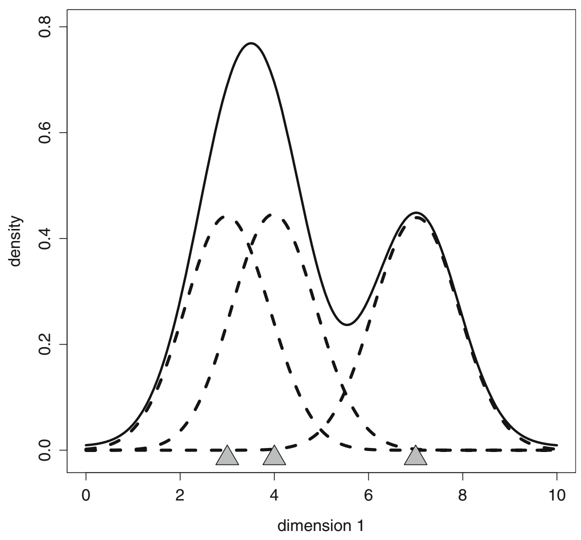


Fig. 10.12 A one-dimensional example of a kernel density estimator for three points. For each of the observations (i.e. 3, 4 and 7), a distribution (e.g. *Gaussian curve*) is placed over them (the *dashed lines*); these distributions are aggregated to obtain the cumulative curve (the *full black line*). This cumulative curve is the kernel density estimator of these points. The width of initial distributions used is the smoothing factor and is a parameter the researcher has to select. Kernel home range estimation works in a similar way in 2D with for instance a bivariate normal distribution to smooth the locations

therefore be helpful for your analysis of animal space use data in R (see Bivand et al. 2008, or the vignettes available for '*sp*'). You create the *SpatialPointsDataFrame* from a dataframe, which you obtained from the *ltrj* using the *ld* function (i.e. list to dataframe). Note: '*SpatialPoints*' cannot contain missing coordinates; therefore, you keep only those rows where you have no missing values for the x coordinate (i.e. *!is.na(trj$x)*, the '!' means 'not' in R). The resulting pixel map in Fig. 10.13 shows the areas that are most intensely used by this individual.

During our data exploration, we found that our roe deer occupy a separate summer and winter range: Are both ranges of similar size? For this, you have to compute the home range separately for summer and winter. In Fig. 10.9, you can see that the summer range is occupied at least from day 150 (beginning of June) to day 300 (end of October) and that the winter range is occupied from days 350 (mid-December) till 100 (end of March). You can use these dates to split compute the kernel for summer and winter separately; the function *kernel.area* computes the area within a percentage contour. We demonstrate the computation for the 50, 75 and 95 %:

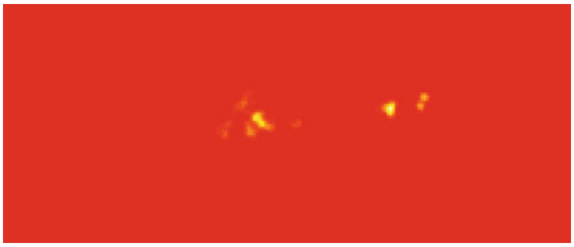


Fig. 10.13 Kernel UD of the first individual, in *yellow*, is the intensely used areas

```
trj$yday <- yday(trj$date)
trj$season <- ifelse(trj$yday>150 & trj$yday<300, "summer", NA)
trj$season <- ifelse(trj$yday>350 | trj$yday<100, "winter", trj$season)

trj <- trj[!is.na(trj$season),]
(kUD <- kernelUD(SpatialPointsDataFrame(trj[c("x","y")],
                 data=data.frame(id=paste(trj$id, trj$season))),
                 h =100, grid=200, kern = "epa"))

## ********** Utilization distribution of several Animals ************
##
## Type: probability density
## Smoothing parameter estimated with a  specified smoothing parameter
## This object is a list with one component per animal.
## Each component is an object of class estUD
## See estUD-class for more information

area <- kernel.area(kUD, percent = c(50, 75, 95), unin = "m", unout = "ha")
(areas <- data.frame(A50=unlist(area[1,]), A75=unlist(area[2,]),
                     A95=unlist(area[3,]), id=rep(c(1:5), each=2),
                     seas=rep(c("S", "W"), 5)))

##               A50   A75    A95 id seas
## X1.summer  9.931 22.51  52.30  1    S
## X1.winter  6.902 14.19  33.06  1    W
## X2.summer 10.915 21.92  40.47  2    S
## X2.winter 11.277 27.07  62.65  2    W
## X3.summer 12.977 30.24  62.96  3    S
## X3.winter  7.873 19.79  50.84  3    W
## X4.summer 10.230 24.25  83.17  4    S
## X4.winter  5.747 11.14  28.43  4    W
## X5.summer 12.498 24.31  49.14  5    S
## X5.winter 18.344 41.74 112.72  5    W
```

You can visualise these results clearly with boxplots:

```
par(mfrow = c(1, 3))
boxplot(areas$A50 ~ areas$seas, cex = 2, main = "a")
boxplot(areas$A75 ~ areas$seas, cex = 2, main = "b")
boxplot(areas$A95 ~ areas$seas, cex = 2, main = "c")
```

Figure 10.14 shows that more than a change in the mean range size, there seems to be a marked change in the individual variation between seasons. During the winter season, there seems to be much larger individual variation in range size than there is in summer; however, more data will be required to further investigate this seasonal variation in range sizes.

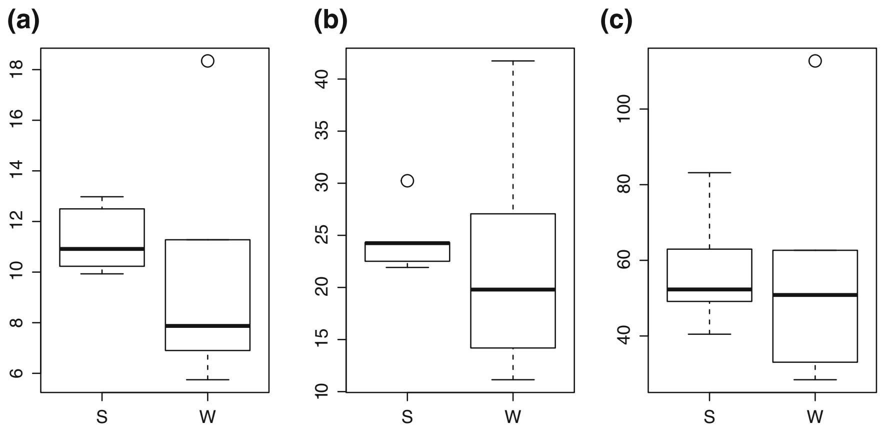


Fig. 10.14 The boxplots of the areas (in ha) of the seasonal ranges from left to right the 50, 75 and 95 % kernel contours. Each panel depicts summer (S) on the *left*, and winter (W) on the *right*

Habitat Use and Habitat Selection Analysis

In the previous exercise, you saw that roe deer change the location of their activities from summer to winter. Such seasonal migrations are often triggered by changes in the environment. You can then wonder which environmental conditions are changing when the animal moves between its seasonal ranges. For instance, snowfall is a driver for many migratory ungulates in northern and alpine environments, and winter ranges are often characterised by less snow cover than the summer ranges in winter (e.g. Ball et al. 2001). Thus, you would expect that roe deer will be moving down in altitude during the winter to escape from the snow that accumulates at higher altitudes.

If roe deer move to lower altitudes during winter, then they probably also move closer to roads, which are usually found in valley bottoms. You would not necessarily expect roe deer to show a seasonal response directly toward roads, but you do expect this as a side effect from the shift in altitude. Such closer proximity to roads can have substantial effects on road safety, as animals close to roads are at a higher risk of road crossings and thus traffic accidents. Ungulate vehicle collisions are an important concern for road safety and animal welfare. From an applied perspective, it is thus an interesting question to see whether there is a seasonal movement closer to roads, which could partly explain seasonal patterns often observed in ungulate vehicle collisions.

You first add the environmental data to your inspected trajectory with the *merge* function:

```
trj <- ld(ltrj)
trj <- merge(trj, locs, by.x=c("id", "date"),
             by.y=c("animals_id", "UTC_time"), all.x=T)
```

You inspect then whether there is a relationship between the distance to the roads and the altitude:

```
library(lattice)
xyplot(roads_dist ~ altitude_srtm| factor(id),
       xlab = "altitude", ylab = "dist. road", col = 1,
       strip = function(bg = "white", ...) strip.default(bg = "white", ...),
       data = trj)
```

Figure 10.15 shows an interesting relationship between altitude and distance to roads. Each individual shows two clusters, which are possibly corresponding with the two seasonal ranges you detected before. Within each cluster, there is a positive relationship between altitude and distance to roads; i.e. at higher altitudes, the distance to roads is greater. However, when you compare both clusters, it seems that the cluster at higher altitude is often also closer to roads (except for Animal 1). Overall, it seems that in your data, there is no obvious positive relationship between altitude and distance to roads.

Let us now investigate the hypothesis that there is a seasonal change in roe deer altitude:

```
xyplot(altitude_srtm ~ as.POSIXlt(acquisition_time)$yday | factor(id),
       xlab = "day of year", ylab = "altitude", col = 1,
       strip = function(bg = "white", ...) strip.default(bg = "white", ...),
       data = trj)
```

Figure 10.16 shows that there are marked seasonal changes in the altitude of the roe deer positions. As you expected, roe deer are at lower altitudes during the winter than they are during the summer. This pattern explains the occurrence of the two clusters of points for each individual in Fig. 10.15.

You can now proceed by testing the statistical significance of these results. You use the same seasonal cutoff points as before:

```
trj$yday <- yday(trj$date)
trj$season <- ifelse(trj$yday > 150 & trj$yday < 300, "summer", NA)
trj$season <- ifelse(trj$yday > 350 | trj$yday < 100, "winter", trj$season)
trj2 <- trj[!is.na(trj$season), ]
fit <- lm(altitude_srtm ~ as.factor(season), data = trj2)
```

The function *lm* is used to fit a linear model to the data (note: for this simple case a Student's *t* test would have been sufficient).

These results show that as expected the roe deer move to lower altitudes during the winter:

	Estimate	Std. error	*t* value	Pr(>|*t*|)
(Intercept)	1,581.6609	2.6212	603.41	0.0000
as.factor(season)winter	−572.3900	4.2519	−134.62	0.0000

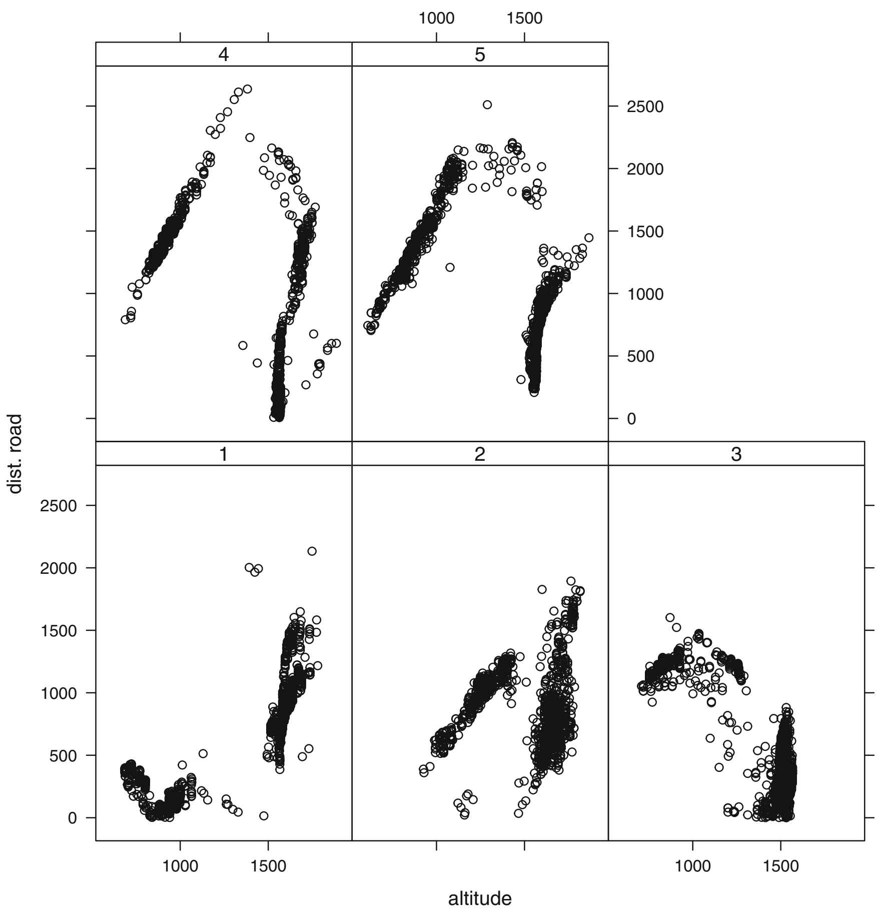


Fig. 10.15 The changing distance to roads as a function of the altitude for each individual

In winter, the roe deer are on average 572 m lower than during the summer, which is a 33 % decrease.

A treatment on model validation falls outside the scope of this book. We refer the reader to introductory books in statistics; several such books are available using examples in R (e.g. Crawley 2005; Zuur et al. 2007).

In this example, you have focused on the habitat use of roe deer. Often researchers are not only interested in the habitat characteristics used by the animals, but also in the comparison between use and availability—i.e. habitat selection. In habitat-selection studies, the used habitat characteristics are compared against the characteristics the animal could have used or the available habitat. Thus, to perform habitat-selection analysis, you have to sample from the available points and obtain the habitat characteristics for these points.

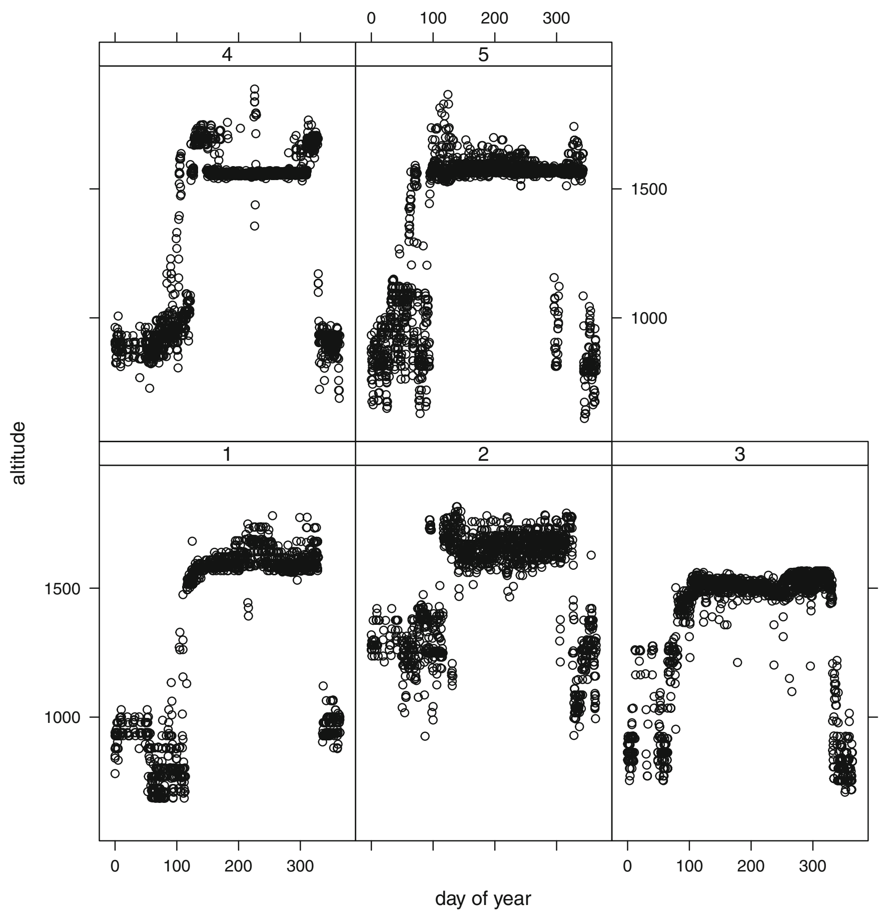


Fig. 10.16 The altitude of roe deer locations as a function of the day of the year. You see marked seasonal changes

The available locations are commonly sampled randomly at two scales: within the study area to study home range placement, or within the individual home range to study home range use (respectively, called second- and third-order habitat selection following Johnson 1980). We will demonstrate third-order habitat selection and use a minimum convex polygon (MCP) to characterise the area available for each roe deer, from which we sample 2,000 random locations. The *mcp*-function in R requires the use of a *SpatialPointsDataFrame* when using multiple animals (for a single animal a *SpatialPoints* object suffices):

```
trj2 <- na.omit(trj[,c("id", "x","y")])
sptrj <- SpatialPointsDataFrame(SpatialPoints(trj2[,c("x","y")]),
                                data=data.frame(id=trj2$id))
ranges <- mcp(sptrj, percent=100)
```

Then, you sample for each individual randomly from its available range, and you place the coordinates from these sampled locations together with the animal's id in a *data.frame*. Operations like this, in which something needs to be repeated for a number of individuals, are easily performed using the *list* format, which is also the reason that '*adehabitatLT*' uses a *list* to store trajectories. However, a database works with *data.frames*; therefore, you will have to bind the *data.frames* in the list together in one large *data.frame*.

```
set.seed(0) #to ensure replication of the same randomly sampled points
rndpts <- lapply(c(1:5),
              function(x, ranges){spsample(ranges[ranges@data$id==x,], n=2000,
                                   type="random", iter=100)},ranges=ranges)
rndpts <- lapply(c(1:5),}{function(x,rndpts)data.frame(rndpts[[x]]@coords,id=x),
                 rndpts=rndpts)
rndpts <- do.call("rbind", rndpts)
```

Figure 10.17 shows the areas you considered available for each individual roe deer, from which you sampled the random locations:

```
plot(ranges)
points(rndpts[c(1:100), c("x", "y")], pch = 16)  #shows the first 100 random points
```

The easiest way to obtain the environmental data for these random points is to simply upload them into the database and extract the required information from there. To facilitate the ordering of your random locations, you add a column *nb* numbered 1 to the number of random points. The function *dbWriteTable* writes a table to the database; you can specify a schema (*analysis*) in addition to the table name (*rndpts_tmp*):

```
rndpts$nb <- c(1:nrow(rndpts))
dbWriteTable(con, c("analysis", "rndpts_tmp"), rndpts)

## [1] TRUE
```

Next, you use the database to couple the locations in this table to the environmental data stored in the database. The easiest way to do this is by first adding a geometry column for the random locations:

```
dbSendQuery(con,
  "ALTER TABLE analysis.rndpts_tmp ADD COLUMN geom geometry(point, 4326);")

## <PostgreSQLResult:(6096912,1,7)>

dbSendQuery(con,
  "UPDATE analysis.rndpts_tmp
   SET geom = st_transform((st_setsrid(st_makepoint(x,y),23032)),4326);" )

## <PostgreSQLResult:(6096912,1,8)>
```

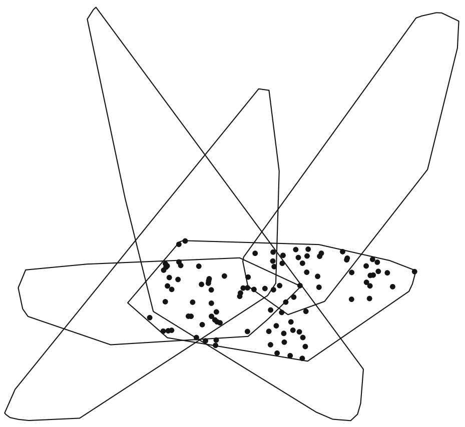


Fig. 10.17 The available areas for each roe deer estimated by a MCP. The first 100 locations sampled randomly from the area of individual 1 are represented by *black dots*

You extract the altitude and land cover for the random locations from the rasters stored in the database with the following queries (the details of these SQL queries were discussed earlier in this book):

```
altitude <- fetch(dbSendQuery(con,
    "SELECT st_value(srtm_dem.rast, geom) as altitude
     FROM env_data.srtm_dem, analysis.rndpts_tmp
     WHERE st_intersects(srtm_dem.rast,geom) ORDER BY nb;"), -1)
landcover <- fetch(dbSendQuery(con,
    "SELECT st_value(corine_land_cover.rast, st_transform(geom,3035)) as landcover
     FROM env_data.corine_land_cover, analysis.rndpts_tmp
     WHERE st_intersects(corine_land_cover.rast, st_transform(geom,3035))
                                                     ORDER BY nb;"), -1)
```

You extract the distance to the closest road for the random locations from the roads stored in the database with the following query (this query can require a few minutes):

```
mindist <- fetch(dbSendQuery(con,
    "SELECT min(distance) as dist_roads
     FROM (SELECT nb, st_distance(roads.geom::geography,
           rndpts_tmp.geom::geography)::integer as distance
           FROM env_data.roads, analysis.rndpts_tmp) AS a
           GROUP BY a.nb ORDER BY nb;"), -1)
```

You add these environmental data to the randomly sampled available locations:

```
rndpts <- cbind(rndpts, altitude, landcover, mindist)
head(rndpts)

##        x       y id nb altitude landcover dist_roads
## 1 660909 5097377  1  1      940        23         30
## 2 657779 5096660  1  2     1650        25       1555
## 3 658739 5096304  1  3     1778        24       1500
## 4 657638 5097232  1  4     1678        25       1135
## 5 658984 5097551  1  5     1510        18        289
## 6 659459 5097810  1  6     1459        18         82
```

Now that you no longer need these locations in the database, you can remove the table:

```
dbRemoveTable(con, c("analysis", "rndpts_tmp"))

## [1] TRUE
```

A discussion on habitat selection falls outside the scope of this chapter. More information on exploratory habitat selection using R can be found in the *vignette(adehabitatHS)*; for a general discussion on the use of resource selection functions on habitat selection, we refer the reader to the book by Manly et al. (2002). We will merely visualise the difference in the habitat types between the used and the available locations.

To ensure proper comparison of used and available habitats, you provide the same levels to both data sets. Moreover, to allow our seasonal comparison, you also need to allocate the random locations to the summer and winter season:

```
trj <- na.omit(trj[,c("id", "x", "y", "roads_dist","corine_land_cover_code",
                      "altitude_srtm", "season")])
names(trj) <- c("id", "x", "y", "dist_roads", "landcover", "altitude", "season")
trj$landcover <- factor(trj$landcover, levels=c(18, 21, 23:27, 29, 31, 32))
#allocate the random locations to each season
rndpts$season <- c("summer", "winter")
rndpts$landcover <- factor(rndpts$landcover, levels=c(18, 21, 23:27, 29, 31, 32))
```

Now, you will compute for each individual the number of locations inside each habitat type:

```
use_win <- table(trj$id[trj$season=="winter"],
                 trj$landcover[trj$season=="winter"])
ava_win <- table(rndpts$id[rndpts$season=="winter"],
                 rndpts$landcover[rndpts$season=="winter"])
use_sum <- table(trj$id[trj$season=="summer"],
                 trj$landcover[trj$season=="summer"])
ava_sum <- table(rndpts$id[rndpts$season=="summer"],
                 rndpts$landcover[rndpts$season=="summer"])
#determine the proportions of each class
calc.prop <- function(x){(x/sum(x))}
use_win <- t(apply(use_win, 1, calc.prop))
ava_win <- t(apply(ava_win, 1, calc.prop))
use_sum <- t(apply(use_sum, 1, calc.prop))
ava_sum <- t(apply(ava_sum, 1, calc.prop))
#to avoid division by zero, we add a small number to each element in the table
use_win <- use_win+0.01
ava_win <- ava_win+0.01
use_sum <- use_sum+0.01
ava_sum <- ava_sum+0.01
library(adehabitatHS)
sr_win <- widesIII(use_win,ava_win, avknown = FALSE, alpha = 0.05)
sr_sum <- widesIII(use_sum,ava_sum, avknown = FALSE, alpha = 0.05)
```

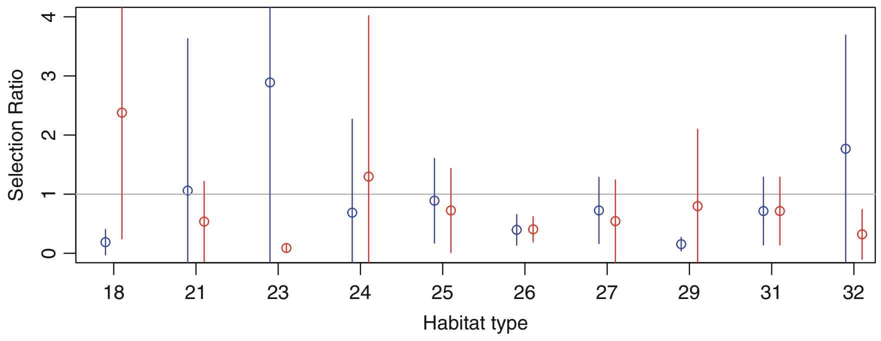


Fig. 10.18 The ratio of the proportion used and the proportion available (known as the selection ratio). Values above 1 are used more than their availability and vice versa. Blue points are for winter and red for summer. You can find the corresponding habitat types in the corine_land_cover_legend table from your database

The function *widesIII* in the '*adehabitatHS*' library computes the selection ratios for individually tracked animals; it also provides a number of statistical tests.

```
plot(c(1:10)-0.1, sr_win$wi, xaxt="n", ylab="Selection Ratio",
     xlab="Habitat type", col="blue", ylim=c(0,4))
axis(1, at=c(1:10), labels=c(18, 21, 23:27, 29, 31, 32))
abline(h=1, col="dark grey")
points(c(1:10)+0.1, sr_sum$wi, col="red")
segments(c(1:10)-0.1, sr_win$ICwiupper, c(1:10)-0.1, sr_win$ICwilower,col="blue")
segments(c(1:10)+0.1, sr_sum$ICwiupper, c(1:10)+0.1, sr_sum$ICwilower,col="red")
```

For a more extensive discussion of selection ratios, we refer you to the aforementioned references. Here, you limit yourselves to visualising the selection ratios for both seasons. Figure 10.18 shows that the roe deer seems to select more for pastures (class 18) during summer, whereas for most roe deer, their use of broad-leaved forests seem to be higher during the winter months (class 23). Great care should be taken not to over-interpret the unreliable results from classes that are hardly present in the study area (e.g. classes 26–31). These types of figures and tables are highly suitable to inspecting categorical data such as land cover. Continuous data such as altitude are better represented using histograms.

In the previous demonstrations, you have been using R to visualise and analyse data stored in the database, and you also used R as an interface to interact with the database. An alternative approach is to use R from within the database, which we will demonstrate in the next chapter.

Acknowledgments M. Basille provided valuable comments on the earlier drafts of this chapter. The work for this chapter was funded by Norwegian Research Council project number 208434-F40 'Management of biodiversity and ecosystem services in spatially structured landscapes'.

References

Ball JP, Nordengren C, Wallin K (2001) Partial migration by large ungulates: characteristics of seasonal moose Alces alces ranges in northern Sweden. Wild Biol 1:39–47

Bivand RS, Pebesma EJ, Rubio VG (2008) Applied spatial data: analysis with R. Springer, New York

Burt WH (1943) Territoriality and home range concepts as applied to mammals. J Mammal 24(3):346–352

Calenge C (2006) The package 'adehabitat' for the R software: a tool for the analysis of space and habitat use by animals. Ecol Model 197(3):516–519

Calenge C, Dray S, Royer-Carenzi M (2009) The concept of animals' trajectories from a data analysis perspective. Ecol Inform 4(1):34–41

Cooper WE Jr (1978) Home range criteria based on temporal stability of areal occupation. J Theor Biol 73(4):687–695

Crawley MJ (2005) Statistics: an introduction using R. Wiley, New York

Fortin D, Beyer HL, Boyce MS, Smith DW, Duchesne T, Mao JS (2005) Wolves influence elk movements: behavior shapes a trophic cascade in Yellowstone National Park. Ecology 86(5):1320–1330

Johnson DH (1980) The comparison of usage and availability measurements for evaluating resource preference. Ecology 61(1):65–71

Kie JG, Matthiopoulos J, Fieberg J, Powell RA, Cagnacci F, Mitchell MS, Moorcroft PR (2010) The home-range concept: are traditional estimators still relevant with modern telemetry technology? Philos Trans R Soc B: Biol Sci 365(1550):2221–2231

Manly BFJ, McDonald LL, Thomas DL, McDonald TL, Erickson WP (2002) Resource selection by animals: statistical analysis and design for field studies. Kluwer, The Netherlands

Turchin P (1998) Quantitative analysis of movement: measuring and modeling population redistribution in animals and plants. Sinauer Associates, Sunderland

Van Winkle W (1975) Comparison of several probabilistic home-range models. J Wild Manag 39:118–123

Zuur AF, Ieno EN, Smith GM (2007) Analysing ecological data. Springer, New York