Working with wildlife GPS tracks in R

Introduction:

R is becoming an increasingly popular way of examining spatial ecological data, both because of its rich functionality and because it is open source. Here, we will look at one of several R packages within a suite called adehabitat, all of which focus on spatial ecological analysis. Because of the rapid growth in its analytical functions, the original adehabitat package is now split into four packages:

- adehabitatLT: the focus of this exercise, which looks specifically at wildlife trajectories
- adehabitatHR: tools for working out species' home ranges
- adehabitatHS: tools for exploring habitat suitability
- adehabitatMA: tools for mapping data concerning species

This exercise is a simplified exploration based on part of an excellent exercise by Clement Calenge (hereafter 'the Calenge exercise'), the author of the adehabitat packages, which is available here: https://cran.r-project.org/web/packages/adehabitatLT/vignettes/adehabitatLT.pdf. Thinking particularly of those relatively new to R, we step through some of the early parts of this exercise with a little more explanation of the data processing. If you are particularly interested in the analysis of habitat tracks, you could go further and work through the other parts of the Calenge exercise. If you are new to R, you may also wish to look at the R Studio 'quick start' pdf accompanying this document.

Loading the Relevant Packages:

We first need to install the R packages that we will be working with. Alongside the adehabitatLT package, we will also use the 'raster' package to help with conversion and display of some raster layers in the data set we will use:

```
> install.packages("raster")
> install.packages("adehabitatLT")
```

Having installed the packages, we now need to load the packages into our current session ready for use:

```
> library(adehabitatLT)
> library(raster)
```

Exploring Some Track Data:

Many R packages come with associated data sets and this is also true of adehabitatHR. We can load an example data set as follows:

```
>data("puechabonsp")
```

If we wish to find out more about this data set (or indeed any R command), then we can do so as follows:

```
>?puechabonsp
```

As you can hopefully now see from the associated help file, the data set concerns the movements of four individual wild boar in southern France. Within it, there are two further objects, held within a list R structure. One (`map`) is a series of raster map layers, held in an object of class SpatialPixelsDataFrame, whilst the other (`relocs`) contains the summer locations of the boar, held as a SpatialPointsDataFrame class object (remember: to find out more about both object classes: simply type `?SpatialPixelsDataFrame).` We can explore these a little more simply by entering their names (note the `$` enables us to reference the individual items within the R list structure):

```
>puechabonsp$map

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
x            697900      100        79
y           3156800      100        84

Variables measured:
     Elevation Aspect    Slope Herbaceous
2017        68      3 1.548994          0
2018        69      3 1.118594          0
2019        70      3 1.358634          0
2020        70      3 1.724311          0
2021        70      3 1.944885          0
2022        71      3 2.024868          0
```

…so the rasters are based on a 100 metre pixel size and comprise elevation, aspect, slope and aspect…

What about the other item in the list?

```
> puechabonsp$relocs
class       : SpatialPointsDataFrame
features    : 119
extent      : 698626, 701410, 3157848, 3161678  (xmin, xmax, ymin, ymax)
coord. ref. : NA
variables   : 4
names       :  Name, Age, Sex,   Date
min values  : Brock,   2,   1, 920729
max values  :  Jean,   3,   2, 930831
```

So we have a set of 119 relocations (positional fixes), with associated information on the names, age and sex of the wild boar, as well as the dates that the positional fixes relate to.

For mapping out the various raster layers, it is helpful to convert these to separate raster objects. This can be done via the raster utility:

```
> elevat<-raster(puechabonsp$map, layer=1)
```

Note that we could create rasters for other layers within the map SpatialPixelsDataFrame, such as slope, simply by changing the layer number:

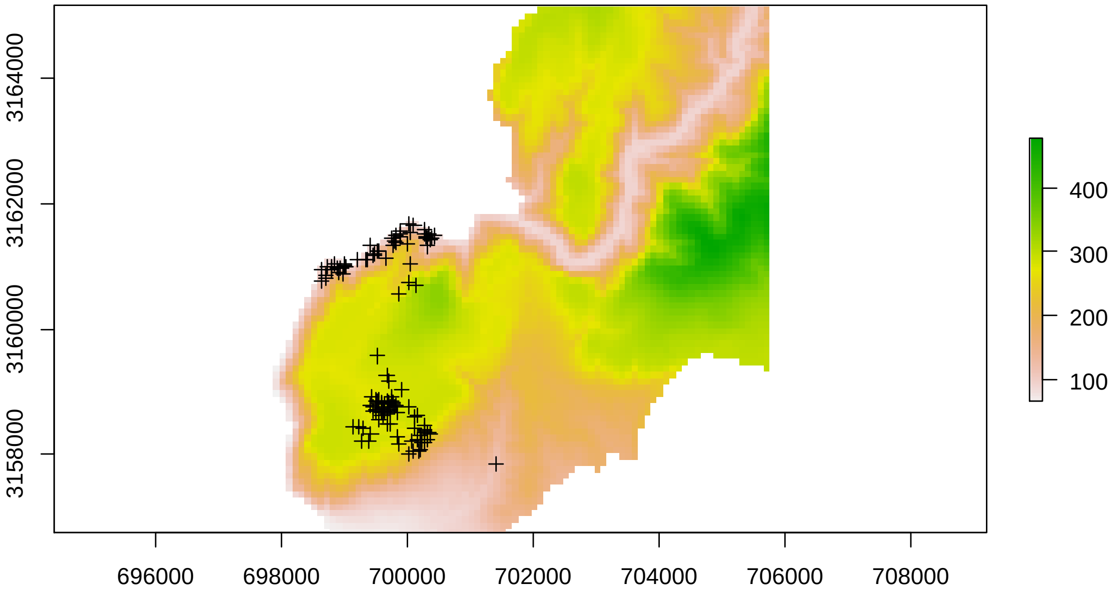
```
> slope<-raster(puechabonsp$map, layer=3)
```

We can now map out the locations of the boar, relative to elevation using the `plot` tool, starting with the raster that we just created:

```
> plot(elevat)
```

`add=TRUE` lets us plot further data on top of our most recent plot:

```
> plot(puechabonsp$relocs, add=TRUE)
```



Now we have the boar locations plotted on top of elevation.

Pre-processing of the tracks:

At this point, we can pick up the tracks and convert them to trajectory format for manipulation with adehabitatLT, drawing on the very good online materials on this (https://cran.r-project.org/web/packages/adehabitatLT/vignettes/adehabitatLT.pdf).

The code in the exercise (Section 2.4) converts the relocs SpatialPointsDataFrame object into a more generic R data frame object:

```
> locs <- puechabonsp$relocs
> locs <- as.data.frame(locs)
> locs
     Name Age Sex   Date      X       Y
1   Brock   2   1 930701 699889 3161559
2   Brock   2   1 930703 700046 3161541
3   Brock   2   1 930706 698840 3161033
4   Brock   2   1 930707 699809 3161496
5   Brock   2   1 930708 698627 3160941
6   Brock   2   1 930709 698719 3160989
7   Brock   2   1 930713 698991 3161015
etc
```

This shows you just the first few records instead of all records:

```
> head(locs)
```

You can see from this that there are four wild boar and we have the date and X and Y coordinates for each one. To convert these into trajectory format, we need an ID for each track, plus a timestamp and the X and Y coordinates, held within an R structure called a data frame. The timestamp is the tricky, technical part. It uses some R classes for handling time that are based around one particular standard for handling time in a compatible way across platforms, the Portable Operating System Interface (POSIX) standard.

First, we have to convert the date numbers into character strings. The str R function gives us some details of the structure of any object:

```
> str(locs)
'data.frame':     119 obs. of  6 variables:
 $ Name: Factor w/ 4 levels "Brock","Calou",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Age : int  2 2 2 2 2 2 2 2 2 2 ...
 $ Sex : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Date: int  930701 930703 930706 930707 930708 930709 930713 930714
930715 930720 ...
 $ X   : num  699889 700046 698840 699809 698627 ...
 $ Y   : num  3161559 3161541 3161033 3161496 3160941 ...
```

At the moment, name is int (integers or whole numbers), and we first need to convert this to be a string of characters, as a first step to then converting to a date:

```
> da <- as.character(locs$Date)
> head(da)
> str(da)
 chr [1:119] "930701" "930703" "930706" "930707" "930708" "930709" "930713"
"930714" "930715" "930720" "930723" ...
```

The chr is telling us we are dealing with data of type character (strings), that we have a vector consisting of 119 such strings, and then we see the individual records. The quotation marks indicate that these are now stored as character strings, not numbers.

```
> da <- strptime(da,"%y%m%d", tz="Europe/Paris")
> str(da)
 POSIXlt[1:119], format: "1993-07-01" "1993-07-03" "1993-07-06" "1993-07-
07" "1993-07-08" "1993-07-09" "1993-07-13" "1993-07-14" ...
```

Having converted our vector to be strings rather than numbers, we can now convert it to a vector of `POSIXlt` objects, in other words dates. As the str (structure) command above indicates, you can see the structure has changed. Notice also that da can appear on both the left-hand and right-hand side of the assignment statement <-.

We are still not quite there. We need `POSIXct` objects (i.e. a single number, measured as seconds elapsed since a notional start time in 1970), not `POSIXlt` objects (i.e. a list comprising the date, the time of day etc separately). Fortunately, again 'as' lets us convert between different object classes:

```
> da<-as.POSIXct(da)
> str(da)
 POSIXct[1:119], format: "1993-07-01" "1993-07-03" "1993-07-06" "1993-07-
07" "1993-07-08" "1993-07-09" "1993-07-13" "1993-07-14" ...
```

The other thing that we need to generate our trajectory object is a data frame containing the X and Y locations of the wild boar. We can do this as follows:

```
> boarcoord=locs[,c("X","Y")]
> str(boarcoord)
'data.frame':     119 obs. of  2 variables:
 $ X: num  699889 700046 698840 699809 698627 ...
 $ Y: num  3161559 3161541 3161033 3161496 3160941 ...
```

The [] notation enables us to create a subset of the existing data frame, locs. There are two terms within the square brackets, separated by a comma. The first subsets the rows and the second subsets the columns. Here, the rows part is 'blank' in order to retain all the existing rows, hence why we see [, - there is no subsetting of rows. Had we typed this:

```
> test<-locs[1,]
> str(test)
'data.frame':     1 obs. of  6 variables:
 $ Name: Factor w/ 4 levels "Brock","Calou",..: 1
 $ Age : int 2
 $ Sex : int 1
 $ Date: int 930701
 $ X   : num 7e+05
 $ Y   : num 3161559
```

…we would have ended up with just the first row of the locs data frame. The second part after the comma `c("X","Y")` makes a vector with the names of two elements of the data frame, the ones containing the coordinates. These two elements are then subsetted from the data frame, so we end with a smaller data frame, just containing coordinates.

Now, after all that work, we have a data frame just containing coordinates, and a vector of `POSIXct` objects, so we are ready to create our trajectory.

```
> puech <- as.ltraj(xy = boarcoord, date = da, id = locs$Name)
```

So, what is the difference between storing this information as a trajectory and storing it as a set of points? Simply typing the name of the object gives us some clues:

```
> puech

*********** List of class ltraj ***********

Type of the traject: Type II (time recorded)
* Time zone: Europe/Paris *
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
     id burst nb.reloc NAs date.begin   date.end
1 Brock Brock       30   0 1993-07-01 1993-08-31
2 Calou Calou       19   0 1993-07-03 1993-08-31
3  Chou  Chou       40   0 1992-07-29 1993-08-30
4  Jean  Jean       30   0 1993-07-01 1993-08-31
```

You can see that there are four 'bursts', which in this case are collections of relocations (i.e. successive positional fixes) relating to one of four animals, each with different names (Brock, Calou, Chou and Jean) as specified by the ID part of `as.ltraj`. Looking at the burst characteristics, there are 30 relocations (positional fixes) for the animal called Brock, with no missing positional fixes (`NAs)`, and the positional fixes for Brock relate to a period between July and August 1993. Notice the longer period for the boar called 'Chou'. If the same animal was tracked over multiple, separated periods, we might want to further split out the data for each of these periods (which is what the Calenge exercise does subsequently). The R structure used to store the trajectories is a list. If we wish to refer to one particular 'burst' or item within the list, following standard R syntax, we therefore need to use double square brackets, e.g. `puech[[2]]` refers to the second burst.

If you look at an individual 'burst', you can see that alongside the coordinates and associated timestamp, additional information that has been calculated:

```
> head(puech[[1]])
       x       y       date    dx   dy      dist     dt     R2n   abs.angle  rel.angle
1 699889 3161559 1993-07-01   157  -18  158.0285 172800       0 -0.11415127         NA
2 700046 3161541 1993-07-03 -1206 -508 1308.6252 259200   24973 -2.74292194 -2.6287707
3 698840 3161033 1993-07-06   969  463 1073.9320  86400 1377077  0.44574032 -3.0945230
4 699809 3161496 1993-07-07 -1182 -555 1305.8135  86400   10369 -2.70260603  3.1348390
5 698627 3160941 1993-07-08    92   48  103.7690  86400 1974568  0.48088728 -3.0996920
6 698719 3160989 1993-07-09   272   26  273.2398 345600 1693800  0.09529869 -0.3855886
```

Each row in the output is a relocation, with some associated properties. These are clearly explained in the Calenge exercise on pp. 5-6. For each 'burst' (set of positional fixes sharing a unique identifier). As some additional clarification alongside this explanation:

- dt is the time in seconds between each positional fix, since a `POSIXct` object measures time in seconds. You can see that the first two relocations above are 2 days apart, which translates into 172,800 seconds.
- Dx, dy, dist and R2n are all measured in metres.
- Abs.angle and rel.angle are measured in radians, rather than degrees.

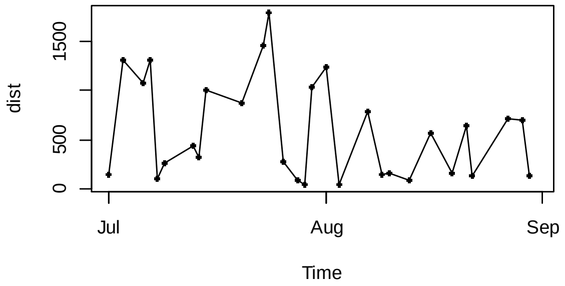
All of this could provide extra analytical insights on animal movement not available from data stored as points.

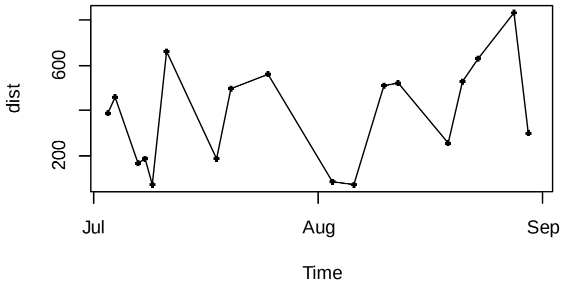
We can get a handle on each of these movement characteristics using a graphing tool specifically designed for plotting trajectory characteristics over time called plotltr. We can find out more about this:

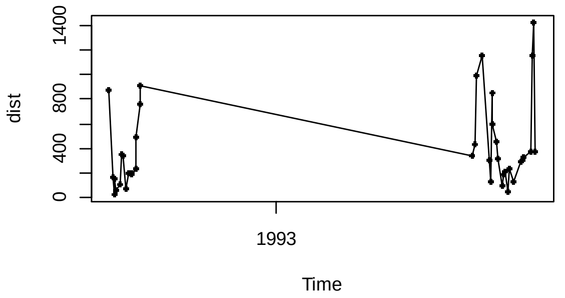
```
> ?plotltr
```

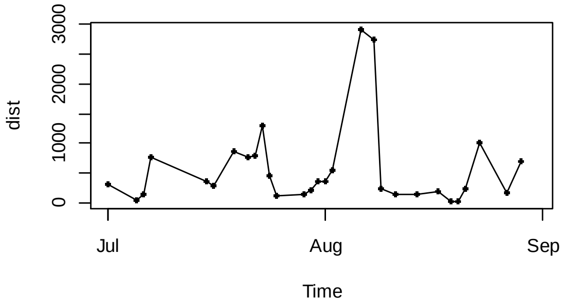
So, typing this would enable us to plot out the distances between successive relocations over time:

```
> plotltr(puech, "dist")
```









You could also experiment with using plotltr to explore other trajectory characteristics, such as moving angles. Notice the big gap in time between two of the successive relocations for the animal shown in the bottom-left graphic (likely to be the boar called 'Chou' if you think back to the earlier information on the periods covered by each 'burst'). Notice also that across all four animals' trajectories, the time intervals between successive relocations vary in length – the distance along the x-axis between successive data points varies.

If we were to get a really good handle on the nature of boar movement, we would need to fix both issues first. Clearly, the distance or movement angle calculated between two relocations that are many months apart is really very different to that between relocations a day apart – hence why splitting the bottom-left graph out into two separate bursts would be sensible. Similarly, in comparing distances and movement angles, it would help greatly to use a regular interval between successive relocations (e.g. 1 day), rather than an irregular interval that could be 1 day or a week. Although we will stop here, the Calenge exercise goes on to address both issues. It splits up the track for the boar called Chou into two separate bursts, reflecting the two periods of data collection. It also explores irregularities in the time intervals between successive relocations, correcting these by interpolating movement between known positional fixes.