Home Range Estimation in R: the `adehabitatHR` Package

Clement Calenge,
Office national de la classe et de la faune sauvage
Saint Benoist – 78610 Auffargis – France.

Mar 2011

Contents

1 History of the package adehabitatHR

The package `adehabitatHR` contains functions dealing with home-range analysis that were originally available in the package `adehabitat` (Calenge, 2006). The data used for such analysis are generally relocation data collected on animals monitored using VHF or GPS collars.

I developped the package `adehabitat` during my PhD (Calenge, 2005) to make easier the analysis of habitat selection by animals. The package `adehabitat` was designed to extend the capabilities of the package `ade4` concerning studies of habitat selection by wildlife.

Since its first submission to CRAN in September 2004, a lot of work has been done on the management and analysis of spatial data in R, and especially with the release of the package `sp` (Pebesma and Bivand, 2005). The package `sp` provides classes of data that are really useful to deal with spatial data...

In addition, with the increase of both the number (more than 250 functions in Oct. 2008) and the diversity of the functions in the package `adehabitat`, it soon became apparent that a reshaping of the package was needed, to make its content clearer to the users. I decided to "split" the package `adehabitat` into

four packages:

- `adehabitatHR` package provides classes and methods for dealing with home range analysis in R.
- `adehabitatHS` package provides classes and methods for dealing with habitat selection analysis in R.
- `adehabitatLT` package provides classes and methods for dealing with animals trajectory analysis in R.
- `adehabitatMA` package provides classes and methods for dealing with maps in R.

We consider in this document the use of the package `adehabitatHR` to deal with home-range analysis. All the methods available in `adehabitat` are also available in `adehabitatHR`, but the classes of data returned by the functions of `adehabitatHR` are completely different from the classes returned by the same functions in `adehabitat`. Indeed, the classes of data returned by the functions of `adehabitatHR` have been designed to be compliant with the classes of data provided by the package `sp`.

Package `adehabitatHR` is loaded by

```
> library(adehabitatHR)

deldir 0.0-13

     Please note: The process for determining duplicated points
     has changed from that used in version 0.0-9 (and previously).
```

2 Basic summary of the functions of the packages

The package `adehabitatHR` implements the following home range estimation methods:

1. The Minimum Convex Polygon (Mohr, 1947);
2. Several kernel home range methods:
   - The "classical" kernel method (Worton, 1989)
   - the Brownian bridge kernel method (Bullard, 1999, Horne et al. 2007);

- The Biased random bridge kernel method, also called "movement-based kernel estimation" (Benhamou and Cornelis, 2010, Benhamou, 2011);
- the product kernel algorithm (Keating and Cherry, 2009).

3. Several home-range estimation methods relying on the calculation of convex hulls:
   - The modification by Kenward et al. (2001) of the single-linkage clustering algorithm;
   - The three LoCoH (Local Convex Hull) methods developed by Getz et al. (2007)
   - The characteristic hull method of Downs and Horner (2009).

I describe these functions in this vignette.

2.1 The classes of data required as input

The package `adehabitatHR`, as all the brother "adehabitat" packages, relies on the classes of data provided by the package `sp`. Two types of data can be accepted as input of the functions of the package `adehabitatHR`:

- objects of the class `SpatialPoints` (package `sp`). These objects are designed to store the coordinates of points into space, and are especially well designed to store the relocations of **one** animal monitored using radio-tracking.

- objects of the class `SpatialPointsDataFrame` (package `sp`). These objects are also designed to store the coordinates of points into space, but also store additionnal information concerning the points (attribute data). This class of data is especially well designed to store the relocations of several animals monitored using radio-tracking. In this case, the attribute data should contain only one variable: a factor indicating the identity of the animals.

All the home-range estimation methods included in the package can take an object belonging to one of these two classes as the argument `xy`. For example, generate a sample of relocations uniformly distributed on the plane:

```
> xy <- matrix(runif(60), ncol = 2)
> head(xy)
```

```
          [,1]       [,2]
[1,] 0.9501194 0.05213269
[2,] 0.5319651 0.46129166
[3,] 0.3474738 0.45718668
[4,] 0.8702528 0.33284070
[5,] 0.8099797 0.56469108
[6,] 0.6582251 0.47367876
```

The object `xy` is a matrix containing 30 relocations of a virtual animal. Convert it into the class `SpatialPoints`:

```
> xysp <- SpatialPoints(xy)
```

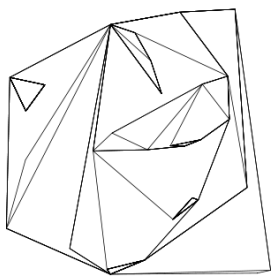
Consider the function `clusthr`, which implements the single-linkage clustering algorithm (see next sections). Using the function `clusthr` on this object returns an object of class `SpatialPolygonsDataFrame`:

```
> clu <- clusthr(xysp)
> class(clu)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

Show the results:

```
> plot(clu)
```



Now, consider the relocations of another animal:

```
> xy2 <- matrix(runif(60), ncol = 2)
```

We can bind the two matrices together:

```
> xyt <- rbind(xy, xy2)
```

and generate a vector containing the ID of the animals for each relocation of the object `xyt`:

```
> id <- gl(2, 30)
```

We can convert the object `id` into a `SpatialPointsDataFrame`:

```
> idsp <- data.frame(id)
> coordinates(idsp) <- xyt
> class(idsp)

[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

And we can use the same function `clusthr` on this new object:

```
> clu2 <- clusthr(idsp)
> class(clu2)

[1] "MCHu"

> clu2

********** Multiple convex hull Home range of several Animals ************

This object is a list with one component per animal.
Each component is an object of class SpatialPolygonsDataFrame
The home range has been estimated for the following animals:
[1] "1" "2"
```

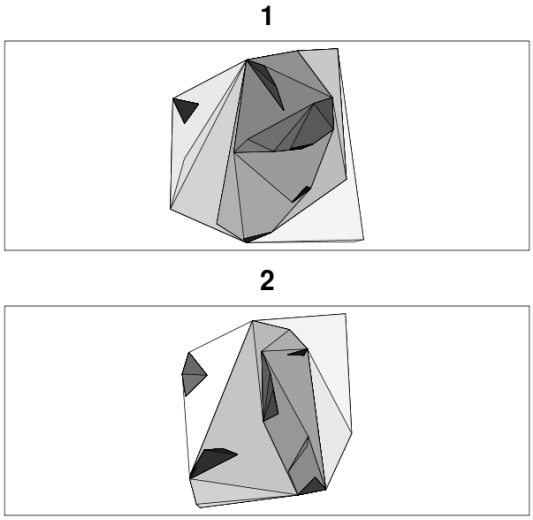
An object of the class `"MCHu"` is basically a list of objects of class `SpatialPolygonsDataFrame`, with one element per animal. Thus:

```
> length(clu2)

[1] 2

> class(clu2[[1]])

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

```
> class(clu2[[2]])

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

Although each `SpatialPolygonsDataFrame` of the list can be easily handled by the functions of the package `sp` and relatives (`maptools`, etc.), the package `adehabitatHR` also contains functions allowing to deal with such lists. For example:

```
> plot(clu2)
```



Therefore, the same home-range estimating function can accept an object of class `SpatialPoints` (one animal) or `SpatialPointsDataFrame` (several animals). This is the case of all home-range estimation methods in `adehabitatHR`.

2.2 Our example data set

I will illustrate the use of the functions in `adehabitatHR` using a subset of the data collected by Daniel Maillard (1996; *Office national de la chasse et de la faune sauvage*, Montpellier) on the wild boar at Puechabon (near Montpellier, South of France). First load the dataset `puechabonsp`:

```
> data(puechabonsp)
> names(puechabonsp)

[1] "map"    "relocs"
```

This dataset is a list with two components: the component `relocs` is a `SpatialPointsDataFrame` containing the relocations of four wild boars monitored using radio-tracking. The following variables are available for each relocations:

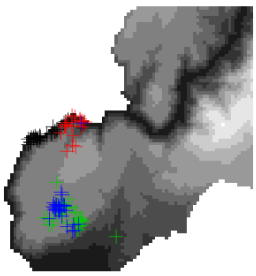
```
> head(as.data.frame(puechabonsp$relocs))
```

```
   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
```

The first variable is the identity of the animals, and we will use it in the estimation process.

The component `map` is a `SpatialPixelsDataFrame` containing the maps of four environmental variables on the study area (Elevation, slope, aspect and herbaceous cover). Have a look at the distribution of the relocations on an elevation map:

```
> ## Map of the elevation
> image(puechabonsp$map, col=grey(c(1:10)/10))
> ## map of the relocations
> plot(puechabonsp$relocs, add=TRUE,
+      col=as.data.frame(puechabonsp$relocs)[,1])
> NA
```



3 The Minimum Convex Polygon (MCP)

3.1 The method

The MCP is probably the most widely used estimation method. This method consists in the calculation of the smallest convex polygon enclosing all the relocations of the animal. This polygon is considered to be the home range of the animal. Because the home range has been defined as the area traversed by the animal during its **normal** activities of foraging, mating and caring for young (Burt, 1943), a common operation consists to first remove a small percentage of the relocations farthest from the centroid of the cloud of relocations before the estimation. Indeed, the animal may sometimes make occasionnal large moves to unusual places outside its home range, and this results into outliers that cannot be considered as "normal activities".

3.2 The function `mcp` and interaction with the package `sp`

The function `mcp` allows the MCP estimation. For example, using the `puechabonsp` dataset, we estimate here the MCP of the monitored wild boars (after the removal of 5% of extreme points). Remember that the first column of `puechabonsp$relocs` is the identity of the animals:

```
> data(puechabonsp)
> cp <- mcp(puechabonsp$relocs[, 1], percent = 95)
```

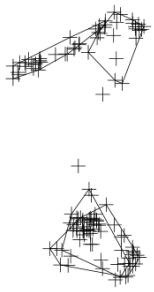
The results are stored in an object of class `SpatialPolygonsDataFrame`:

```
> class(cp)
```

```
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

Therefore the functions available in the package `sp` and related packages (`maptools`, etc.) are available to deal with these objects. For example, the function `plot`:

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



Similarly, the function `writePolyShape` from the package `maptools` can be used to export the object `cp` to a shapefile, so that it can be read into a GIS:

```
> library(maptools)
> writePolyShape(cp, "homerange")
```

The resulting shapefils can be read in any GIS.

3.3 Overlay operations

The function `overlay` of the package `sp` allows to combine points (including grid of pixels) and polygons for points-in-polygon operations (see the help page of this function). For example, the element `map` of the dataset `puechabonsp` is a map of the study area according to four variables:

```
> head(as.data.frame(puechabonsp$map))
```

```
     Elevation Aspect    Slope Herbaceous      x       y
2017        68      3 1.548994          0 698800 3156800
2018        69      3 1.118594          0 698900 3156800
2019        70      3 1.358634          0 699000 3156800
2020        70      3 1.724311          0 699100 3156800
2021        70      3 1.944885          0 699200 3156800
2022        71      3 2.024868          0 699300 3156800
```

This object is of the class `SpatialPixelsDataFrame`. We may identify the pixels of this map enclosed in each home range using the function `overlay` from the package `sp`. For example, to identify the pixels enclosed in the home range of the second animal:

```
> enc <- overlay(puechabonsp$map, cp[2, ])
> head(enc)
```

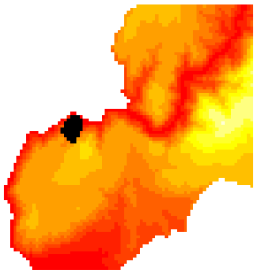
```
[1] NA NA NA NA NA NA
```

The result is a vector, taking the value NA if the pixel is outside the home-range, and 1 otherwise. We may transform this vector into a `SpatialPixelsDataFrame`, using the functions of the package `sp`:

```
> enc <- data.frame(enc)
> coordinates(enc) <- coordinates(puechabonsp$map)
> gridded(enc) <- TRUE
```

The result is shown below, in black, on an elevation map:

```
> image(puechabonsp$map)
> image(enc, add = TRUE, col = "black", useRasterImage = FALSE)
```

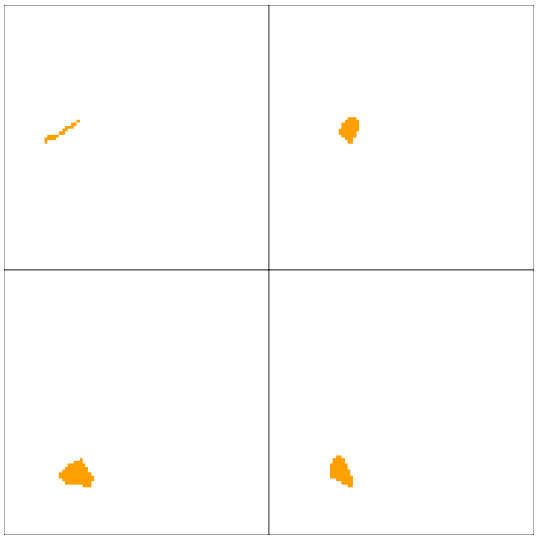


However, because the rasterization of home ranges is a common operation, I included a function `hr.rast` to make this operation more straightforward:

```
> cprast <- hr.rast(cp, puechabonsp$map)
```

The resulting object is a `SpatialPixelsDataFrame` with one column per animal:

```
> par(mar = c(0, 0, 0, 0))
> par(mfrow = c(2, 2))
> for (i in 1:4) {
+     image(cprast[, i], useRasterImage = FALSE)
+     box()
+ }
```



3.4 Computation of the home-range size

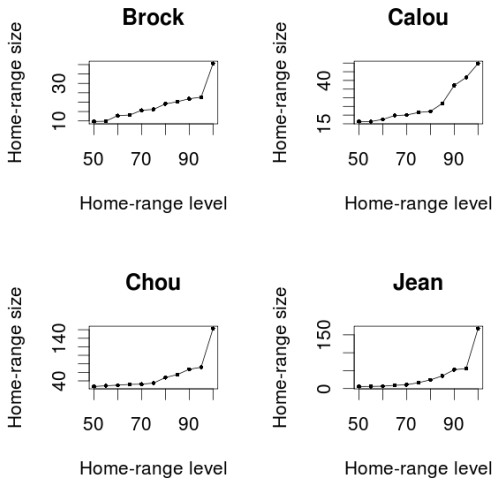
The home range size is automatically computed by the function `mcp`. Thus, coercing the object returned by the function to the class `data.frame` gives access to the home range size:

```
> as.data.frame(cp)

         id     area
Brock Brock 22.64670
Calou Calou 41.68815
Chou   Chou 71.93205
Jean   Jean 55.88415
```

The area is here expressed in hectares (the original units in the object `puechabonsp$relocs` were expressed in metres). The units of the results and of the original data are controlled by the arguments `unin` and `unout` of the function `mcp` (see the help page of the function `mcp`). We did not change the units in our example estimation, as the default was a suitable choice.

In this example, we have chosen to exclude 5% of the mst extreme relocations, but we could have made another choice. We may compute the home-range size for various choices of the number of extreme relocations to be excluded, using the function `mcp.area`:

```
> hrs <- mcp.area(puechabonsp$relocs[, 1], percent = seq(50, 100,
+     by = 5))
```



Here, except for Calou, the choice to exclude 5% of the relocations seems a sensible one. In the case of Calou, it is disputable to remove 20% of the relocations to achieve an asymptote in the home range size decrease. 20% of the time spent in an area is no longer an exceptionnal activity... and this area could be considered as part of the home range. Under these conditions, the choice to compute the MCP with 95% of the relocations for all animals seems a sensible one.

Note that the results of the function `mcp.area` are stored in a data frame of class `"hrsize"` and can be used for further analysis:

```
> hrs
```

```
       Brock    Calou      Chou      Jean
50   9.68735 16.31155  26.76340   5.72115
55   9.85540 16.31155  28.51200   5.99155
60  12.84635 17.49680  29.97835   6.64385
65  13.12805 19.74600  31.75860   9.05275
70  15.66545 20.05745  32.25850  10.58390
75  16.14625 21.53025  34.77430  16.40945
80  19.06725 22.15015  47.81925  24.29185
85  20.22535 26.75180  54.76550  35.31225
90  21.82440 37.12975  67.18565  52.59315
95  22.64670 41.68815  71.93205  55.88415
100 40.59635 49.75050 162.84550 167.63690
```

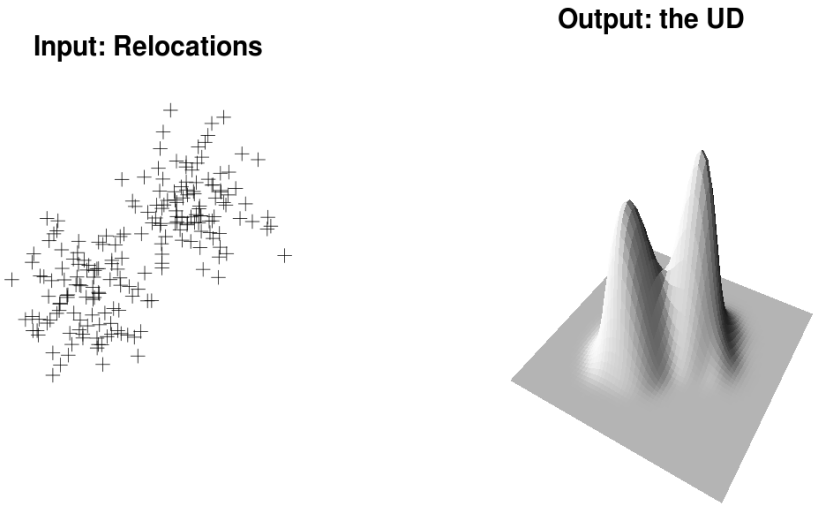
This class of data is common in `adehabitatHR` and we will generate again such objects with other home range estimation methods.

4 The kernel estimation and the utilization distribution

4.1 The utilization distribution

The MCP has met a large success in the ecological literature. However, many authors have stressed that the definition of the home range which is commonly used in the literature was rather imprecise: "*that area traversed by the animal during its normal activities of food gathering, mating and caring for young*" (Burt, 1943). Although this definition corresponds well to the feeling of many ecologists concerning what is the home range, it lacks formalism: what is an *area traversed*? what is a *normal activity*?

Several authors have therefore proposed to replace this definition by a more formal model: the *utilization distribution* (UD, van Winkle, 1975). Under this model, we consider that the animals use of space can be described by a bivariate probability density function, the UD, which gives the probability density to relocate the animal at any place according to the coordinates (x, y) of this place. The study of the space use by an animal could consist in the study of the properties of the utilization distribution. The issue is therefore to estimate the utilization distribution from the relocation data:



The seminal paper of Worton (1989) proposed the use of the kernel method (Silverman, 1986; Wand and Jones, 1995) to estimate the UD using the relocation data. The kernel method was probably the most frequently used function in the package `adehabitat`. A lot of useful material and information on its use can be found on the website of the group Animove (`http://www.faunalia.it/animov/`). I give here a summary of the discussion found

on this site (they include answers to the most frequently asked questions).

The basic principle of the method is the following: a bivariate *kernel function* is placed over each relocation, and the values of these functions are averaged together.

Many kernel functions K can be used in the estimation process provided that:

$$\int K(\mathbf{x}) = 1$$

and

$$K(\mathbf{x}) > 0 \tilde{} \tilde{} \tilde{} \tilde{} \forall \mathbf{x}$$

where $\mathbf{x}$ is a vector containing the coordinates of a point on the plane. However, there are two common choices for the kernel functions: the bivariate normal kernel:

$$K(\mathbf{x}) = \frac{1}{2\pi} \exp(-\frac{1}{2}\mathbf{x}^t\mathbf{x})$$

and the Epanechnikov kernel:

$$K(\mathbf{x}) = \frac{3}{4}(1 - \mathbf{x}^t\mathbf{x}) \tilde{} \tilde{} \tilde{} \tilde{} \text{if} \tilde{} x < 1 \tilde{} \text{or} \tilde{} 0 \tilde{} \text{otherwise}$$

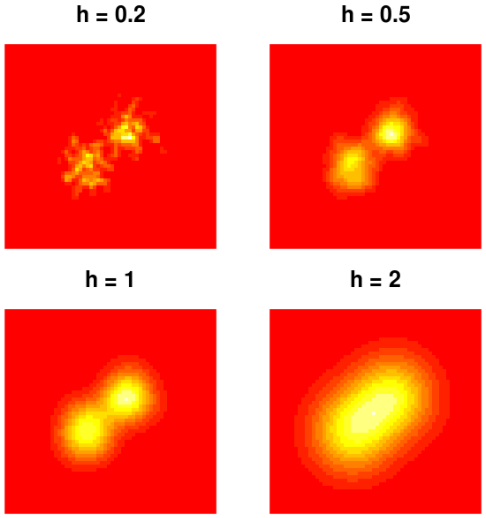
The choice of a kernel function does not greatly affect the estimates (Silverman, 1986; Wand and Jones, 1995). Although the Epanechnikov kernel is slightly more efficient, the bivariate normal kernel is still a common choice (and is the default choice in `adehabitatHR`).

Then, the kernel estimation of the UD at a given point $\mathbf{x}$ of the plane is obtained by:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^2} \sum_{i=1}^{n} K \left\{ \frac{1}{h}(\mathbf{x} - \mathbf{X}_i) \right\}$$

where h is a smoothing parameter, n is the number of relocations, and $\mathbf{X}_i$ is the i^{th} relocation of the sample.

The smoothing parameter h controls the "width" of the kernel functions placed over each point. On an example dataset:



There has been a considerable amount of work on the correct choice of a smoothing parameter for home range estimation. However, the two most common choices are the so-called "reference bandwidth". When a bivariate normal kernel has been chosen, this "reference bandwidth" is equal to:

$$h = \sigma \times n^{-1/6}$$

where

$$\sigma = 0.5 \times (\sigma_x + \sigma_y)$$

and σ_x and σ_x are the standard deviations of the x and y coordinates of the relocations respectively. If an Epanechnikov kernel is used, this value is multiplied by 1.77 (Silverman, 1986, p. 86). The reference bandwidth supposes that the UD is a bivariate normal distribution, which is disputable in most ecological studies. When the animals uses several centers of activity, the reference smoothing parameter is often too large, which results into a strong oversmoothing of the data (the estimated UD predicts the frequent presence of the animal in areas which are not actually used).

Alternatively, the smoothing parameter h may be computed by Least Square Cross Validation (LSCV). The estimated value then minimizes the Mean Integrated Square Error (MISE), i.e. the difference in volume between the true UD and the estimated UD.

Finally, a subjective visual choice for the smoothing parameter, based on successive trials, is often a sensible choice (Silverman, 1986; Wand and Jones, 1995).

4.2 The function kernelUD: estimating the utilization distribution

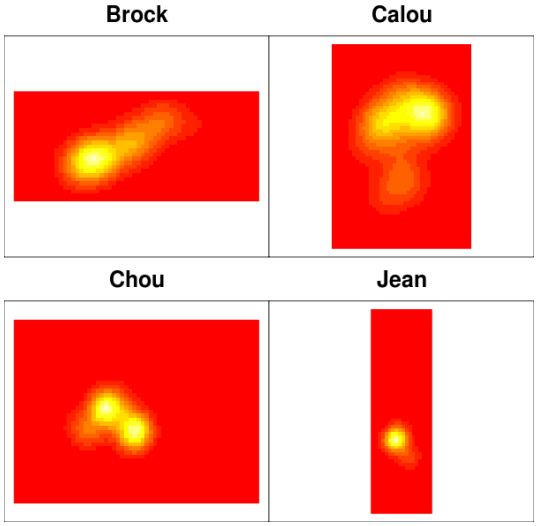
The function kernelUD of the package adehabitatHR implements the method described above to estimate the UD. The UD is estimated in each pixel of a grid superposed to the relocations. I describe in the next sections how the grid parameters can be controlled in the estimation. Similarly to all the functions allowing the estimation of the home range, the functions may take as argument either a SpatialPoints object (estimation of the UD of one animal) or a SpatialPointsDataFrame object with one column corresponding to the identity of the animal (estimation of the UD of several animals). The argument h controls the value of the smoothing parameter. Thus, if h = "href", the "reference" bandwidth is used in the estimation. If h = "LSCV", the "LSCV" bandwidth is used in the estimation. Alternatively, it is possible to pass a numeric value for the smoothing parameter.

I give below a short example of the use of kernelUD, using the puechabonsp dataset. Remember that the first column of the component relocs of this dataset contains the identity of the animals:

```
> data(puechabonsp)
> kud <- kernelUD(puechabonsp$relocs[, 1], h = "href")
> kud

********** Utilization distribution of several Animals ************

Type: probability density
Smoothing parameter estimated with a  href 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
```

The resulting object is a list of the class "estUD". This class extends the class SpatialPixelsDataFrame (it just contains an additional attribute storing the information about the smoothing parameter). Display the results:

```
> image(kud)
```



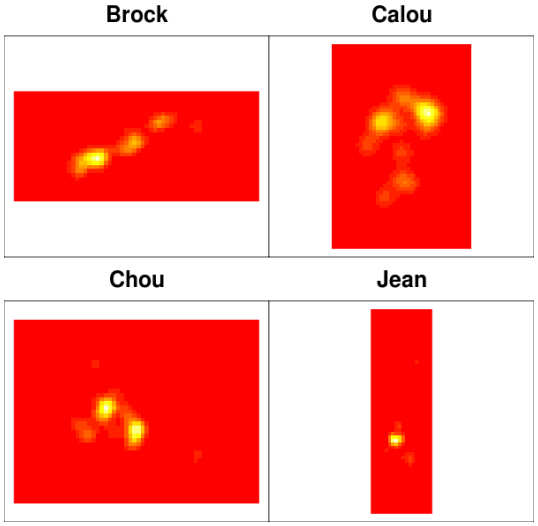
The values of the smoothing parameters are stored in the slot `"h"` of each element of the list. For example, to get the h-value for the first animal:

```
> kud[[1]]@h

$h
[1] 211.6511

$meth
[1] "href"
```

4.3 The Least Square Cross Validation

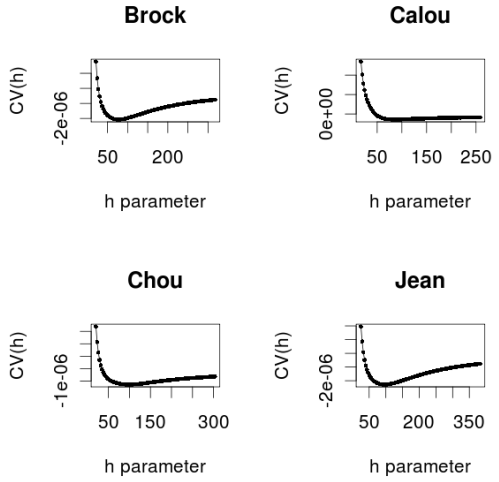
Alternatively, we could have set the argument `h` equal to `"LSCV"`. The LSCV algorithm searches for the optimum value of h in the interval specified by the parameter `hlim`. As an example, estimate the UD with a h parameter chosen with the LSCV algorithm for the `puechabonsp` dataset:

```
> kudl <- kernelUD(puechabonsp$relocs[, 1], h = "LSCV")
> image(kudl)
```



The resulting UD fits the data more closely than the use of "href". However, note that the cross-validation criterion cannot be minimized in some cases. According to Seaman and Powell (1998) "*This is a difficult problem that has not been worked out by statistical theoreticians, so no definitive response is available at this time*". Therefore, it is essential to look at the results of the LSCV minimization using the function `plotLSCV`:

```
> plotLSCV(kudl)
```



Here, the minimization has been achieved in the specified interval. When the algorithm does not converge toward a solution, the estimate should not be used in further analysis. Solutions are discussed on the animove website (http://www.faunalia.it/animov/), and especially on the mailing list. In particular, see the following messages:

- http://www.faunalia.com/pipermail/animov/2006-May/000137.html
- http://www.faunalia.com/pipermail/animov/2006-May/000130.html
- http://www.faunalia.com/pipermail/animov/2006-May/000165.html

Finally, the user may indicate a numeric value for the smoothing parameter. Further discussion on the choice of h-values can be found on the animove website, at the URL: https://www.faunalia.it/dokuwiki/doku.php?id=public:animove.

4.4 Controlling the grid

The UD is estimated at the center of each pixel of a grid. Although the size and resolution of the grid does not have a large effect on the estimates (see Silverman, 1986), it is sometimes useful to be able to control the parameters defining this grid. This is done thanks to the parameter `grid` of the function `kernelUD`.

4.4.1 Passing a numeric value

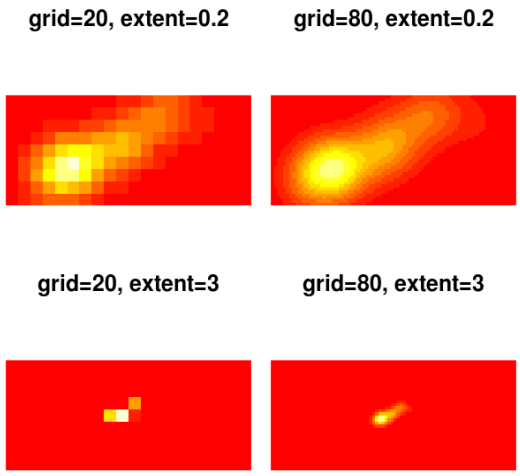
When a numeric value is passed to the parameter `grid`, the function `kernelUD` automatically computes the grid for the estimation. This grid is computed in the following way:

- the minimum and maximum coordinates $(X_{min}, X_{max}, Y_{min}, Y_{max})$ of the relocations are computed.
- the range of X and Y values is also computed (i.e., the difference between the maximum and minimum value). Let R_x and R_Y be these ranges.
- The coverage of the grid is defined thanks to the parameter `extent` of the function `kernelUD`. Let e be this parameter. The minimum and maximum X coordinates of the pixels of the grid are equal to $X_{min} - e \times R_x$ and $X_{max} + e \times R_x$ respectively. Similarly, the minimum and maximum Y coordinates of the pixels of the grid are equal to $Y_{min} - e \times R_Y$ and $Y_{max} + e \times R_Y$ respectively.

- Finally, these extents are split into g intervals, where g is the value of the parameter `grid`.

Consequently, the parameter `grid` controls the resolution of the grid and the parameter `extent` controls its extent. As an illustration, consider the estimate of the first animal:

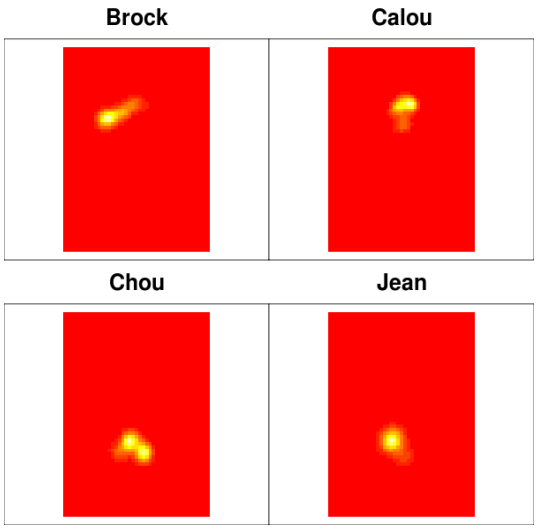
```
> ## The relocations of "Brock"
> locs <- puechabonsp$relocs
> firs <- locs[as.data.frame(locs)[,1]=="Brock",]
> ## Graphical parameters
> par(mar=c(0,0,2,0))
> par(mfrow=c(2,2))
> ## Estimation of the UD with grid=20 and extent=0.2
> image(kernelUD(firs, grid=20, extent=0.2))
> title(main="grid=20, extent=0.2")
> ## Estimation of the UD with grid=20 and extent=0.2
> image(kernelUD(firs, grid=80, extent=0.2))
> title(main="grid=80, extent=0.2")
> ## Estimation of the UD with grid=20 and extent=0.2
> image(kernelUD(firs, grid=20, extent=3))
> title(main="grid=20, extent=3")
> ## Estimation of the UD with grid=20 and extent=0.2
> image(kernelUD(firs, grid=80, extent=3))
> title(main="grid=80, extent=3")
> NA
```



Note that the parameter `same4all` is an additional parameter allowing the control of the grid. If it is equal to TRUE, the same grid is used for all animals,

thus for example:

```
> kus <- kernelUD(puechabonsp$relocs[, 1], same4all = TRUE)
> image(kus)
```



Because all the UD are estimated on the same grid, it is possible to coerce the resulting object as a `SpatialPixelsDataFrame`:

```
> ii <- estUDm2spixdf(kus)
> class(ii)
```

```
[1] "SpatialPixelsDataFrame"
attr(,"package")
[1] "sp"
```

This object can then be handed with the functions of the package `sp`.

4.4.2 Passing a `SpatialPixelsDataFrame`

It is sometimes useful to estimate the UD in each pixel of a raster map already available. This can be useful in habitat selection studies, when the value of the UD is to be later related to the value of environmental variables. In such cases, it is possible to pass a raster map to the parameter `grid` of the function. For example, the dataset `puechabonsp` contains an object of class `SpatialPixelsDataFrame` storing the environmental information for this variable. This map can be passed to the parameter `grid`:

```
> kudm <- kernelUD(puechabonsp$relocs[, 1], grid = puechabonsp$map)
```

And as in the previous section, the object can be coerced into a `SpatialPixelsDataFrame` using the function `estUDm2spixdf`. Note also that it is possible to pass a list of `SpatialPixelsDataFrame` (with one element per animal) to the parameter `grid` (see the help page of the function `kernelUD`).

4.5 Estimating the home range from the UD

The UD gives the probability density to relocate the animal at a given place. The home range deduced from the UD as the minimum area on which the probability to relocate the animal is equal to a specified value. For example, the 95% home range corresponds to the smallest area on which the probability to relocate the animal is equal to 0.95. The functions `getvolumeUD` and `getverticeshr` provide utilities for home range estimation.

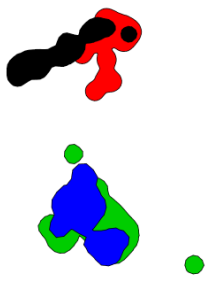
4.5.1 Home ranges in vector mode

The most useful function is the function `getverticeshr`. For example, to deduce the 90% home range from the UD estimated using the LSCV algorithm:

```
> homerange <- getverticeshr(kudl)
> class(homerange)

[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
```

The resulting object is of the class `SpatialPolygonsDataFrame`. Therefore, as for the MCP (see previous section), the functions of the package `sp` and `maptools` are available to deal with this object, or to export it toward a GIS (see section 3.2 for examples). Therefore, to display the home range:

```
> plot(homerange, col = 1:4)
```



4.5.2 Home ranges in raster mode

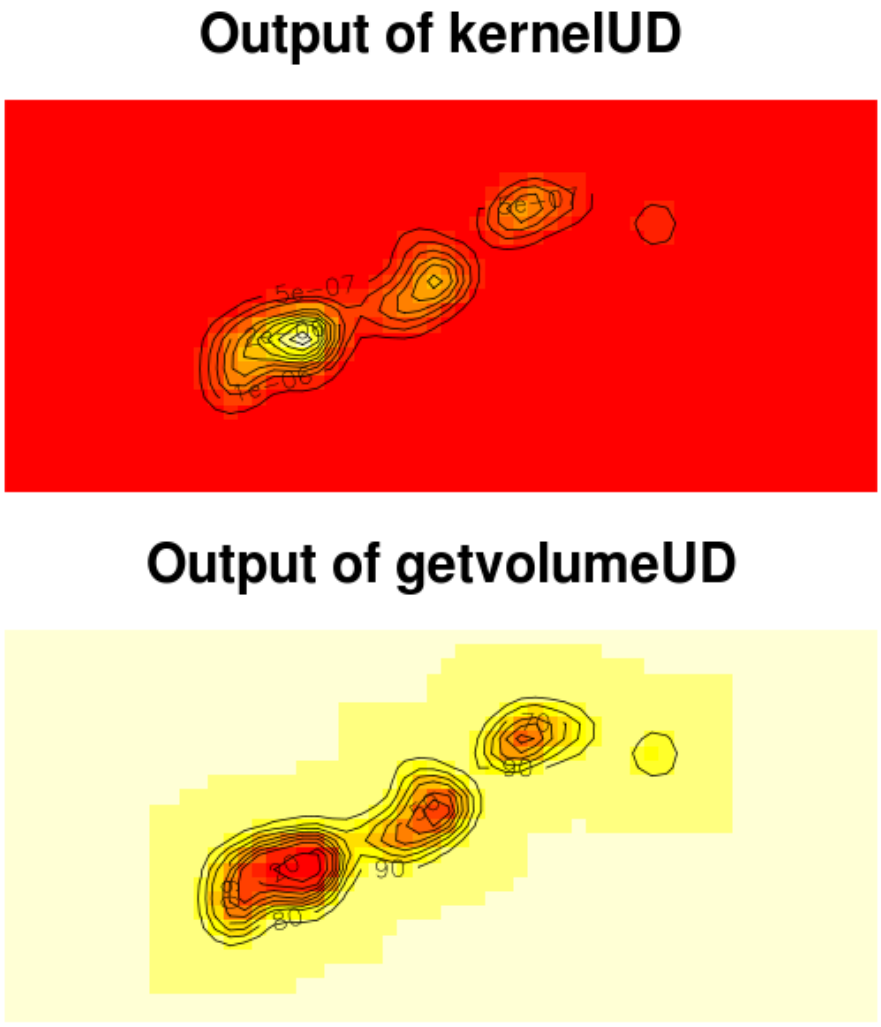
However, the function `getverticeshr` returns an object of class `SpatialPolygonsDataFrame`, i.e. an object in mode vector.

The function `getvolumeUD` may also be useful to estimate the home range in raster mode, from the UD. Actually, this function modifies the UD component of the object passed as argument, so that the value of a pixel is equal to the percentage of the smallest home range containing this pixel:

```
> vud <- getvolumeUD(kudl)
> vud

********** Utilization distribution of several Animals ************

Type: volume under UD
Smoothing parameter estimated with a  LSCV 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
```

To make clear the differences between the output of `kernelUD` and `getvolumeUD` look at the values on the following contourplot:

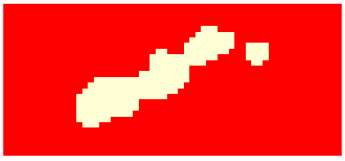
```
> ## Set up graphical parameters
> par(mfrow=c(2,1))
> par(mar=c(0,0,2,0))
> ## The output of kernelUD for the first animal
> image(kudl[[1]])
> title("Output of kernelUD")
> ## Convert into a suitable data structure for
> ## the use of contour
> xyz <- as.image.SpatialGridDataFrame(kudl[[1]])
> contour(xyz, add=TRUE)
> ## and similarly for the output of getvolumeUD
> par(mar=c(0,0,2,0))
> image(vud[[1]])
> title("Output of getvolumeUD")
> xyzv <- as.image.SpatialGridDataFrame(vud[[1]])
> contour(xyzv, add=TRUE)
> NA
```



Whereas the output of `kernelUD` is the raw UD, the output of `getvolumeUD` can be used to compute the home range (the labels of the contour lines correspond to the home ranges computed with various probability levels). For example, to get the rasterized 95% home range of the first animal:

```
> ## store the volume under the UD (as computed by getvolumeUD)
> ## of the first animal in fud
> fud <- vud[[1]]
> ## store the value of the volume under UD in a vector hr95
> hr95 <- as.data.frame(fud)[,1]
> ## if hr95 is <= 95 then the pixel belongs to the home range
> ## (takes the value 1, 0 otherwise)
```

```
> hr95 <- as.numeric(hr95 <= 95)
> ## Converts into a data frame
> hr95 <- data.frame(hr95)
> ## Converts to a SpatialPixelsDataFrame
> coordinates(hr95) <- coordinates(vud[[1]])
> gridded(hr95) <- TRUE
> ## display the results
> image(hr95)
> NA
```



4.6 The home range size

As for the MCP, the `SpatialPolygonsDataFrame` returned by the function `getverticeshr` contains a column named "area", which contains the area of the home ranges:

```
> as.data.frame(homerange)
```

```
         id      area
Brock Brock  72.36203
Calou Calou  81.31172
Chou   Chou 132.75889
Jean   Jean 105.25208
```

And, as for the MCP, the units of these areas are controlled by the parameters `unin` and `unout` of the function `getverticeshr`.

I also considered that it would be useful to have a function which would compute the home-range size for several probability levels. This is the aim of

the function `kernel.area` which takes the UD as argument (here we use the UD estimated with the LSCV algorithm).

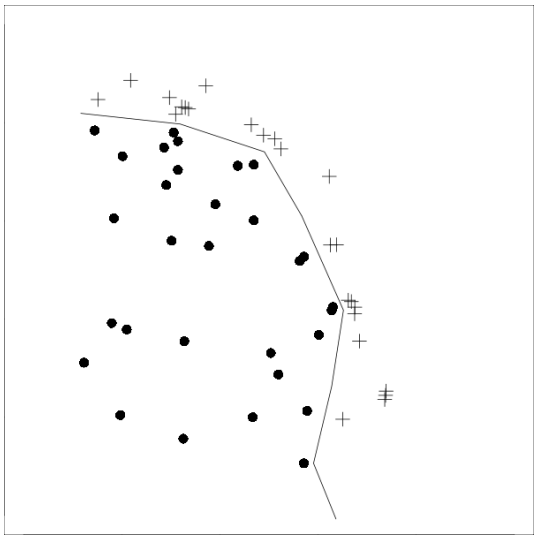
```
> ii <- kernel.area(kudl, percent = seq(50, 95, by = 5))
> ii
```

```
      Brock    Calou      Chou      Jean
50 16.74837 19.48793  27.45380  16.29855
55 19.24281 22.97831  32.82519  19.01497
60 22.44994 27.05042  39.39023  24.44782
65 25.65708 31.41339  45.95527  28.52246
70 29.93326 36.35809  53.71395  35.31352
75 34.20944 41.88452  62.66628  42.10459
80 39.91101 48.28354  74.00589  51.61207
85 46.68163 55.84602  87.13597  62.47777
90 55.94668 65.73542 103.84698  77.41811
95 70.20061 80.42409 130.70396 100.50772
```

The resulting object is of the class `hrsize`, and can be plotted using the function `plot`. Note that the home-range sizes returned by this function are slightly different from the home-range size stored in the `SpatialPolygonsDataFrame` returned by the function `getverticeshr`. Indeed, while the former measures the area covered by the rasterized home range (area covered by the set of pixels of the grid included in the home range), the latter measures the area of the vector home range (with smoother contour). However, note that the difference between the two estimates decrease as the resolution of the grid becomes finer.

4.7 Taking into account the presence of physical boundary over the study area

In a recent paper, Benhamou and Cornelis (2010) proposed an interesting approach to take into account the presence of physical boundaries on the study area. They proposed to extend to 2 dimensions a method developed for one dimension by Silverman 1986. The principle of this approach is described below:



On this figure, the relocations are located near the lower left corner of the map. The line describes a physical boundary located on the study area (e.g. a cliff, a pond, etc.). The animal cannot cross the boundary. The use of the classical kernel estimation will not take into account this boundary when estimating the UD. A classical approach when such problem occur is to use the classical kernel approach and then to set to 0 the value of the UD on the other side of the boundary. However, this may lead to some bias in the estimation of the UD near the boundary (as the kernel estimation will "smooth" the use across the boundary, whereas one side of the boundary is used and the other unused). Benhamou and Cornelis (2010) extended an approach developed for kernel estimation in one dimension: this approach consists in calculating the mirror image of the points located near the boundary, and adding these "mirror points" to the actual relocations before the kernel estimation (These mirror points correspond to the crosses on the figure). **Note that this approach cannot be used for all boundaries**. There are two constraints that have to be satisfied:

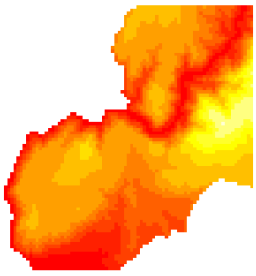
- The boundary should be defined as the union of several segments, *and each segment length should at least be equal to* $3 \times h$, where h is the smoothing parameter used for kernel smoothing;

- The angle between two successive line segments should be greater that $\pi/2$ or lower than $-\pi/2$.

Therefore, the boundary shape should be fairly simple, and not too tortuous.

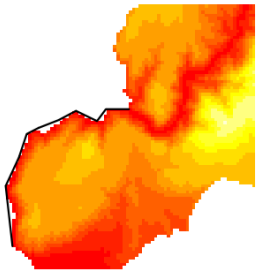
Now, let us consider the case of the wild boar monitored in Puechabon. Consider the map of the elevation on the study area:

```
> image(puechabonsp$map)
```



This area is traversed by the Herault gorges. As can be seen on this figure, the slopes of the gorges are very steep, and it is nearly impossible for the wild boars monitored to cross the river (although it is possible in other places). Therefore, the gorges are a boundary that cannot be crossed by the animal. Using the function `locator()`, we have defined the following boundary:

```
> bound <- structure(list(x = c(701751.385381925, 701019.24105475,
+     700739.303517889, 700071.760160759, 699522.651915378, 698887.40904327,
+     698510.570051342, 698262.932999504, 697843.026694212, 698058.363261028),
+     y = c(3161824.03387414, 3161824.03387414, 3161446.96718494,
+         3161770.16720425, 3161479.28718687, 3161231.50050539,
+         3161037.5804938, 3160294.22044937, 3159389.26039528,
+         3157482.3802813)), .Names = c("x", "y"))
> image(puechabonsp$map)
> lines(bound, lwd = 3)
```

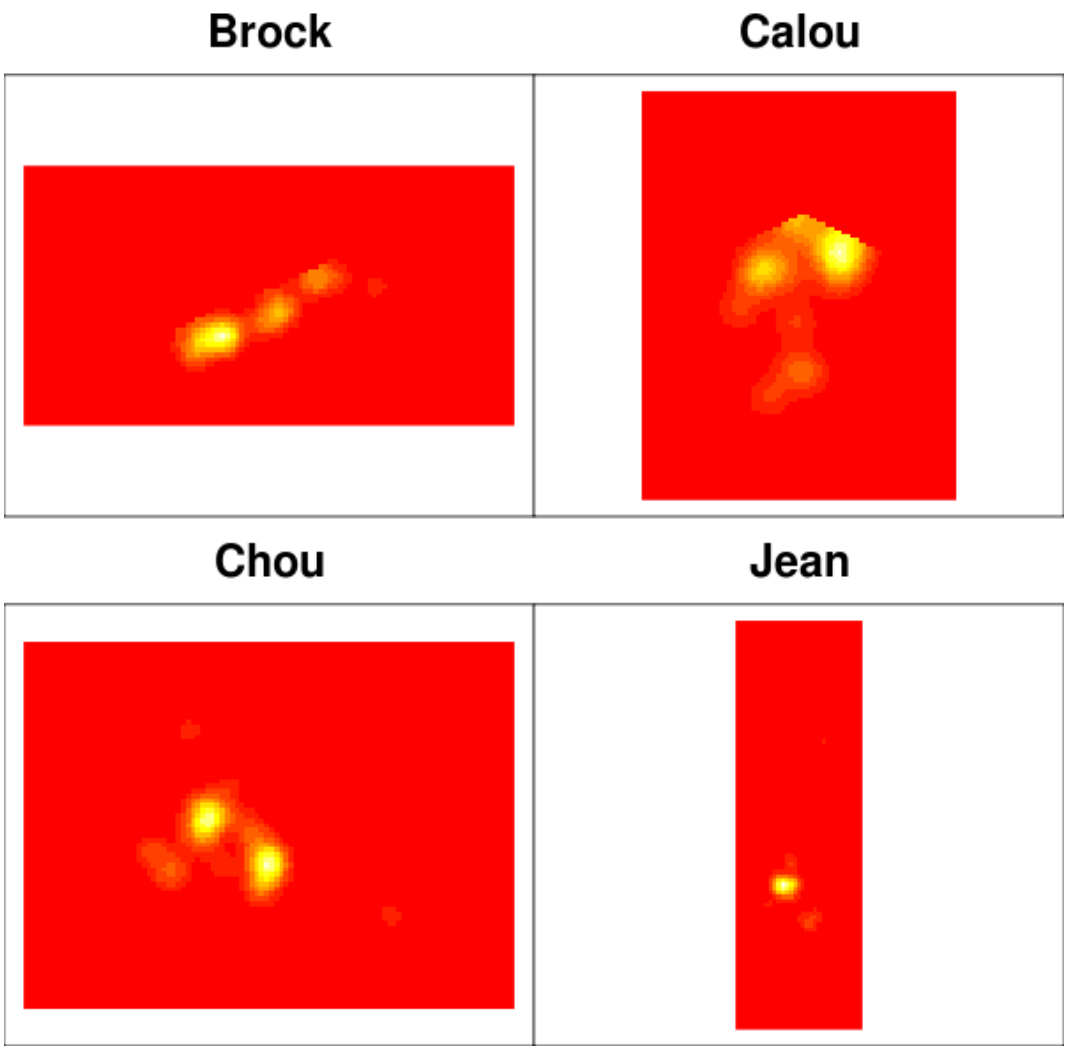


We now convert this boundary as a SpatialLines object:

```
> bound <- do.call("cbind", bound)
> Slo1 <- Line(bound)
> Sli1 <- Lines(list(Slo1), ID = "frontier1")
> barrier <- SpatialLines(list(Sli1))
```

We can now set the parameter `boundary` of the function `kernelUD`, which indicates that we want to use this approach:

```
> kud <- kernelUD(puechabonsp$relocs[, 1], h = 100, grid = 100,
+     boundary = barrier)
> image(kud)
```



The effect of the boundary correction is clear on the UD of Brock and Calou.

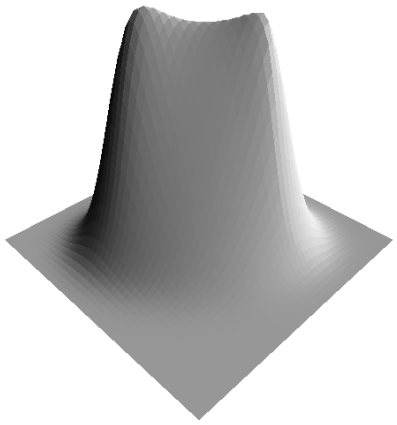
5 Taking into account the time dependence between relocation

5.1 The Brownian bridge kernel method

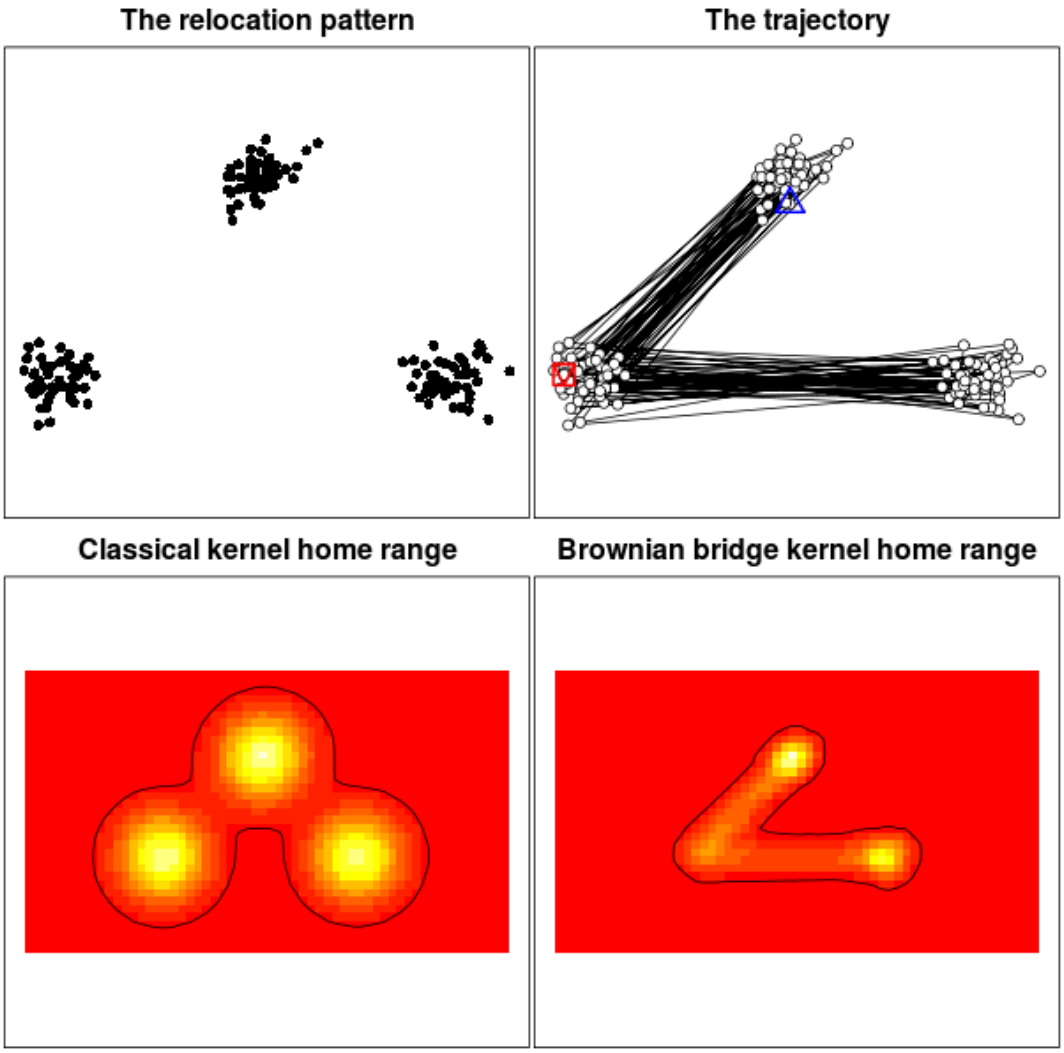
5.1.1 Description

Bullard (1999) proposed an extension of the kernel method taking into account the time dependence between successive relocations. Whereas the classical ker-

nel method places a kernel function above each *relocation*, the Brownian bridge kernel method places a kernel function above each *step* of the trajectory (a step being the straight line connecting two successive relocations). This kernel function is a combination of two bivariate normal pdf (two "bumps") and a Brownian bridge pdf (see Bullard, 1991 and Horne et al. 2007 for additional details):



This method takes into account not only the position of the relocations, but also the path travelled by the animal between successive relocations. The following figure may help to understand the method:



In this example, the relocations are located into three patches. However, the order of use of the patches is not random. Using the Brownian bridge method allows to identify that some areas between the patches are actually not used by the animal.

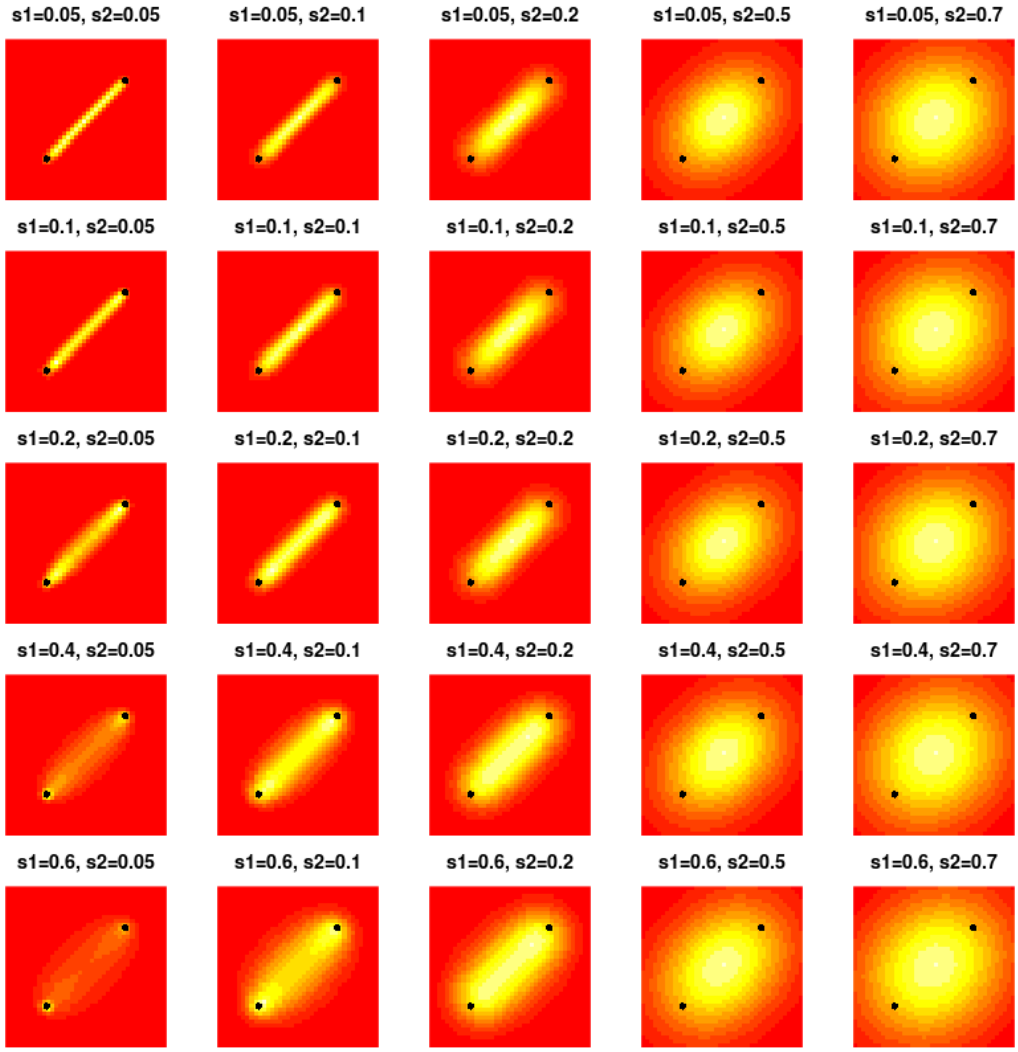
The kernel Brownian bridge method is implemented in the function `kernelbb`. Two smoothing parameters need to be set:

- `sig1`: This parameter controls the width of the "bridge" connecting successive relocations. The larger it is and the larger the bridge is. This

parameter is therefore *related* to the speed of the animal. But be careful: it is not the speed of the animal. Horne et al. (2007) call `sig1`^{2} the "Brownian motion variance parameter". It is even not the variance of the position, but a parameter used to compute the variance (which is itself related to the speed). The variance of the position of the animal between two relocations at time t is $S^2 = \texttt{sig1}^2 \times t \times (T - t)/T$, where T is the duration of the trajectory between the two relocations and t the time at the current position. Therefore, `sig1` is related to the speed of the animal in a very complex way, and its biological meaning is not that clear. The definition of `sig1` can be done subjectively after visual exploration of the result for different values, or using the function `liker` that implements the maximum likelihood approach developed by Horne et al. (2007);

- `sig2`: This parameter controls the width of the "bumps" added over the relocations (see Bullard, 1999; Horne et al. 2007). It is similar to the smoothing parameter of the smoothing parameter h of the classical kernel method, and is therefore related to the imprecision of the relocations.

The following figure shows the kernel function placed over the step built by the two relocations (0,0) and (1,1), for different values of `sig1` and `sig2`:



5.1.2 Example

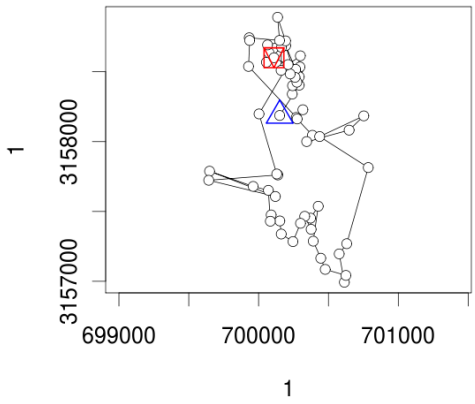
The Brownian bridge kernel method is implemented in the function `kernelbb` of the package. This function takes as argument an object of class `ltraj`. This class has been developed in the brother package `adehabitatLT` to store animals trajectory sampled using GPS, radio-tracking, Argos data, etc. The interested reader should read the help page of the function `as.ltraj` as well as the vignette of the package `adehabitatLT`.

First consider as an example the night monitoring (every 10 minutes) of one wild boar:

```
> data(puechcirc)
> x <- puechcirc[1]
> x

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

Type of the traject: Type II (time recorded)
Regular traject. Time lag between two locs: 600 seconds

Characteristics of the bursts:
    id    burst nb.reloc NAs          date.begin            date.end
1 CH93 CH930824       80  16 1993-08-24 17:00:00 1993-08-25 06:10:00
```

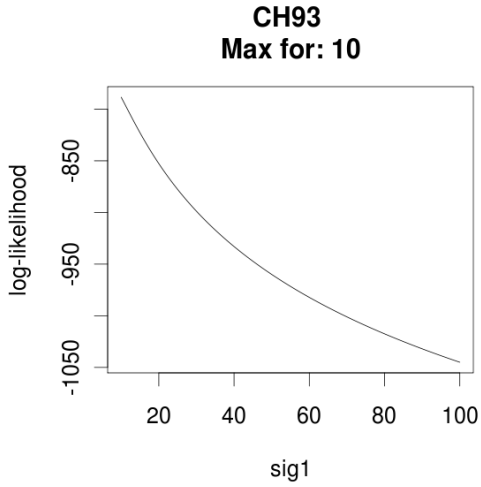
Have a look at the trajectory of the animal:

```
> plot(x)
```



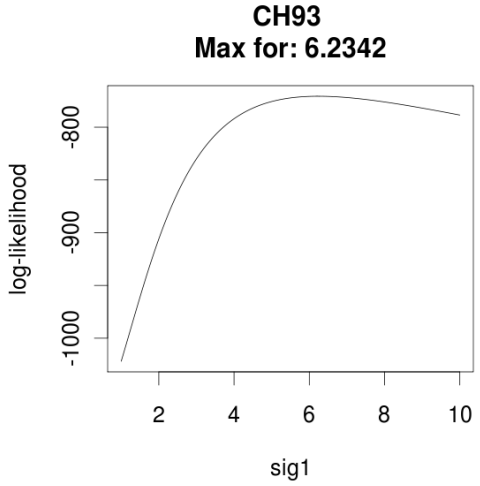
A study has shown that the mean standard deviation of the relocations (relocations as a sample of the actual position of the animal) is equal to 58 meters on these data (Maillard, 1996, p. 63). Therefore, we set `sig2 = 58` metres. Then we use the function `liker` to find the maximum likelihood estimation of the parameter `sig1`. We search the optimum value in the range [10, 100]:

```
> lik <- liker(x, sig2 = 58, rangesig1 = c(10, 100))
```



We expected a too large standard deviation... Let us try again, and search in the interval [1, 10]:

```
> lik2 <- liker(x, sig2 = 58, rangesig1 = c(1, 10))
```



Actually, the optimum value is 6.23:

```
> lik2

****************************************
```

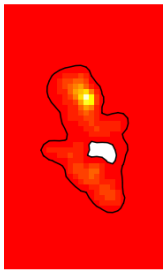
```
Maximization of the log-likelihood for parameter
sig1 of brownian bridge

CH93 : Sig1 = 6.2342 Sig2 = 58
```

We can now estimate the kernel Brownian bridge home range with the parameters `sig1=6.23` and `sig2=58`:

```
> tata <- kernelbb(x, sig1 = 6.23, sig2 = 58, grid = 50)
> tata

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
     cellcentre.offset cellsize cells.dim
Var2            699072 77.42857        30
Var1           3156044 77.42857        50

Variables measured:
            ud
1 3.012177e-19
2 2.672510e-18
3 2.058395e-17
4 1.375991e-16
5 7.981098e-16
6 4.015373e-15
...
```

The result is an object of class `estUD`, and can be managed as such (see section 4.2). For example to see the utilization distribution and the 95% home range:

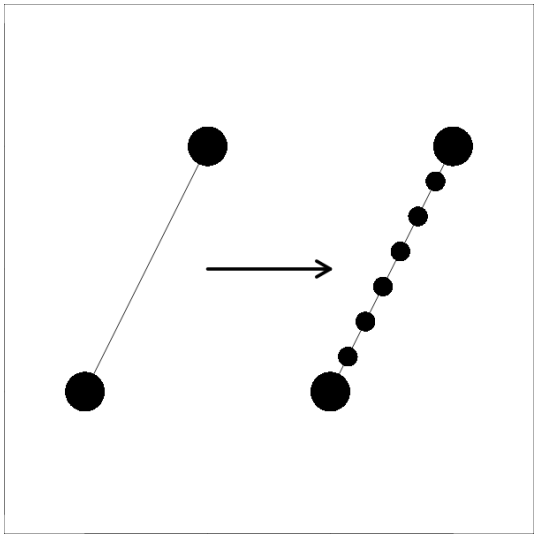
```
> image(tata)
> plot(getverticeshr(tata, 95), add = TRUE, lwd = 2)
```



5.2 The biased random bridge kernel method

5.2.1 Description

Benhamou and Cornelis (2010) developed an approach similar to the Brownian bridge method for the analysis of trajectories. They called it "movement based kernel estimation" (MBKE). The principle of this method consists in dividing each step into several sub-steps, i.e. in adding new points at regular intervals on each step prior to a classical kernel estimation, that is:



Then a kernel estimation is carried out with the known (large points on the figure) and interpolated (small points) relocations, with a variable smooth-

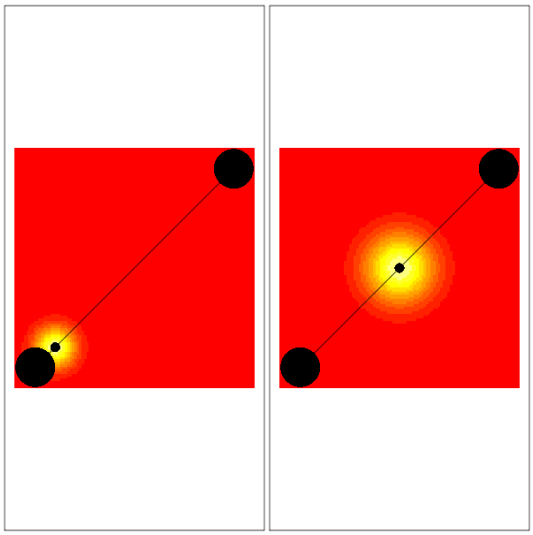
ing parameter (the smoothing parameter depends on the time spent between a known relocation and an interpolated relocation: it is minimum when the relocation is known and larger as the interpolated time between the known relocation and the interpolated location increases).

In a later paper, Benhamou (2011) demonstrated that the movement-based kernel approach takes place in the framework of the biased random walk model. If we describe a trajectory as a succession of "steps", each being characterized by a speed and an angle with the east direction, the process generating the trajectory is a biased random walk when the probability density distribution of the angles is not an uniform distribution (i.e. there is a preferred direction of movement). The main quality of the biased random walk is that it does not suppose a purely diffusive movement (the brownian bridge method supposes only a diffusion), but takes into account an advection component in the trajectory (i.e. a "drift" between successive relocations). This model is therefore more realistic when animal movements are studied.

Now, consider two successive relocations $\mathbf{r}_1 = (x_1, y_1)$ and $\mathbf{r}_2 = (x_2, y_2)$, collected respectively at times t_1 and t_2, and building a step of the trajectory. Our aim is to estimate the function giving the probability density (pdf) that the animal was located at a given place $\mathbf{r} = (x, y)$ at time t_i (with $t_1 < t_i < t_2$), and *given that the animal is moving according to a biased random walk*. The movement generated by a biased random walk *given* the start and end relocation is called a *biased random bridge* (BRB). Benhamou (2011) noted that the BRB can be approximated by a bivariate normal distribution:

$$f(\mathbf{r}, t_i|\mathbf{r}_1, \mathbf{r}_2) = \frac{t_2 - t_1}{4\pi D(t_2 - t_i)} \exp\left[\frac{\mathbf{r}_m \mathbf{D} \mathbf{r}_m}{4p_i(t_2 - t_i)}\right] \quad (1)$$

where the mean location corresponds to $\mathbf{r}_m = (x_1 + p_i(x_2 - x_1), y_1 + p_i(y_2 - y_1))$, and $p_i = (t_i - t_1)/(t_2 - t_1)$. The variance-covariance matrix $\mathbf{D}$ of this distribution usually cannot be estimated in practice (Benhamou, 2011). Consequently, the biased random bridge is approximated by a random walk with a diffusion coefficient D on which a drift $\mathbf{v}$ is appended. And, under this assumption, the required pdf can be approximated by a circular normal distribution fully independent of the drift, i.e. the matrix $\mathbf{D}$ in equation 1 is supposed diagonal, with both diagonal elements corresponding to a *diffusion coefficient* D. The figure below illustrates how the pdf of relocating the animal changes with the times t_i:



We can see that the variance of this distribution is larger when the animal is far from the known relocations.

Benhamou (2011) demonstrated that the MBKE can be used to estimate the UD of the animal under the hypothesis that the animal moves according to a biased random walk, with a drift allowed to change in strength and direction from one step to the next. However, it is supposed that the drift is constant during each step. For this reason, it is required to set an upper time threshold T_{max}. Steps characterized by a longer duration are not taken into account into the estimation of the UD. This upper threshold should be based on biological grounds.

As for the Brownian bridge approach, this conditional pdf based on BRB takes an infinite value at times $t_i = t_1$ and $t_i = t_2$ (because, at these times, the relocation of the animal is known exactly). Benhamou (2011) proposed to circumvent this drawback by considering that the true relocation of the animal at times t_1 and t_2 is not known exactly. He noted: "*a GPS fix should be considered a punctual sample of the possible locations at which the animal may be observed at that time, given its current motivational state and history. Even if the recording noise is low, the relocation variance should therefore be large enough to encompass potential locations occurring in the same habitat patch as the recorded location*". He proposed two ways to include this "relocation uncertainty" component in the pdf: (i) either the relocation variance progressively merges with the movement component, (ii) or the relocation variance has a constant weight.

In both cases, the minimum uncertainty over the relocation of an animal is observed for $t_i = t_1$ or t_2. This minimum standard deviation corresponds to the parameter h_{min}. According to Benhamou and Cornelis (2010), "h_{min} *must be*

at least equal to the standard deviation of the localization errors and also must integrate uncertainty of the habitat map when UDs are computed for habitat preference analyses. Beyond these technical constraints, h_{min} also should incorporate a random component inherent to animal behavior because any recorded location, even if accurately recorded and plotted on a reliable map, is just a punctual sample of possible locations at which the animal may be found at that time, given its current motivational state and history. Consequently, h_{min} should be large enough to encompass potential locations occurring in the same habitat patch as the recorded location".

Therefore, the smoothing parameter used in the MKBE approach can be calculated with:

$$h_i^2 = h_{min}^2 + 4p_i(1-p_i)(h_{max}^2 - h_{min}^2)T_i/T_max$$

where $h_{max}^2 = h_{min}^2 + D \times T_{max}/2$ if we suppose that the relocation variance has a constant weight or $h_{max}^2 = D \times T_{max}/2$ if we suppose that it progressively merges with the movement component.

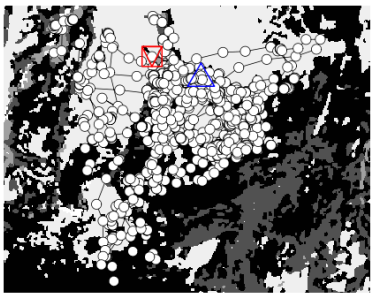
5.2.2 Implementation

The BRB approach is implemented in the function `BRB` of `adehabitatHR`. We first load the dataset `buffalo` used by Benhamou (2011) to illustrate this approach:

```
> data(buffalo)
> tr <- buffalo$traj
> tr

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

Type of the traject: Type II (time recorded)
Irregular traject. Variable time lag between two locs

Characteristics of the bursts:
   id burst nb.reloc NAs          date.begin            date.end
1 buf   buf     1309   0 2001-05-22 19:30:36 2001-06-19 03:30:18


 infolocs provided. The following variables are available:
[1] "act"
```

Note that this object has an infolocs component describing the proportion of time between relocation i and relocation $i+1$ during which the animal was active. We show the trajectory of the animal below:

```
> plot(tr, spixdf = buffalo$habitat)
```



The first step of the analysis consists in the estimation of the diffusion coefficient D. We want to estimate one diffusion parameter for each habitat type. As Benhamou (2011), we set $T_{max} = 180$ minutes and $L_{min} = 50$ metres (the smallest distance below which we consider that the animal is not moving). Because it is more sensible to consider the time of activity in the implementation of the method, we define the argument `activity` of the function as the name of the variable in the infolocs component storing this information:

```
> vv <- BRB.D(tr, Tmax = 180 * 60, Lmin = 50, habitat = buffalo$habitat,
+     activity = "act")
> vv

$buf
          n        D
global 431 7.362258
1       20 5.825270
2        0      NaN
3       19 3.855205
4       61 5.270402

attr(,"class")
[1] "DBRB"
```

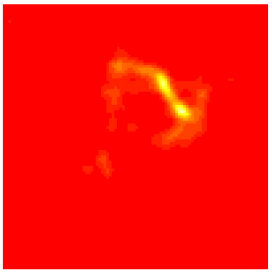
The function `BRB.D` implements the estimation of the diffusion coefficient described in Benhamou (2011). Note that a maximum likelihood approach, similar to the approach used by Horne et al. (2007) is also available in the function `BRB.likD`. The values of the diffusion parameters are given in $m^2.s^{-1}$. Note that the number of steps in habitat 2 was to small to allow the calculation of a diffusion parameter for this habitat type. We can then use the function

BRB to estimate the UD from these relocations. The parameter tau controls the time lag between interpolated relocations. Similarly to Benhamou (2011), we set $\tau = 5$ min, and $h_{min} = 100$ metres:

```
> ud <- BRB(tr, D = vv, Tmax = 180 * 60, tau = 300, Lmin = 50,
+     hmin = 100, habitat = buffalo$habitat, activity = "act",
+     grid = 100, b = 0, same4all = FALSE, extent = 0.01)
> ud

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
x          440275.5 75.69636       100
y         1375107.2 75.69636       100

Variables measured:
  dens
1    0
2    0
3    0
4    0
5    0
6    0
...
```

The missing dispersion parameter for habitat 2 was replaced in the estimation by the global dispersion parameter. As for all kernel methods, this function returns an object of class estUD or estUDm, depending on the number of animals in the object of class ltraj passed as argument. We show the resulting UD:

```
> image(ud)
```



Note that it is also possible to specify a physical boundary that cannot be crossed by the animals in this function.

5.3 The product kernel algorithm

5.3.1 Description

Until now, we have only considered utilization distribution in space. However, Horne et al. (2007) noted that it could also be useful to explore utilization distributions in space and time. That is, to smooth the animals distribution not only in space, but also in time. This approach is done by placing a three dimensional kernel function (X and Y coordinates, and date) over each relocation, and then, by summing these kernel functions. The three-dimensional kernel function K correspond to the combination of three one-dimensional kernel functions K_j:

$$K(\mathbf{x}) = \frac{1}{h_x h_y h_t} \sum_{i=1}^{n} \left(\prod_{j=1}^{3} K_j \left(\frac{x_j - X_{ij}}{h_j} \right) \right)$$

where h_x, h_y, h_t are the smoothing parameters for the X, Y and time dimension, n is the number of relocations, and X_{ij} is the coordinate of the i^{th} relocation on the j^{th} dimension (X, Y or time).

This method is implemented in the function `kernelkc` of the package `adehabitatHR`. The univariate kernel functions associated to the spatial coordinates are the biweight kernel, that is:

$$K(v_i) = \frac{15}{16}(1 - v_i^2)^2$$

when $v_i < 1$ and 0 otherwise. For the time, the user has the choice between two kernel functions. Either the time is considered to be a linear variable, and the biweight kernel is used, or the time is considered to be a circular variable (e.g. January 3^{th} 2001 is considered to be the same date as January 3^{th} 2002 and January 3^{th} 2003, etc.) and the wrapped Cauchy kernel is used. The wrapped Cauchy kernel is:

$$K(v_i) = \frac{h\left(1-(1-h)^2\right)}{2\pi\left(1+(1-h)^2-2(1-h)\cos(v_i h)\right)}$$

where $0 < h < 1$.

5.3.2 Application

We now illustrate the product kernel algorithm on an example. We load the `bear` dataset. This dataset describes the trajectory of a brown bear monitored using GPS (from the package `adehabitatLT`):

```
> data(bear)
> bear

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

Type of the traject: Type II (time recorded)
Regular traject. Time lag between two locs: 1800 seconds

Characteristics of the bursts:
     id burst nb.reloc NAs          date.begin            date.end
1 W0208 W0208     1157 157 2004-04-19 16:30:00 2004-05-13 18:30:00
```

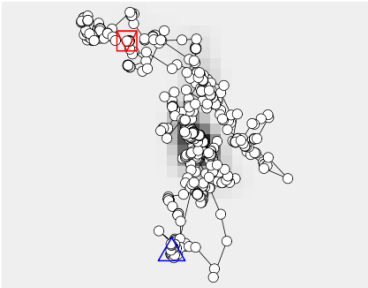
The data are stored as an object of class `ltraj` (see section 5.1.2). Let us compute the UD of this bear for the date 2004-05-01. After a visual exploration, we noted that a smoothing parameter of 1000 meters for X and Y coordinates, and of 72 hours for the time returned interesting results. We can compute the UD with:

```
> uu <- kernelkc(bear, h = c(1000, 1000, 72 * 3600), tcalc = as.POSIXct("2004-05-01"),
+     grid = 50)
> uu

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
     cellcentre.offset cellsize cells.dim
Var2            512114 379.6327        39
Var1           6807486 379.6327        50

Variables measured:
```

```
  ud
1  0
2  0
3  0
4  0
5  0
6  0
...
```

The result is an object of class `estUD`, and can be managed as such (see section 4.2). For example to see both the trajectory and the UD, we type:

```
> plot(bear, spixdf = uu)
```



To compute the 95% home-range, there is a subtlety. Note that the UD integral over the whole plane is not equal to 1:

```
> sum(slot(uu, "data")[, 1]) * (gridparameters(uu)[1, 2]^2)
```

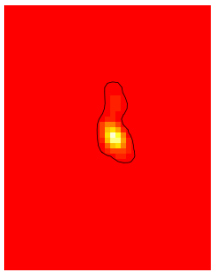
```
[1] 5.062715e-07
```

This is because the UD is defined in space *and time*. That is, the integral over the X, Y and time direction is equal to 1, and not the integral over the X, Y direction at a given time t. Therefore, the 95% home range at a given time t relies on a "restandardization":

$$UD'(x, y|t_i) = \frac{UD(x, y, t_i)}{\int_{x',y'} UD(x', y', t_i)}$$

This is done by setting to `TRUE` the argument `standardize` of the functions `getvolumeUD` or `getverticeshr`. Thus, to calculate the home range of the bear at this date, type:

```
> ver <- getverticeshr(uu, percent = 95, standardize = TRUE)
> image(uu)
> plot(ver, add = TRUE)
```



Note that it is possible to estimate the UD at a succession of t_i close in time, then to draw the images corresponding and save them in a file, and finally to combine these images into a movie. The movie can then be explored to identify patterns. For example, the following code can be used to generate image files (because it is very long, we do not execute it in this vignette:

```
> vv <- seq(min(bear[[1]]$date), max(bear[[1]]$date), length = 50)
> head(vv)
> re <- lapply(1:length(vv), function(i) {
+     uu <- kernelkc(bear, h = c(1000, 1000, 72 * 3600), tcalc = vv[i],
+         grid = 50)
+     jpeg(paste("UD", i, ".jpg", sep = ""))
+     image(uu, col = grey(seq(1, 0, length = 10)))
+     title(main = vv[i])
+     plot(getverticeshr(uu, 95, standardize = TRUE), lwd = 2,
+         border = "red", add = TRUE)
+     dev.off()
+ })
```

The images can then be combined using the program `mencoder` (Linux) or `ImageMagick` (Windows). For Linux users, the following command line can be used:

```
mencoder mf://*.jpg -mf w=320:h=240:fps=15:type=jpeg -ovc lavc -o UD.avi
```

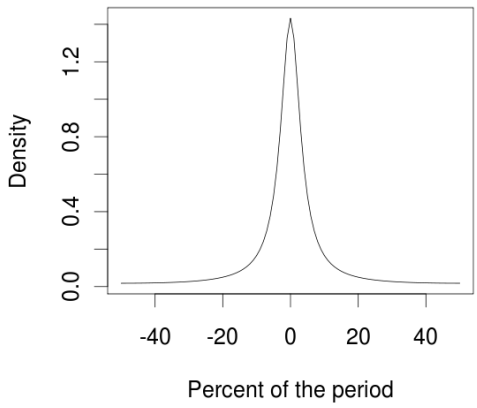
5.3.3 Application with time considered as a circular variable

Now, consider the time as a circular variable. We consider a time cycle of 24 hour. We will estimate the UD at 03h00. We have to define the beginning of the cycle (argument `t0` of the function `kernelkc`). We consider that the time cycle begins at midnight (there is no particular reason, it is just to illustrate the function). We have to set, as the argument `t0`, any object of class POSIXct corresponding to a date at this hour, for example the 12/25/2012 at 00H00:

```
> t0 <- as.POSIXct("2012-12-25 00:00")
```

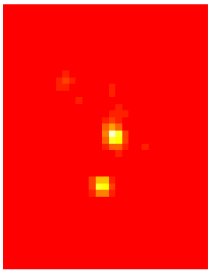
We define $h = 0.2$ for the time dimension (remember that h should be comprised between 0 and 1). We can have a look at the amount of weight given to the neighbouring time points of the point 3h00:

```
> exwc(0.2)
```



The function `exwc` draws a graph of the wrapped Cauchy distribution for the chosen h parameter (for circular time), so that it is possible to make one's mind concerning the weight that can be given to the neighbouring points of a given time point. Now use the function `kernelkc` to estimate the UD:

```
> uu <- kernelkc(bear, h = c(1000, 1000, 0.2), cycle = 24 * 3600,
+     tcalc = 3 * 3600, t0 = t0, circular = TRUE)
> image(uu)
```



Remark: In these examples, we have used the function `kernelkc` to estimate the UD at a given date, with an object of class `ltraj` as argument. However, note that the product kernel algorithm can be used on other kind of data. Thus, the function `kernelkcbase` accepts as argument a data frame with three columns (x, y and time). The example section on the help page of this function provides examples of use.

6 Convex hulls methods

Several methods relying on the calculation of multiple convex hulls have been developed to estimate the home range from relocation data. The package `adehabitatHR` implements three such methods. I describe these three methods in this section.

Remark: note that the function `getverticeshr` is generic, and can be used for all home-range estimation methods of the package `adehabitatHR` (except for the minimum convex polygon).

6.1 The single linkage algorithm

6.1.1 The method

Kenward et al. (2001) proposed a modification of the single-linkage cluster analysis for the estimation of the home range. Whereas this method does not rely on a probabilistic definition of the home range (contrary to the kernel method), it can still be useful to identify the structure of the home range.

The clustering process is the following (also described on the help page of the function `clusthr`): the three locations with the minimum mean of nearest-neighbour joining distances (NNJD) form the first cluster. At each step of the clustering process, two distances are computed: (i) the minimum mean NNJD between three locations (which corresponds to the next potential cluster) and (ii) the minimum of the NNJD between a cluster "c" and the closest location. If (i) is smaller that (ii), another cluster is defined with these three locations. If (ii) is smaller than (i), the cluster "c" gains a new location. If this new location belong to another cluster, the two cluster fuses. The process stop when all relocations are assigned to the same cluster.

At each step of the clustering process, the proportion of all relocations which are assigned to a cluster is computed (so that the home range can be defined to enclose a given proportion of the relocations at hand, i.e. to an uncomplete process). At a given step, the home range is defined as the set of minimum convex polygons enclosing the relocations in the clusters.

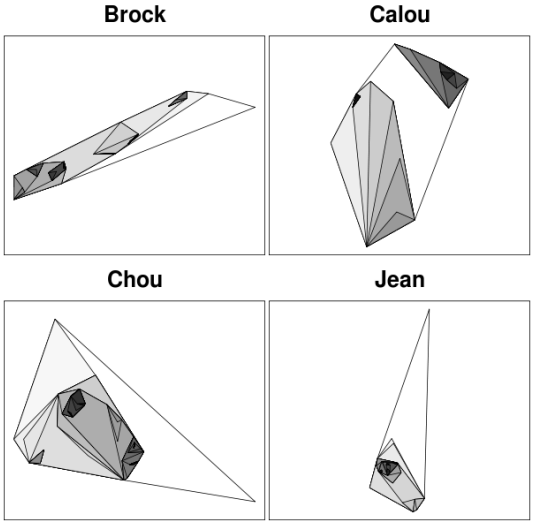
6.1.2 The function `clusthr`

Take as example the `puechabonsp` dataset:

```
> uu <- clusthr(puechabonsp$relocs[, 1])
> class(uu)

[1] "MCHu"
```

The class `MCHu` is returned by all the home range estimation methods based on clustering methods (LoCoH and single linkage). The object `uu` is basically a list of `SpatialPolygonsDataFrame` objects (one per animal). We can have a display of the home ranges by typing:

```
> plot(uu)
```



The information stored in each `SpatialPolygonsDataFrame` object is the following (for example, for the third animal):

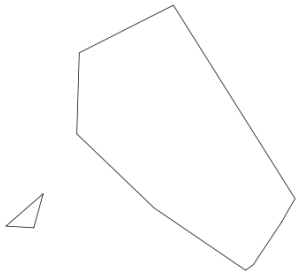
```
> as.data.frame(uu[[3]])
```

```
   percent      area
1      7.5   0.02505
2     15.0   0.06240
3     22.5   0.11930
4     30.0   0.17420
5     37.5   0.18520
6     37.5   1.18020
7     40.0   1.62740
8     40.0   2.52375
9     42.5   3.05210
10    50.0   3.28650
11    50.0   3.71345
12    52.5   4.10105
13    55.0   4.85180
14    55.0   5.79000
15    57.5   6.70530
16    60.0   6.98955
17    67.5   7.66045
18    75.0   8.08420
19    77.5   9.33975
20    77.5  13.27250
21    77.5  30.72235
22    80.0  32.28935
23    82.5  37.80750
```

```
24    85.0  37.80750
25    87.5  43.34810
26    90.0  44.71795
27    90.0  64.17895
28    92.5  68.91885
29    95.0  71.93205
30    97.5 100.12995
31   100.0 162.84550
```

From this data frame, it is possible to extract a home range corresponding to a particular percentage of relocations enclosed in the home range. For example, the smallest home range enclosing 90% of the relocations corresponds to the row 26 of this data frame:

```
> plot(uu[[3]][26, ])
```



6.1.3 Rasterizing an object of class `MCHu`

It is possible to rasterize the estimated home ranges using the function `MCHu.rast`. For example, using the `map` component of the `puechabonsp` dataset:

```
> uur <- MCHu.rast(uu, puechabonsp$map, percent = 90)
> image(uur, 3)
```

6.1.4 Computing the home-range size

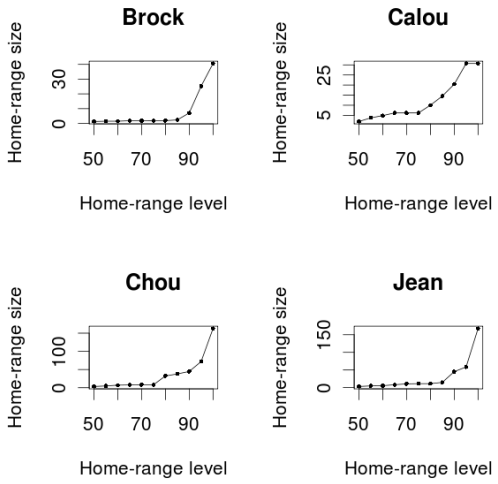
We already noticed that the object of class `MCHu` returned by the function `clusthr` contains the area of each home range. For example, for the first animal:

```
> head(as.data.frame(uu[[1]]))
```

```
   percent    area
1 10.00000 0.01790
2 13.33333 0.04565
3 23.33333 0.06635
4 26.66667 0.09470
5 36.66667 0.27170
6 36.66667 0.79285
```

and as usual, the units of these areas are controlled by the parameters `unin` and `unout` of the function `clusthr`. I also included a function `MCHu2hrsize`, which generates an object of class `hrsize`, and makes easier the exploration of the relationship between the home range size and the percentage of relocations included in the home range:

```
> ii <- MCHu2hrsize(uu, percent = seq(50, 100, by = 5))
```



Here, the choice of 90% seems a good choice for the estimation, as the area-percentage relationship seems to reach a rough asymptote (no steep fall in the decrease below this value).

6.2 The LoCoH methods

The LoCoH (Local Convex Hulls) family of methods has been proposed by Getz et al. (2007) for locating Utilization Distributions from relocation data. Three possible algorithms can be used: Fixed k LoCoH, Fixed r LoCoH, and Adaptive LoCoH. The three algorithms are implemented in `adehabitatHR`. All the algorithms work by constructing a small convex hull for each relocation and its neighbours, and then incrementally merging the hulls together from smallest to largest into isopleths. The 10% isopleth contains 10% of the points and represents a higher utilization than the 100% isopleth that contains all the points (as for the single linkage method). I describe the three methods below (this description is also on the help page of the functions `LoCoH.k`, `LoCoH.r` and `LoCoH.a`):

- **Fixed k LoCoH**: Also known as k-NNCH, Fixed k LoCoH is described in Getz and Willmers (2004). The convex hull for each point is constructed from the (k-1) nearest neighbors to that point. Hulls are merged together from smallest to largest based on the area of the hull. This method is implemented in the function `LoCoH.k`;

- **Fixed r LoCoH**: In this case, hulls are created from all points within r distance of the root point. When merging hulls, the hulls are primarily sorted by the value of k generated for each hull (the number of points contained in the hull), and secondly by the area of the hull. This method

is implemented in the function `LoCoH.r`;

- **Adaptive LoCoH**: Here, hulls are created out of the maximum nearest neighbors such that the sum of the distances from the nearest neighbors is less than or equal to d. Use the same hull sorting as Fixed r LoCoH. This method is implemented in the function `LoCoH.a`;

All of these algorithms can take a significant amount of time.

6.3 Example with the Fixed k LoCoH method

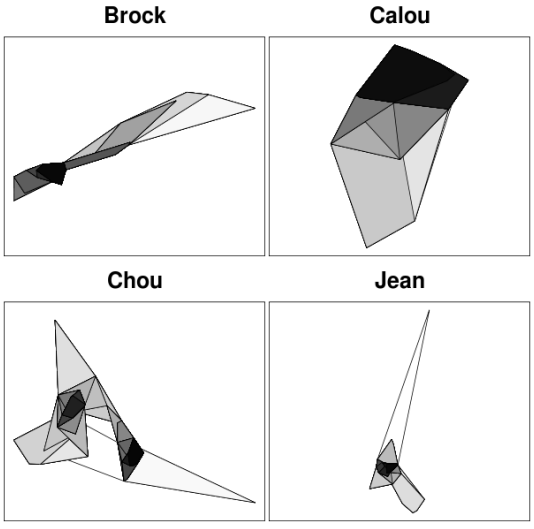
We use the fixed k LoCoH method to estimate the home range of the wild boars monitored at Puechabon, for example with $k = 5$:

```
> res <- LoCoH.k(puechabonsp$relocs[, 1], 10)
> class("res")
```

```
[1] "character"
```

The object returned by the function is of the class `MCHu`, as the objects returned by the function `clusthr` (see previous section). Consequently, all the functions developed for this class of objects can be used (including rasterization with `MCHu.rast`, computation of home-range size with `MCHu2hrsize`, etc.). A visual display of the home ranges can be obtained with:

```
> plot(res)
```



Note that the relationship between the home range size and the number of chosen neighbours can be investigated thanks to the function `LoCoH.k.area`:

```
> LoCoH.k.area(puechabonsp$relocs[, 1], krange = seq(4, 15, length = 5),
+     percent = 100)
```

Just copy and paste the above code (it is not executed in this report). An asymptote seems to be reached at about 10 neighbours, which justifies the choice of 10 neighbours for an home range estimated with 100% of the relocations (a more stable home range size).

6.4 The characteristic hull method

Downs and Horner (2009) developed an interesting method for the home range estimation. This method relies on the calculation of the Delaunay triangulation of the set of relocations. Then, the triangles are ordered according to their area (and not their perimeter). The smallest triangles correspond to the areas the most intensively used by the animals. It is then possible to derive the home range estimated for a given percentage level.

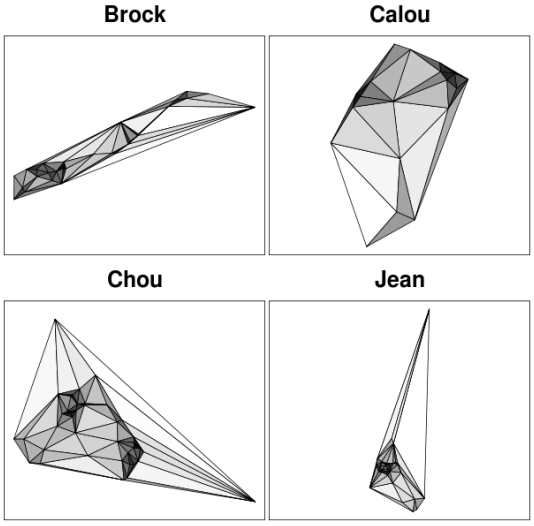
For example, consider the `puechabonsp` dataset.

```
> res <- CharHull(puechabonsp$relocs[, 1])
> class("res")
```

```
[1] "character"
```

The object returned by the function is also of the class `MCHu`, as the objects returned by the functions `clusthr` and `LoCoH.*` (see previous sections). Consequently, all the functions developed for this class of objects can be used (including rasterization with `MCHu.rast`, computation of home-range size with `MCHu2hrsize`, extraction of the home-range contour with `getverticeshr` etc.). A visual display of the home ranges can be obtained with:

```
> plot(res)
```



7 Conclusion

I included in the package `adehabitatHR` several functions allowing the estimation of the home range and utilization distribution of animals using relocation data. These concepts are extremely useful for the study of animal space use, but they consider the use of space from a purely spatial point of view. In other words, time is not taken into account...

The package `adehabitatLT` represents an attempt to shift our point of view on relocation data toward a more dynamic point of view. Indeed, with the increasing use of GPS collars, the time dimension of the monitoring becomes more and more important in studies of animals space use. This package contains several functions developed for the analysis of trajectories. The class `ltraj`, which is at the very core of this package, has been developped to make possible the analysis of animals trajectories.

Another important aspect in the study of animals space use is the study of their relationships with the environment (i.e., habitat selection). The package `adehabitatHS` proposes several methods allowing to explore this aspect of animals space use. Several functions are implemented, relying on the study of the distribution of the time spent by the animals in the multidimensional space defined by the environmental variables. In addition to common methods in the analysis of habitat selection (compositional analysis of habitat use, selection ratios, etc.), the package `adehabitatHS` proposes several exploratory tools useful to identify patterns in habitat selection. The study of habitat selection by animals is possible through the interaction with the other brother packages (of the suite `adehabitat`), but also thanks to te tools provided by the package `sp`,

which allow to handle raster and vector maps of environmental variables. Note that the package `adehabitatMA` provides additionnal tools which can be useful in the particular field of habitat selection study.

References

Bullard, F. 1999. Estimating the home range of an animal: a Brownian bridge approach. Johns Hopkins University. Master thesis.

Burt, W.H. 1943. Territoriality and home range concepts as applied to mammals. Journal of Mammalogy, 24, 346-352.

Benhamou, S. and Cornelis, D. 2010. Incorporating movement behavior and barriers to improve kernel home range space use estimates. Journal of Wildlife Management, 74, 1353–1360.

Benhamou, S. 2011. Dynamic approach to space and habitat use based on biased random bridges. PLOS ONE, 6, 1–8.

Calenge, C. 2005. Des outils statistiques pour l'analyse des semis de points dans l'espace ecologique. Universite Claude Bernard Lyon 1.

Calenge, C. 2006. The package adehabitat for the R software: a tool for the analysis of space and habitat use by animals. Ecological modelling, 197, 516–519.

Calenge, C., Dray, S. and Royer, M. in press. The concept of animals trajectories from a data analysis perspective. Ecological Informatics.

Downs, J.A. and Horner, M.W. 2009. A Characteristic-Hull Based Method for Home Range Estimation. Transactions in GIS, 13: 527–537.

Getz, W., Fortmann-Roe, S., Cross, P., Lyons, A., Ryan, S. and Wilmers, C. 2007. LoCoH: Nonparameteric Kernel Methods for Constructing Home Ranges and Utilization Distributions. PloS one, 2, e207.

Horne, J.S., Garton, E.O., Krone, S.M. and Lewis, J.S. 2007. Analyzing animal movements using Brownian bridges. Ecology, 88, 2354–2353.

Keating, K. and Cherry, S. 2009. Modeling utilization distributions in space and time Ecology, in press.

Kenward, R., Clarke, R., Hodder, K. and Walls, S. 2001. Density and linkage estimators of home range: nearest neighbor clustering defines multinuclear cores. Ecology, 82, 1905–1920.

Maillard, D. 1996. Occupation et utilisation de la garrigue et du vignoble mediterraneens par le sanglier (Sus scrofa L.). Universite de droit, d'economie et des sciences d'Aix-Marseille III, PhD thesis.

Mohr, C. 1947. Table of equivalent populations of North American small mammals. American Midland Naturalist, 37, 223–249.

Pebesma, E. and Bivand, R.S. 2005. Classes and Methods for Spatial data in R. R News, 5, 9–13.

Silverman, B. 1986. Density estimation for statistics and data analysis. Chapman and Hall, London.

van Winkle, W. (1975) Comparison of several probabilistic home-range models. Journal of Wildlife Management, 39, 118–123.

Wand, M. and Jones, M. 1995. Kernel smoothing. Chapman and Hall/CRC.

Worton, B. 1989. Kernel methods for estimating the utilization distribution in home-range studies. Ecology, 70, 164–168.