# Classes of map objects available in adehabitat

Clément Calenge

September 14, 2004

# Contents

# Chapter 1

# Package objective

This package was primarily developed for the analysis of radio-tracking data. Common analyses are available for home-range estimation or study of habitat selection. However, **adehabitat** has been useful in other fields, such as biogeography (Spichiger *et al.*, 2004, The geographical zonation in the Neotropics of tree species characteristic of the Paraguay-Paraná Basin, *Journal of Biogeography*, 31, 1489-1501). Basically, this package was developed to provide an interface between Geographic Information Systems (G.I.S.) and the R package **ade4**, useful for the multivariate analysis of ecological data (Chessel *et al.*, 2004, The ade4 package - I : One table methods, *R News*, 4/1, 5-10).

Roughly speaking, the functions available in this package are general enough to provide useful tools to ecologists who want to match one or several sets of points (animals, plant species, etc.) with a set of raster maps. Some basic GIS operations are available (labeling of connected features, spatial join of a set of points with a set of maps, etc.). In this document, we present the different classes of objects used in **adehabitat** to store maps.

First of all, you need to load the package:

```
> library(adehabitat)
```

```
This package requires ade4 to be installed

Loading required package: ade4
```

# Chapter 2

# The storage mode of raster maps in adehabitat

## 2.1 Single maps : the class `"asc"`

Two kinds of maps must be distinguished in geographical analysis (see Figure 2.1):

- *the vector maps* divide the area into a set of polygons of various sizes, with different attributes;
- *the raster maps* divide the area of interest into a set of square cells, the pixels, all of the same size, each pixel having a value for the mapped variable.

Figure 2.1: (left) a vector map; (right) a raster map.

Only raster maps can be imported in R with **adehabitat**. When vector maps are to be imported into R , you may consider the wonderful package **maptools**. Wildlife scientists often prefer to work with raster maps, because several environmental variables are often under study. With raster maps, all the pixels have the same shape and the same area. This equal importance of the sampling units makes easier the analyses of such multivariate data.

Raster maps may be either of type `numeric` (a numerical variable is mapped, *e.g.* the elevation) or of type `factor` (a categorical variable is mapped, *e.g.* the vegetation type). Both types of maps are handled with **adehabitat**.

These maps are stored in R as matrices of class `"asc"`. Each cell of the matrix corresponds to a pixel of the map. These maps have attributes related to the position and the resolution of the map (see Figure 2.2):

- `xll` : the x coordinate of the centre of the lower left pixel of the map;
- `yll` : the y coordinate of the centre of the lower left pixel of the map;
- `cellsize` : the size of the side of a pixel on the studied map.

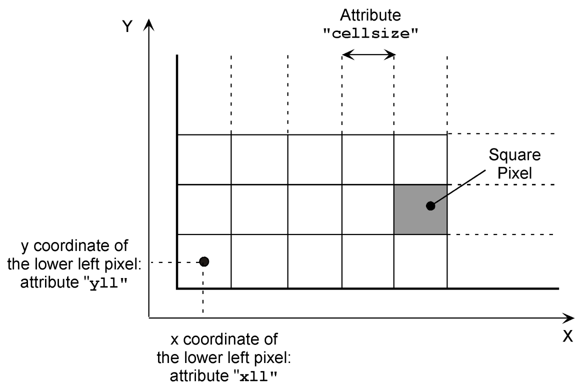


Figure 2.2: Detail of a raster map of class `"asc"` illustrating the attributes related to its position and its resolution.

Two other attributes are related to the mapped variables

- `type` : either `"numeric"` or `"factor"`;
- `levels` : if the type of the map is `"factor"`, the levels of the factor are considered.

Objects of class `"asc"` can be created using the function `as.asc`. We generate here a map with random numbers (Figure 2.3):

```
> mat <- matrix(rnorm(10000), 100, 100)
> asc <- as.asc(mat)
> image(asc)
> box()
```

However, the most common way to create objects of class `"asc"` is to import Arcview ASCII raster files (files with the extension ".asc"), with the function `import.asc()`. For example, **adehabitat** contains a raster map named "elevation.asc" in the directory "ascfiles". The path for this file can be obtained with the command:

```
> (path.to.file <- paste(system.file(package = "adehabitat"), "ascfiles/elevation.asc",
+     sep = "/"))

[1] "C:/rw1090/library/adehabitat/ascfiles/elevation.asc"
```

Of course, this path may vary from one machine to another. The map is then imported into R (Figure 2.4):

```
> el <- import.asc(path.to.file)
> image(el, main = "Elevation")
```

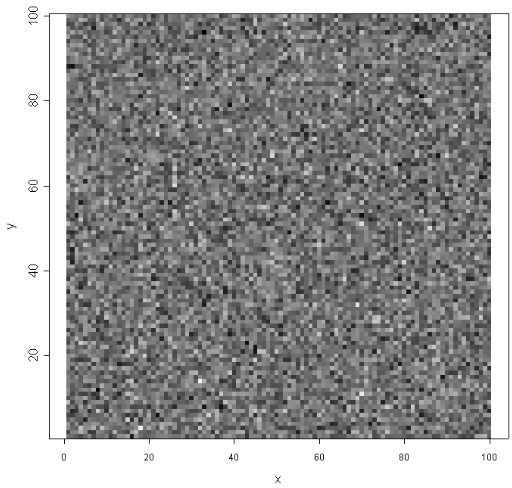


Figure 2.3: An object of class "asc" generated with as.asc() on a matrix of random numbers.

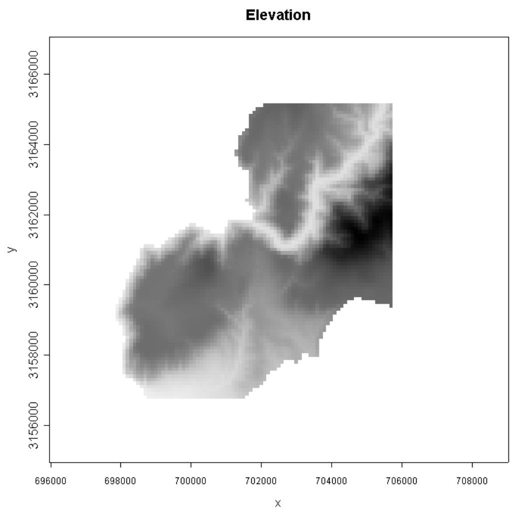


Figure 2.4: A raster map of type "numeric" imported into R with the function import.asc()(Map of the elevation at Puéchabon, South of France).

Conversely, the exportation of a matrix of class "asc" can be done using the function `export.asc()`:

```
> export.asc(el, "toto.asc")
```

For factor maps, it is also necessary to define the levels of the variable. For example, consider the map "aspect.asc", located in the directory "ascfiles" of the package. This map describes the aspect of the study area, and has four levels (`North`, `East`, `West`, `South`). However, these levels are not stored in the ASCII raster file:

```
> (path.to.map <- paste(system.file(package = "adehabitat"), "ascfiles/aspect.asc",
+     sep = "/"))
[1] "C:/rw1090/library/adehabitat/ascfiles/aspect.asc"
> asp <- import.asc(path.to.map, type = "factor")
> levels(asp)
[1] "1" "2" "3" "4"
```

It is therefore necessary to specify the labels associated with each level of the variable when importing a factor map:

```
> asp <- import.asc(path.to.map, type = "factor", lev = c("North",
+     "East", "West", "South"))
> levels(asp)
[1] "North" "East"  "West"  "South"
```

Another way of specifying the levels of the factor map is to export the theme table from Arcview. To proceed, select the theme in Arcview, then choose the menu Theme -> Table, and finally export the table as a delimited text file from the menu File -> Export. The file "aspect.txt" is located in the directory "ascfile" of the package:

```
> (path.to.table <- paste(system.file(package = "adehabitat"),
+     "ascfiles/aspect.txt", sep = "/"))
[1] "C:/rw1090/library/adehabitat/ascfiles/aspect.txt"
> file.show(path.to.table)
```

This text file has three columns, which are separated by commas.

```
"Value","Count","NewField1"
 1,537,North
 2,1504,East
 3,1262,South
 4,1076,West
```

The first column gives the levels of the map, and the third column gives the corresponding labels (The second column is no interest for our purpose). This file may then be specified when importing the map:

```
> asp <- import.asc(path.to.map, type = "factor", lev = path.to.table,
+     levnb = 1, labnb = 3)
> levels(asp)
[1] "North" "East"  "South" "West"
> co <- colasc(asp, North = "blue", East = "yellow", West = "orange",
+     South = "red")
```

`colasc()` links a vector of colors with each level of `asp`. Then, the map is displayed (Figure 2.5):

```
> image(asp, clfac = co)
> legend(696662, 3166028, legend = levels(asp), fill = co)
```

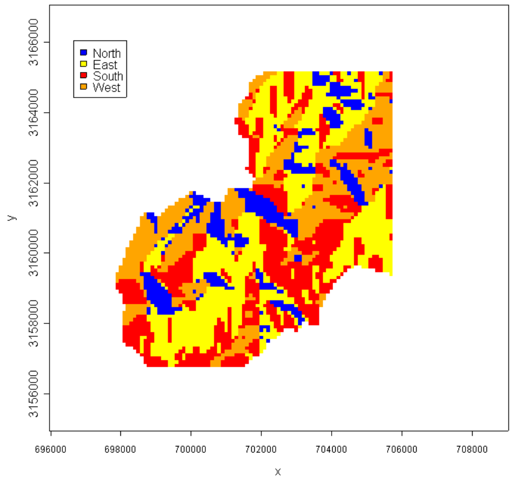


Figure 2.5: A raster map of type "factor" imported into R with the function `import.asc()`(map of the aspect at Puéchabon, South of France).

Finally, there are other ways to create objects of class "asc": the functions `asc2im()` and `im2asc()` provide an interface with the class "im" of the package **spatstat**.

*Note:* The central place of this class of objects in **adehabitat** arises from the large availability of **Arcview GIS** among ecologists (`http://www.esri.com/`). The class "asc" originates from the ASCII raster maps created by Arcview GIS (under the File menu: Export Data source). Note, however, that the format coming from other GIS softwares may be converted to ASCII raster maps using the free software **Landserf**, downloadable at the following URL: `http://www.soi.city.ac.uk/~jwo/landserf/`.

## 2.2 Multiple raster maps in a unique object: the class "kasc"

Ecologists are usually interested in several environmental variables. Therefore, they have to deal with a collection of $K$ raster maps of the same area. These maps often have the same attributes (same number of rows and of columns, same resolution, same coordinates for the lower left pixel). Such a collection of maps may be stored in objects of class "kasc".

This class inherits from the class "data.frame". Each column of this data frame corresponds to the map of one variable, with one row per pixel of the map (Figure 2.6).

Thus, all sub-setting rules that are applied to the class "data.frame" can also be applied to the class "kasc". Objects of class "kasc" have the same attributes as objects of the class "asc" (described in the previous section), but they also have two additional attributes:

- `nrow`: the number of rows of the raster map. *Warning!* the number of rows of the map is equal to the number of columns of the matrix of class "asc" coding this map;
- `ncol`: the number of columns of the raster map. *Warning!* the number of columns of the map is equal to the number of rows of the matrix of class "asc" coding this map.

An example of object of class "kasc" is provided in the data set `puechabon`. The component named `kasc` of this list is a map of class "kasc":

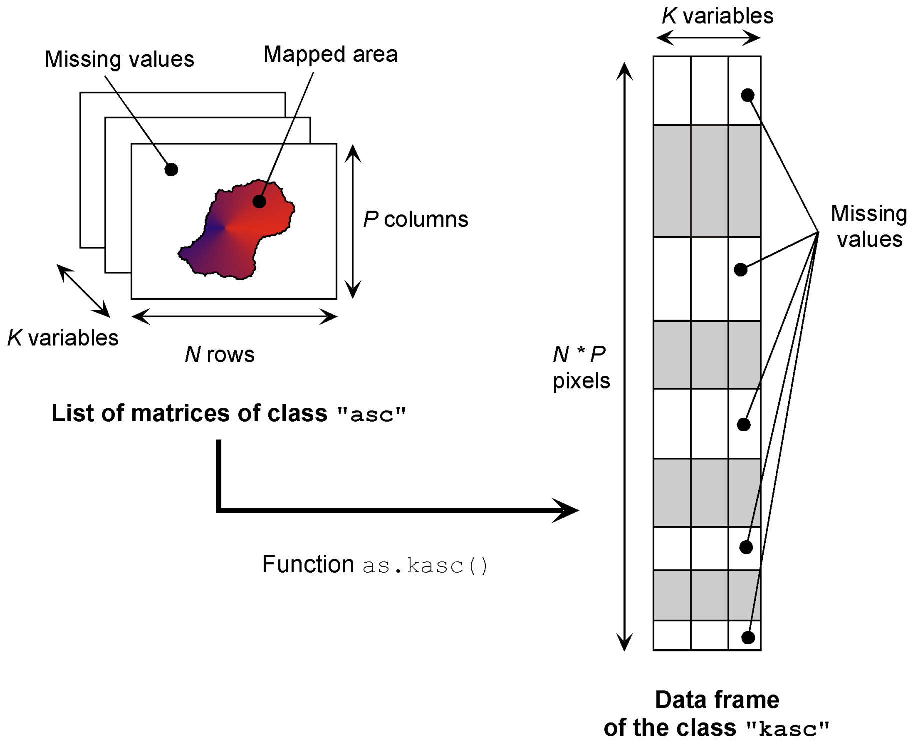


Figure 2.6: A data frame of the class `"kasc"` is created from several raster maps of class `"asc"`. Note that when the mapped area does not fit the whole map (which is the case here), this leads to a considerable amount of missing values in the object of class `kasc`.

```
> data(puechabon)
> kasc <- puechabon$kasc
> image(kasc)
```

The image returned by these instructions is displayed Figure 2.7.

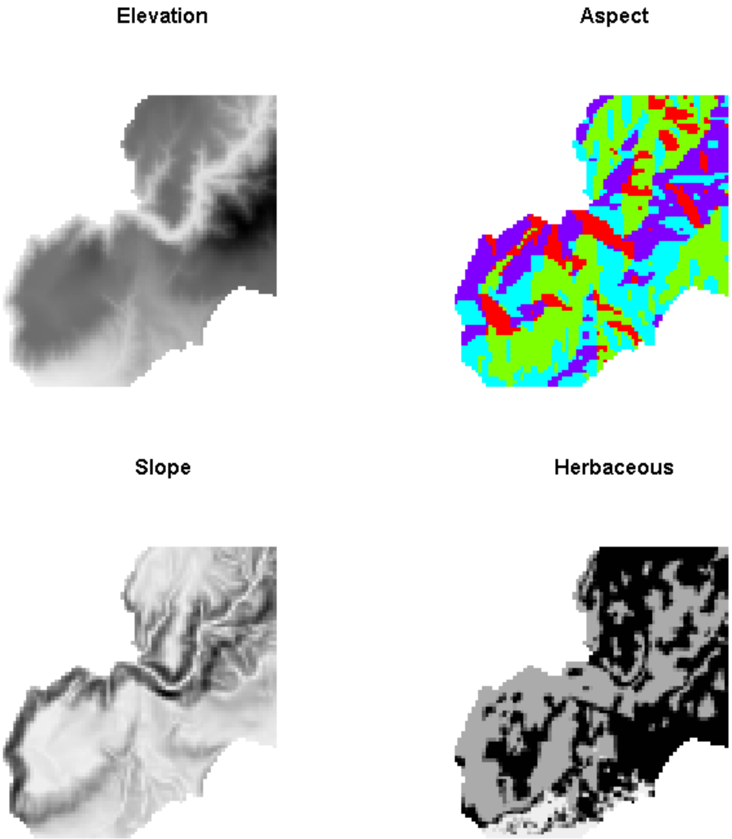


Figure 2.7: Example of object of class "kasc" (maps of Elevation, Slope, Aspect and Herbaceous cover at Puéchabon, South of France).

The most common way to create objects of class "kasc" is to use the function `as.kasc()`. In this example, don't forget that the maps `el` and `asp` have already been imported in the previous section. Note that each element of the list passed to the function `as.kasc` should be named:

```
> (obj <- as.kasc(list(Elevation = el, Aspect = asp)))

Raster map of class "kasc":
Cell size:  100
Number of rows:  121
Number of columns:  111

Variables measured:
1. Elevation: numeric
2. Aspect: factor
```

## 2.3 Basic operations with maps of classes "asc" and "kasc"

### 2.3.1 Introduction

We describe here some basic operations that can be done on maps with **adehabitat**. We mainly use the data set `puechabon` available in the package **adehabitat**. This data set stores information on the use of space by four wild boars (*Sus scrofa*) monitored by radio-tracking in the South of France (see the help page of this data set for further information and source). We use here the components: (i) `puechabon$kasc`, an object of class "kasc" that describes several variables on the study area; and (ii) `puechabon$locs`, a data frame containing the coordinates of the relocations of the wild boars resting sites in summer, as well as information on wild boars in factors `Name`, `Sex`, `Age`. We first load the data set:

```
> data(puechabon)
> puechabon$kasc
```

```
Raster map of class "kasc":
Cell size:  100
Number of rows:  121
Number of columns:  111

Variables measured:
1. Elevation: numeric
2. Aspect: factor
3. Slope: numeric
4. Herbaceous: numeric
```

```
> puechabon$locs[1:4, ]
```

```
   Name Age Sex      X       Y   Date
1 Brock   2   1 699889 3161559 930701
2 Brock   2   1 700046 3161541 930703
3 Brock   2   1 698840 3161033 930706
4 Brock   2   1 699809 3161496 930707
```

A general view of the data is displayed using (Figure 2.8):

```
> el <- getkasc(puechabon$kasc, "Elevation")
> opar <- par(mfrow = c(2, 2), mar = c(0, 0, 4, 0))
> for (i in levels(puechabon$locs$Name)) {
+     image(el, main = paste("Wild boar named", i), axes = FALSE)
+     points(puechabon$locs[puechabon$locs$Name == i, c("X", "Y")],
+         pch = 16)
+ }
> par(opar)
```

In this section, we also use another data set, `chamois`. This data set stores locations of chamois (*Rupicapra rupicapra*) during the hunting seasons of two years (1992 and 1997), collected in the Chartreuse mountain (French Alps) by census operations (see the help page of this data set for further information and source). This list has two elements: (i) `chamois$map`, an object of class "kasc" that describes three habitat variables on the study area; and (ii) `chamois$locs`, a data frame containing the coordinates of the locations of the chamois. We also load this data set:

```
> data(chamois)
> chamois$map
```

```
Raster map of class "kasc":
Cell size:  50
Number of rows:  353
Number of columns:  353

Variables measured:
1. Vegetation: factor
2. Disteco: numeric
3. Slope: numeric
```

```
> chamois$locs[1:4, ]
```

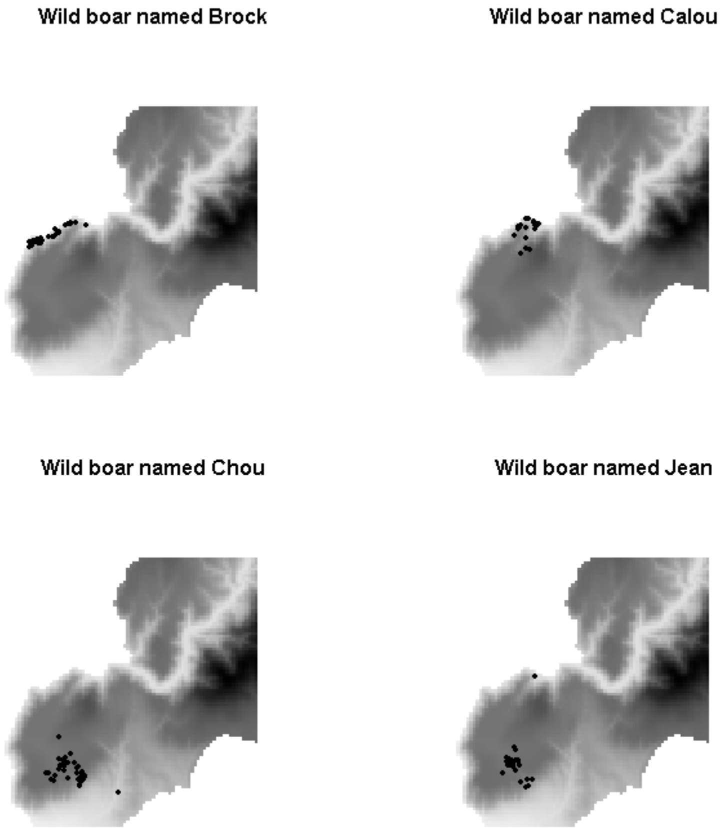


Figure 2.8: Distribution of the relocations of four wild boars monitored using radio-tracking at Puéchabon, South of France).

```
            X       Y
72  862793.5 2038625
180 867550.2 2047080
17  872687.6 2046367
277 866479.5 2043730
```

A general view of the data is displayed using (Figures 2.9):

```
> sl <- getkasc(chamois$map, "Slope")
> image(sl, main = "Distribution of chamois occurrences in the Chartreuse mountain")
> points(chamois$locs, pch = 16)
```

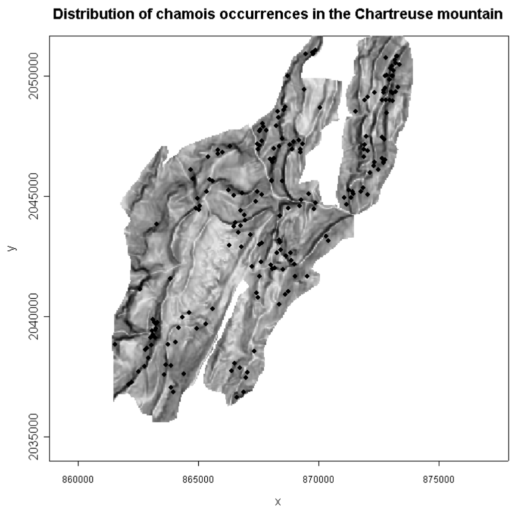


Figure 2.9: Distribution of Chamois occurrences collected during two years (1992 and 1997) by census operation in the Chartreuse mountain (French Alps).

### 2.3.2 Getting a component of an object of class `"kasc"`

Several functions in **adehabitat** return objects of class `"kasc"`. For example, the user can convert one of the variables of the object of class `"kasc"` into a matrix of class `"asc"`, either for further analyses or for exportation toward a Geographic Information System. This is done with the help of the function `getkasc()`. For example, using the class object `"kasc"` of the data set `puechabon`:

```
> kasc <- puechabon$kasc
> (el <- getkasc(kasc, "Elevation"))

Raster map of class "asc":
Cell size:  100
Number of rows:  121
Number of columns:  111
Type:  numeric
```

`el` is the map of the elevation on the area. It is displayed in Figure 2.4.

### 2.3.3 Mathematical morphology

Some basic operations of mathematical morphology are available in **adehabitat**. Erosion and dilatation can be performed with the function `morphology()`. It may be useful to define buffers around mapped features. For example, using the map `el` created in the previous section:

```
> er8 <- morphology(el, operation = "erode", nt = 8)
> di8 <- morphology(el, operation = "dilate", nt = 8)
```

The maps `er8` and `di8` are objects of class `"asc"`, with pixels taking the value `1` if they belong to the mapped features, and missing values otherwise. The argument `nt` of the function indicates the number of times that the operation is to be processed. In the example above, the size of the pixel is 100 m, and `nt = 8`. This means that buffer areas of 800 m have been defined inside and outside the boundaries of the study area (See Figure 2.10):

```
> image(di8, col = "black")
> image(el, col = "gray", add = TRUE)
> image(er8, col = "white", add = TRUE)
> arrows(703530, 3165169, 703530, (3165169 - 800), code = 3, lwd = 2,
+     length = 0.1)
> text(704156, 3164775, "800 m")
> arrows(704295, 3159355, 706588, 3157294, col = "red", lwd = 2,
+     code = 1)
> text(706240, 3156738, "Boundary of the study area")
> legend(696000, 3165841, c("Buffer area inside the boundary",
+     "Buffer area outside the boundary"), fill = c("gray", "black"),
+     cex = 0.7)
```

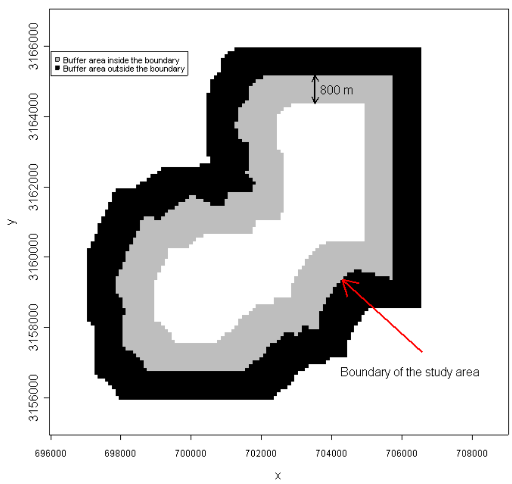


Figure 2.10: Example of morphological operations performed on the maps of the data set `puechabon` with the function `morphology()`. Buffer areas of 800 m have been defined inside and outside the study area.

### 2.3.4 Computing buffers around points

This operation, somewhat related to the previous one, can be processed using the function `buffer()`. For example, consider the component `locs` of the data set `puechabon`. This data frame contains the coordinates of the relocations of wild boars monitored using radio-tracking. Also consider the map `el`, created in the section 2.3.2:

```
> data(puechabon)
> puechabon$locs[1:4, ]

   Name Age Sex      X       Y   Date
1 Brock   2   1 699889 3161559 930701
2 Brock   2   1 700046 3161541 930703
3 Brock   2   1 698840 3161033 930706
4 Brock   2   1 699809 3161496 930707
```

A map of the relocations on the study area (Figure 2.11) is obtained with:

```
> image(el)
> points(puechabon$locs[, c("X", "Y")], pch = 16)
```

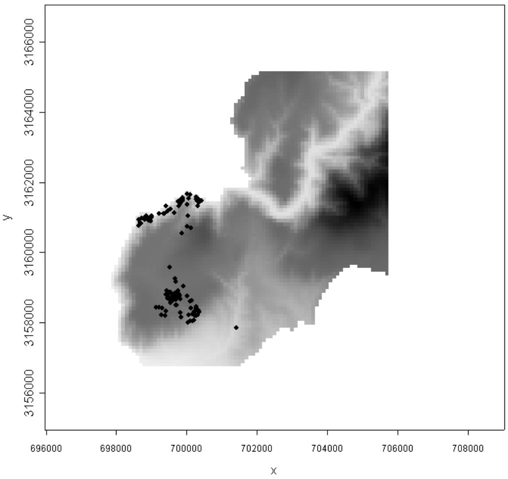


Figure 2.11: Relocations of 4 wild boars monitored using radio-tracking at Puéchabon (South of France).

Here, the four wild boars are not distinguished. A buffer of 500 m is computed around each point (*e.g.* to take into account the lack of precision in the relocations, or to define areas that are available to the animals; Figure 2.12) :

```
> bu <- buffer(puechabon$locs[, c("X", "Y")], el, 500)
> image(bu)
> points(puechabon$locs[, c("X", "Y")], pch = 16)
```

Note, that the map `el` could have been replaced by an object of class `"kasc"` to compute the same buffer (you may try to perform this operation by replacing `el` by `puechabon$kasc`). The map `bu` is an object of class `"asc"`, with pixels taking the value `1` if they belong to the mapped features, and missing values otherwise. It is easy after that to multiply the maps `el` and `bu` to measure the elevation in the neighbouring of the relocations (Figure 2.13). Thus:

```
> bubis <- bu * el
> mean(as.vector(bubis), na.rm = TRUE)

[1] 232.4520

> sd(as.vector(bubis), na.rm = TRUE)

[1] 69.06259

> image(bubis)
```

### 2.3.5 Finding habitat features at every points

This operation can be processed using the function `join.asc()`. For example, using the map `el` and the component `locs` of the data set `puechabon`, it is possible to determine the elevation at each wild boar relocation in `puechabon$locs`:

```
> vec <- join.asc(puechabon$locs[, c("X", "Y")], el)
> length(vec)

[1] 119
```

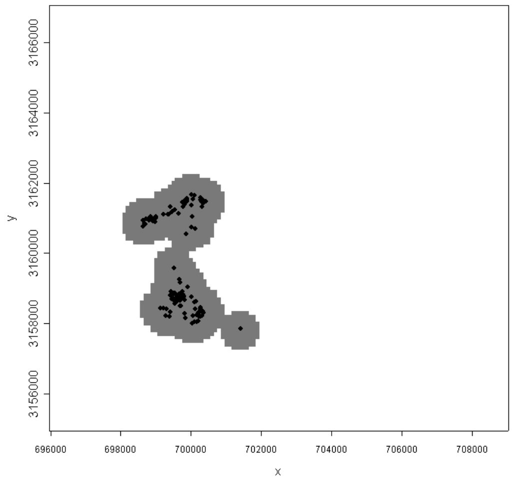


Figure 2.12: A buffer of 500 m around all relocations of the four boars monitored using radio-tracking. The buffer is computed using the function `buffer()`.

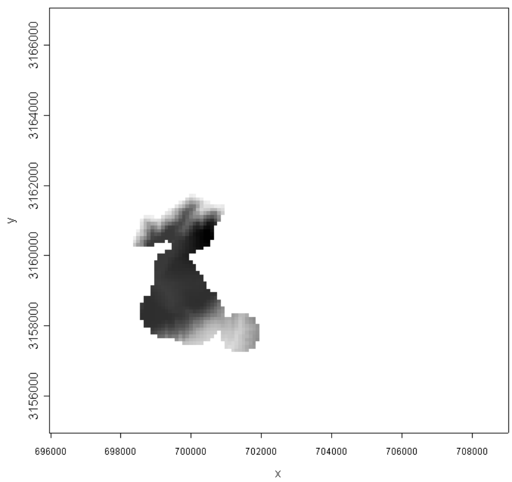


Figure 2.13: The elevation within 500 m from all relocations of the four wild boars monitored at Puéchabon (South of France) using radio-tracking.

```
> nrow(puechabon$locs)
[1] 119
> vec[1:10]
 [1] 108 172 121 113 101 127 102 108 104 151
```

vec contains the elevation of each relocation in puechabon$locs. It can further be used in statistical analyses, *e.g.* to study habitat selection by the boars. The function join.kasc() matchs one set of points with several maps (*i.e.* an object of class kasc):

```
> df <- join.kasc(puechabon$locs[, c("X", "Y")], puechabon$kasc)
> nrow(df)
[1] 119
> nrow(puechabon$locs)
[1] 119
> df[1:10, ]
   Elevation    Aspect    Slope Herbaceous
1        108 NorthWest 19.39472        0.2
2        172 NorthWest 23.67911        0.2
3        121 NorthEast 20.16249        0.2
4        113 NorthWest 22.39815        0.2
5        101 NorthWest 14.97099        0.2
6        127 NorthWest 16.82134        0.2
7        102 NorthEast 18.14621        0.2
8        108 NorthWest 18.08812        0.2
9        104 NorthEast 21.24609        0.2
10       151 NorthWest 29.24554        0.2
```

Each row of df gives the habitat composition for each relocation in locs.

### 2.3.6 Counting the number of points in each pixel of a map

This operation can be processed using count.points(). Consider again the map el and the wild boars relocations puechabon$locs:

```
> (cp <- count.points(puechabon$locs[, c("X", "Y")], el))
Raster map of class "asc":
Cell size:  100
Number of rows:  121
Number of columns:  111
Type:  numeric
> image(cp)
> box()
```

cp contains the number of relocations in each pixel of the map (Figure 2.14). It can be coerced into vector for a further analysis of the counts. When several sets of points are available (for example the locations of several animals monitored by radio-tracking, or several sets of species occurrences on the same area), the function count.points.id() counts, for each set, the number of points in each pixel of the map (Figure 2.15). For example, for the relocations of the wild boars puechabon$locs, the animals identity is indicated in the column Name:

```
> (cp <- count.points.id(puechabon$locs[, c("X", "Y")], puechabon$locs$Name,
+     el))
Raster map of class "kasc":
Cell size:  100
Number of rows:  121
Number of columns:  111

Variables measured:
1. Brock: numeric
2. Calou: numeric
3. Chou: numeric
4. Jean: numeric
> image(cp)
```

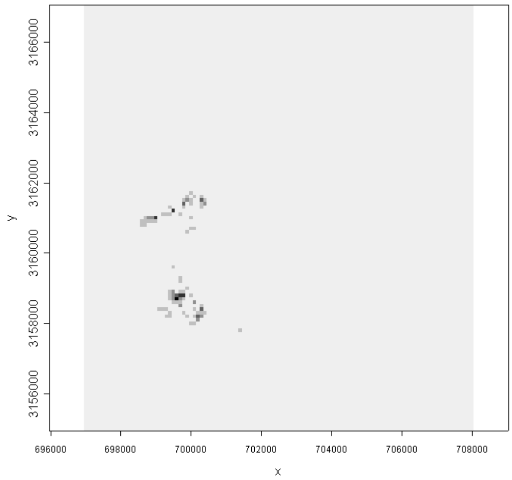


Figure 2.14: The number of wild boars relocations inside each pixel of the raster map `el` (numbered with the function `count.points()`).


Figure 2.15: The number of relocations of each wild boar inside each pixel of the raster map `el` (computed with the function `count.points.id()`).

### 2.3.7 Counting the number of points in each cell of a virtual grid

This operation is very similar to the previous one, except that in this case, the raster map is not available to the analyst. Consider only the wild boars relocations `puechabon$locs`. A virtual grid can be superposed to the study area using the function `ascgen()` (for "*asc generator*"). For example, the chosen cell size of the grid is 500 m (Figure 2.16):

```
> hihi <- ascgen(xy = puechabon$locs[, c("X", "Y")], cellsize = 500)
> image(hihi)
> box()
```

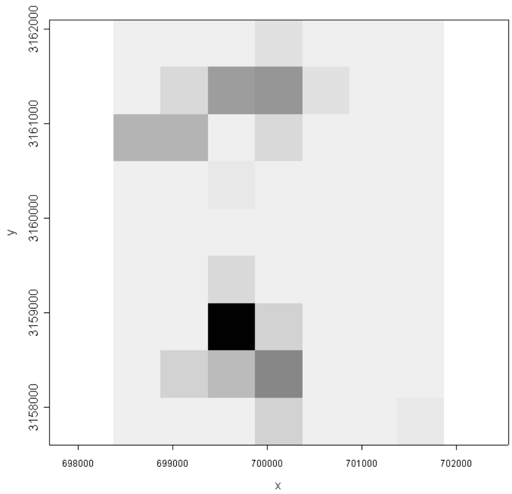


Figure 2.16: The number of wild boars relocations inside each cell of a virtual grid generated with the function `ascgen`.

The function `ascgen()` is useful to define "virtual grids", when no objects of class `"asc"` are available initially. The name of the animals monitored are available in the column `Name` of the object `puechabon$locs`. Once the virtual grid `hihi` has been defined, the number of relocations of each animal falling in each cell of the grid can be drawn using the function `count.points.id()` (Figure 2.17):

```
> tmpbis <- count.points.id(xy = puechabon$locs[, c("X", "Y")],
+     id = puechabon$locs$Name, hihi)
> image(tmpbis)
```

### 2.3.8 Copying attributes of a map into a matrix with the same dimensions

The function `getascattr()` is extremely useful when one is working with objects of class `"asc"`. For example, we want to divide the elevation of the study area (stored in the object `el` created in section 2.3.2) into two intervals (lower than 200 m a.s.l., and upper than 200 m a.s.l.). This operation can be done by typing the following commands:

```
> el <- getkasc(puechabon$kasc, "Elevation")
> elcat <- el < 200
> class(elcat)

[1] "matrix"

> names(attributes(elcat))

[1] "dim"
```

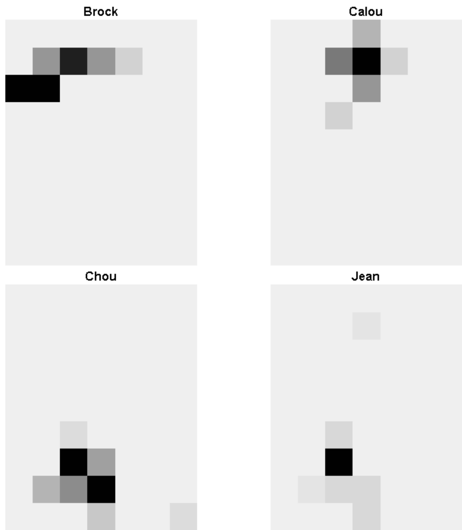


Figure 2.17: The number of wild boars relocations inside each cell of a virtual grid, for each boar.

Note that the result is not an object of class "asc". Further, all the attributes of the map (cellsize, xll, yll, see above) have been lost. The function getascattr() copies the attributes of an object of class "asc" into a matrix of the same dimensions (Figure 2.18):

```
> (elcat <- getascattr(el, elcat, type = "factor", lev = c("> 200 m",
+     "< 200 m")))
```

```
Raster map of class "asc":
Cell size:  100
Number of rows:  121
Number of columns:  111
Type:  factor
```

```
> image(elcat)
> legend(698000, 3165000, levels(elcat), fill = rainbow(2))
```

The function getkascattr() is an extension of the function getascattr() for objects of class "kasc".

### 2.3.9 Maps of different areas

It often occurs that the different maps available to an ecologist do not cover exactly the same area (see Figure 2.19). Although the maps have the same dimension, some pixels have values measured for some variables, while other variables are not mapped at this place; the mapped area is not exactly the same for all variables. Most analyses do not deal with these "partially missing pixels". The function managNAkasc() sets to NA all the pixels that are not mapped for all variables.

The following example renders even clearer this point:

```
> kasc <- puechabon$kasc
> el <- getkasc(kasc, "Elevation")
> sl <- getkasc(kasc, "Slope")
> el[el < 200] <- NA
> tmp <- as.kasc(list(Elevation = el, Slope = sl))
> image(tmp)
```

Note that the two maps in tmp do not cover the same area (Figure 2.20);

```
> tmp <- managNAkasc(tmp)
> image(tmp)
```

And now, the two maps are covering exactly the same area (Figure 2.21).

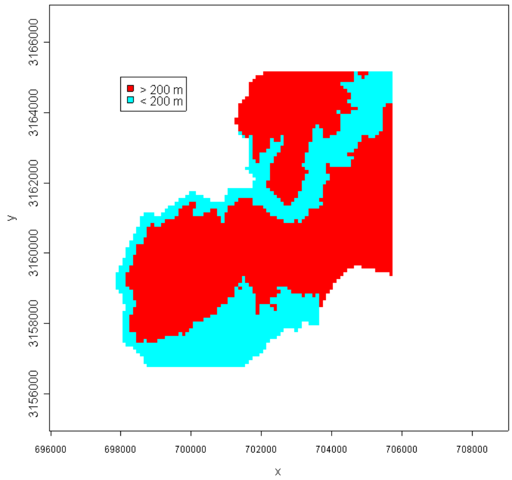


Figure 2.18: The elevation at Puéchabon divided into two classes (< 200 m and > 200 m, see text).

### 2.3.10 The multivariate analysis of objects of class "kasc"

As objects of class "kasc" store the information about several environmental variables, a multivariate analysis can generate a global view of the area. Actually, the class "kasc" is a middle step between the data frames and the maps. However, the object kasc contains a lot of missing values (the pixels where the variables are not mapped). The functions kasc2df() and df2kasc() are intended to convert an object of class "kasc" into an object of class "data.frame", and conversely. The following example renders this point clearer:

```
> data(puechabon)
> kasc <- puechabon$kasc
> toto <- kasc[1:10, ]
> class(toto) <- "data.frame"
> toto
```

```
   Elevation Aspect Slope Herbaceous
1         NA   <NA>    NA         NA
2         NA   <NA>    NA         NA
3         NA   <NA>    NA         NA
4         NA   <NA>    NA         NA
5         NA   <NA>    NA         NA
6         NA   <NA>    NA         NA
7         NA   <NA>    NA         NA
8         NA   <NA>    NA         NA
9         NA   <NA>    NA         NA
10        NA   <NA>    NA         NA
```

To have an idea of the structures of the variables on the study area, it is necessary to remove these missing values (cf Figure 2.6 and 2.19), as the functions available in the R package **ade4** do not deal with the missing values:

```
> huhu <- kasc2df(kasc)
> names(huhu)
```

```
[1] "index" "tab"
```

```
> huhu$index[1:4]
```

```
[1] 2017 2018 2019 2020
```

```
> huhu$tab[1:4, ]
```

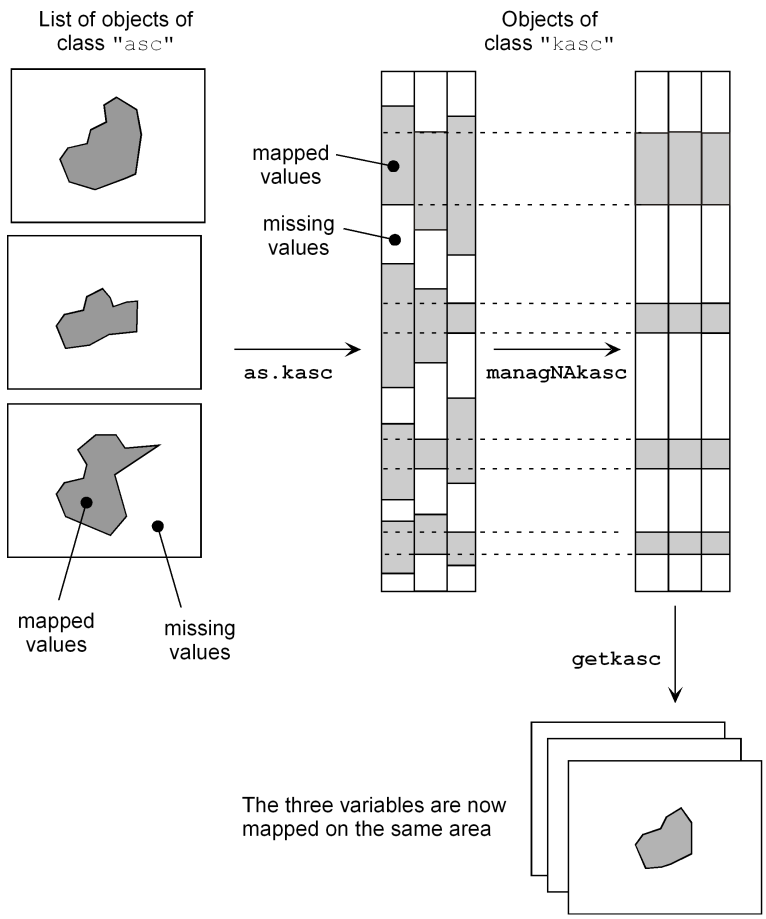


Figure 2.19: The use of `managNAkasc`: A list of objects of class `"asc"` is available, but the areas mapped are not the same for the three variables, even if the maps cover the same area. The function `as.kasc` converts this list into a data frame of class `"kasc"` (see Figure 2.6). Then, the function `managNAkasc` sets to NA all the pixels that are not mapped for all the variables.


Figure 2.20: Two maps (elevation and slope) of Puéchabon stored in the object `tmp`. Note that the two maps do not cover the same area.

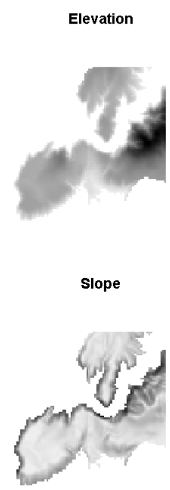


Figure 2.21: The maps of elevation and slope at Puéchabon. Only areas with elevation > 200 m a.s.l. are mapped.

```
     Elevation    Aspect    Slope Herbaceous
2017        68 SouthWest 1.548994          0
2018        69 SouthWest 1.118594          0
2019        70 SouthWest 1.358634          0
2020        70 SouthWest 1.724311          0
```

The data frame `huhu$tab` contains the pixels for which the mapped variables are not missing, and the vector `huhu$index` contains the indices of the rows of the object `kasc` that are not missing. This vector can be used to back-transform the data frame into an object of class `"kasc"` (see Figure 2.22).

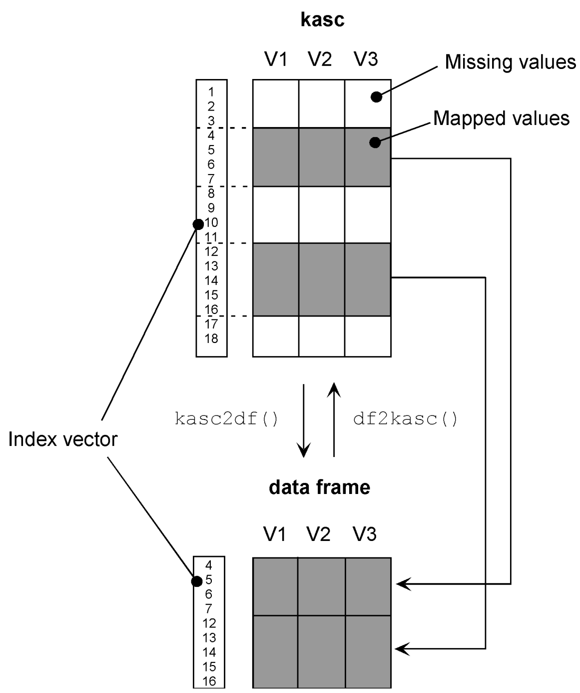


Figure 2.22: The rationale behind the functions `kasc2df` and `df2kasc`.

Once the object of class `"kasc"` has been transformed into a data frame, a multivariate analysis can be done. For example, we keep all the continuous variables (Aspect through out) to compute a Principal Component Analysis:

```
> huhu$tab$Aspect <- NULL
> (pc <- dudi.pca(huhu$tab, scannf = FALSE, nf = 2))
```

```
Duality diagramm
class: pca dudi
$call: dudi.pca(df = huhu$tab, scannf = FALSE, nf = 2)

$nf: 2 axis-components saved
$rank: 3
eigen values: 1.449 0.8475 0.7036
  vector length mode    content
1 $cw    3      numeric column weights
2 $lw    4379   numeric row weights
3 $eig   3      numeric eigen values

  data.frame nrow ncol content
1 $tab       4379 3    modified array
2 $li        4379 2    row coordinates
3 $l1        4379 2    row normed scores
4 $co        3    2    column coordinates
```

```
5 $c1        3    2    column normed scores
other elements: cent norm
```

Then, the first two principal axes can be mapped by the function `df2kasc()`, using as argument the index vector previously returned by the function `kasc2df()` (Figure 2.23):

```
> map <- df2kasc(pc$li, huhu$index, kasc)
> image(map)
```

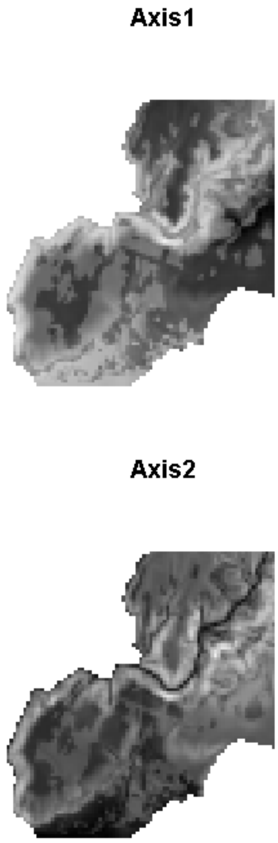


Figure 2.23: Map of the pixels scores in the principal component analysis of the variables elevation, slope and herbaceous cover (Puéchabon, South of France).

### 2.3.11 Diminishing the resolution of a map

Since R is not a Geographic Information System, it is current that the maps imported into objects of class `asc` or `kasc` are too large, and require too much memory for R . This is the case when high-resolution maps cover a large area. However, a high resolution is not necessarily of major importance in Ecology. Indeed, the ecological studies are often carried out at a given scale, and the precision of the data collected is often linked to this scale. For example, if the distribution of a species is to be studied at the continental scale, it is not essential to have maps with a pixel size of 10 metres. When such cases occur, the function `lowres()` can be used to diminish the resolution of the map. We use here the data set `chamois`, described and loaded in the section 2.3.1:

```
> (kasc <- chamois$map)
```

```
Raster map of class "kasc":
Cell size:  50
Number of rows:  353
Number of columns:  353

Variables measured:
1. Vegetation: factor
2. Disteco: numeric
3. Slope: numeric
```

```
> (si1 <- object.size(kasc))
```

```
[1] 3988880
```

The maps of the area are displayed on Figure 2.24:

```
> image(kasc)
```


Figure 2.24: Maps of the Chartreuse mountain, in the French Alps (data set `chamois`). The resolution of the map is of 50 m.

Sometimes, we need to get a less precise resolution for a map. For instance, the map `kasc` is rather large and we want to diminish the resolution to 200 m instead of 50 m. So we merge together all pixels contained in a square of 4*4 adjacent pixels of side length 50 m into one pixel of side length 200 m :

```
> (m <- lowres(kasc, np = 4))
```

```
Raster map of class "kasc":
Cell size:  200
```

```
Number of rows:  88
Number of columns:  88

Variables measured:
1. Vegetation: factor
2. Disteco: numeric
3. Slope: numeric
```

```
> image(m)
```


Figure 2.25: Same maps as in Figure 2.24, but with a resolution diminished to 200 m.

The size of the maps is diminished here:

```
> (si2 <- object.size(m))
[1] 249192
> (si1 - si2)/si1
[1] 0.9375283
```

This results in a large economy of memory (reduction of 93%), without erasing the main features of the area.

### 2.3.12 Subsetting an area

This operation is somewhat related to the previous one. In some cases, it may be useful to work on a smaller portion of the study area. This can be achieved using the function `subsetmap()`. For example, we subset the map of the Chartreuse mountain, to work on a smaller portion of the map (Figure 2.26):

```
> data(chamois)
> slope <- getkasc(chamois$map, "Slope")
> def.par <- par(no.readonly = TRUE)
> layout(matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 2), ncol = 3, byrow = TRUE))
> par(mar = c(0, 0, 0, 0))
> image(slope, axes = FALSE)
> box()
> x <- c(863603.8, 867286.5)
> y <- c(2042689, 2045797)
> polygon(x = c(x[1], x[2], x[2], x[1]), y = c(y[1], y[1], y[2],
+     y[2]), lwd = 2)
> sl2 <- subsetmap(slope, xlim = x, ylim = y)
> par(mar = c(0, 0, 2, 0))
> image(sl2, axes = FALSE, main = "Reduced map")
> box()
> par(def.par)
```

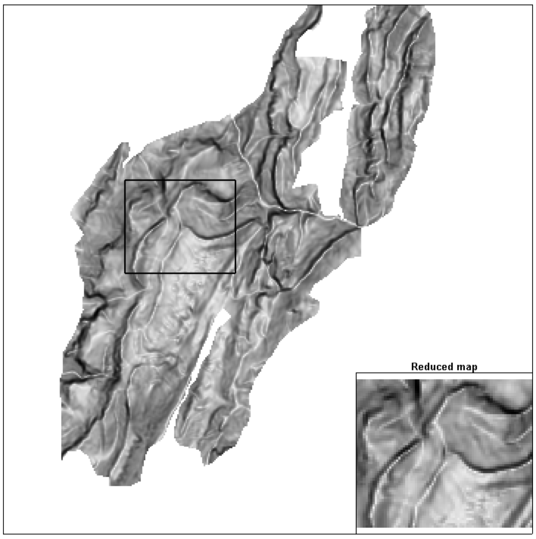


Figure 2.26: The use of the function `subsetmap()` allows to select a part of a given map (delimited by a rectangle), and to store it in another map (labelled "reduced map" in this figure)

In this case, the limits of the new maps were specified in `x` and `y`. Alternatively, when no limits are indicated for the new maps, the user is asked to click on the old map to delimit the upper-left and the bottom-right corners of the new map (try it!). This function can be used on both `"asc"` and `"kasc"` maps.

### 2.3.13 Other operations

The list presented here does not include all the functions available in **adehabitat** but provides the most frequent basic operations that can be done with the package. Numerous other basic functions are available to the user for programming purposes. For example, the function `labcon()` can be used to label connected features. The function `getXYcoords()` can be used to get the coordinates of the rows and the columns of pixels on the maps. We recommend the user to have a look to the help pages of these functions (type `help.start()`, then choose `packages`, and finally `adehabitat`), and especially to the examples for a better view of what can be done with the package.

# Chapter 3

# The storage mode of vector maps in adehabitat

## 3.1 Description of the class "area"

The class "area" is a class that has been developed in the R package **ade4**. This is the only class of vector maps that can be used with **adehabitat**. The most simple way to create such objects is to use the function `as.area()`. The data set `elec88` of the package **ade4** contains an object that can be converted to this class. The data frame `elec88$area` contains the coordinates of the boundaries of the French departments:

```
> data(elec88)
> ar <- elec88$area
> ar[1:5, ]

  V1  V2  V3
1 D1 432 213
2 D1 442 199
3 D1 448 204
4 D1 448 219
5 D1 451 227
```

This data frame has three columns. The first variable is a factor defining the polygons. The second and third variables are the x and y coordinates of the polygon vertices in the order where they are found. This format is the standard input to `as.area()`:

```
> ar <- as.area(ar)
> area.plot(ar)
```

The object `ar` is now a data frame of class "area".

Many functions in **adehabitat** return or require objects of class "area". Thus, the functions `mcp()` (to compute the minimum convex polygon home range), `getverticeshr()` (to compute the kernel home range), or `getcontour()` (to compute the contour polygon of a raster object) return this class of object. This class can also be used in many graphical functions of the R packages **ade4** and **adehabitat**, to overlay vector maps and other types of information (*e.g.* `s.label()`, `plot.traj()`, etc.).

*Note:* The polygons stored in the object may or may not overlap each other. For example, in the data set `elec88`, displayed previously, the mapped features are the French departments. Since one given location always belongs to only one department, the polygons defined in the object `ar` do not overlap each other. However, in other objects of class "area", the polygons do overlap. For example, let us consider the data set `puechabon`, presented in the section 2.3.1:

```
> data(puechabon)
> lo <- puechabon$locs
```

Figure 3.1: Map of France, stored in R as an object of class "area"

```
> cp <- mcp(lo[, c("X", "Y")], lo[, "Name"])
> class(cp)
```

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

The object cp contains the coordinates of the vertices of the home ranges of the four wild boars monitored using radio-tracking, computed as the minimum convex polygon encompassing all relocations. These home ranges may be mapped using the function area.plot() of the package **ade4** (Figure 3.2). Since these animals are not territorial, their home-range overlap each other:

```
> opar <- par(mar = c(0, 0, 0, 0))
> area.plot(cp)
> points(puechabon$locs[, c("X", "Y")], pch = 16, col = as.numeric(puechabon$locs$Name))
> box()
> par(opar)
```

It is then possible to export an object of class "area" toward a GIS software using the function area2dxf(). This function exports an object of class "area" to the DXF file format (http://www.autodesk.com/techpubs/autocad/acad2000/dxf/). The DXF file format can be read by nearly all GIS softwares. Using the object cp, previously defined:

```
> area2dxf(cp, file = "myfile.dxf")
```

This command creates a new file in your working directory, named "myfile.dxf", that can be read into a GIS. The help page of this function explains how variables measured for each polygon can be exported in the DXF file (*e.g.* exportation of the proportion of tree cover inside each home range).

## 3.2 Conversion between raster and vector maps

Several functions are available to allow the conversion between raster and vector maps. First, the function getcontour() can be used to convert the connected features of an object of class "asc" into polygons. Consider the example of the data set puechabon:

```
> el <- getkasc(puechabon$kasc, "Elevation")
> cont.el <- getcontour(el)
> class(cont.el)
```

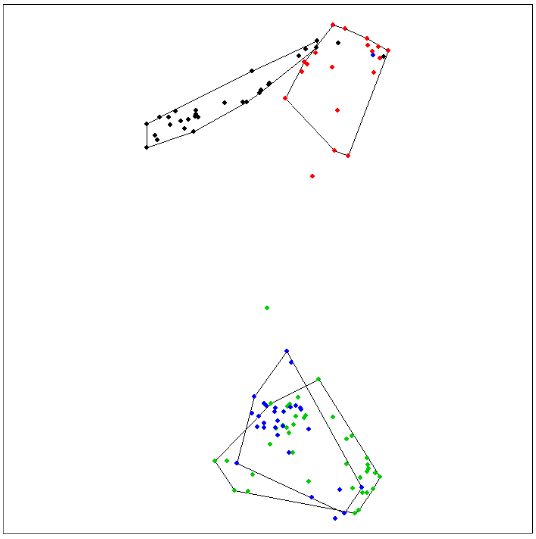

Figure 3.2: Overlapping polygons stored in an object of class area (home ranges of 4 wild boars monitored using radio-tracking at Puéchabon). The relocations of the boars are also displayed.

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

```
> nlevels(cont.el[, 1])
```

```
[1] 1
```

The object `cont.el` is an object of class `"area"`. There is only one connected feature on the map `el`, and this is transformed into one polygon (Figure 3.3):

```
> image(el)
> polygon(cont.el[, 2:3], lwd = 3)
```

Note that more complex maps can also be used. On the other hand, the converse operation (rasterization of vector polygons) is also possible, using the functions `mcp.rast()` and `hr.rast()`. The function `mcp.rast()` converts only one polygon to raster (*i.e.* a data frame with two columns, the x-y coordinates of the vertices of the polygon), whereas the function `hr.rast()` converts an object of class `"area"` to raster. For example, using the data set `puechabon`:

```
> lo <- puechabon$locs
> kasc <- puechabon$kasc
> cp <- mcp(lo[, c("X", "Y")], lo[, "Name"])
```

We define again `cp`, an object of class `area` containing the vertices of the wild boars home ranges (see above). We transform this object into a raster map of class `"kasc"`, using the function `hr.rast()`:

```
> (rast <- hr.rast(cp, kasc))
```

```
Raster map of class "kasc":
Cell size:  100
Number of rows:  121
Number of columns:  111

Variables measured:
1. Brock: numeric
2. Calou: numeric
3. Chou: numeric
4. Jean: numeric
```

```
> def.par <- par(no.readonly = TRUE)
> layout(matrix(c(1, 1, 2, 4, 3, 5), 2, 3))
> par(mar = c(0, 0, 4, 0))
```

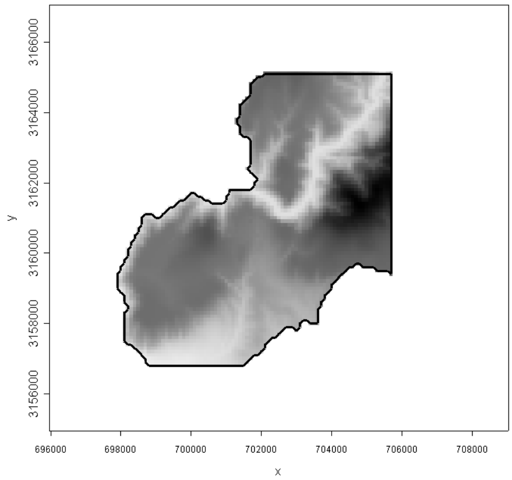


Figure 3.3: The boundaries of the Puéchabon area stored as an object of class "area" (computed using the function getcontour()). The vertices of the polygon are the centres of the pixels located on the boudary of the mapped area.

```
> area.plot(cp)
> points(puechabon$locs[, c("X", "Y")], pch = 16, col = as.numeric(puechabon$locs$Name))
> box()
> for (i in names(rast)) {
+     image(getkasc(rast, i), main = paste("Wild boar named", i),
+         axes = FALSE)
+     polygon(cont.el[, 2:3])
+     box()
+ }
> par(def.par)
```

The resulting object is an object of class "kasc", with one column per level of the factor lo$Name. Each map in this object corresponds to the (raster) home-range of one animal (Figure 3.4).

## 3.3 Using masks

The use of masks is a common operation in GIS. A mask allows to specify on a map the limits of a small area, where the subsequent statistical analyses are to be done. The mask is a map of class "asc", that should contain missing values for those areas where processing should not occur and any value for the cells to be processed. We illustrate here an example of use of masks, because it implies knowledge of the functions described previously. A new function, setmask(), is introduced here. First, consider the map of elevation at Puechabon and the polygon pol, which delimits the plateau (see Figure 3.5):

```
> el <- getkasc(puechabon$kasc, "Elevation")
> pol <- data.frame(x = c(700658, 699222, 698342, 698643, 700427,
+     701029), y = c(3160768, 3160676, 3159402, 3158336, 3158869,
+     3159657))
> image(el)
> polygon(pol, lwd = 2)
```

We rasterize the polygon, to define a mask, and we use the resulting map as argument for the function setmask() to keep areas mapped inside the polygon (Figure 3.6):

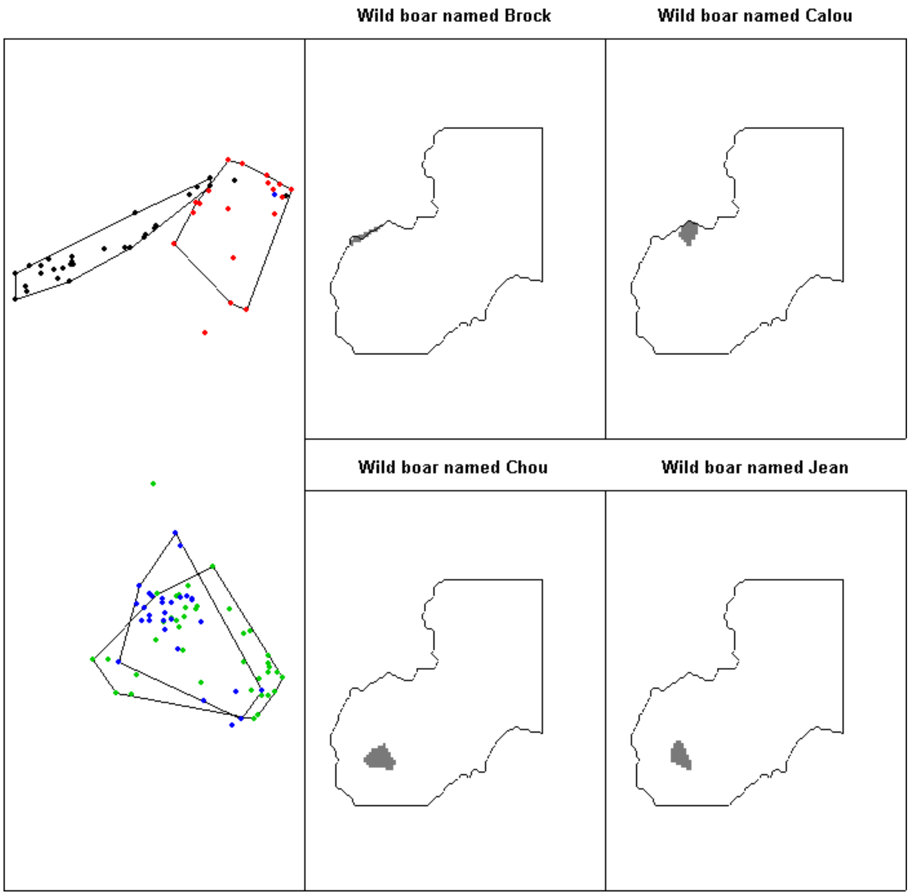


Figure 3.4: left: the home ranges of the four wild boars monitored by radio-tracking, stored as an object of class `"area"`; right: the rasterized home-ranges, stored in an object of class `"kasc"` (computed using the function `hr.rast()`).

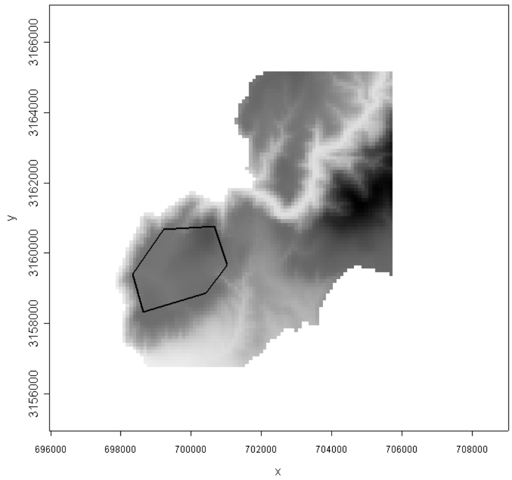


Figure 3.5: The map of elevation at Puéchabon (South of France). The polygon delimits the plateau of Puéchabon.

```
> pr <- mcp.rast(pol, el)
> masked.kasc <- setmask(puechabon$kasc, pr)
> image(masked.kasc, xlim = c(696999, 702373), ylim = c(3156784,
+     3162297))
```

Note that in this case, the coordinates of the masking polygon were available. When the user wants to click on the map to define himself a masking polygon, it is straightforward to write a function using the function `locator()` of the **base** package. For example, the following function is intended to achieve this aim (try it, to defined your own polygon in the example above). The argument `x` is the number of vertices of the polygon:

```
> def.pol <- function(x) {
+     toto <- locator(1)
+     for (i in 2:x) {
+         tutu <- locator(1)
+         toto$x <- c(toto$x, tutu$x)
+         toto$y <- c(toto$y, tutu$y)
+         lines(toto$x, toto$y)
+     }
+     polygon(toto)
+     return(toto)
+ }
```

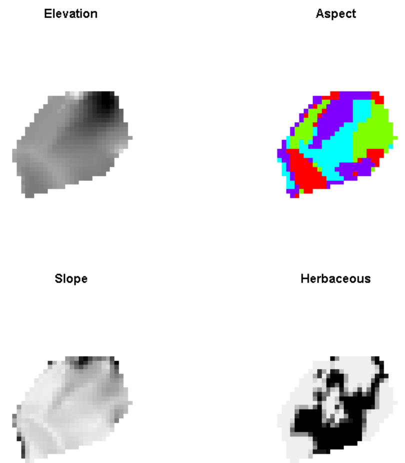


Figure 3.6: The maps of the plateau of Puéchabon (South of France), after the use of the function `setmask`, to mask areas outside the limits of the plateau.

# Chapter 4

# Conclusion

We have presented here the main classes of map objects available in **adehabitat**. Numerous more specific functions are available to analyse data related to the spatial distribution of animals. These functions concern:

1. the estimation of the home range of animals, using the kernel method (`kernelUD()`) or the minimum convex polygon (`mcp()`);
2. the exploration of autocorrelated locations in radio-tracking analysis, using:
   - the computation of turning angles (`angles()`);
   - the computation of travel speeds (`speed()`) ;
3. The analysis of habitat selection, using :
   - selection ratios (`wi()`);
   - compositional analysis (`compana()`);
   - habitat suitability maps (algorithm DOMAIN: `domain()`; Mahalanobis distances: `mahasuhab()`; Resource Selection Functions may also be computed with the function `glm()` of the package **base** and the functions presented in this tutorial);
   - multivariate analyses relying on the concept of ecological niche (ENFA: `enfa()`; K-select analysis: `kselect()`; niche analysis, from the package **ade4**: `niche()`).

We encourage the reader look at the help page of these functions. We hope that this package will be useful to biologists concerned by the analysis of the space use by animals.

# Chapter 5

# Acknowledgements

I warmly thank Anne-Béatrice Dufour and Mathieu Basille (Laboratoire de Biométrie, University of Lyon, France) for their useful comments on earlier drafts of this manuscript. I am also grateful to Mathieu Garel, Gaelle Darmon, Jodie Martin (Laboratoire de Biométrie, University of Lyon, France), and Cyrille Chatelain (Conservatoires et Jardins Botaniques de Genève, Switzerland), who tested **adehabitat** and advised me about the improvements to bring to the package.