Tools for the Analysis of Mapped Data: the `adehabitatMA` Package

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

Feb 2011

Contents

1 History of the package adehabitatMA **1**

2 The aim of the package **3**
2.1 sp and adehabitatMA 3
2.2 Controlling the options 4

3 Working with raster maps only **5**
3.1 Exploring a raster map interactively 5
3.2 Labelling connected features on a map 6
3.3 Computing the contour of a mapped area on a raster map 10
3.4 Mathematical morphology 11
3.5 Changing the resolution of a map 13
3.6 Subsetting a map 16

4 Working with points and maps **17**
4.1 Counting the number of points in pixels 18
4.2 Finding the value of mapped variables at specified locations . . . 21
4.3 Generating a raster map from points data 22

5 Computing buffer regions **23**

6 Conclusion **25**

1 History of the package adehabitatMA

The package `adehabitatMA` contains functions allowing the analysis of mapped data that were originally available in the package `adehabitat` (Calenge, 2006).

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:

- `adehabitatMA` package provides methods for dealing with maps in R.
- `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.

We consider in this document the use of the package `adehabitatMA` to deal with the analysis of mapped data. All the methods available in `adehabitatMA` are also available in `adehabitat`, but the classes of data returned by the functions of `adehabitatMA` 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`. Note that functions allowing the conversion from old classes of `adehabitat` to new classes of `adehabitatMA` and `sp` are described on the help page of the function `kasc2spixdf`. We therefore suppose that the user is familiar with this package.

Package `adehabitatMA` is loaded by

```
> library(adehabitatMA)
```

2 The aim of the package

2.1 sp and adehabitatMA

The package `sp` is really useful for the analysis of mapped data, and for this reason, I encourage the user to become familiar with this package. However, the package `adehabitat` originally contained functions which were also useful in spatial analysis though not available in `sp`, and especially in the field of analysis of animal space use. For this reason, I included these functions in the package `adehabitatMA`. I demonstate the use of these functions in this document.

I will demonstrate the use of this package using the example dataset `lynxjura`.

```
> data(lynxjura)
```

This dataset contains the results of the monitoring of the lynx in the Jura mountains (France) by the French lynx network of the *Office national de la chasse et de la faune sauvage*. This dataset has two components: `$locs` is a `SpatialPointsDataFrame` containing the locations of presence indices of the lynx, along with the date of collection and the type of indices (attacks on livestocks, captured animal, tracks, etc.). Look at the first rows of this object:

```
> head(lynxjura$locs)
```

```
        coordinates     Date Type
3 (856208, 2135450) 19801231   EP
4 (852942, 2134370) 19810101   EP
5 (857408, 2134780) 19810101    E
7 (857425, 2135190) 19810114    E
8 (853543, 2133730) 19810114    E
9 (852771, 2132560) 19810114    E
```

The component `$map` of this dataset is a `SpatialPixelsDataFrame` contains the maps of four environmental variables measured on the study area:

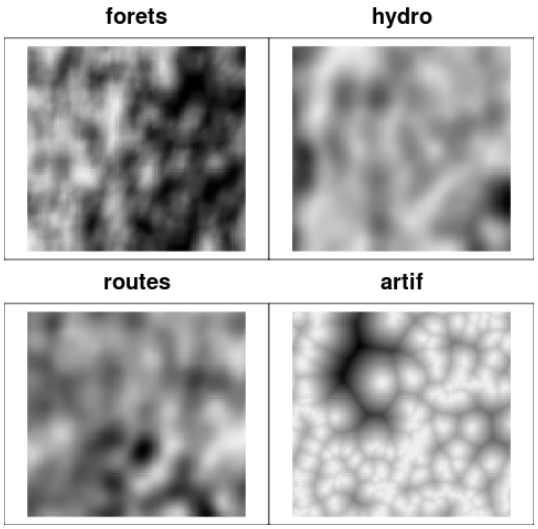
```
> lynxjura$map
```

```
Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
x            828916      500        96
y           2130700      500        90

Variables measured:
     forets     hydro    routes     artif
1  5.154568 11.230920 11.622950 275.80010
2 11.643950 10.054470 11.389700  21.72639
```

```
3 16.126420  8.814120 10.972950  55.53905
4 18.757530  7.549436 10.417370  95.46117
5 19.779750  6.323049  9.784133 448.96580
6 22.390620  5.249640  9.140086 699.53190
...
```

The whole content of this object can be displayed using the function `mimage`:

```
> mimage(lynxjura$map)
```



2.2 Controlling the options

The user familiar with the use of the package `sp` might have noted, in the previous section, that the content of the object of class `SpatialPixelsDataFrame` was printed in an unusual way. Actually, the package `adehabitatMA` defines its own printing methods. The use of these printing methods can be controlled with the function `adeoptions`. The default value for the `shortprint` option is `TRUE`:

```
> adeoptions(shortprint = TRUE)
```

which causes the `Spatial` objects to be printed in that way;

```
> lynxjura$map

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
```

```
x            828916       500         96
y           2130700       500         90

Variables measured:
     forets     hydro    routes     artif
1  5.154568 11.230920 11.622950 275.80010
2 11.643950 10.054470 11.389700  21.72639
3 16.126420  8.814120 10.972950  55.53905
4 18.757530  7.549436 10.417370  95.46117
5 19.779750  6.323049  9.784133 448.96580
6 22.390620  5.249640  9.140086 699.53190
...
```

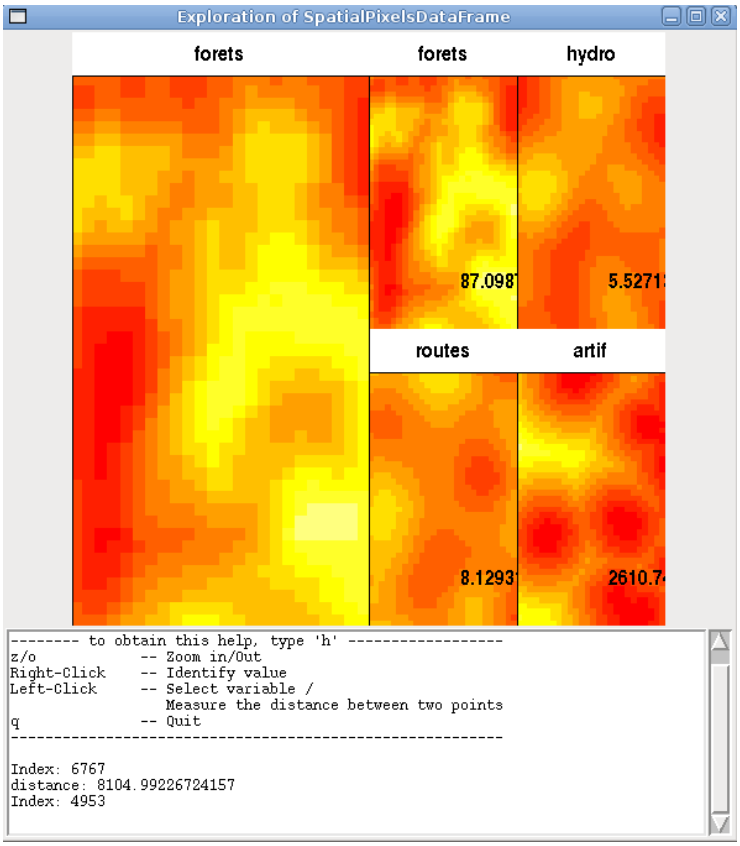
The user might find disturbing the use of these methods, and prefer the original printing methods proposed by the package `sp`. In this case, the `shortprint` option should be turned to `FALSE`.

3 Working with raster maps only

3.1 Exploring a raster map interactively

`adehabitatMA` provides one function allowing the exploration of objects of class `SpatialPixelsDataFrame`, the function `explore`. This allows to interactively explore a map, to find the value of the variables at a given place, to measure the distance between two points, to zoom on a given part. Note that this function requires the package `tkrplot`:

```
> explore(lynxjura$map)
```



This function also has an argument named `panel.last`, which allows to pass an expression to be evaluated after plotting has taken place. For example, we can use it to explore the distribution of lynx indices in relationship with the maps. Copy and paste the following expression:

```
> explore(lynxjura$map, panel.last = function() points(lynxjura$locs,
+     pch = 3))
```

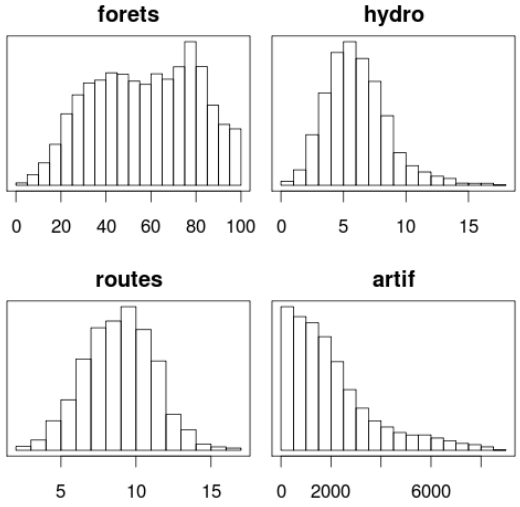
3.2 Labelling connected features on a map

It is sometimes useful to identify automatically separated components on a raster map. For example, consider the maps in the object `lynxjura` loaded previously:

```
> map <- lynxjura$map
```

Look at the values of the mapped variables:

```
> hist(map)
```



Consider the variable "forets", which represents the percentage of forested area in each pixel of the map. We will consider areas with at least 95% of forests:

```
> forest <- map[, 1]
> forest[[1]][forest[[1]] < 95] <- NA
> image(forest, col = "green")
```

We may wonder how many forested areas are identified using this criterium. in the present case, we can count them ourselves: 9 areas. However, the function `labcon` provides an efficient way to do it:

```
> lab <- labcon(forest)
> image(lab)
```

The resulting object is a `SpatialPixelsDataFrame` mapping one integer vector taking a unique value per connected component. Therefore, we can count the number of components in the map.

```
> lab

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
1            828916      500        96
2           2130700      500        90

Variables measured:
     z
1223 7
1224 7
1225 7
1226 7
1227 7
1228 7
...

> max(lab[[1]])

[1] 9
```

It is easy to measure the area of each component, because we know the area covered by a pixel:

```
> gridparameters(lab)
```

```
  cellcentre.offset cellsize cells.dim
1            828916      500        96
2           2130700      500        90
```

Therefore, the area covered by each component can be computed by:

```
> table(lab[[1]]) * 500 * 500
```

```
      1       2       3       4       5       6       7       8
9250000 3750000 18750000 3000000  250000  500000 30250000 6750000
      9
 500000
```

The result is here returned in squared meters as the original units of the maps were in meters.

We can also measure the average density of rivers (second variable, names "hydro" in the object `map`) in the component 1. We first have to transform the objects `lab` and `map` as full grid to allow the comparison:

```
> fullgrid(lab) <- TRUE
> fullgrid(map) <- TRUE
```

Then we compute the mean of the density of rivers in the component 1:

```
> mean(map[[2]][lab[[1]] == 1], na.rm = TRUE)
```

```
[1] 3.944150
```

Or getting the map of "hydro" for the first connected component in a separate map:

```
> comp1 <- map[2]
> comp1[[1]][map[[1]] < 95] <- NA
> comp1[[1]][lab[[1]] != 1] <- NA
> image(comp1)
```

3.3 Computing the contour of a mapped area on a raster map

Consider again the map of forested area built in the previous section:

```
> image(forest, col = "red")
```

It may sometimes be useful to compute the coordinates of the vertices of the contour polygon of the forested area:

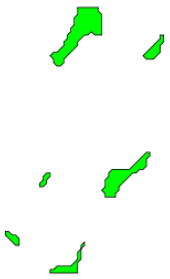
```
> con <- getcontour(forest)
```

```
Warning message:
In getcontour(forest) :
  At least 3 pixels are required to compute a contour.
3 components erased
```

The resulting object is of the class `SpatialPolygons`. It can therefore be handled with other functions of the package `sp`, and can be exported toward a GIS using the functions of the package `maptools`.

However, note the warning here. To understand it, look at the resulting object:

```
> plot(con, col = "green")
```



There are 6 polygons plotted here. However, we saw in the previous section that there are 9 forested areas in the studied regions. Three of the forested areas are built by less than three pixels, so that no contour can be calculated for these "small forests".

3.4 Mathematical morphology

In the last case, we may be annoyed to "loose" these three small forests. It is easy to understand that at least three points are required to build a polygon, and that the function cannot find the contour polygon of only one point (one pixel).

There is however a way to circumvent this drawback. We can use the function `morphology`, which performs operations of mathematical morphology: it allows to erode or to dilate the object (see the help page of the function. In our case, we may dilate each forested area by one pixel:

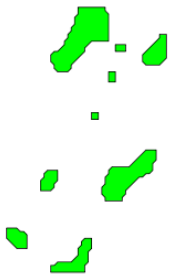
```
> for1 <- morphology(forest, "dilate", nt = 1)
> for1

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
1            828916      500        96
2           2130700      500        90

Variables measured:
     z
1126 1
1127 1
1128 1
1129 1
1130 1
1131 1
...
```

The resulting object is a `SpatialPixelsDataFrame`. Now compare the area covered by the components of `for1` with the area covered by the components of `forest`:

```
> image(for1, col = "blue")
> image(forest, col = "yellow", add = TRUE)
```


Each forested area has been expanded by exactly one pixel on all sides. Note that the components on the boundary of the map cannot be expanded. Now if we use the function `getcontour` again:

```
> plot(getcontour(for1), col = "green")
```



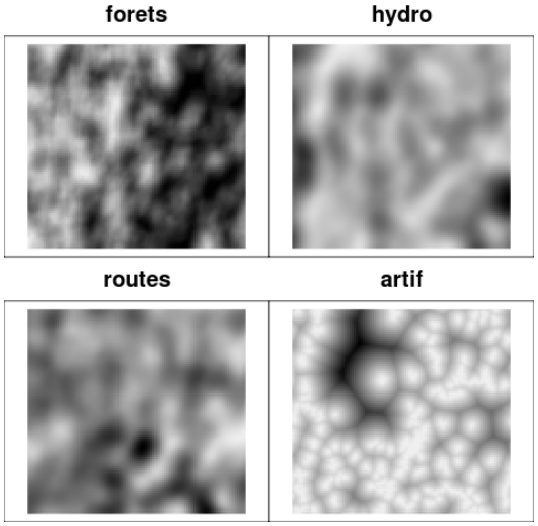
All forested areas are now present in this vectorized object. This operation of course assume that the area increase caused by the use of the function can be considered as negligible.

Note that the inverse operation (erosion) is also possible with the function `morphology`.

3.5 Changing the resolution of a map

The package `adehabitatMA` provides a function allowing to change the resolution of the map. For example, consider the map of the study area used previously:

```
> map <- lynxjura$map
> mimage(map)
```



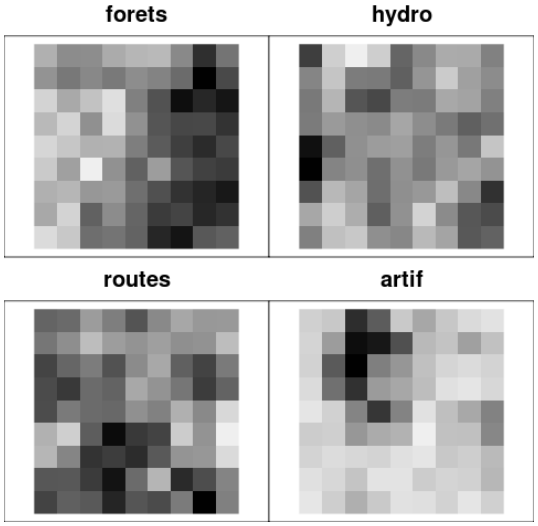
The parameters of this map are:

```
> gridparameters(map)
```

```
  cellcentre.offset cellsize cells.dim
x            828916      500        96
y           2130700      500        90
```

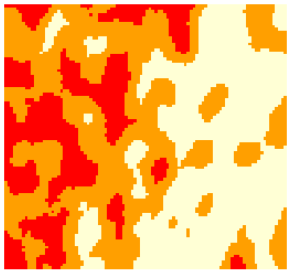
We may want to change the resolution of this map so that pixels cover 5 $\times$ 5 km instead of 500 $\times$ 500 m. That is, we want to merge together 10 pixels into one pixel. The function `lowres` allows to perform this operation. We first define `map` as a `SpatialPixelsDataFrame`:

```
> mimage(lowres(map, 10))
```



The function computes the average value in each pixel of the new map. However, note that in some case, it is not sensible to compute an average to summarize the values observed in a given pixel. For example, imagine that we transform the variable "forets" in the following way:

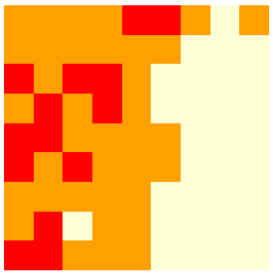
```
> map[[1]] <- as.numeric(cut(map[[1]], 3))
> image(map, 1)
```



The map is now a factor with three classes. In such a case, it does not make sense to compute a mean to summarize the values of the pixels. In such a case, the function `lowres` assigns to the pixel the most frequent level. When several

levels are equally represented in the pixel, the function randomly samples one of these levels. We have to indicate to the function which variable(s) in the object are factors thanks to the argument `which.fac`:

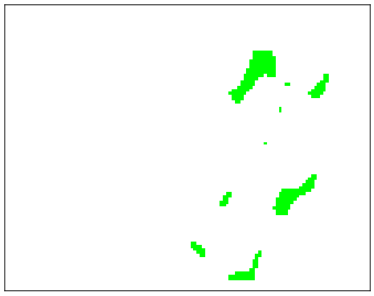
```
> image(lowres(map, 10, which.fac = 1))
```



3.6 Subsetting a map

Consider again the map of the forests:

```
> image(forest, col = "green")
> box()
```

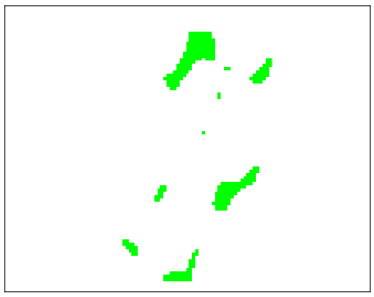


The forested areas are located in the eastern part of the area. We can use the function `subsetmap` to get that part of the map:

```
> for2 <- subsetmap(forest)
```

Then click on the map to indicate the lower-left and upper-right boundaries of the new map:

```
> image(for2, col = "green")
> box()
```



4 Working with points and maps

We now consider the case where we work with points and raster maps. Remember that the functions of `adehabitatMA` were originally available in `adehabitat` to study issues related to habitat selection by animals. Issues involving both points and maps are extremely frequent in this field. The points may be the relocations of animals collected using radio-tracking, or presence indices of animals on an area (as is the case for the `lynxjura` data). The maps therefore represent the environment of the animals.

We programmed two commonly used functions to deal with points and maps:

- `count.points` allows to count the number of points in each pixel of the raster map;
- `join` allows to identify the values of mapped variables at specified locations.

Although the function `overlap` of the package `sp` could easily be used to perform these operations, we prefered to provide specific functions, as these operations are very common in the field of habitat selection studies.

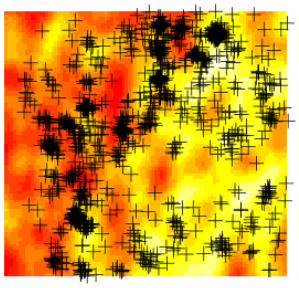
4.1 Counting the number of points in pixels

Consider the data on the lynx in the Jura mountains:

```
> data(lynxjura)
> map <- lynxjura$map
> class(map)

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

> locs <- lynxjura$loc
> class(locs)

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

Display the image:

```
> image(map, 1)
> points(locs, pch = 3)
```

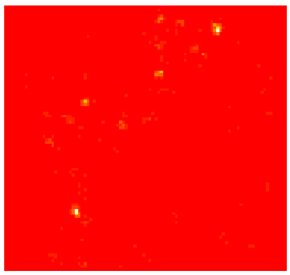


We can count the number of points in each pixel of the map:

```
> cp <- count.points(locs, map)

Warning message:
In count.points(locs, map) :
  several columns in the SpatialPointsDataFrame, no id considered
```

Note the warning message. We will talk about it later. Now, display the result:

```
> image(cp)
```



The warning returned by the functions is explained by the object containing the points. Indeed, this object should be, according to the help page, an object of class `SpatialPoints`, or `SpatialPointsDataFrame` *with one column*. In the latter case, the column is considered as a factor giving, for each point, the membership of the point to a set (e.g. the identity of the animal). In this case, the `SpatialPointsDataFrame` contains more than one column:

```
> head(locs)

        coordinates     Date Type
3 (856208, 2135450) 19801231   EP
4 (852942, 2134370) 19810101   EP
5 (857408, 2134780) 19810101    E
7 (857425, 2135190) 19810114    E
8 (853543, 2133730) 19810114    E
9 (852771, 2132560) 19810114    E
```

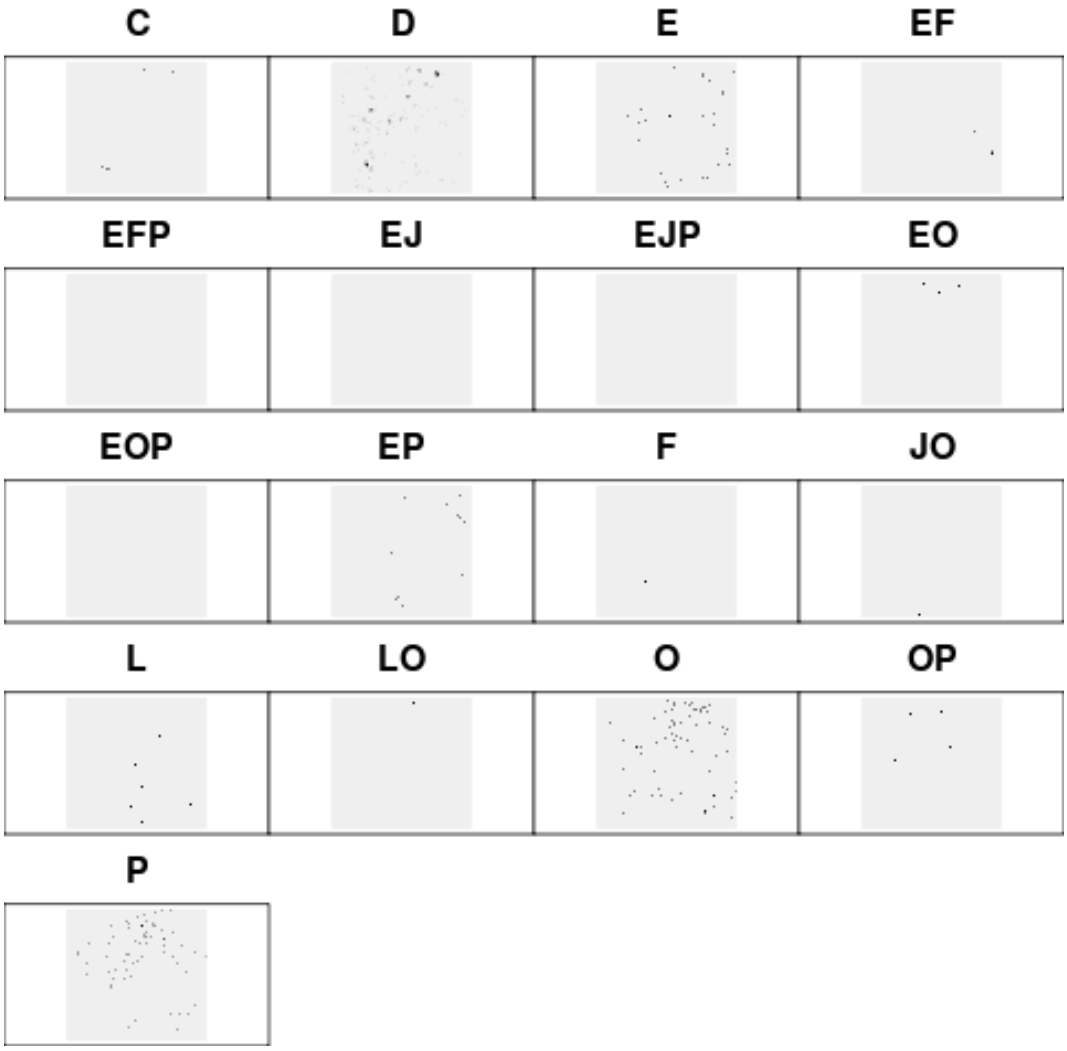
The function does not know which column should be considered as the variable defining the membership of points to a set. Therefore, it treats the object

as a `SpatialPoints` object (which is the desired behaviour). However, we could have wanted to compute several maps, with one map per type of index, indicating the number of indices of each type in each pixel of the map:

```
> cpr <- count.points(locs[, "Type"], map)
> cpr

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
  cellcentre.offset cellsize cells.dim
x            828916      500        96
y           2130700      500        90

Variables measured:
  C D E EF EFP EJ EJP EO EOP EP F JO L LO O OP P
1 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
2 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
3 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
4 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
5 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
6 0 0 0  0   0  0   0  0   0  0 0  0 0  0 0  0 0
...
```

The results can be displayed:

```
> mimage(cpr)
```



There is one map per type of points.

4.2 Finding the value of mapped variables at specified locations

The other possible approach consists to perform a spatial join: in other words, to find the value of mapped variables at specified locations For example:

```
> df <- join(locs, map)
```

The result is a data frame containing the value of the variables at the specified locations:

```
> head(df)

      forets    hydro    routes      artif
920 88.27358 3.107517 10.474140  930.50060
721 80.38864 5.049408  9.390557  666.97920
826 82.70963 3.859995 11.714720 1168.79900
922 82.42469 3.739009 11.310690  915.04360
626 82.05580 5.009838 10.072680  513.04710
433 77.81136 5.458577  9.881883   31.20114
```

This data frame can then be used in further analyses of habitat selection.

4.3 Generating a raster map from points data

In some cases it may be useful to generate a grid of pixels from a set of points. The package `sp` contains a function named `makegrid`, which generates a regular grid of points. However, I also included in `adehabitatMA` a function which computes directly a `SpatialPixelsDataFrame` from a set of points. This function allows to specify either the size of the pixel or the number of rows/columns of the grid. For example, to generate a `SpatialPixelsDataFrame` object from the set of locations of presence indices of the lynx, with a pixel size of 1000×1000 m:

```
> asc <- ascgen(locs, cellsize = 5000)
> asc

Object of class "SpatialPixelsDataFrame" (package sp):

Grid parameters:
     cellcentre.offset cellsize cells.dim
Var2          832051.5     5000        10
Var1         2130483.6     5000        10

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

The resulting object is an object of class `SpatialPixelsDataFrame`, with one column containing the number of points in the pixels of the map.

```
> image(asc)
```

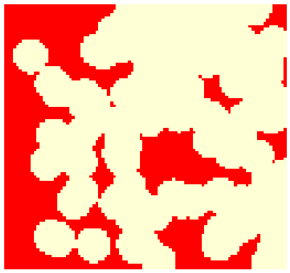
5 Computing buffer regions

`adehabitatMA` provides a function allowing the computation of buffer region. Buffer regions can be computed from points, lines or polygons. For example, consider the presence indices of the lynx of type “sighting” (labelled ”O” in the data)

```
> po <- locs[locs[["Type"]] == "O", ]
```

We may want to identify all areas located within 3000 m from a lynx presence index:

```
> image(buffer(po, map, 3000))
```

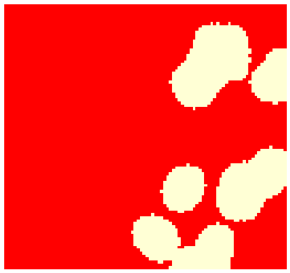


Now consider again the vector map of forested areas that we built in the section 3.3 (map `con`):

```
> plot(con)
```

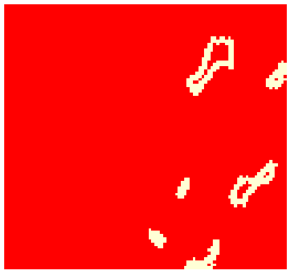
We may want to identify all the areas located within 3000 metres from one of these forested areas:

```
> image(buffer(con, map, 3000))
```



Alternatively, we may have wanted to identify the "ecotone" (boundary habitat) between the forest and the open areas... for example, to identify all the areas located within 500 metres from the boundary:

```
> sl <- as(con, "SpatialLines")
> image(buffer(sl, map, 500))
```



6 Conclusion

I included in the package `adehabitatMA` several functions adding to the existing set of tools available in R to perform spatial analyses. Originally, these

functions were included in the package `adehabitat` to allow the study of very specific ecological issues related to habitat selection, but as most users who sent me questions/suggestions concerning these functions were using them to study issues in other fields (e.g. biogeography, landscape ecology, etc.), I decided to include them in a separate package much more compliant with existing packages for spatial analysis within R.

However, readers interested in the study of animal space use and habitat selection should also have a look at the brother packages. All the packages `adehabitat**` contain a vignette similar to this one, which explains not only the functions, but also in some cases the philosophy underlying the analysis of animal space use.

References

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.

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