Landscape Ecology in R

Managing spatial data, calculating landscape metrics and simulating corridors

by Felipe Martello

Spatial Ecology and Conservation lab – UNESP, Rio Claro – SP, Brazil

Reviewed by
Marina Corrêa Côrtes

June 2016

This booklet was designed as part of the Genetics Landscape discipline, taught by Professor Dr. Marina Corrêa Côrtes in the Ecology and Biodiversity Postgraduate program in University of the State of São Paulo, Rio Claro - Brazil.

With the exception of "gdal_polygonizeR" function, all others functions to be loaded in this booklet, were developed by Felipe Martello.

Sumário

1. Managing Spatial data in R

1.1. Importing spatial data

1.1.1. Shape files

We'll import the shape file using the function "readOGR" from the "rgdal" package

```
library(rgdal)
v01_shp<-readOGR(".","buf500_v01")

## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "buf500_v01"
## with 25 features
## It has 10 fields
```

Let's plot it!

```
plot(v01_shp)
```

First, we should look at the attribute table and the coordinate reference system (CRS)

```
 head(v01_shp@data) #attribute table
```

```
##   Id OBJECTID   XMIN   XMAX    YMIN    YMAX classes       class_nome
## 0  0        4 412800 413150 7662850 7663200      10             agua
## 1  0        4 412800 413150 7662850 7663200      10             agua
## 2  0        4 412800 413150 7662850 7663200      10             agua
## 3  0        4 412800 413150 7662850 7663200      10             agua
## 4  0       26 416300 416650 7661450 7661800      11 pasto_abandonado
## 5  0       26 416300 416650 7661450 7661800      11 pasto_abandonado
##   pto_amos   area_hec
## 0      V01 0.01412432
## 1      V01 1.15868100
## 2      V01 0.10196738
## 3      V01 0.06388624
## 4      V01 0.26345308
## 5      V01 0.67074886
```

```
 v01_shp@proj4string #SRC information
```

```
## CRS arguments:
##  +proj=utm +zone=22 +south +datum=WGS84 +units=m +no_defs
## +ellps=WGS84 +towgs84=0,0,0
```

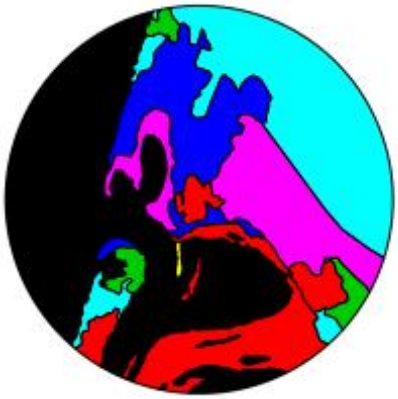
Be careful with the function that you use to read the shapefile: some of them do not recognize the projection. For example function "readShapePoly" from "maptools" package:

```
library("maptools")

    v01_shp2<-readShapePoly("buf500_v01.shp")
    v01_shp2@proj4string
```

```
## CRS arguments: NA
```

Now, let's colors to the shape using the column "class_nome" from the attribute table:

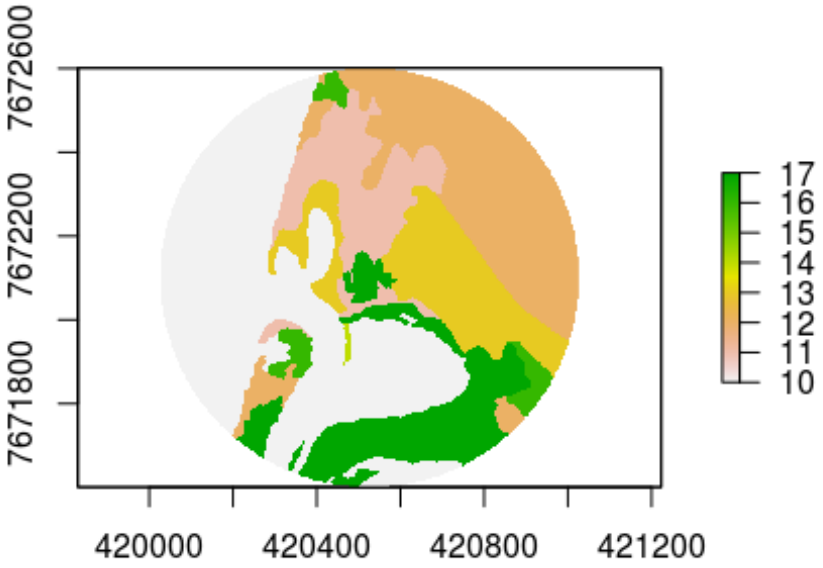
```
 plot(v01_shp,col=v01_shp@data$class_nome)
```



1.2. Raster

To read a raster we will use the "raster" function from the "raster" package:

```
library(raster)
v01_r.file<-raster("v01_1m.tif")
plot(v01_r.file)
```



We can extract the pixels values placing the "[]" after the raster name.To get an informative summary of the raster, we can use the table command to look at the number of pixels assigned to each value (which can be classes).

```
table(v01_r.file[])
```

```
##
##     10     11     12     13     14     16     17
## 304632  85572 174531  98767   1232  20350 100132
```

1.2. Converting spatial data

1.2.1. From shape to raster

To convert a shape file in a raster we will use the "rasterize" function from the "raster" package. First we need to set the extent (xy limits) and the resolution of the raster. Let's start by setting the raster with the same extension of the shape file .

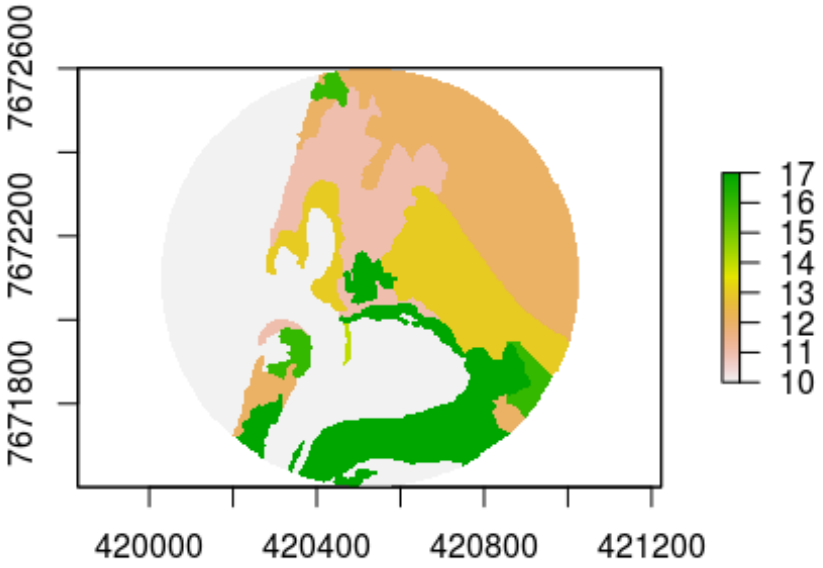
```
r<-raster()
r<-raster(extent(v01_shp))
```

I know that the resolution of the v01_shp shapefile is 1 m. To optimize data processing, however, we will use a 5 m resolution instead. Note that you don't need to set the unit of the raster - it will use the same unit of the shapefile

```
res(r)<-5
```

Finally, we can use the "rasterize" function. You have to indicate in the "field" argument the column of the attribute table that should be used to convert the shape.

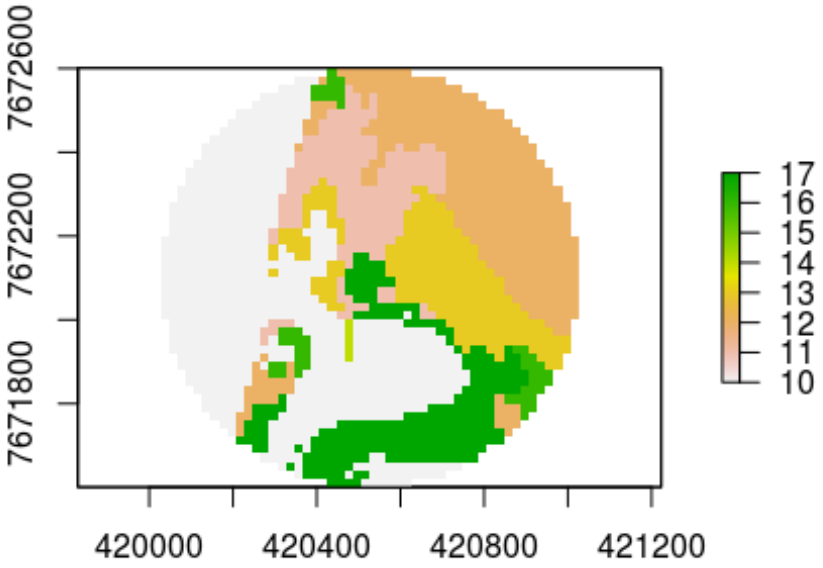
```
v01_shp_rst_5m<-rasterize(v01_shp,r,field="classes")
plot(v01_shp_rst_5m)
```



Note that once we changed the resolution, we can almost see the size of each pixel.

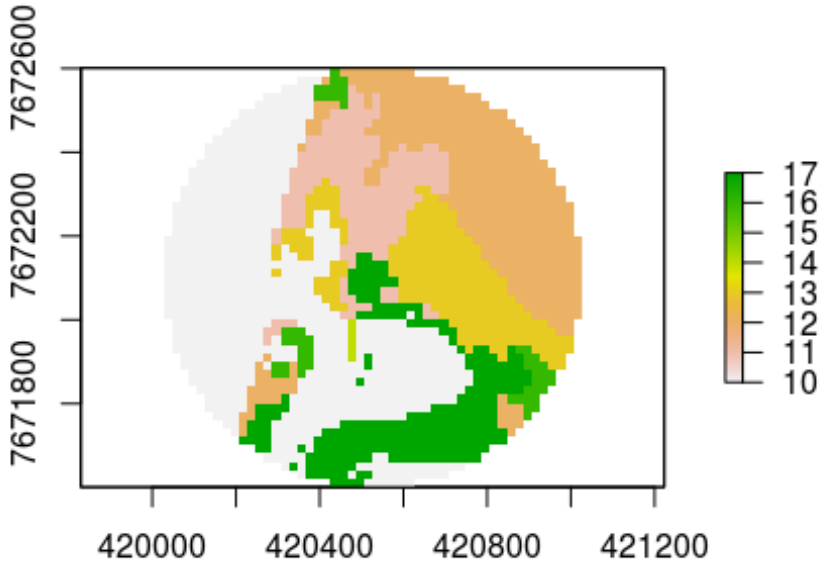
Let's increase the pixel size a little more, decreasing the resolution.

```
res(r)<-20
v01_shp_rst_20m<-rasterize(v01_shp,r,field="classes")
plot(v01_shp_rst_20m)
```



Be careful with the function argument (fun) that you use to assign the value of the pixel inside the "rasterize" function. The default is "last", but we can change it, let's say to "maximum" (with the same resolution - 20m):

```
v01_shp_rst_20m_max<-rasterize(v01_shp,r,field="classes",fun=max)
plot(v01_shp_rst_20m_max)
```



Note that it changes the values of some pixels.

1.2.2. From raster to poligon

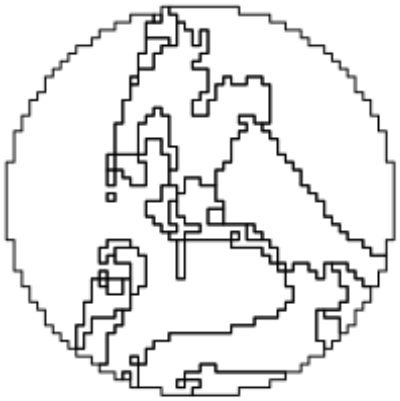
To do this conversion we will use a customized function that we need to load.

```
#gdal_polygonizeR
#function to convert raster in shape
 gdal_polygonizeR <- function(x, outshape=NULL, gdalformat = 'ESRI Shapefile', pypath=NULL,
readpoly=TRUE, quiet=TRUE) {
   if (isTRUE(readpoly)) require(rgdal)
   if (is.null(pypath)) {
     pypath <- Sys.which('gdal_polygonize.py')
   }
   if (!file.exists(pypath)) stop("Can't find gdal_polygonize.py on your system.")
   owd <- getwd()
   on.exit(setwd(owd))
   setwd(dirname(pypath))
   if (!is.null(outshape)) {
      outshape <- sub('\\.shp$', '', outshape)
     f.exists <- file.exists(paste(outshape, c('shp', 'shx', 'dbf'), sep='.'))
     if (any(f.exists))
       stop(sprintf('File already exists: %s',
                    toString(paste(outshape, c('shp', 'shx', 'dbf'),
                                   sep='.')[f.exists])), call.=FALSE)
```

```
  } else outshape <- tempfile()
  if (is(x, 'Raster')) {
    require(raster)
    writeRaster(x, {f <- tempfile(fileext='.tif')})
    rastpath <- normalizePath(f)
  } else if (is.character(x)) {
    rastpath <- normalizePath(x)
  } else stop('x must be a file path (character string), or a Raster object.')
  system2('python', args=(sprintf('"%1$s" "%2$s" -f "%3$s" "%4$s.shp"',
                                  pypath, rastpath, gdalformat, outshape)))
  if (isTRUE(readpoly)) {
    shp <- readOGR(dirname(outshape), layer = basename(outshape), verbose=!quiet)
    return(shp)
  }
  return(NULL)
}
```

Now we can use it.

```
v01_shp_20m<-gdal_polygonizeR(v01_shp_rst_20m)
plot(v01_shp_20m)
```



Let's see the attribute table.

```
head(v01_shp_20m@data)
```

```
##   DN
## 0 12
## 1 16
## 2 12
## 3 12
## 4 13
## 5 11
```

The only column in this table is the value of each polygon.

1.3. Exporting spatial data

1.3.1 Raster

To export a raster we'll use the "writeRaster" function (from the “raster” package).You can export it in several formats. Here will use the "tif" format:

```
writeRaster(v01_shp_rst_5m,"v01_5m.tif")
```

1.3.2 Shape file

To export a shape file we will use the "writeOGR"function (“rgdal”).

```
writeOGR(v01_shp_20m,"v01_shp_20m","v01_shp_20m",driver="ESRI Shapefile")
```

If we go to our output folder, you can see that this function created a new folder wherein it exported 4 files.

1.4. Changing resolution and projection

1.4.1 Resolution

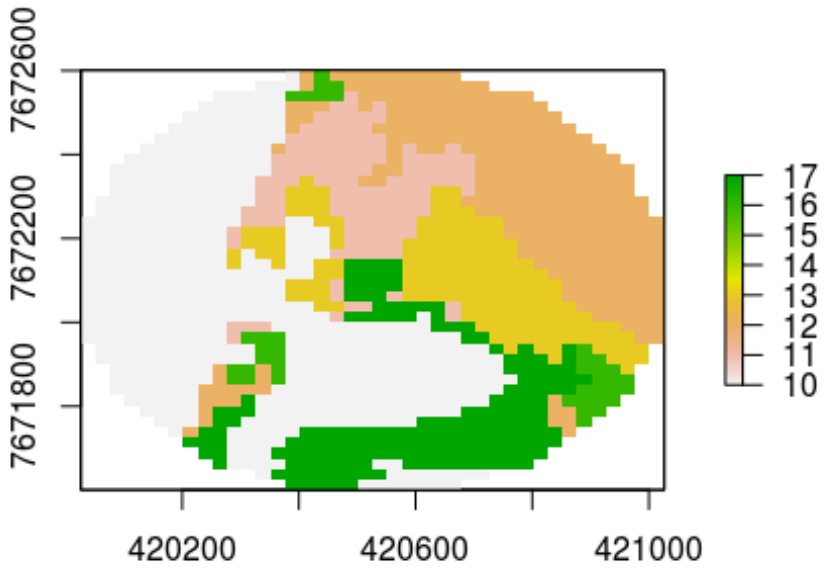
To change the resolution of a raster we will use the "resample" function. First, we need to create an empty raster with the extent and resolution of our desirable resolution. Here we will

create this raster and use the same extent of the original raster. After that, we will set 25 m as resolution.

```
v01_25m<-raster()
extent(v01_25m)<-extent(v01_shp_rst_5m)
res(v01_25m)<-25
```

Now we can use the "resample" function.

```
v01_25m<-resample(v01_shp_rst_5m,v01_25m,na.rm=T,method='ngb')
  plot(v01_25m)
```

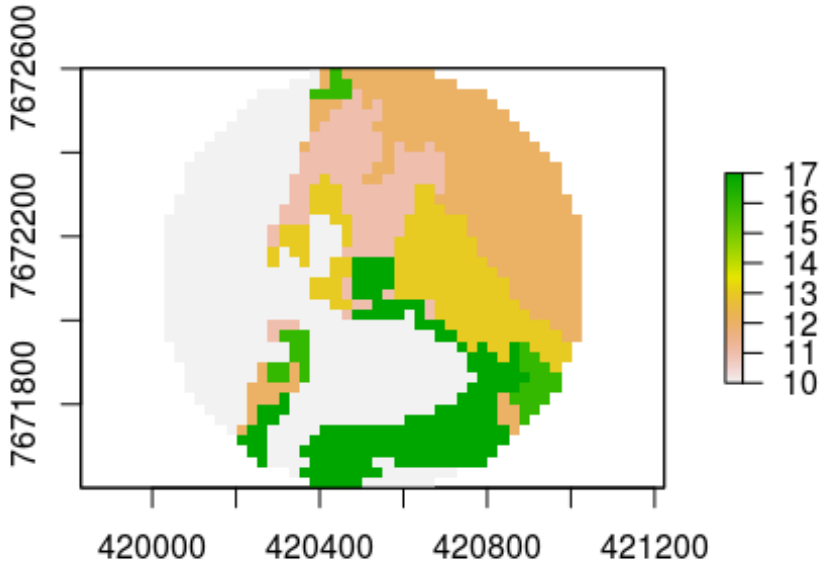


I don't know why, but it seems that these commands changed the projection! So let's put it in the same projection of the original raster (v01_shp_rst_5m).

```
crs(v01_shp_rst_5m)
```

```
## CRS arguments:
##  +proj=utm +zone=22 +south +datum=WGS84 +units=m +no_defs
## +ellps=WGS84 +towgs84=0,0,0
```

```
crs(v01_25m)

## CRS arguments:
##  +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0

crs(v01_25m)<-crs(v01_shp_rst_5m)
plot(v01_25m)
```



1.4.2 Projection

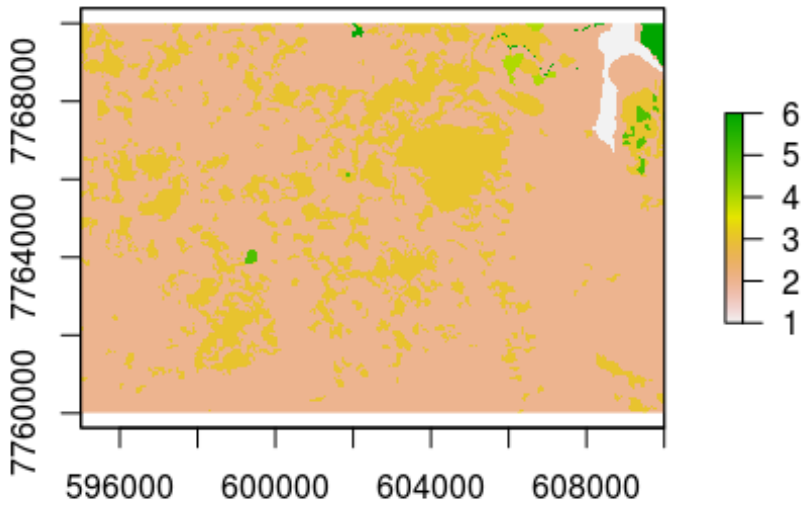
To see and change the projection we will use the function "crs" (“raster”).

```
crs(v01_shp_rst_5m)

## CRS arguments:
##  +proj=utm +zone=22 +south +datum=WGS84 +units=m +no_defs
## +ellps=WGS84 +towgs84=0,0,0

crs(v01_shp_rst_5m)<-"+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
crs(v01_shp_rst_5m)

## CRS arguments:
##  +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

1.5.Inserting points by dataframe

Now we will work with another raster.

```
r_cati<-raster("catimbo_20m.tif")#importing raster
  plot(r_cati)
```

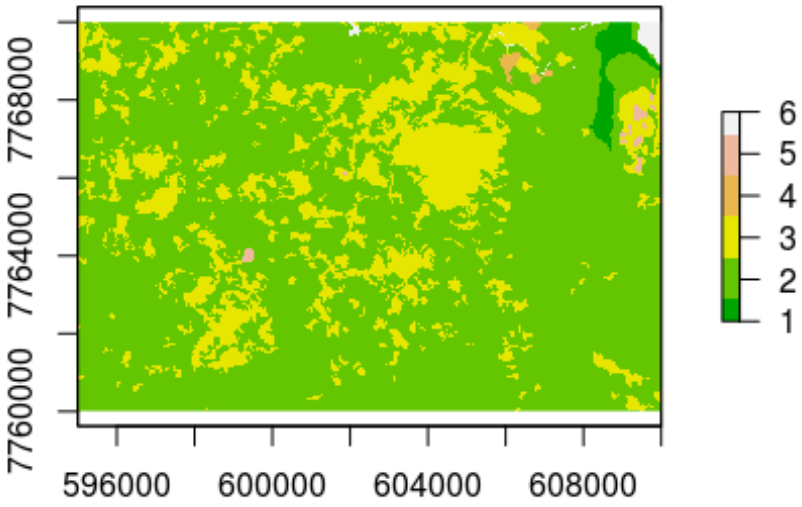



```
  table(r_cati[])
```

```
##
##      1      2      3      4      5      6
##   1880 129629  33406    409    452    724
```

We can change the color of the raster indicating the number of class that we have.

```
plot(r_cati,col = terrain.colors(6))
```



Let's import an attribute table with the spatial coordinatesusing the function "read.dbf"

(“foreign”).

```
library(foreign)
ptos.dbf<-read.dbf("ptos_cati.dbf")
ptos.dbf
```

```
##   id      x       y
## 1  1 598177 7762577
## 2  2 601283 7762630
## 3  3 603044 7763246
## 4  4 596909 7766432
## 5  5 603167 7767919
```

Before I plot my points, I'll convert them to a "SpatialPointDataFrame" object using the

"coordinates" function (“sp”) and setting the projection.

```
ptos<-ptos.dbf
coordinates(ptos)<-c("x","y")
class(ptos)
```

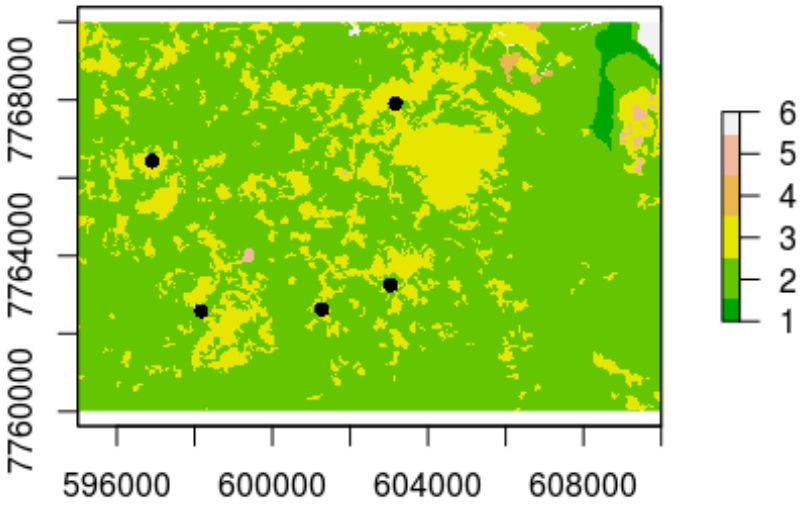
```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
crs(ptos)<-crs(r_cati)
summary(ptos)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##       min     max
## x  596909  603167
## y 7762577 7767919
## Is projected: TRUE
## proj4string :
## [+proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
## +ellps=WGS84 +towgs84=0,0,0]
## Number of points: 5
## Data attributes:
##        id
##  Min.   :1
##  1st Qu.:2
##  Median :3
##  Mean   :3
##  3rd Qu.:4
##  Max.   :5
```

Now, let's plot it together with the raster.

```
plot(r_cati,col = terrain.colors(6))
plot(ptos,pch=16, col="black", add=TRUE,labels=T)
```



Notice that I used the "add=TRUE" parameter to keep the r_cati raster.

1.6. Creating one buffer for each point

Buffers have been used to understand scale effects or just to simplify some steps of the analyses. Here I create the function "buf.by.ptos", which cuts a raster (argument r) centered in each point within a SpatialPointDataFrame (argument ptos) given the radius of the buffer (argument d). This function require the "rgeos" package.

```
buf.by.ptos<-function(ptos,r,d){
    library(rgeos)
    r_cati<-r
    ptos.data<-data.frame(ptos@coords)
    cat_buffer<-list()

    for (i in 1:length(ptos)){
      temp_ptos<-ptos.data[i,]
      coordinates(temp_ptos)<-c("x","y")
      pbuffer<-gBuffer(temp_ptos,widt=d)
```

```
      temp_buf<-mask(r_cati,pbuffer)      cat_buffer[[i]]<-temp_buf
      buff.ptos<-stack(cat_buffer)
    }
    buff.ptos<-stack(cat_buffer)
    return(buff.ptos)
  }
```

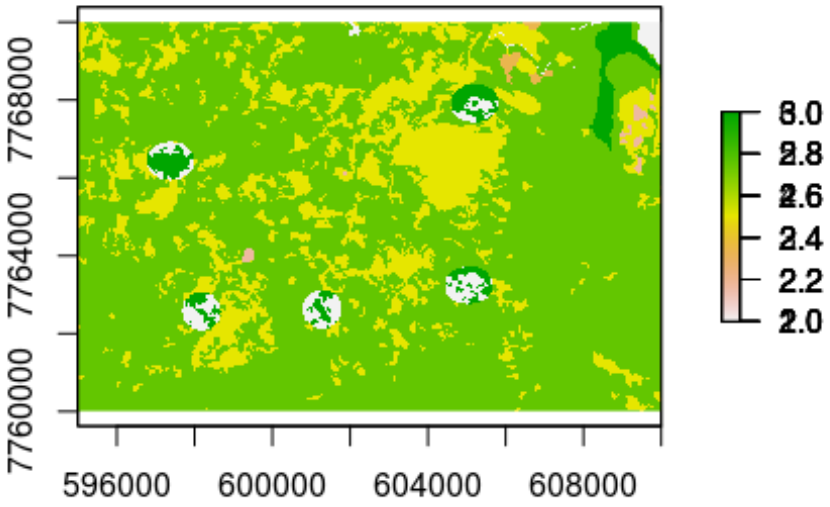
Let's try it.

```
buff_cat<-buf.by.ptos(ptos,r_cati,500)
class(buff_cat)
```

```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

We can see that the output is a "stack" object, which is a set of rasters with the same extent and resolution of the original r_cati raster. Each element of this object is one raster, that here we plot together with the original raster file.

```
plot(r_cati,col = terrain.colors(6))
  plot(buff_cat[[1]],add=T)
  plot(buff_cat[[2]],add=T)
  plot(buff_cat[[3]],add=T)
  plot(buff_cat[[4]],add=T)
  plot(buff_cat[[5]],add=T)
```



1.7. Dealing with raster values

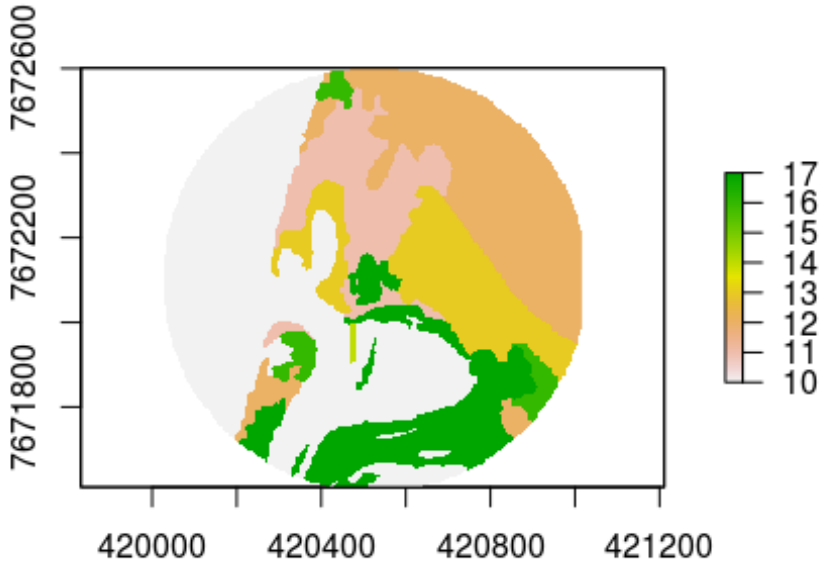
1.7.1 Histograms

We can now look at the frequencies of pixel values in a raster by making histograms.Let's start by importing three rasters.

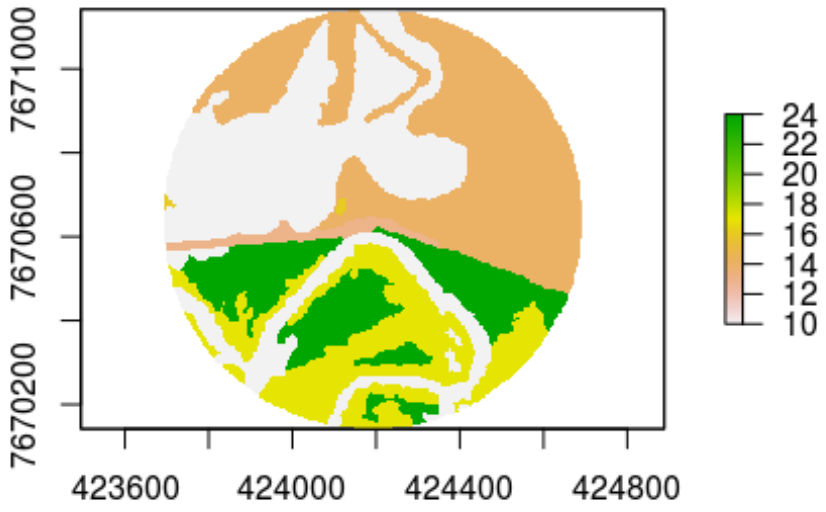
```
v01<-raster("v01_buf500_res5m.tif")
v02<-raster("v02_buf500_res5m.tif")
v03<-raster("v03_buf500_res5m.tif")
```

Now let's plot it.

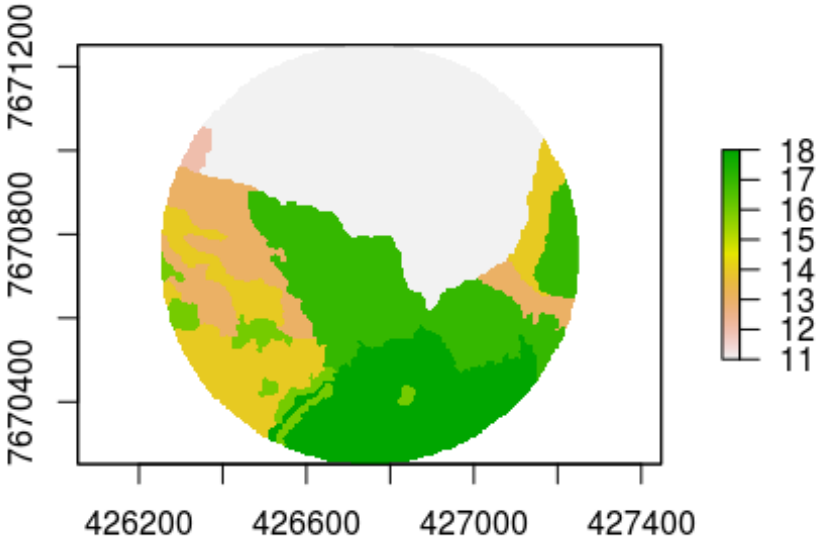
```
plot(v01)
```




```
plot(v02)
```

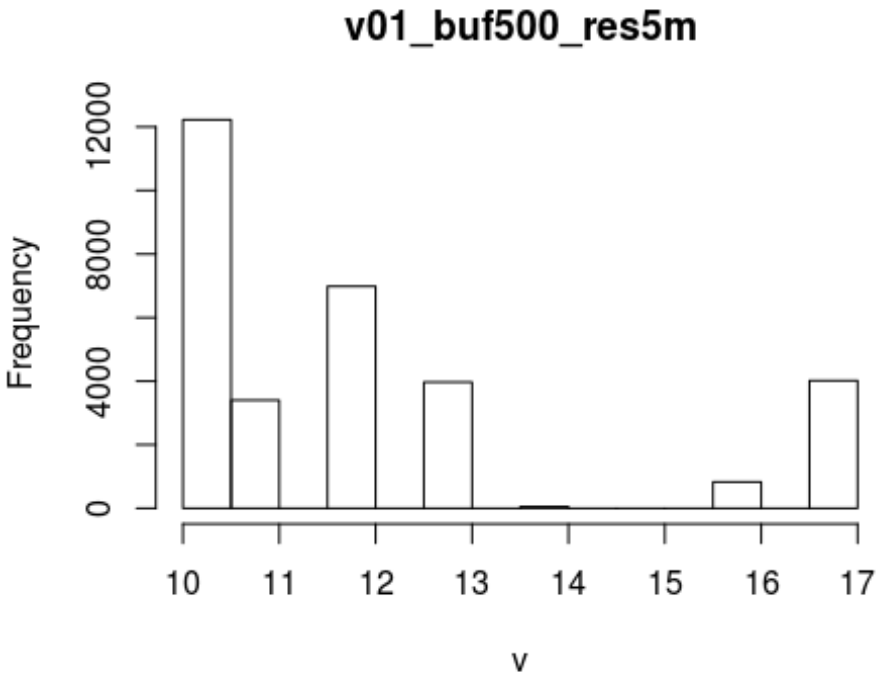



```
plot(v03)
```



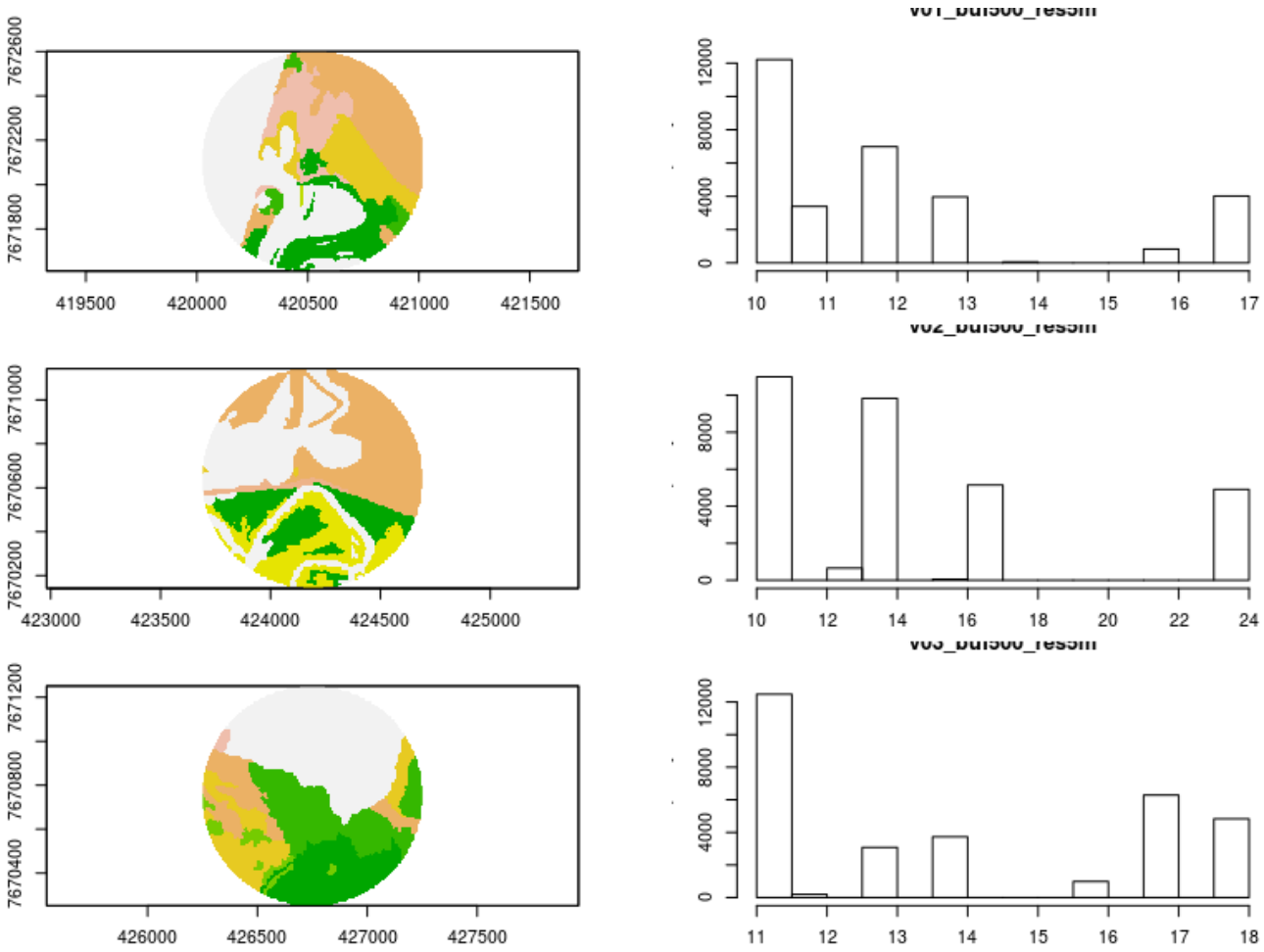
We can see the histogram of a raster using the "hist" function.

```
hist(v01)
```



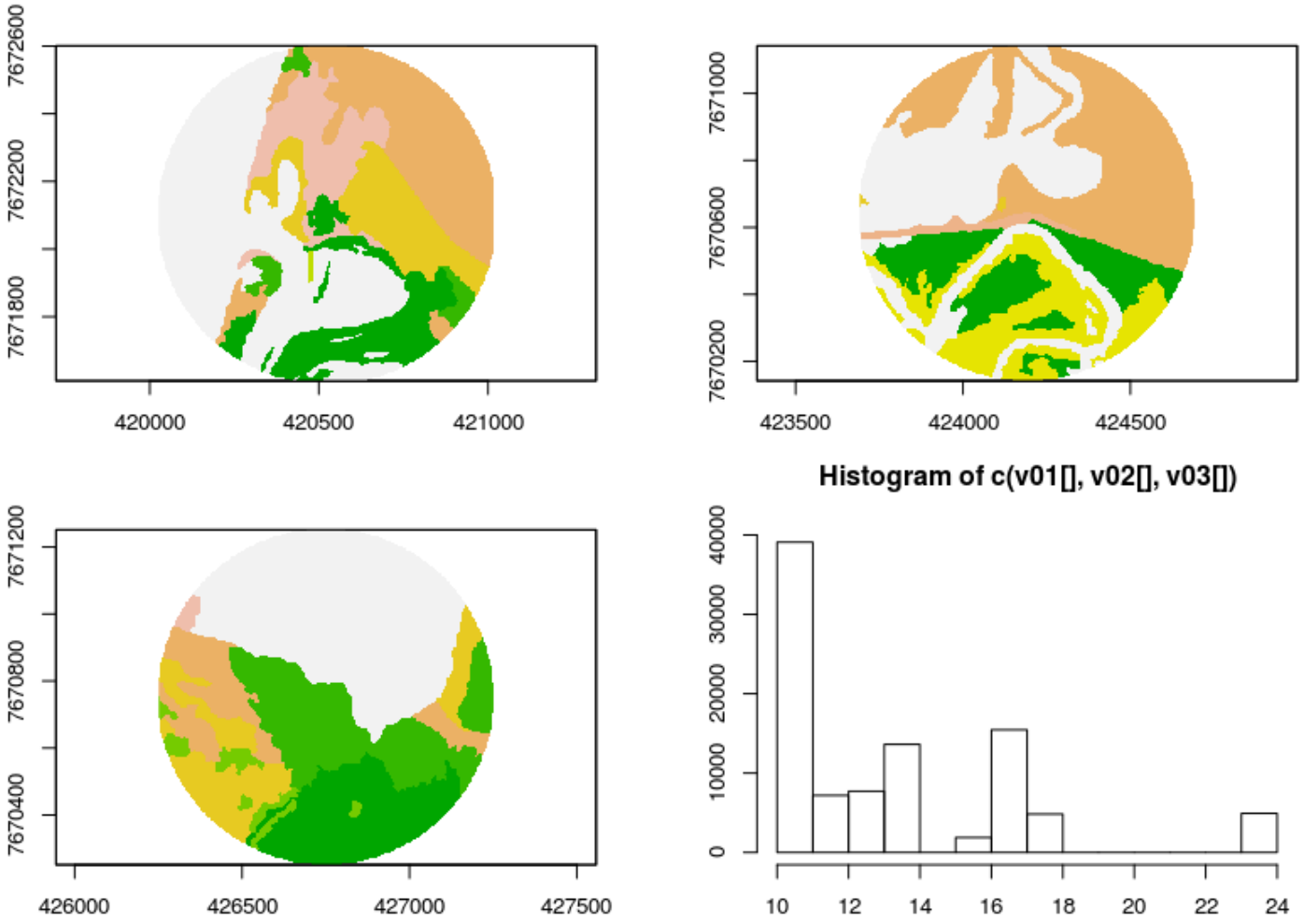
We can put all plots together using "par" function.

```
par(mfrow=c(3,2))
  plot(v01)
  hist(v01)
  plot(v02)
  hist(v02)
  plot(v03)
  hist(v03)
```



Now with the histogram of all rasters together

```
par(mfrow=c(2,2))
  plot(v01)
  plot(v02)
  plot(v03)
  hist(c(v01[],v02[],v03[]))
```



1.7.2. Changing pixels values

We can change the values of pixels as we do in any matrix or data frame. I know that both classes 16 and 17 are forest in the raster "v02", so we will merge them into a single class, by changing the values of 17 to 16.

```
v02.j<-v02
v02.j[v02.j==17]<-16 #class 17->16
```

Before we plot the new raster we will compare the frequencies of pixel values between the original and new raster to confirm that the class 17 disappeared.

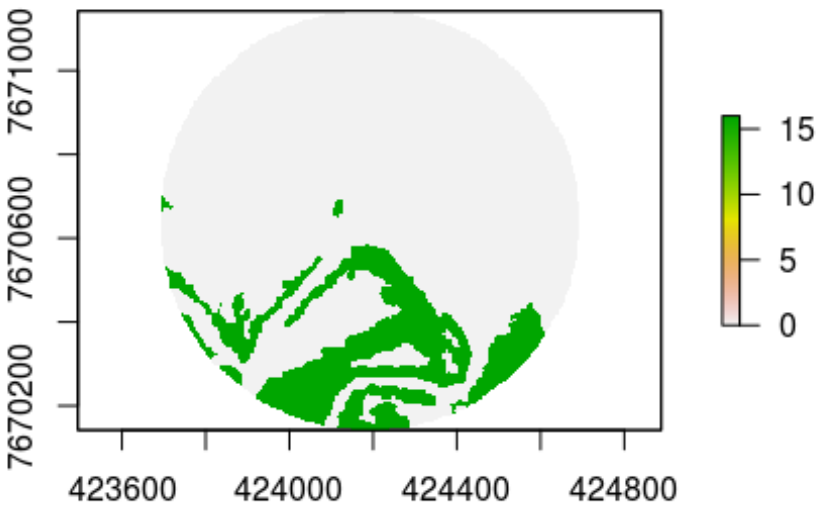
```
table(v02[])

##
##    10    13    14    16    17    24
## 10997   654  9830    56  5147  4910

table(v02.j[])

##
##    10    13    14    16    24
## 10997   654  9830  5203  4910
```

1.7.3 Building binaries rasters

Using the same idea, we can also build a raster with just one class. This kind of raster is used to calculate some metrics, as well as to create other maps, for instance maps of Euclidian distance from some class (we will see this latter).

We will use the raster “v02.j” that we had just created. This time we want a new raster containing only the class 16.

```
v02_cl16<-v02.j
v02_cl16[v02_cl16!=16]<-0#  values!=16<-0
plot(v02_cl16)
```


7671000
7670600
7670200
423600
424000
424400
424800
15
10
5
0

2. Landscape metrics

We will use the three rasters that we imported previously ("v01","v02","v03")

2.1. Class area

We already know how to get the frequency of pixels values of each raster. Now we will create an object that contains the frequencies of pixel values for all three rasters, so that the rows indicate the raster name and columns indicate the classes used to represent the rasters..

First we will create a table with the frequency of pixels for each raster. We will also include the name of each raster in each table.

```
   v01_tab<-cbind.data.frame("v01",table(v01[]))
    colnames(v01_tab)<-c("land","class","freq")
    v02_tab<-cbind.data.frame("v02",table(v02[]))
    colnames(v02_tab)<-c("land","class","freq")
    v03_tab<-cbind.data.frame("v03",table(v03[]))
    colnames(v03_tab)<-c("land","class","freq")
```

Now let's join these three objects in one.

```
(land_class<-rbind(v01_tab,v02_tab,v03_tab))
```

```
##    land class  freq
## 1   v01    10 12223
## 2   v01    11  3400
## 3   v01    12  6985
## 4   v01    13  3969
## 5   v01    14    55
## 6   v01    16   830
## 7   v01    17  4019
## 8   v02    10 10997
## 9   v02    13   654
```

```
## 10  v02    14  9830
## 11  v02    16    56
## 12  v02    17  5147
## 13  v02    24  4910
## 14  v03    11 12484
## 15  v03    12   200
## 16  v03    13  3079
## 17  v03    14  3726
## 18  v03    16   986
## 19  v03    17  6287
## 20  v03    18  4832
```

As I said before, to build our desirable table, the column "land" must become rows containing the names of the rasters, the column "class" must be separated in a way that each class will be one column, and finally the column "freq" will be the number of pixels for each class inside our table. To do this "magic" we will use the function "dcast" (“reshape2”).

```
library(reshape2)
land_class<-dcast(land_class,land~class,value.var="freq")
land_class

##   land    10    11   12   13   14  16   17   24   18
## 1  v01 12223  3400 6985 3969   55 830 4019   NA   NA
## 2  v02 10997    NA   NA  654 9830  56 5147 4910   NA
## 3  v03    NA 12484  200 3079 3726 986 6287   NA 4832
```

We have to change the values NA to 0.

```
land_class[is.na(land_class)]<-0
land_class

##   land    10    11   12   13   14  16   17   24   18
## 1  v01 12223  3400 6985 3969   55 830 4019    0    0
## 2  v02 10997     0    0  654 9830  56 5147 4910    0
## 3  v03     0 12484  200 3079 3726 986 6287    0 4832
```

We will now put the first column (land) as rownames.

```
rownames(land_class)<-land_class[,1]
land_class<-land_class[,-1]
```

The values in our table represent the number of pixels of each class. We can change it to area (m^2) if we know the resolution and if the projection is in UTM (which I always recommend). I

know that the resolution of these rasters is 5m, so if we multiply the number of pixels by the area of each pixel (5x5) we have the area in m^2.

```
land_class<-land_class*(5*5)
land_class
```

```
##         10     11     12    13     14    16     17     24     18
## v01 305575  85000 174625 99225   1375 20750 100475      0      0
## v02 274925      0      0 16350 245750  1400 128675 122750      0
## v03      0 312100   5000 76975  93150 24650 157175      0 120800
```

Now if we divide these values by 10000, we have the area in hectare.

```
land_class.hec<-land_class/10000
land_class.hec
```

```
##          10    11      12     13      14    16      17     24    18
## v01 30.5575  8.50 17.4625 9.9225  0.1375 2.075 10.0475  0.000  0.00
## v02 27.4925  0.00  0.0000 1.6350 24.5750 0.140 12.8675 12.275  0.00
## v03  0.0000 31.21  0.5000 7.6975  9.3150 2.465 15.7175  0.000 12.08
```

Let's put it in a new table that we'll use for all metrics.

```
land_metric<-land_class.hec
```

We can also calculate the total area of each landscape by summing all values in each row.

```
land_metric$total_area<-rowSums(land_class)
land_metric
```

```
##          10    11      12     13      14    16      17     24    18
## v01 30.5575  8.50 17.4625 9.9225  0.1375 2.075 10.0475  0.000  0.00
## v02 27.4925  0.00  0.0000 1.6350 24.5750 0.140 12.8675 12.275  0.00
## v03  0.0000 31.21  0.5000 7.6975  9.3150 2.465 15.7175  0.000 12.08
##     total_area
## v01     787025
## v02     789850
## v03     789850
```

We can notice that the total area of raster “v01” is a bit smaller than the others raster, probably due to an error in the mapping process (we don’t need to worry about it here).

2.2. Landscape diversity

2.2.1 Richness of class

To calculate the number of classes (richness)in each landscape we just have to turn all values in the "land_class" table in 0 or 1 (expressing the presence or absence of each class) and sum the values of each row. Then we can put it in our metric table.

```
  bin_land_class<-land_class
  bin_land_class[bin_land_class>0]<-1
  bin_land_class
```

```
##     10 11 12 13 14 16 17 24 18
## v01  1  1  1  1  1  1  1  0  0
## v02  1  0  0  1  1  1  1  1  0
## v03  0  1  1  1  1  1  1  0  1
```

```
  (class_rich<-rowSums(bin_land_class))
```

```
## v01 v02 v03
##   7   6   7
```

```
  land_metric$rich<-class_rich
```

2.2.2 Shannon index

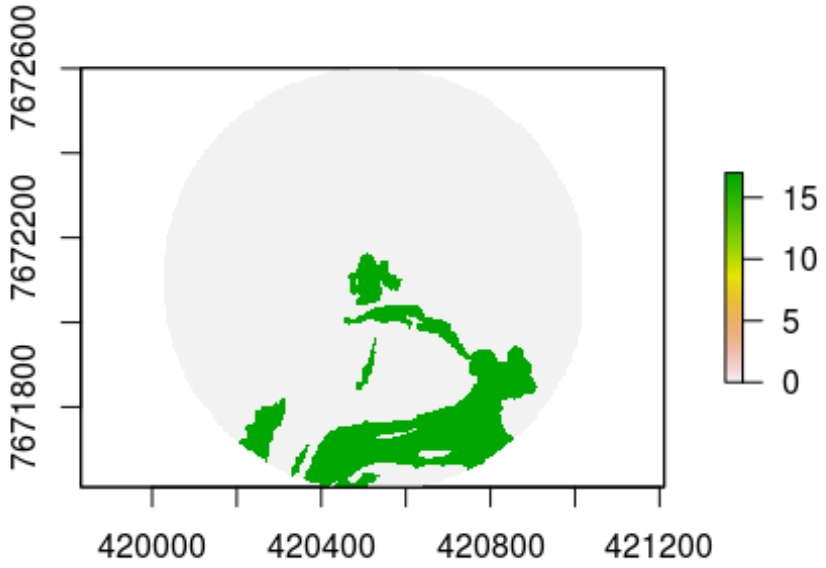
To calculate the Shannon index, we just have to use the function "diversity" (vegan) in our "land_class" table.

```
library(vegan)

## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.3-5

land_shan<-diversity(land_class,"shannon")
land_shan

##       v01       v02       v03
## 1.572575 1.407026 1.594601

land_metric$shan<-land_shan
```

2.3 Edge and core area

We will calculate the core and edge area of the class 17 in each landscape. For that, first we need to create a raster containing this one class for each landscape.

```
v01_cl17<-v01
v01_cl17[v01_cl17!=17]<-0
plot(v01_cl17)
```



Then we will use the function "edge.core" to calculate these metrics. We have to load this function.

This function has two arguments: "x", which is the raster with just one value different from 0 (here the class 17), and "d": the distance of influence of the edge effect inside the patch. The "d" distance is in pixel, so you have to know the distance of your raster.

```
edge.core<-function(x,d){
  temp<-unique(x[])
  temp<-temp[!is.na(temp)]
  temp<-temp[temp>0]
  if( length(temp)>1) warning("Raster needs to have just one value different from 0")
  x[x>0]<-1
  temp_x<-focal(x, w=matrix(1,d,d), fun=mean, na.rm=T, pad=T)
  temp_x<-temp_x*x
  temp_x[temp_x==1]<-2
  temp_x[temp_x>0&temp_x<2]<-1

  temp_tb<-data.frame(table(temp_x[]))
  temp_tb<-temp_tb[-1,]
```

```
  temp_tb$Environment<-c("edge","core")
  temp_tb<-temp_tb[,-1]
  temp_tb<-temp_tb[,c(2,1)]
  return(list(raster=temp_x,table=temp_tb))
  }
```

As I said before, the resolution of these rasters is 5 m, so if the distance of edge influence is 25m, we need to divide by the resolution. Thus our "d" is 5.

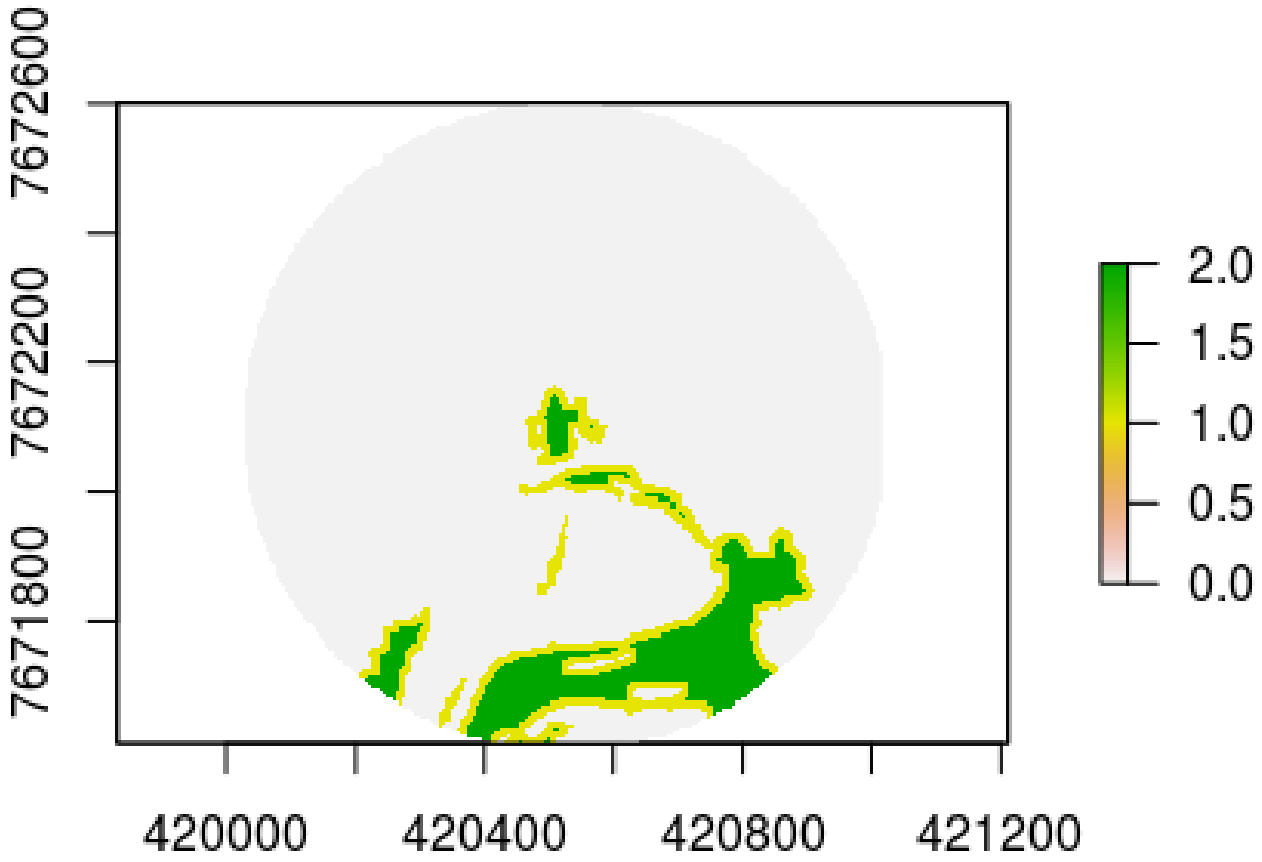
```
edg_v01_cl17<-edge.core(v01_cl17,5) #edge.core = load function
class(edg_v01_cl17)#output is a list with: 1)raster ; 2)table
```

```
## [1] "list"
```

The output of this function is a list with two elements.

The first element is a raster wherein the core has value=2 and the edge value=1. The second element is a table with the frequency of both core and edge pixels.

```
plot(edg_v01_cl17[[1]])
```




```
edg_v01_cl17[[2]]
```

```
##   Environment Freq
## 2        edge 1778
## 3        core 2241
```

Let's do it for the others landscapes.

```
 v02_cl17<-v02
  v02_cl17[v02_cl17!=17]<-0
  edg_v02_cl17<-edge.core(v02_cl17,5)

  v03_cl17<-v03
  v03_cl17[v03_cl17!=17]<-0
  edg_v03_cl17<-edge.core(v03_cl17,5)
```

Now let's create a table with all values for all landscapes and rearrange it as we did in session 2.1 with the "dcast" function.

```
edge_tab<-rbind(edg_v01_cl17[[2]],edg_v02_cl17[[2]],edg_v03_cl17[[2]])
edge_tab$Land<-c("v01","v01","v02","v02","v03","v03")
edge_tab
```

```
##    Environment Freq Land
## 2         edge 1778  v01
## 3         core 2241  v01
## 21        edge 2641  v02
## 31        core 2506  v02
## 22        edge 1420  v03
## 32        core 4867  v03
```

```
edge_tab<-dcast(edge_tab,Land~Environment,value.var="Freq")
edge_tab
```

```
##   Land core edge
## 1  v01 2241 1778
## 2  v02 2506 2641
## 3  v03 4867 1420
```

Once again we will put the first column as rownames.

Then we will convert pixels frequency to hectare and add this information in our metric table.

```
rownames(edge_tab)<-edge_tab[,1]
  edge_tab<-edge_tab[,-1]
  edge_tab
```

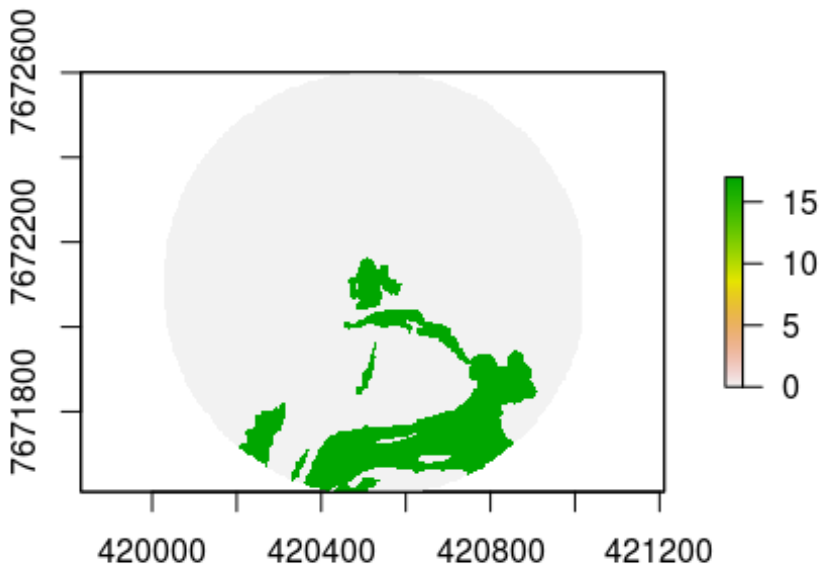
```
##     core edge
## v01 2241 1778
## v02 2506 2641
## v03 4867 1420
```

```
  edge_tab<-(edge_tab*(5*5))/10000 #converting in hectare

  land_metric$core_cl17<-edge_tab[,1]
  land_metric$edge_cl17<-edge_tab[,2]
```

2.4 number of patches & patches size of a specific class

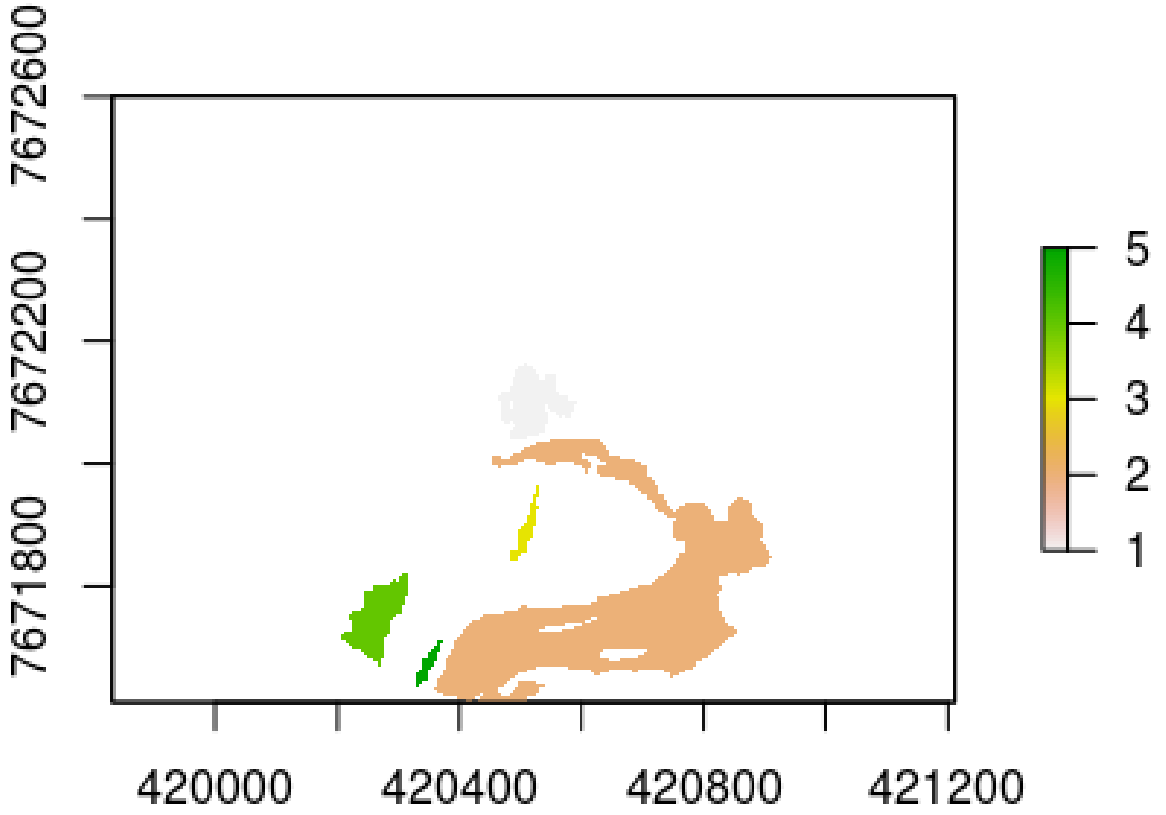
Now we will calculate the number and sizes of patches of a desirable class (in our case, we will stick with the class 17!).

```
plot(v01_cl17)
```



The function "clump" (“raster”) groups pixels by the patch it belongs. This function assigns one number for each patch.

```
clump_v01_cl17<-clump(v01_cl17) #classify pixels by patches
## Loading required namespace: igraph
  plot(clump_v01_cl17)
```



Let's put the pixels frequency in a table and convert to hectare.

```
frag.area_v01_cl17<-table(clump_v01_cl17[])
(frag.area_v01_cl17<-(frag.area_v01_cl17*(5*5))/10000)

##
##      1      2      3      4      5
## 0.8825 8.0950 0.1800 0.7975 0.0925
```

Now we'll do the same with the other landscapes.

```
clump_v02_cl17<-clump(v02_cl17)
frag.area_v02_cl17<-table(clump_v02_cl17[])
frag.area_v02_cl17<-(frag.area_v02_cl17*(5*5))/10000
clump_v03_cl17<-clump(v03_cl17)
frag.area_v03_cl17<-table(clump_v03_cl17[])
frag.area_v03_cl17<-(frag.area_v03_cl17*(5*5))/10000
```

The number of patches is just the length of this object. Let's put it in our metric table.

```
(n.patches_v01<-length(frag.area_v01_cl17))

## [1] 5

n.patches_v02<-length(frag.area_v02_cl17)
n.patches_v03<-length(frag.area_v03_cl17)
land_metric$n.patch<-c(n.patches_v01,n.patches_v02,n.patches_v03)
```

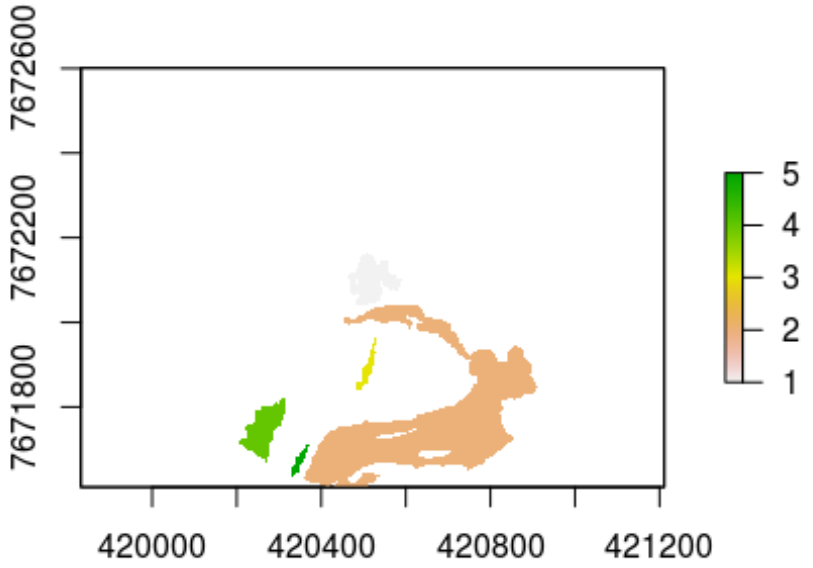
It's easy now to calculate the mean size of patches using the "mean" function.

```
(avg.patches_v01<-mean(frag.area_v01_cl17))

## [1] 2.0095

avg.patches_v02<-mean(frag.area_v02_cl17)
avg.patches_v03<-mean(frag.area_v03_cl17)
land_metric$avg.patch<-c(avg.patches_v01,avg.patches_v02,avg.patches_v03)
```

2.5. Minimum Distance between patches

To calculate the distance between patches of the same class we will use the raster with just one class (17), wherein the values are numeric codes that identify the patches.

```
plot(clump_v01_cl17)
```



Now we have to load the function "dist.patch" to calculate the distance between patches of the same class. For this function the only argument is a raster wherein the values represent the code of the patch that the pixel belongs (output of the "clump" function).

```
dist.patch<-function(r){

  str.conec.resu<-list()

  dist_lis<-list()
  for (i in 1:length(table(r[]))){
    r.1<-r
    r.1[r.1!=i]<-NA
    dis.r<-distance(r.1)
    dis.r[dis.r==0]<-99999999999
    dist_lis[[i]]<-dis.r}

  foo<-stack(dist_lis)
  dist.all<-min(foo)

  px_dist<-list()
  dist.rast<-list()
```

```
  for (i in 1:length(table(r[]))){
    r.2<-r
    r.2[r.2!=i]<-NA
    r.2[r.2>1]<-1
    r.t<-r.2*dist.all
    dist.rast[[i]]<-r.t
    px_dist[[i]]<-r.t[r.t>0]}

  foo2<-stack(dist.rast)
  dist.frag<-min(foo2,na.rm=T)

  str.conec.resu[[1]]<-dist.frag
  str.conec.resu[[2]]<-px_dist

  return(str.conec.resu)
}
```

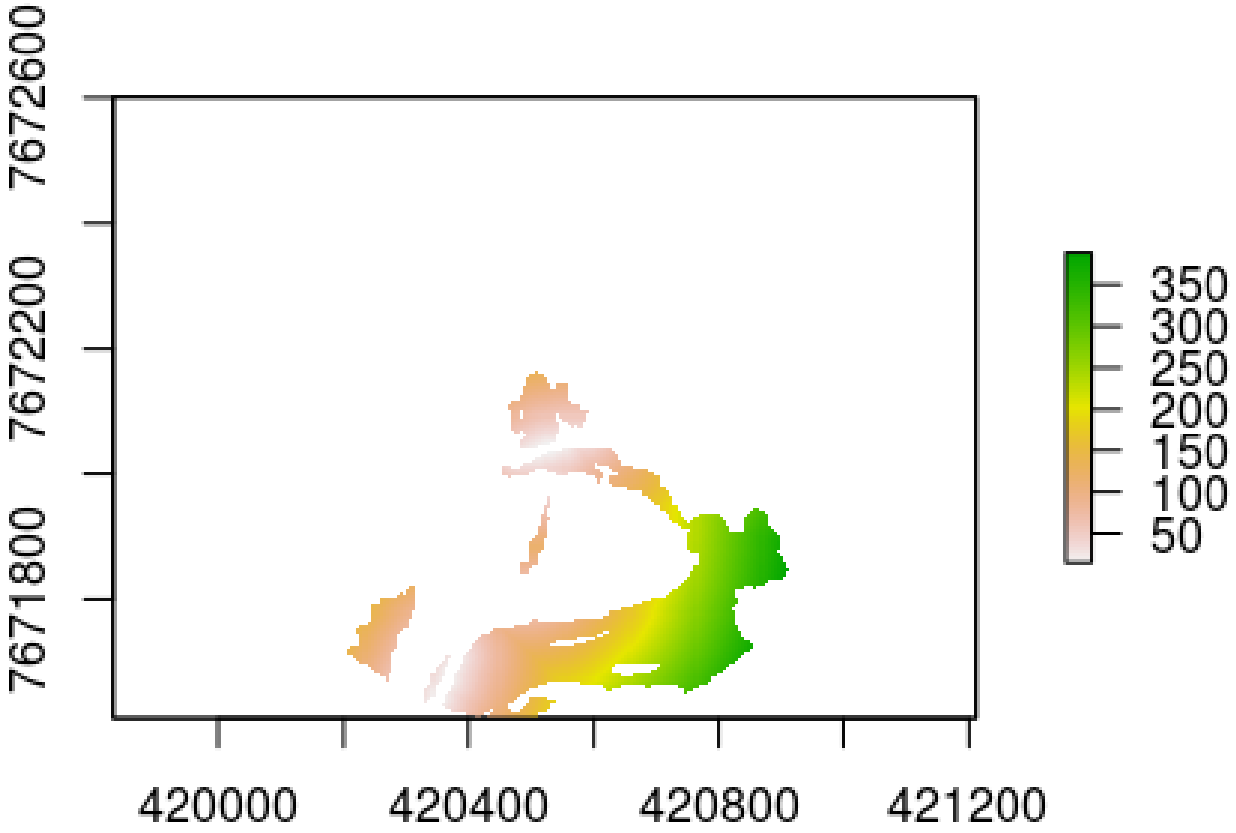
Let use it!

```
dist.patv01<-dist.patch(clump_v01_cl17)
class(dist.patv01)
```

```
## [1] "list"
```

The output is a list, wherein the first element is a raster indicating the value of each pixel given the distance to the nearest pixel of the same class belonging to another patch. The second element is another list, wherein the number of elements matches the number of patches. These elements are vectors wherein the values are distances of each pixel to the nearest pixel of another patch of the same class.

```
plot(dist.patv01[[1]])
```




```
class(dist.patv01[[2]])
```

```
## [1] "list"
```

With this last list we can calculate the mean, maximum and minimum distance for each patch, as well as, the variation within each patch and the landscape. Let's do it for the others landscapes.

```
dist.patv02<-dist.patch(clump_v02_cl17)
dist.patv03<-dist.patch(clump_v03_cl17)
```

Now we will calculate the mean distance between pixels of different patches belonging to class 17 in each landscape. Then we'll put it in our metric table.

```
(avg.dist_v01<-mean(unlist(dist.patv01[[2]])))
## [1] 173.7485
(avg.dist_v02<-mean(unlist(dist.patv02[[2]])))
## [1] 88.60735
(avg.dist_v03<-mean(unlist(dist.patv03[[2]])))
## [1] 348.3099
land_metric$avg.dis_cl17<-c(avg.dist_v01,avg.dist_v02,avg.dist_v03)
```

2.6. Functional Connectivity

Here we will calculate functional connectivity considering the dispersal ability of a specific species. The function that I created ("funct.conect") can also be used to calculate other metrics, such as the number of patches connected considering the dispersal ability of the organism.

The arguments of this function are the raster (x) with just on class different from 0 and the dispersal distance of the focal species (d).

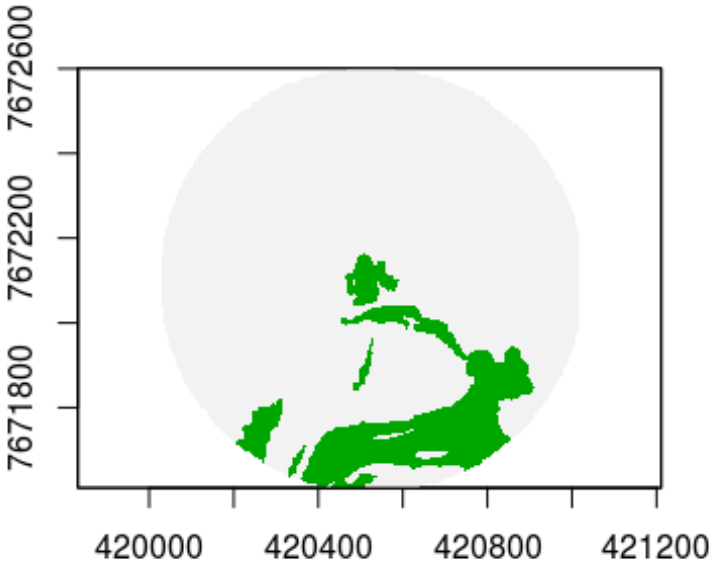
Let's load it!

```
funct.conect<-function(x,d){
  temp<-unique(x[])
  temp<-temp[!is.na(temp)]
  temp<-temp[temp>0]
  if( length(temp)>1) warning("Raster needs to have just one value different from 0")
  x[x>0]<-1
  temp_x<-focal(x, w=matrix(1,d,d), fun=mean, na.rm=T, pad=T)
  temp_x[temp_x>0]<-1
  temp_mask<-x
```

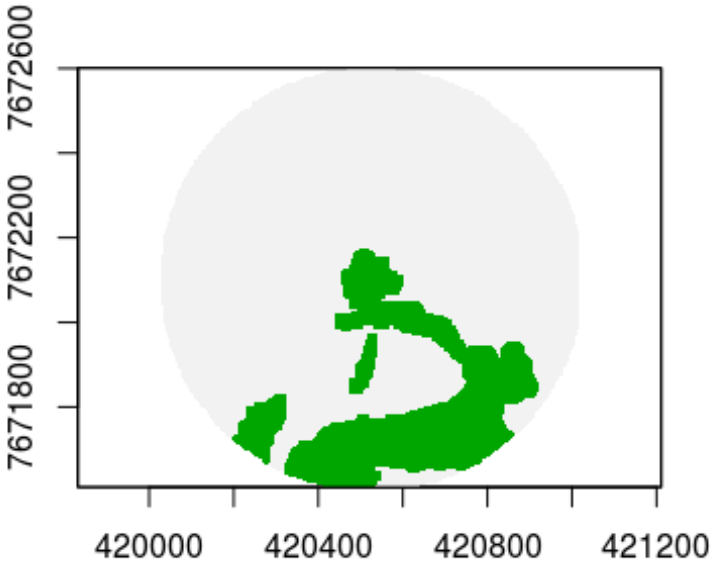
```
  temp_mask[!is.na(temp_mask)]<-1
  temp_x<-temp_x*temp_mask
  return(temp_x)
}
```

Here we will run the function and plot everything together to see the differences in area before and after considering the dispersal ability. The dispersal distance that we will use here is 25m, which we need to divide by the resolution (5m).

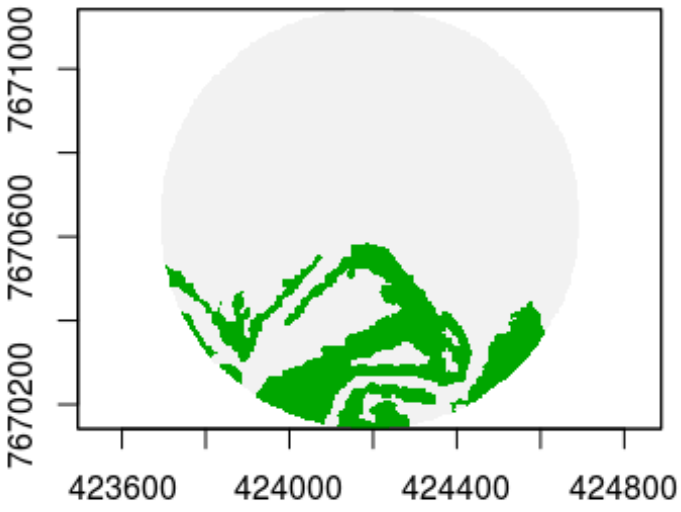
```
plot(v01_cl17,legend=FALSE)
```



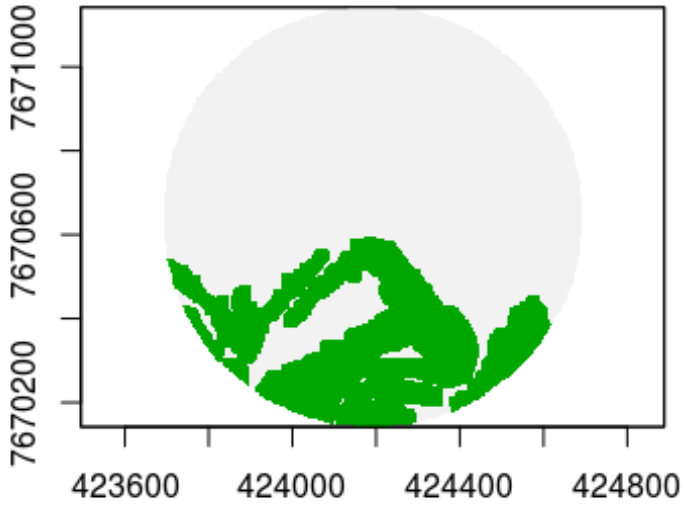

```
fc_v01_cl17<-funct.conect(v01_cl17,5)
plot(fc_v01_cl17,legend=FALSE)
```



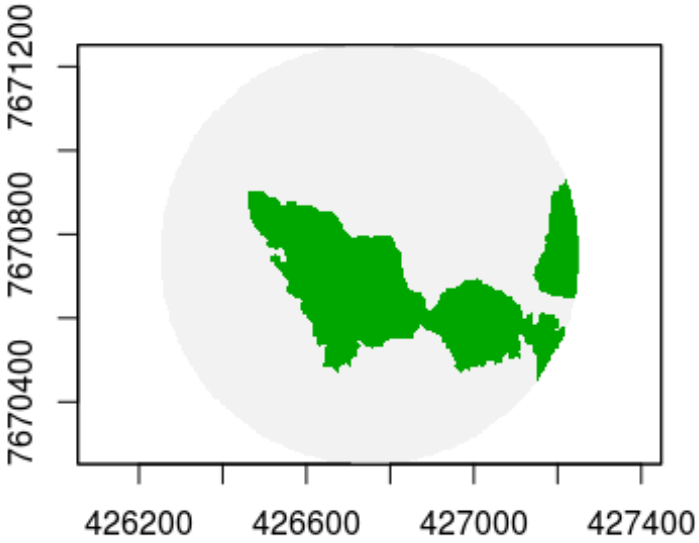

```
plot(v02_cl17,legend=FALSE)
```



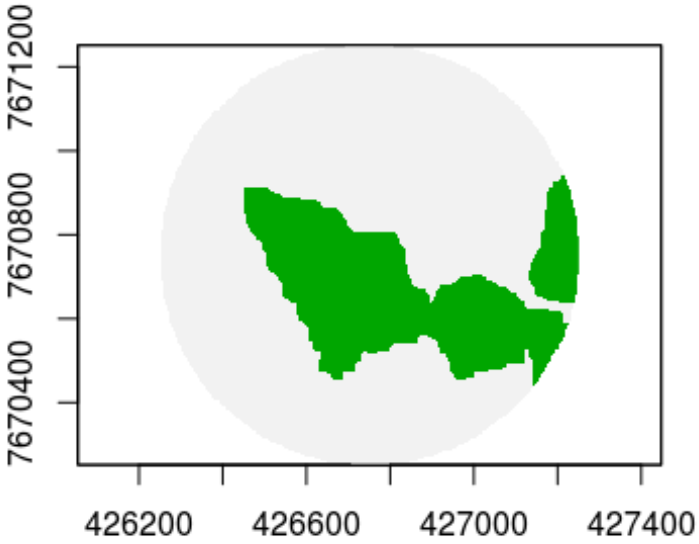
```
fc_v02_cl17<-funct.conect(v02_cl17,5)
plot(fc_v02_cl17,legend=FALSE)
```




```
plot(v03_cl17,legend=FALSE)
```



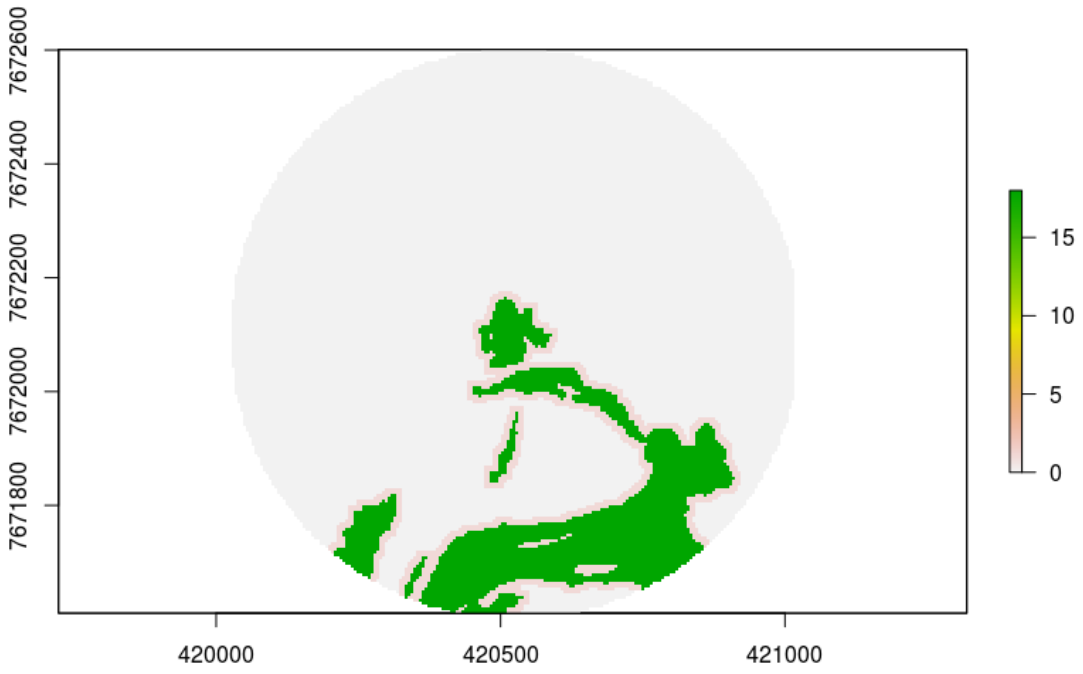

```
fc_v03_cl17<-funct.conect(v03_cl17,5)
plot(fc_v03_cl17,legend=FALSE)
```



We can plot this raster differentiating functional connectivity area. For this, we first add the original map with other that we have just created. With this addition, the new map now has values 0, 1 and 18, since the connectivity map is a binary map 0 and 1 and the original map the code class of interest was 17.

In this case this map is only for viewing, if you want to keep the class 17 with the original code suggest you do it the same way we did in section 1.7.2

```
fc_v03_cl17<-funct.conect(v03_cl17,5)
plot(fc_v03_cl17,legend=FALSE)
```

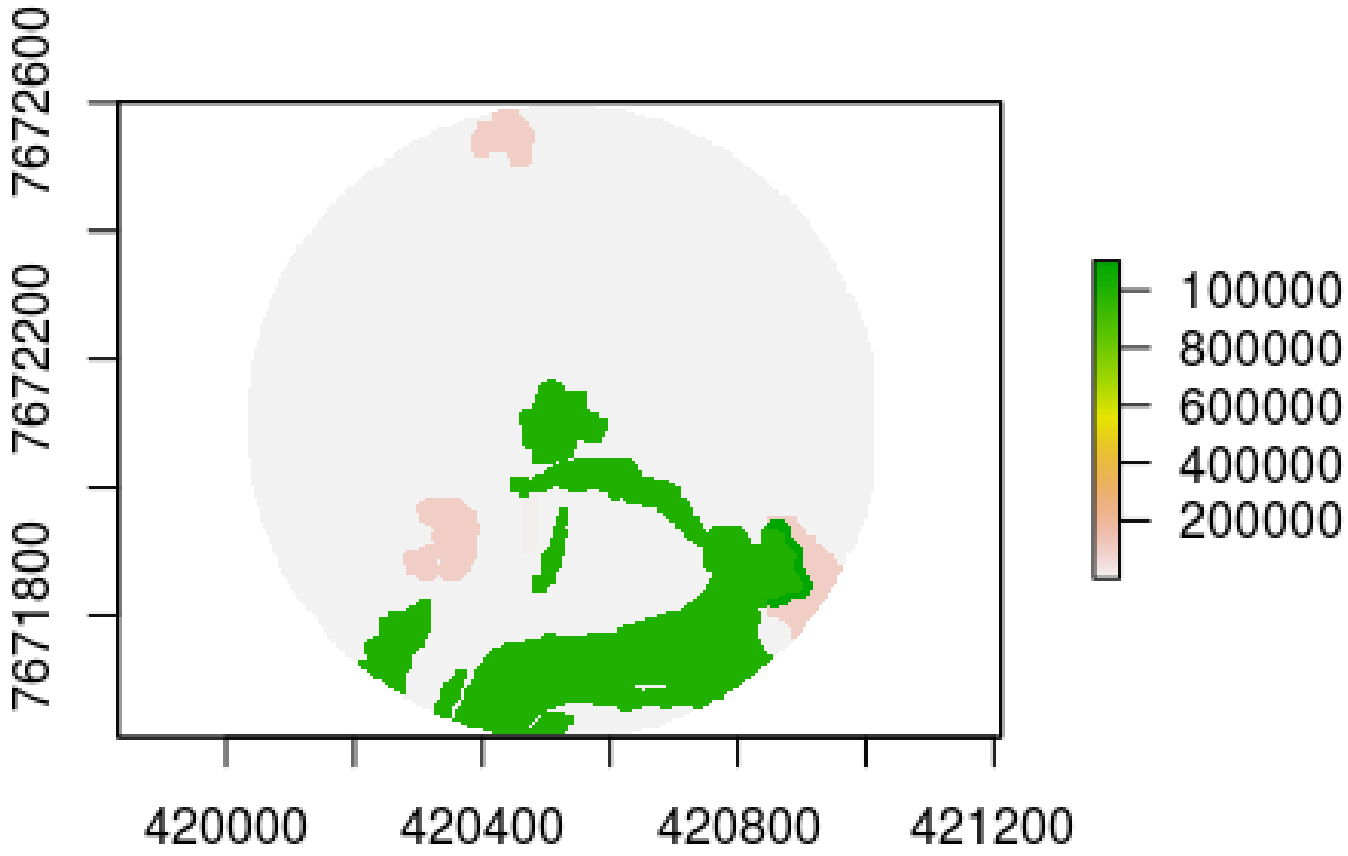


2.7. Richness of edges

To access the types and richness of interactions between patches of different classes (land use) within landscapes we will load another function: "multi.edge". This function creates a raster wherein it classifies each edge by a code that correspond to an interaction between land uses. The parameters of this function are the raster with all class (r), and the distance of edge influence (d).

```
multi.edge<-function(r,d){

  multi.edg<-list()
  r.edges<-list()
  land<-sort(unique(r[]))
  options("scipen"=100,"digits"=4)
```

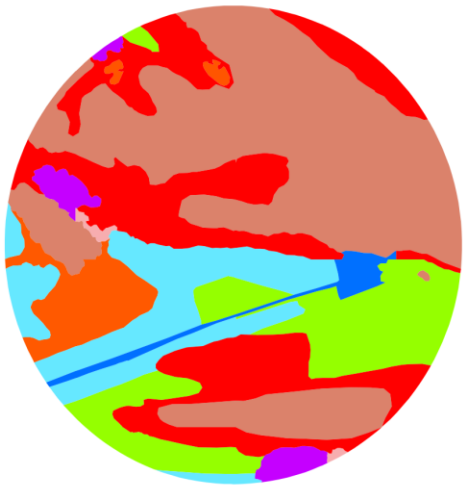
```
  val<-vector()
  for (i in 1:length(land)){
    val[i]<-10^i
  }

  land.val<-cbind(land,val)
  multi.edg[[2]]<-land.val

  for (j in 1: nrow(land.val)){#loop to create the buffer for each landuse
    land.temp<-land.val[j,1]
    map.temp<-r
    map.temp[map.temp!=land.temp]<-1

    map.mov<-focal(map.temp, w=matrix(1,d,d), fun=max, na.rm=F, pad=T)
    map.mov[map.mov!=land.temp]<-0
    map.mov[map.mov==land.temp]<-land.val[j,2]

    table(map.mov[])

    r.edges[[j]]<-map.mov
  }

  r.edges<-stack(r.edges)
  multi.edg[[1]]<-sum(r.edges)

  return(multi.edg)
}

m.edg_v01<- multi.edge(v01,3)
plot(m.edg_v01[[1]])
```



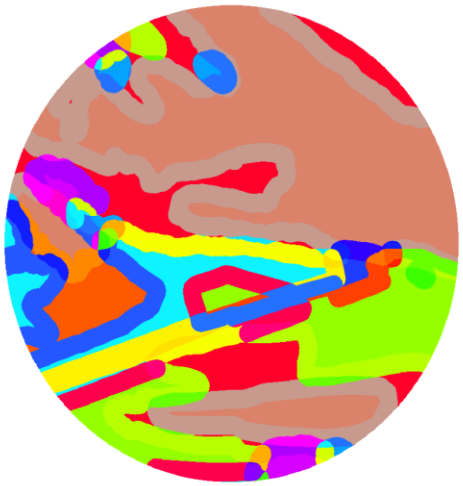
I know that this plot do not help us to see all edges, but this happens because it is difficult to assign distinguishable colors to so many different interactions between classes in this particular landscape.

Fortunately I have treated one raster in a GIS program, that shows all kinds of edges. This raster does not refer to any of these that we are using here, but I believe it is a good example of this function use.

Original raster



Raster with all types of edge



We can see how many classes and interactions that we have using the "table" function.

```
table(m.edg_v01[[1]][])
```

```
##
##       10      100      110     1000     1010     1100     1110    10000
##    10753     2601      206     6046      232      502        7     3243
##    10010    10100    10110    11000    11100   100000   100010  1000000
##      354      499       11      262        5       16       75      497
##  1000010  1000100  1000110  1001000  1001010  1001100  1010000 10000000
##      150       64       21      162       19       10       75     2949
## 10000010 10000100 10000110 10001000 10001010 10010000 10010100 10100000
##     1142      301        4      159        6      165       36        3
## 10100010 11000000 11001000 11010000
##        9       96        5        8
```

These classes are different of the original raster:I changed the original classes in a way that each one is an exponential of 10. With this it is easier to identify which classes constitute which edges. For example, imagine that we have two classes in the original raster: 1 and 2. In

the output raster this classes now can be 10 and 100, and the edge between this two classes will have the code 110. To get the classes codes we have to call the second element of the output object.

```
m.edg_v01[[2]]
```

```
##      land      val
## [1,]   10       10
## [2,]   11      100
## [3,]   12     1000
## [4,]   13    10000
## [5,]   14   100000
## [6,]   16  1000000
## [7,]   17 10000000
```

Let's do it for the others landscapes.

```
m.edg_v02<- multi.edge(v02,3)
m.edg_v03<- multi.edge(v03,3)
```

The richness of edgesis just the length of the object after applying the “table” function. Let's calculate and add it to our metric table.

```
edge.ric_v01<-length(table(m.edg_v01[[1]][]))
edge.ric_v02<-length(table(m.edg_v02[[1]][]))
edge.ric_v03<-length(table(m.edg_v03[[1]][]))

land_metric$edge_ric<-c(edge.ric_v01,edge.ric_v02,edge.ric_v03)
```

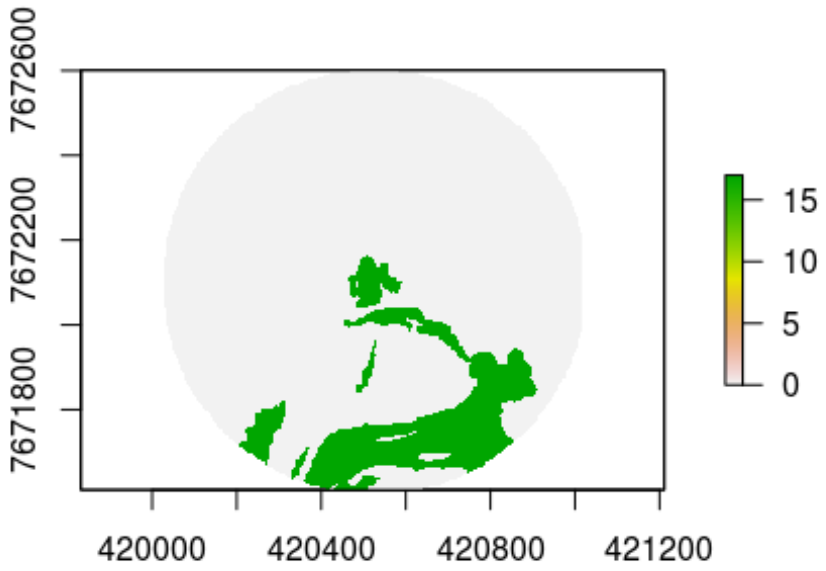
2.8 Usefull rasters

Two types of rasters that are commonly generated from an original land use raster are: 1) “Euclidean distance”: which expresses in each pixel the Euclidean distance from an element; and 2) “Moving window”: which expresses in each pixel the values that the neighboring pixels have.

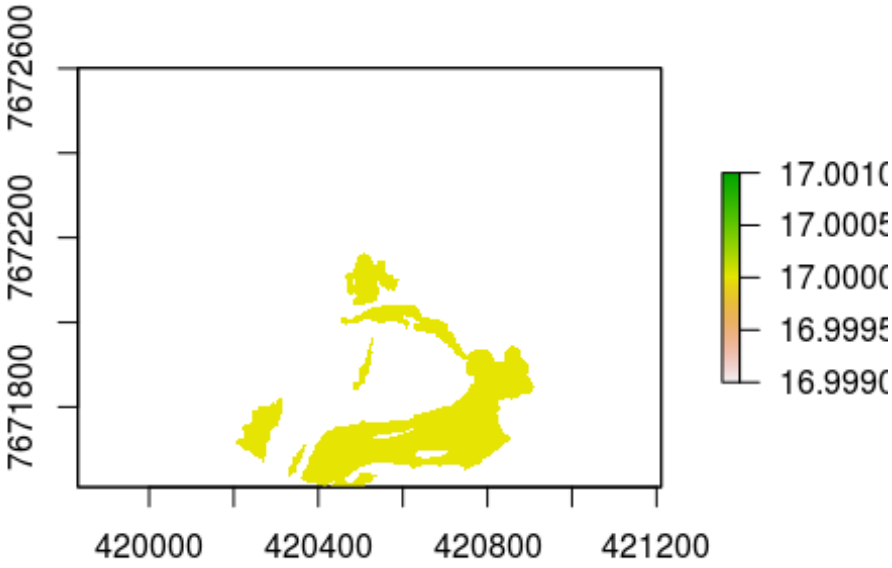
2.8.1 Euclidian distance raster

To create a raster of Euclidean distance all the pixels from other classes must be NA.

```
plot(v01_cl17)
```

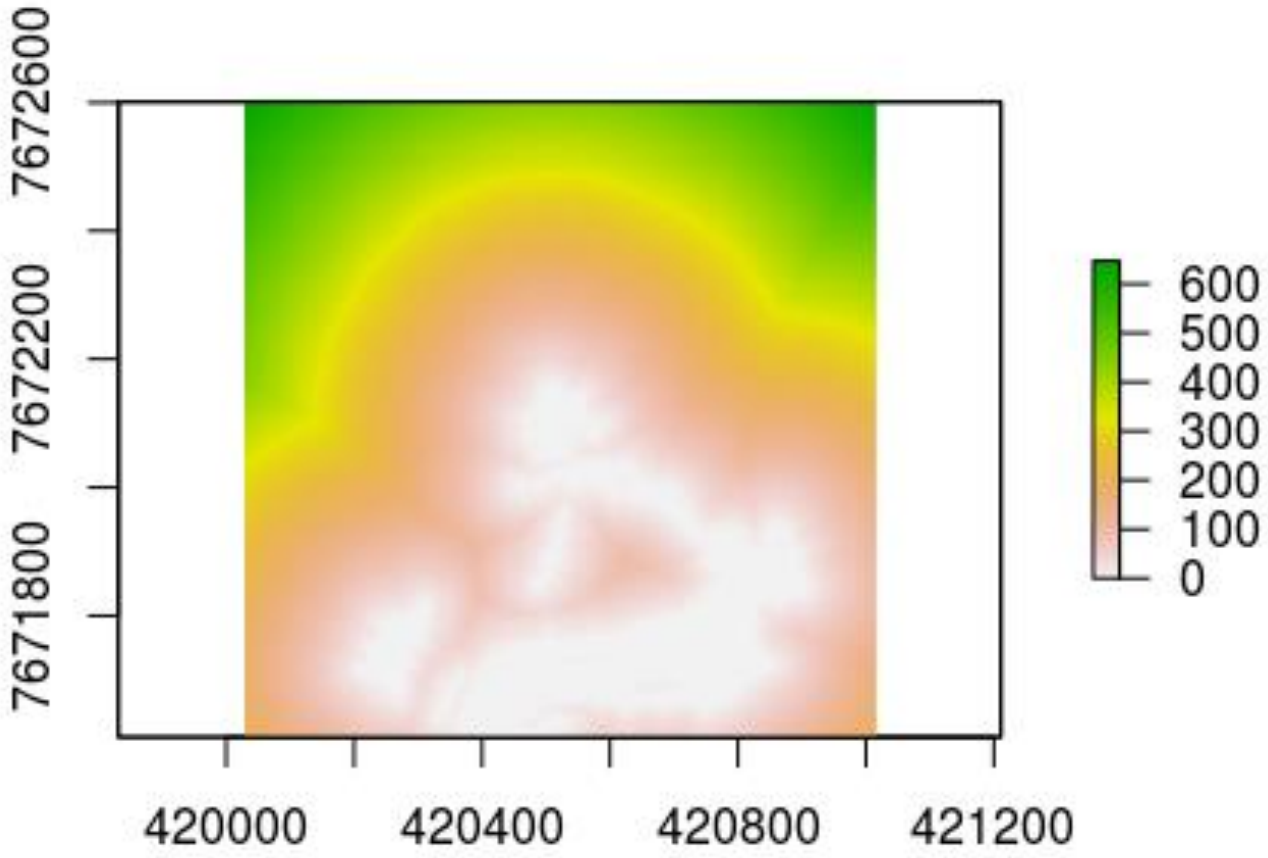



```
dist_v01<-v01_cl17
dist_v01[dist_v01==0]<-NA
plot(dist_v01)
```



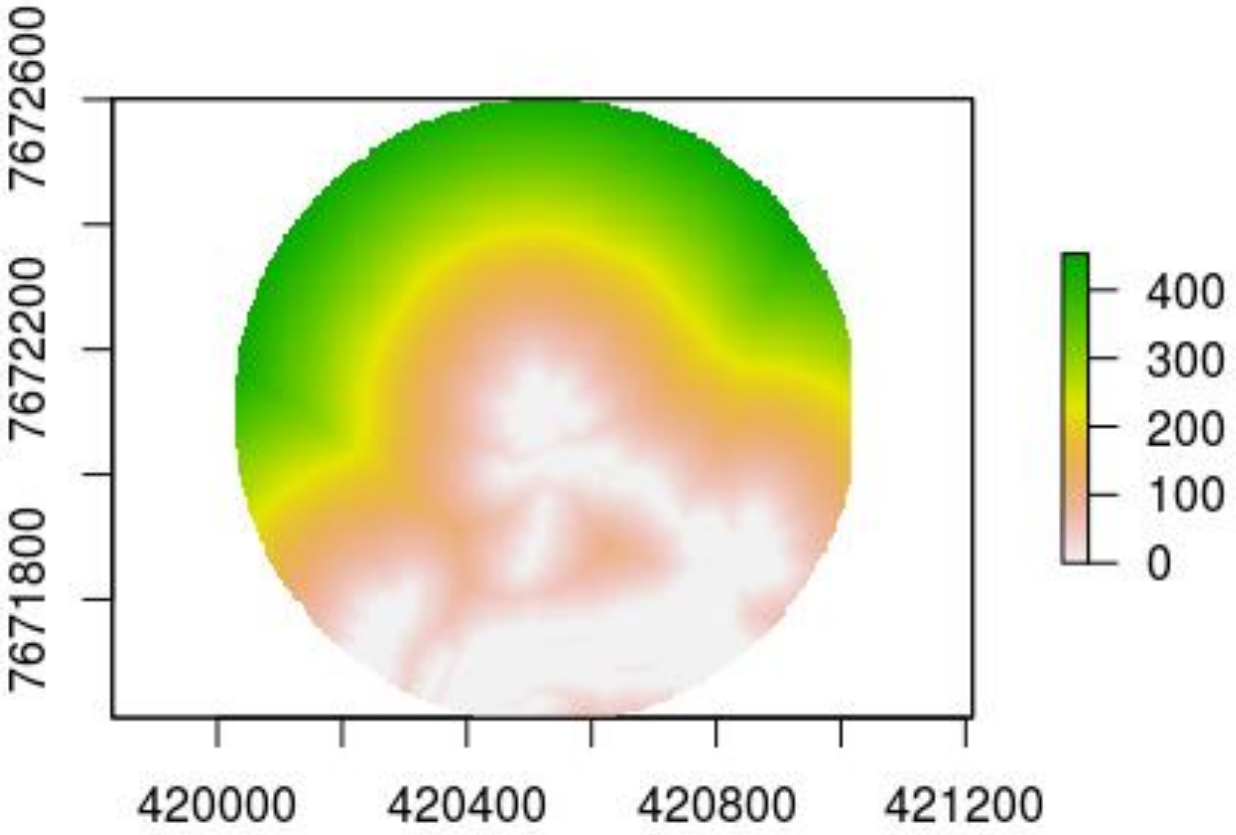
Let's do it using the function "distance" (“raster”).

```
dist_v01<-distance(dist_v01)
plot(dist_v01)
```



We now crop this raster to visualize only our area of interest (the buffer of the v01_cl17). Let's do it using the mask function.

```
dist_v01<-mask(dist_v01,v01)
plot(dist_v01)
```

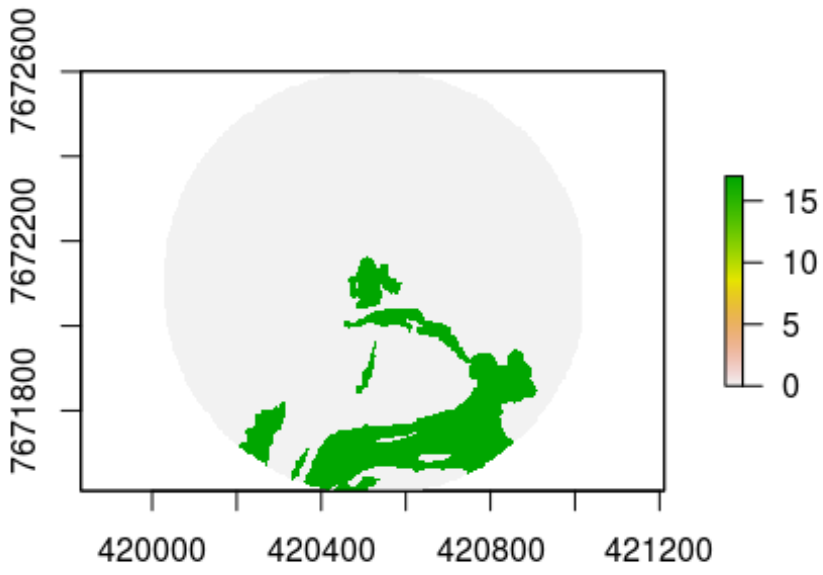


2.8.2 Moving window raster

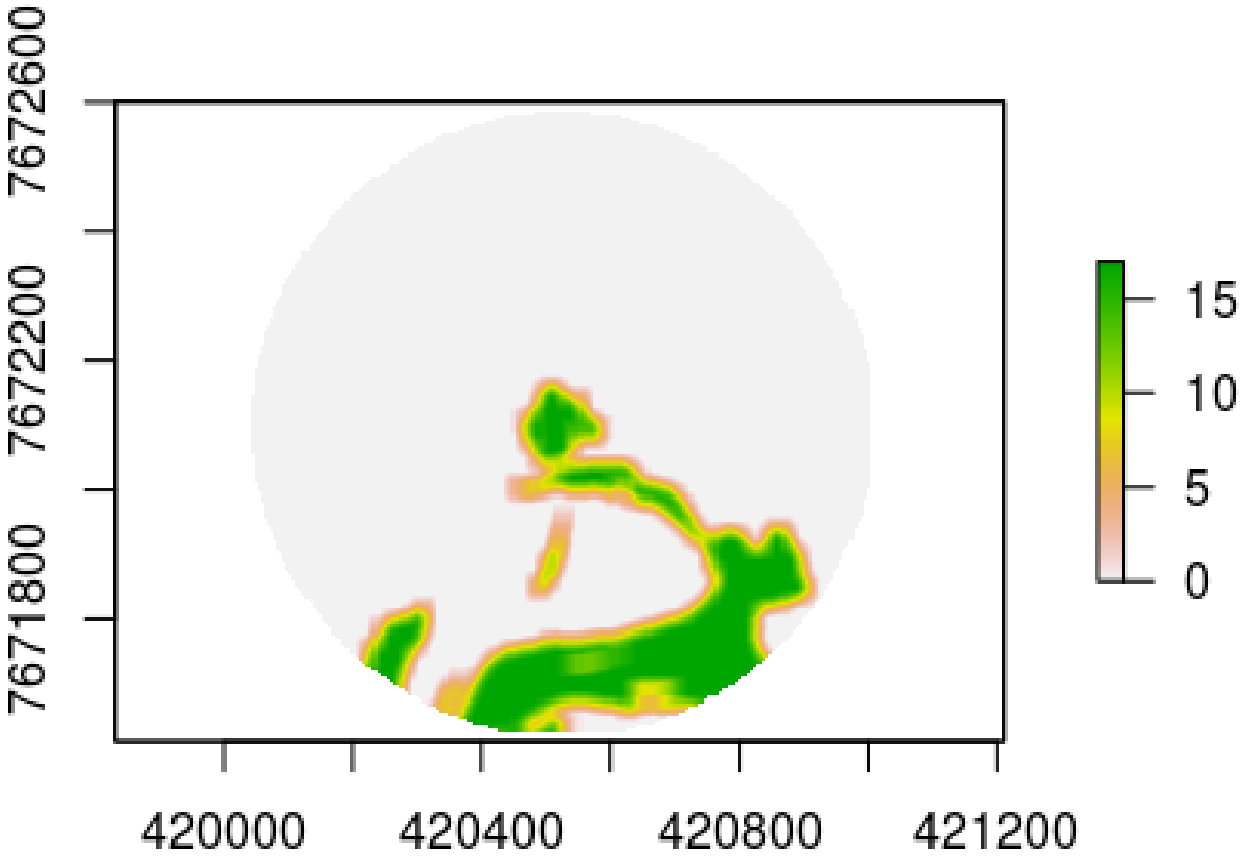
To make moving window rasters we will use the function "focal" (“raster”). To do this we need to set the size of our window by creating a matrix with the number of pixels of the window. Here we use 7, once that we know that the resolution of the original raster is 5m, our window is 35m x 35m.

This “focal” function assigns a value to each pixel, taking as reference the neighboring pixels that are within the window. Therefore it is necessary to indicate which function will be used to synthesize all the pixels values in a single value. The default is the mean, that we will use, but you can use other functions such as the maximum value, standard deviation, or even create a new function.

```
plot(v01_cl17)
```




```
mvw_v01<-focal(v01_cl17,matrix(1,7,7),fun=mean)
plot(mvw_v01)
```


7672600
7672200
7671800
420000
420400
420800
421200
15
10
5
0

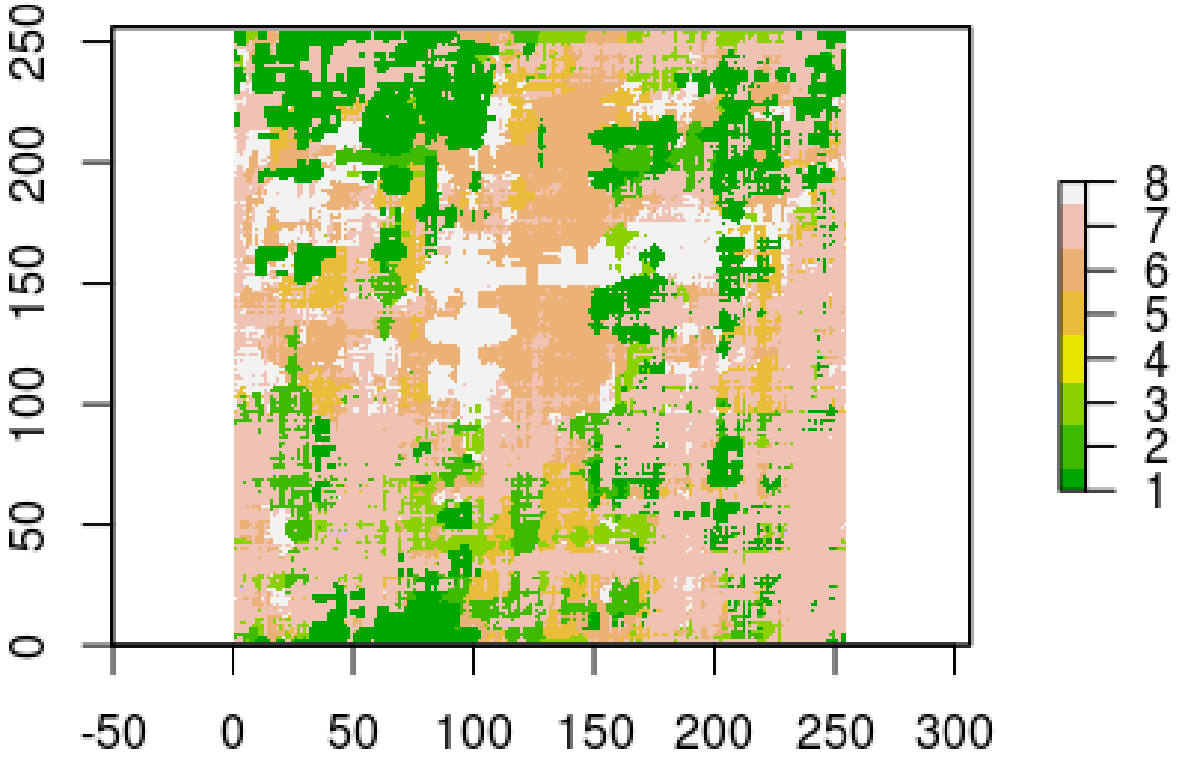
3. Simulating corridors

The simulation of dispersion routes between native habitat fragments is a practice employed for various purposes, such as to develop conservation strategies and to prioritize areas for conservation. In, landscape genetics studies the estimation of dispersion routes are commonly used as covariates for modeling genetic distance between fragmented populations and, conversely, genetic data can be used to inform or refine the delineation of dispersion routes. This technique, called simulation of ecological corridors, is based on the physical theory of fluids and consists in the use of an algorithm that delineate the least cost route between two points, given a raster wherein for each land use is assigned a cost of the focus species to move or use, that environment. This raster is called resistance surface. So, here we will first create resistance maps, then select our fragments, whose we want to connect and finally simulate corridors.

3.1 Creating raster of resistance surface

First let's import a new raster.

```
library(raster)
land<-raster("land_corri.tif")
plot(land,col = terrain.colors(8))
```



We will create a table with the land use code, in which we will insert the resistance value of each land use.

```
(table(land[])) # get the number of pixels in each of the seven classes of land use (note
#that class 4 doesn't exist in our raster)

##
##     1     2     3     5     6     7     8
## 11521  3801  2898  4275 10204 24588  7229

(land.id<-as.numeric(names(table(land[]))))

## [1] 1 2 3 5 6 7 8
```

```
resist<-data.frame(land.id)
```

Now we will assign resistance values to each land use. Here we will create two scenarios, each one with a different set of resistance values for each land use. (You may have some trouble to use some functions that I created if you have resistance values that are the same of some land use code. In this case I suggest you to change the resistance value (ex. instead of 5 you can put 5.1) or create the raster manually) .

```
resist$r1<-c(1,4,10,15,20,25,30)
resist$r2<-c(1,4,10,100,50,200,20)
resist
```

```
##   land.id r1  r2
## 1       1  1   1
## 2       2  4   4
## 3       3 10  10
## 4       5 15 100
## 5       6 20  50
## 6       7 25 200
## 7       8 30  20
```

The way to change the original values of each land use to resistance values is really simple and we had done several times in this workbook, but that to do for all land uses we will need to use the "for" function to make a loop for each land use. I created the function “resist.map” to do this for us.

We need to load it.

To use the function "resist.map" we just need the raster with the land use (r) and a table (d) with two columns: the first is the code of land use and the second is the resistance value of the land use.

```
resist.map<-function(r,d){
  for (i in 1:nrow(d)){
    r[r==d[i,1]]<-d[i,2]
  }
  return(r)}
```

We will create rasters for both resistance columns ("r1" and "r2" in "resist" table).

```
resist1<-resist.map(land,resist[,c(1,2)])
resist2<-resist.map(land,resist[,c(1,3)])
```

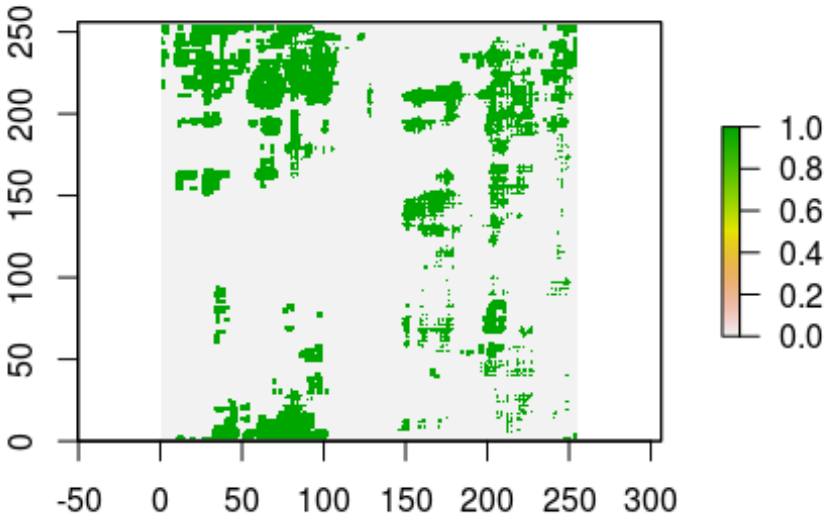
Note that the frequency of pixels is the same of the original raster, only the values were changed.

```
table(land[])

##
##     1     2     3     5     6     7     8
## 11521  3801  2898  4275 10204 24588  7229

table(resist1[])

##
##     1     4    10    15    20    25    30
## 11521  3801  2898  4275 10204 24588  7229
```

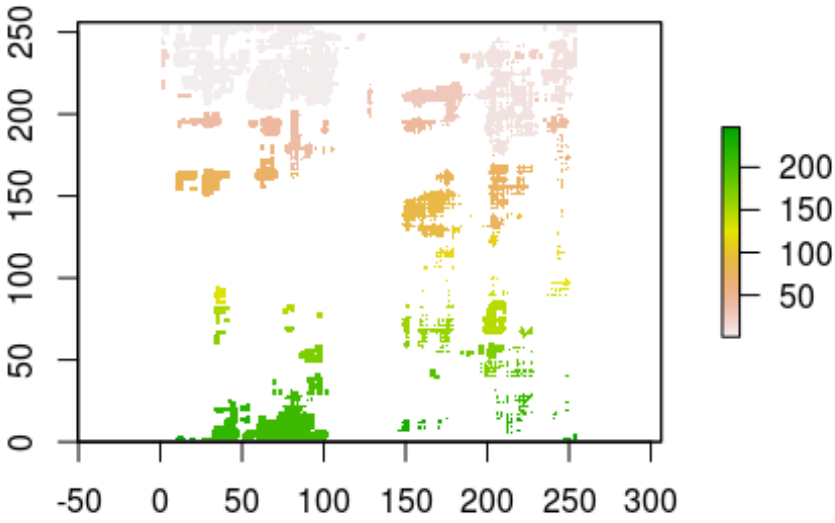
3.2. Selecting source-target patches

The patches between which we want to create the corridors are called "source" and "target" (ST). Here we will consider the class "1" as habitat and we will select two patches as S and T. To do this we will use the function "clump" (“raster”), given a raster with just one class (1). So first we need to create a raster with just one class.

```
 for.land<-land
  for.land[for.land!=1]<-0 # creating map with just one landuse
  plot(for.land)
```




```
pat.land<-clump(for.land) #numering patches
plot(pat.land)
```



```
  length(table(pat.land[]))
```

```
## [1] 247
```

The function "table" shows 247 patches of the class 1. To select the ST patches you may want to export the clump raster ("pat.land") to some GIS software, such as Quantum GIS, to find the code of patches that you want to use . Here we will jump this step and select four patches: 3,11,214 and 210.

```
ST.list<-c(3,11,214,210)
```

Now we need to find the coordinates of these patches in the raster. Because these patches are composed by more than one pixel, we have multiple possible coordinates. I created the "ST.coord" function that randomly selects a xy coordinate for each selected patch. Again we need to load it. This function needs two arguments, the raster with just one class, wherein the pixels values represent the code of the patch that it belongs (result from "clump function"), and a vector (st) with the code of patches that you want to know the coordinates.

```
ST.coord<-function(r,st){
  ST.xy<-data.frame()
  for (i in 1:length(st)){
    r.t1<-r
    r.t1[r.t1[]!=st[i]]<-NA
    xy.t<-rasterToPoints(r.t1)
    xy.t<-xy.t[sample(nrow(xy.t),1),]
    ST.xy<-rbind(ST.xy,xy.t)
  }
  colnames(ST.xy)<-c("x","y","patch")
  return(ST.xy)}
```

Now let's use it.

```
  st.coord<-ST.coord(pat.land,c(3,11,214,210)) # function to load
  st.coord
```

```
##       x     y patch
## 1  38.5 239.5     3
## 2 246.5 238.5    11
## 3  34.5   6.5   214
## 4 221.5  16.5   210
```

3.3 Simulating corridor

3.3.1 Simulating single corridor

We will simulate corridors using two functions; the first simulates a single route, which corresponds to the least cost path (LCP) analysis. To do this we need to load the function "uni.cor", which has three arguments: the surface resistance raster (r), the xy coordinates of the two patches (s and t).

```
library(gdistance)
uni.corr<-function(r,s,t){
  T <- transition(r, function(x) 1/mean(x), 8)
  T <- geoCorrection(T)
  ls_corridor<-shortestPath(T, coordinates(s), coordinates(t), output="SpatialLines")

  corr.dist<-SpatialLinesLengths(ls_corridor)
  euclid.dist<-(sqrt((s[1]-s[2])^2+(t[1]-t[2])^2))*res(r)[1]
  cust.val<-extract(r,ls_corridor)

  uni.lscorr<-list()
  uni.lscorr<-c(ls_corridor,corr.dist,euclid.dist,cust.val)
  return(uni.lscorr)
}
```

First we will create two objects with the coordinates of each patch.

```
s<-st.coord[1,1:2]
t<-st.coord[3,1:2]
```

Now we will simulate the corridor using the resistance map 1.

```
corr.res1<-uni.corr(resist1,s,t)
class(corr.res1)
```

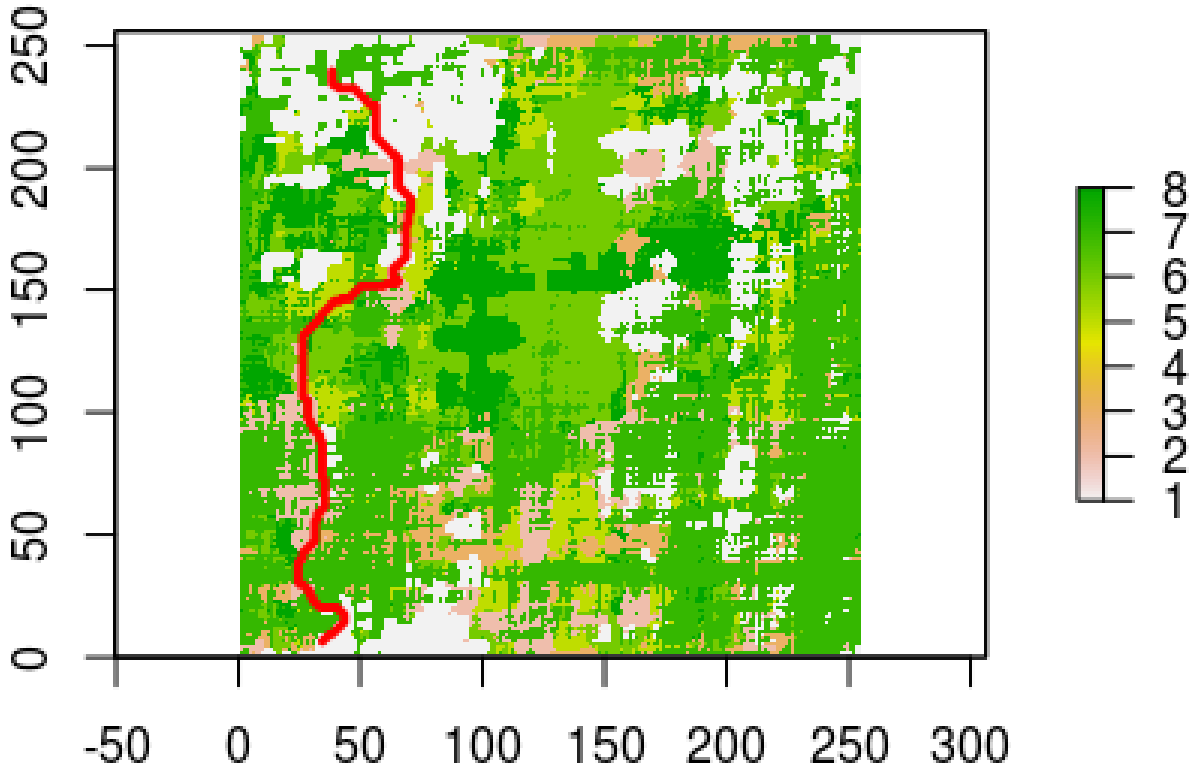
```
## [1] "list"
```

The output result is a list with 4 elements. The first is the shape of the route.

```
plot(corr.res1[[1]]) # corridor
```

Let's plot it again with the original raster.

```
plot(land)
plot(corr.res1[[1]],add=TRUE,lwd=3,col="red")
```



The other three elements are, respectively: the corridor length (in meters), the Euclidean distance (in meters) between selected ST coordinates and the resistance values for each pixel that the corridor had passed.

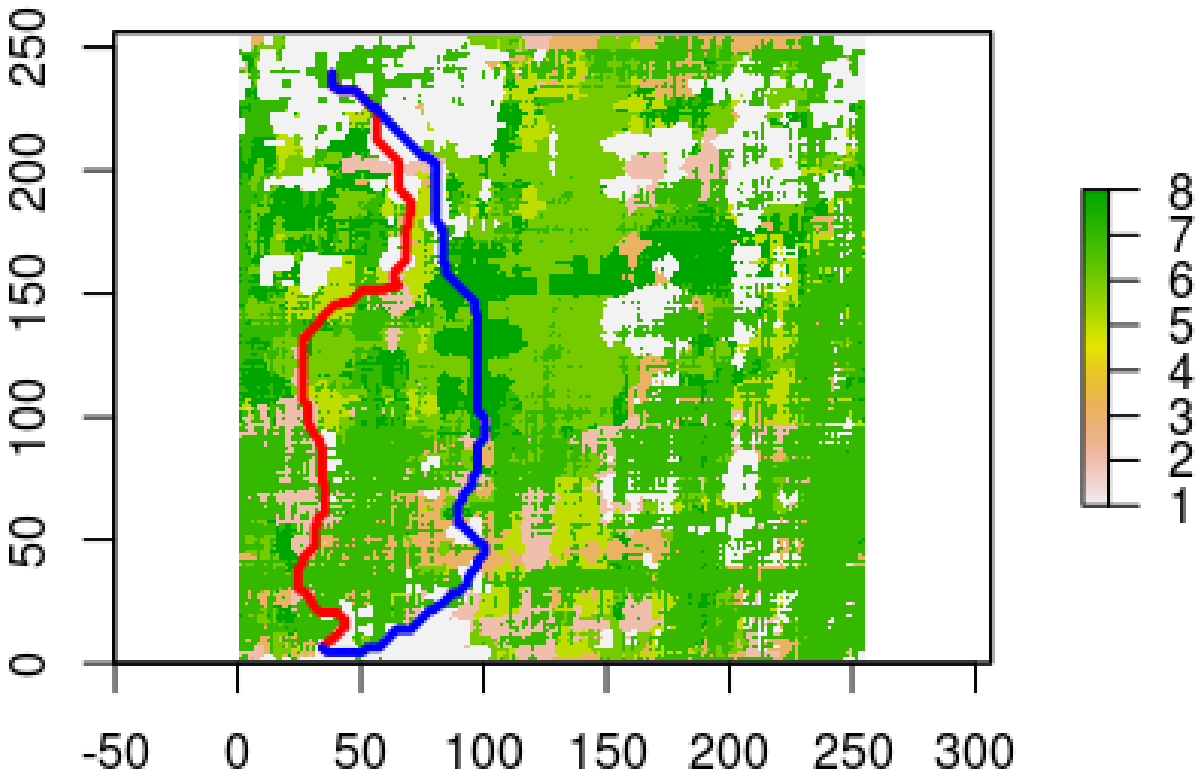
```
corr.res1[[2]] # corridor length
## [1] 307.1
corr.res1[[3]] # euclidian distance
## [1] 202.9
```

```
corr.res1[[4]] #resistance of each pixel
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [24]  1  1  1  1  1  1  1  1 15  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [47]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  4  1  1  4  4  4  1
##  [70]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 15 15 15 15 15  1 15
##  [93]  1  1  1  1  1  1 10 10  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
## [116]  1  1  1  1  1  4  1  1  1  1  1 25  4  4  4  4  4  4 15  4  4  4  4
## [139] 25  4  4  4  4  4 25 25 25 25 15 25 15  4 15 15 15 15 15 15 15 15 15
## [162] 15 15 15 15 15 15 15 15 15 15 15 30 30 25 30 30 25 30 25 25 25 20 20
## [185] 20 25  4 20  4  4  4  4  4  4  4  4  4 20  4  4  4  4  4  4  4  4  4
## [208]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 25  4  4 25  4  4
## [231]  4  4  4  4 25  1  1  1  1  1  1  1  1 25  1  1  1  1  1  1  1  1  1
## [254]  1  1  1 25 25  4 25  1  1  1  1  1  1  1  1  1 25  1  1  4  4  4  4
## [277]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 10  4  4  4  4
## [300]  4  4 25  4  4 25 25 25  4  4 25  4  4  4  4  4  4 25  4  4 25 25 25
## [323] 25 25 25 25 25  1  1  1  1  1  1  1  1  1  1  1  1 25  1  1  1  1  1
## [346]  1  1  1  1  1 25  1  1  1  1  1  1  1  1  1  1  1
```

Now let's repeat this routine with the second surface resistance and compare the routes.

```
corr.res2<-uni.corr(resist2,s,t)
plot(land)
plot(corr.res1[[1]],add=TRUE,lwd=3,col="red")
plot(corr.res2[[1]],add=TRUE,lwd=3,col="blue")
```



3.3.2 Simulating multiples corridors

Assuming that the organisms will not necessarily use the same least cost route, we will use the "multi.corr" function to create multiple routes between the fragments. This function may be useful to identify priority areas for conservation, so that the more corridors cross a certain pixel or region, the higher is the probability that this region is actually used by the organisms.

Similar to the previous function we need to insert the resistance raster (r) and the coordinates of ST patches (s and t), but here we need two additional arguments: n, the numbers of replicates of each corridors; and z, which is a variability parameter expressing how much those replicates will vary spatially among them. There is no right value of this parameter, and the best way to set it is to try different values and see how the routes respond to it. Here we will use the value 2. Let's load the function.

```
multi.corr<-function(r,s,t,n,z){

uni.corr<-function(r,s,t){
  T <- transition(r, function(x) 1/mean(x), 8)
  T <- geoCorrection(T)
  ls_corridor<-shortestPath(T, coordinates(s), coordinates(t), output="SpatialLines")

  corr.dist<-SpatialLinesLengths(ls_corridor)
  euclid.dist<-(sqrt((s[1]-s[2])^2+(t[1]-t[2])^2))*res(r)[1]
  cust.val<-extract(r,ls_corridor)

  uni.lscorr<-list()
  uni.lscorr<-c(ls_corridor,corr.dist,euclid.dist,cust.val)
  return(uni.lscorr)
}

multi.corr.tab<-data.frame()
list.corr<-list()
for (i in 1:n){
  jitter_val<-round(runif(length(r[]),1/z,z),3)
  r<-r/jitter_val

  foo<-uni.corr(r,s,t)

  foo2<-cbind(i,foo[[2]],foo[[3]],sum(foo[[4]]))
  colnames(foo2)<-c("replic","corr.dist","eucli.dist","cust")
  multi.corr.tab<-rbind(multi.corr.tab,foo2)

  r.line<-r
  r.line[]<-NA
  r.line<-rasterize(foo[[1]],r.line)
  list.corr[[i]]<-r.line
}
```

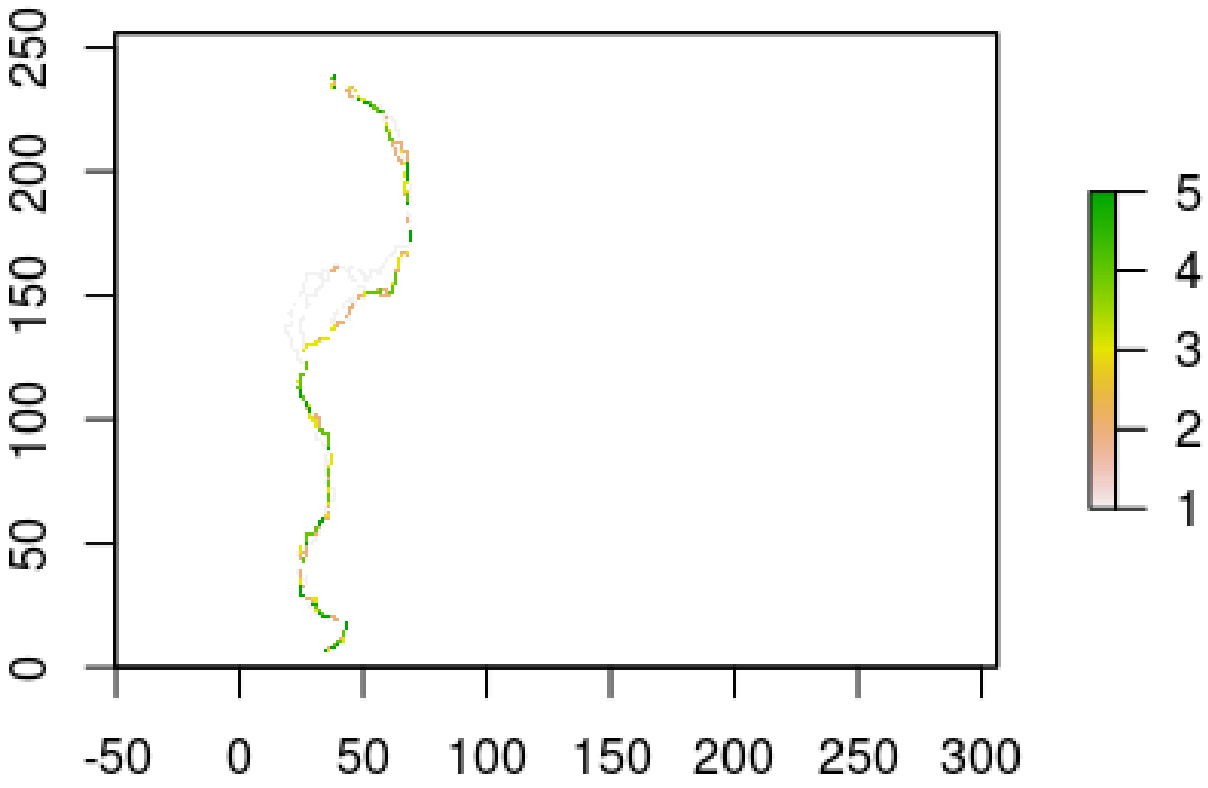
```
list.coor<-stack(list.corr)
sum.coor<-stackApply(list.coor,nrow(multi.corr.tab),fun=sum)
sum.coor[sum.coor==0]<-NA

multi.corr.resu<-list()
multi.corr.resu[[1]]<-sum.coor
multi.corr.resu[[2]]<-data.frame(multi.corr.tab)
return(multi.corr.resu)
}
```

Now we will use this function with the resistance surface raster 1, and we will make 5 corridors.

```
multi.res1<-multi.corr(resist1,s,t,5,2)
```

Again, the output is a list, but now with just two elements. The first is a raster with the simulated corridors. The pixels values correspond to the frequency of routes, where the maximum values can be up to the number replicates.

```
plot(multi.res1[[1]])
```

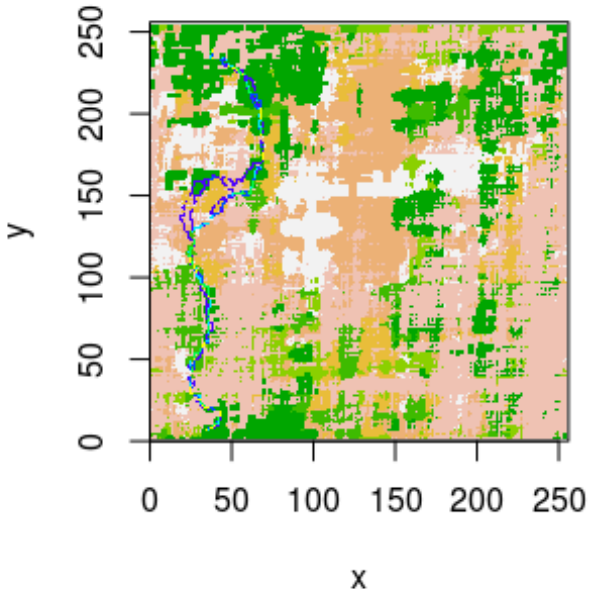



```
table(multi.res1[[1]][])
```

```
##
##   1   2   3   4   5
## 400 152 120 131  91
```

Let's plot it with the original raster.

```
par(pty="s")
image(land,col=terrain.colors(8),legend=FALSE)

image(multi.res1[[1]],add=TRUE,col=topo.colors(5),legend=FALSE)
```



The second element is a table wherein the rows are each corridor replicate and the columns are respectively: corridor length, Euclidian distance and the cumulative cost, which is the sum of all resistance values that the corridors have crossed.

```
multi.res1[[2]]
```

```
##   replic corr.dist eucli.dist cust
## 1      1     317.5      202.9 1838
## 2      2     342.7      202.9 1547
## 3      3     332.3      202.9 1540
## 4      4     339.4      202.9 1370
## 5      5     362.0      202.9 1303
```

The end!

- If you want to know a little more about the research that we do in the Spatial Ecology and Conservation lab, visit us in: http://leec.eco.br
- If you have comments and suggestions to this booklet please write me: felipemartello@gmail.com.

Thanks, Felipe Martello