Quick Introduction to **R** and the Work with **GIS**

Miguel Alvarez[*1] and Peter Borchardt[†2]

[1]Vegetation Ecology, INRES, University of Bonn, Germany
[2]Biogeography and Landscape Ecology, University of Hamburg, Germany

November 9th–13th, 2015

ARBONETH

Contents

[*]malvarez@uni-bonn.de
[†]borchardt@geowiss.uni-hamburg.de

1 Introduction to the Workshop

This is a quick introduction to the applications of **R** handling and analysing **GIS** data sets. This workshop is organised by the project **ARBONETH** (The Ethiopian Arboretum Project, `http://www.arboneth.com/`), which is founded by the **German Academic Exchange Service** (DAAD) and the **Federal Ministry for Economic Cooperation and Development** (BMZ).

1.1 Installing Instructions

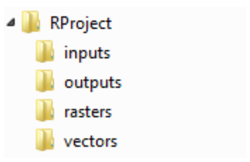
For the workshop's sessions you have to install RStudio, R (usually installed through RStudio) and QGIS (used only to visualise GIS data) in your computer. Additionally you may require to install Java (required for some of the previous software), Adobe Acrobat Reader (required to read this document) and Google Earth.

For the work with **R** there are a series of packages required for the sessions. Such packages will be distributed by the tutors but they can also be downloaded from The Comprehensive R Archive Network (CRAN).

1.2 Data Sets

The current handout is a tutorial including exercises that will be discussed in more detail during theoretical sessions. Some additional exercises are also provided for the afternoon sessions.

Files required for the exercises will be distributed among the participants to the workshop, organized in a `working directory` (project folder) containing four subfolders, namely `inputs` (input tables), `vectors` (vector files such as ESRI shapefiles), `rasters` (raster data sets), and `outputs` (folder to write outputs generated during **R** sessions).



2 Basics on R

The basic component of **R** is the `console`, which is the interface where you write and execute command lines. You will also work with `scripts`, that are text files compiling command lines. The `workspace` is the virtual place containing information structured in `objects`. The `working directory` is the folder, where **R** will search for data to load and where data will be written. Finally, the `history` is a record of command lines executed during a `session`.

Objects are structures containing data or functions in **R**. Such objects may belong to a `class`, which determines the properties or `attributes` of the object. Herewith the basic class of object in **R** is the `vector`.

2.1 Vectors and Matrices

A `vector` is a concatenation of values. Besides the `length` (the only dimension of a vector), the values contained in the vector may belong to a `mode`. Herewith only one mode is allowed for the content of a vector. Modes for data are `logical`, `numeric`, `factor`, `complex` and `character`. Additional modes used for programming purposes are `function`, `formula` and `expression`. The following are some alternative ways to generate vectors are:

```
> rep(5, times=10)
 [1] 5 5 5 5 5 5 5 5 5 5
> seq(from=10, to=100, by=10)
 [1]  10  20  30  40  50  60  70  80  90 100
```

```
> sample(letters, size=10, replace=TRUE)
 [1] "u" "n" "d" "v" "e" "r" "m" "s" "t" "t"
> c("Peter","Piper","picked","a","peck","of","pickled","peppers")
[1] "Peter"   "Piper"   "picked"  "a"       "peck"    "of"      "pickled"
[8] "peppers"
```

The previously generated vectors were displayed in the `console`, but they are not available for further routines. For it, you have to create new objects in the `workspace`, containing the respective values. That is to say, you may assign the values to new objects. Such operation is carried out by using the arrow symbol (`<-`).

```
> A <- seq(from=1, to=10)
> B <- seq(from=10, to=1)
```

The vectors `A` and `B` are numeric ones and this can be confirmed by `is.numeric(A)`, then **R** will answer you `TRUE`. You can also ask using `class(A)`.

While in the previous example vectors are generated by functions, some operations may also result in vectors.

```
> # Mathematical operations
> A + B
 [1] 11 11 11 11 11 11 11 11 11 11
> A*B
 [1] 10 18 24 28 30 30 28 24 18 10
> # Logical operations
> set.seed(58)
> C <- sample(letters, size=10, replace=TRUE)
> C == "p" # are values of C equal to "p"?
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
> C != "a" # are values of C different from "a"?
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

There are several ways allowing access to elements contained in a vector. The most common is using square brackets as displayed in the console. In such brackets you can indicate the position of elements through `integer` values. Negative integers indicate elements to be excluded. It is also possible to use `logic` values, whereupon `TRUE` indicates elements to be included and `FALSE` indicates elements to be excluded.

```
> # Access to elements of a vector
> B[5] # using number
> B[1:5] # using numeric vector
> B[-5] # using negative number
> B[B != 7] # using logical vector
```

Additionally to it, there is also the possibility to name the elements of a vector and access to them using those names as identity.

```
> names(B) <- LETTERS[1:length(B)]
> B
 A  B  C  D  E  F  G  H  I  J 
10  9  8  7  6  5  4  3  2  1 
> B[c("C","F","H")]
C F H 
8 5 3 
```

The `matrix` is in **R** a `vector` with 2 dimensions assigned as attribute. Notice that while in mathematics a vector is a special case of matrix, in **R** is the other way round. To produce a `matrix` you can use the functions `matrix`, `rbind` (binding rows), or `cbind` (binding columns).

```
> # matrix using function matrix
> matrix(1:20, nrow=5)
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
> matrix(1:20, ncol=5, byrow=TRUE)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
> # matrix using functions cbind and rbind
> cbind(A,B)
   A  B
A  1 10
B  2  9
C  3  8
D  4  7
E  5  6
F  6  5
G  7  4
H  8  3
I  9  2
J 10  1
> rbind(A,B)
   A B C D E F G H I  J
A  1 2 3 4 5 6 7 8 9 10
B 10 9 8 7 6 5 4 3 2  1
```

The access to single elements in the matrix is analogous to the access for vectors, but two values (separated by commas) are required, the first value indicates the row and the second, the column.

```
> # Creating the matrix in the workspace
> M <- cbind(A,B)
> M[1,] # access to first row
> M[,1] # access to first column
> M["A","B"] # access by names of rows and columns
```

2.2 Lists and Data Frames

Lists are more complex but at the same time more flexible. A `list` is an object with elements of different classes (even surrogated lists). The access to elements of a list is similar as for vectors. The use of double square brackets (`[[]]`) and the use of the dollar symbol (`$`) are two ways of access frequently used on lists.

```
> # Creating a list
> MyList <- list(First="Hello", Second=A, Third=M)
> # Access to elements
> MyList[1:2] # using a numeric index
> MyList[["Third"]] # using element's name
> MyList$First # using dollar symbol
> MyList$Second[3] # inside of an element
> MyList$Third[5,2]
```

One of the most common objects used to handle data in **R** is the `data.frame`. Data frames resemble matrices, but the main difference is that a `matrix` can contain information belonging to only one class (e.g. `numeric`, `factor` or `logical`), while in the data frame every column can be of a different class.

```
> ## The data.frame
> E <- letters[1:10]
> MyDataFrame <- data.frame(First=B, Second=C, Third=E, stringsAsFactors=FALSE)
> summary(MyDataFrame)
```

```
     First           Second             Third          
 Min.   : 1.00   Length:10          Length:10         
 1st Qu.: 3.25   Class :character   Class :character  
 Median : 5.50   Mode  :character   Mode  :character  
 Mean   : 5.50                                        
 3rd Qu.: 7.75                                        
 Max.   :10.00                                        
```

The access to elements of a `data.frame` can be done in the same way as for a `matrix`. Additionally you can access to the columns by using the symbol `$`, as for lists.

```
> MyDataFrame[,"Second"]
```

```
 [1] "i" "d" "r" "y" "v" "j" "g" "s" "g" "p"
```

```
> MyDataFrame$First
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
```

2.3 Functions and Loops

While operations will be reviewed more into detail during theoretical sessions, here there is a short introduction to functions and loops. A `function` is also an object in `R`. Such functions are usually represented as `foo(argument1=value1, ...)`, where `foo` is the name of the function object and in the brackets and separated by commas you may insert the values for the respective arguments of the function. You will steadily deal with functions during **R** sessions, but in some cases you will need to write your own functions.

```
> MyFunction <- function(x) (x - 5) * 10
> MyFunction(6)
```

```
[1] 10
```

```
> MyFunction(c(3,2))
```

```
[1] -20 -30
```

Though loops are not objects, there are some functions executing loops in **R**. The most common one is `for` in combination with `if`.

```
> for(i in letters) {
+         print(i)
+         if(i == "d") break
+ }
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

Here, there curly brackets enclose many command lines that may be evaluated in every loop.

2.4 Data Import and Export

There are many ways to import data to the `workspace` in **R**. One of the most basic ways is using comma separated values (CSV files) as inputs. It is also possible to import Microsoft Excel files by using the package `xlsx`. Previous to the import and export of data, you may check for the `working directory` used in the current `session`. By typing `getwd()` in the `console`, you will get the respective path. Since this path is the place where **R** will look for files to load, you have to set it according to the location of the files, for instance by `setwd("C:/Rproject")`. Notice that the separation between folders and subfolders are either the common slash symbol (/) or twice the backslash (\\).

```
> Juniperus <- read.csv("inputs/Juniperus_procera.csv")
```

Notice that you will always need to assign the loaded data to an `object`, otherwise you just get a print in the `console`, while the content will get lost after the function finish the work.

For the next example we will load the exemplary data `trees` (for details, see `?trees`). This data set will be imported as data frame. There are some functions provided to explore the structure and content of data frames.

```
> data(trees)
> str(trees)
'data.frame':        31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
> summary(trees)
     Girth           Height       Volume
 Min.   : 8.30   Min.   :63   Min.   :10.20
 1st Qu.:11.05   1st Qu.:72   1st Qu.:19.40
 Median :12.90   Median :76   Median :24.20
 Mean   :13.25   Mean   :76   Mean   :30.17
 3rd Qu.:15.25   3rd Qu.:80   3rd Qu.:37.30
 Max.   :20.60   Max.   :87   Max.   :77.00
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
> tail(trees)
   Girth Height Volume
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
```

In this case, `str` display an overview on the structure of the object `trees`, while `summary` shows statistic summaries of each single variable. The functions `head` and `tile` display respectively the first and the last rows. To write this data set into a file we will use the function `write.csv`.

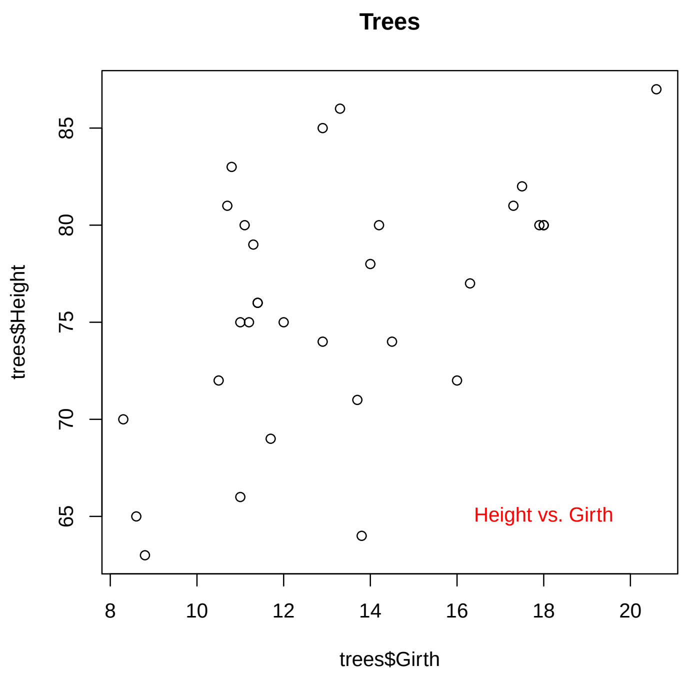
```
> write.csv(trees, "outputs/trees.csv")
```

For general routines to import data in r, look at the help of `read.table` or check the manual `"R Data Import/Export"` in `help.start()`.

2.5 Basics on R Plotting

The very basic function for plotting in R is `plot()` and most of the plotting parameters can be set by using `par()` (take a look in the help file for more details). Plotting functions are classified into two types, on the one side the `high level` functions that produce a whole graphic as in the case of `plot()`, on the other side the `low level` functions are able to introduce single elements in a drawn plot. Some high level functions offer the possibility to use them as low level, usually by setting the argument `add=TRUE`.

```
> plot(trees$Girth, trees$Height, main="Trees") # high level function
> text(18, 65, labels="Height vs. Girth", col="red") # low level function
```

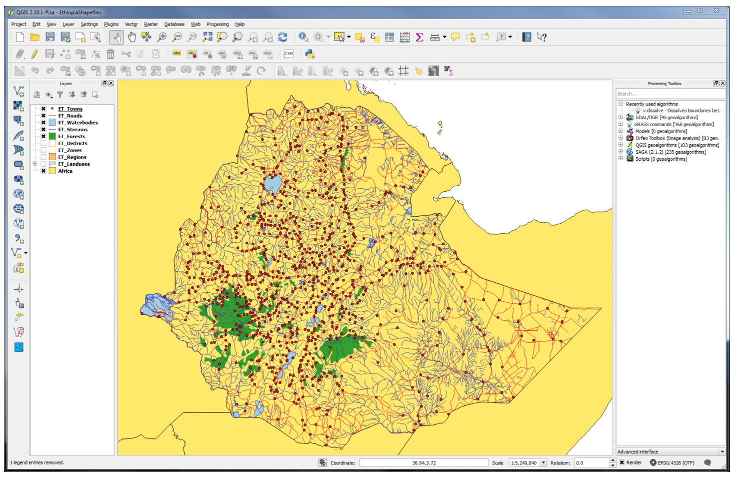


2.6 Exercises

- Create a matrix of 10 rows and 10 columns filling it with random digits, then compute the sums of columns and rows.
- Create a data frame including two variables used for a factorial experiment design (e.g. temperature: 30, 50 and 100 degrees, time: 0, 5, 20 min). Use `expand.grid` to get all possible combinations of levels. Randomize sorting of treatments in the data frame.
- Make box plots for the variable `Height` in the data `trees`. Check `data(iris)` and draw box plots for the variable `Sepal.Length` separated by `Species`.
- Prepare a plot for publication.

3 The Spatial Vector Files in R

There is a series of formats available for handling spatial data sets (GIS), but we will focus on the classes related to `Spatial*DataFrame`, which can content information usually stored in ESRI Shapefiles (including attribute tables) and can be accessed as data frames. Those object classes are provided by the package `sp`.



3.1 Importing ESRI Shapefiles

To import shapefiles we will use the function `readOGR` from the package `rgdal`.

```
> library(rgdal) # load package to the session
> Africa <- readOGR(dsn="vectors", layer="Africa")
```

```
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "Africa"
with 762 features
It has 3 fields
```

The loaded file contains spatial polygons corresponding to the African countries. In such file we can create a subset, for example considering those countries that are members of Eastern Africa.

```
> HornAfrica <- Africa$COUNTRY %in% c("Eritrea","Djibouti","Ethiopia",
+                   "Somalia")
> HornAfrica <- Africa[HornAfrica,]
> # Display
> plot(Africa, col="grey")
> plot(HornAfrica, col="orange", add=TRUE)
```

Further type of shapefiles (points and lines) can be also loaded and plotted in a similar way as done in the QGIS project (alternatively in an ArcGIS project).

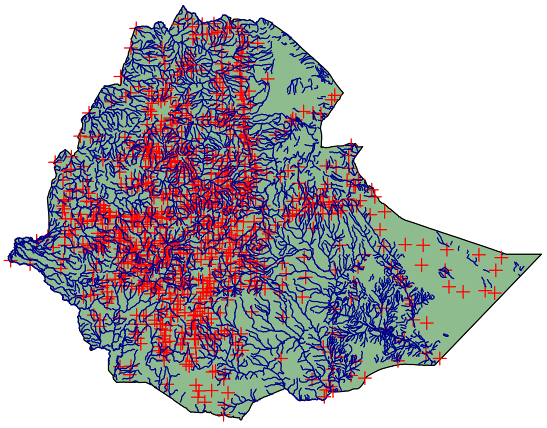
```
> Towns <- readOGR(dsn="vectors", layer="ET_Towns")
```

```
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Towns"
with 934 features
It has 22 fields
```

```
> Streams <- readOGR(dsn="vectors", layer="ET_Streams")
```

```
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Streams"
with 4070 features
It has 5 fields
```

```
> Ethiopia <- subset(Africa, COUNTRY == "Ethiopia")
> # Display
> plot(Ethiopia, col="darkseagreen")
> plot(Towns, col="red", add=TRUE)
> plot(Streams, col="darkblue", add=TRUE)
```



We can calculate distances between cities included in the object `Towns`. Note that such calculation requires a transformation of the coordinates to a projected reference system, for instance the UTM system, which uses meters as coordinate units.

```
> proj4string(Towns)
[1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
> Towns_UTM <- spTransform(Towns, CRS("+proj=utm +zone=37 +north +datum=WGS84"))
> Points <- as.data.frame(Towns_UTM[Towns_UTM$TOWN_NAME %in%
+                                                   c("Addis Abeba","Jimma","Dire Dawa"),])
> rownames(Points) <- Points$TOWN_NAME
> Dist <- dist(Points[,c("coords.x1","coords.x2")])
> round(Dist/1000, digits=0) # Distance in kilometers
            Dire Dawa Addis Abeba
Addis Abeba       347            
Jimma             591         258
```

Looking into a mileage table from Ethiopia, you may realize that the distance values are not that bad, although they are underestimated. The question is, which values are the wrong ones?

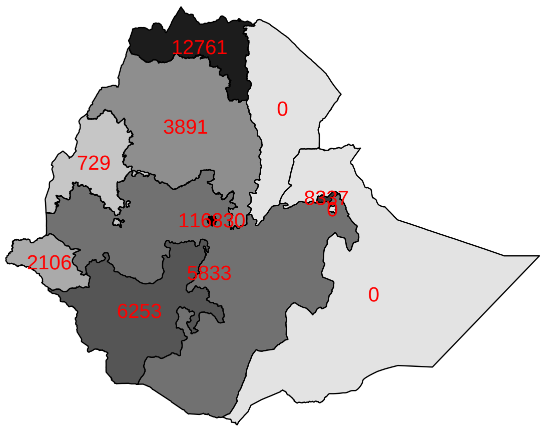
	Dire Dawa	Addis Abeba
Addis Abeba	445	
Jimma	791	355

3.2 Attribute Joins

As in ESRI shapefiles, elements included in the spatial objects have attributexs in an attribute table. Such attributes are stored as data frames in the slot `data` (access through `objectName@data`). New attributes can be passed from different sources, for example the populati9on size to the object `Settlements`. For it, we have to load the table `"KE_Population.csv"` to our workspace.

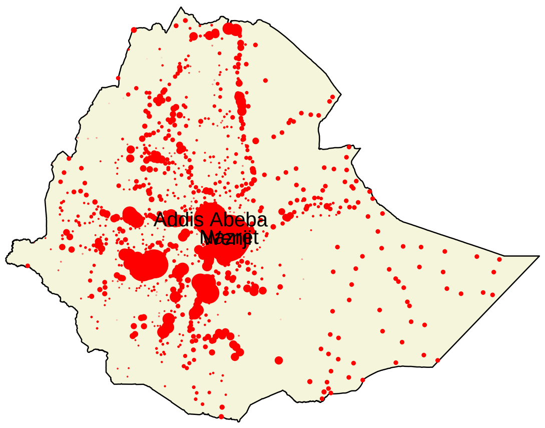
```
> Regions <- readOGR(dsn="vectors", layer="ET_Regions")
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Regions"
with 11 features
It has 16 fields
> Towns <- readOGR(dsn="vectors", layer="ET_Towns")
```

```
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Towns"
with 934 features
It has 22 fields
> # Applying joins for urban population in Regions
> Regions$UrbanPop <- Towns$POPURB[match(Regions$NAME_1, Towns$REGION)]
> # Colors for display
> Palette <- grey(1 - as.numeric(ordered(Regions$UrbanPop))/
+                                 max(as.numeric(ordered(Regions$UrbanPop))))
> # The plot
> plot(Regions, col=Palette)
> text(coordinates(Regions), labels=Regions$UrbanPop, col="red")
```



Accordingly, we can select those cities with a population higher than 250,000 in order to display them in a map. We will also use the function `symbols` to produce a display equivalent to `bubble` (bubble plots).

```
> VeryBigCities <- subset(Towns, POPURB > 100000)
> plot(Ethiopia, col="beige")
> symbols(coordinates(Towns),
+                 circles=Towns$POPURB/max(Towns$POPURB, na.rm=TRUE)*0.6,
+                 fg="red", bg="red", inches=FALSE, add=TRUE)
> text(coordinates(VeryBigCities), labels=VeryBigCities$TOWN_NAME)
```



Calculation of areas by using the function `gArea` from the package `rgeos`. Again as the example of distance calculations, you may transform the coordinate reference system to UTM.

```
> Waterbodies <- readOGR(dsn="vectors", layer="ET_Waterbodies")
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Waterbodies"
with 380 features
It has 5 fields
> plot(Ethiopia, col="gold")
> plot(Waterbodies, col="blue", border="darkblue", add=TRUE)
> # Load package rgeos for area calculation
> library(rgeos)
> Waterbodies_UTM <- spTransform(Waterbodies,
+                CRS("+proj=utm +zone=37 +north +datum=WGS84"))
> # Total area covered by lakes (in square kilometers)
> gArea(Waterbodies_UTM, byid=FALSE)/1000000
[1] 18882.41
> # Respective area size to attribute table
> Waterbodies$Area <- gArea(Waterbodies_UTM, byid=TRUE)/1000000
```

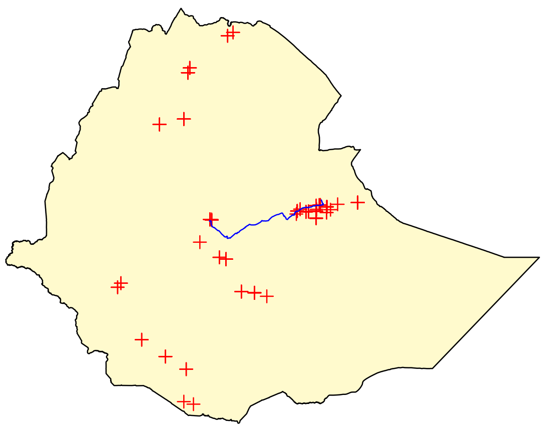
3.3 Creating Spatial Objects

Spatial objects can be created from non-spatial ones, for example using coordinate values from a table. For `SpatialPointsDataFrame` the way to produce it is straight forward. We will take from the data sets a table containing observations of *Juniperus procera* in Ethiopia. Such table content coordinate values in the columns `decimalLongitude` and `decimalLatitude`.

```
> Map <- read.csv("inputs/Juniperus_procera.csv")
> coordinates(Map) <- ~ decimalLongitude + decimalLatitude # get spatial
> proj4string(Map) <- CRS("+proj=longlat +datum=WGS84") # get projection
```

A bit longer is the way to create spatial objects containing lines or polygons. For example, the data set `Route.csv` contains coordinates values along the way from Addis Ababa to Dire Dawa.

```
> Route <- read.csv("inputs/Route.csv")
> Route <- Line(Route) # to Line object
> Route <- Lines(list(Route), ID="AddisToDire") # to Lines object
> Route <- SpatialLines(list(Route),
+                  proj4string=CRS("+proj=longlat +datum=WGS84")) # to SpatialLines
> Route <- SpatialLinesDataFrame(Route,
+                  data.frame(Start="Addis Abeba", End="Dire Dawa"),
+                  match.ID=FALSE) # to SpatialLinaesDataFrame
> # Display
> plot(Ethiopia, col="lemonchiffon")
> plot(Map, col="red", add=TRUE) # Juniperus procera points
> plot(Route, col="blue", add=TRUE) # Route Addis Abeba to Dire Dawa
```



The way to get polygons in spatial objects is analogous but using the sequence of functions `Ploygon`, `Polygons`, `SpatialPolygons` and `SpatialPolygonsDataFrame`. Once done the transformations, we are able to write the output as GIS data sets using the function `writeOGR`.

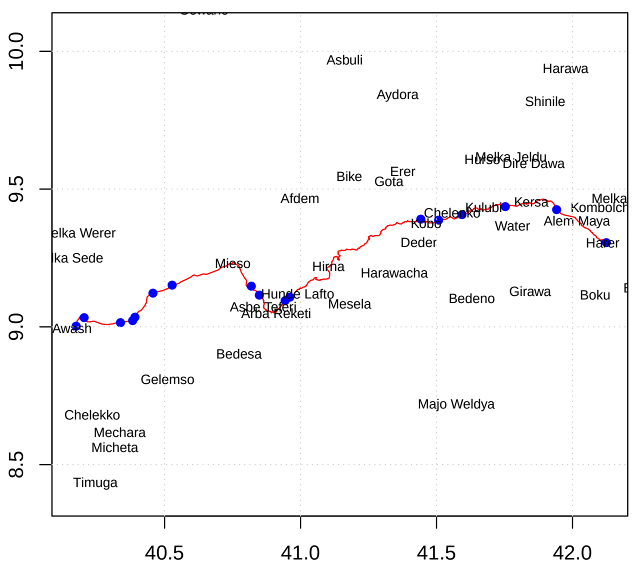
```
> # write ESRI shapefiles
> writeOGR(Map, dsn="outputs", layer="Juniperus", driver="ESRI Shapefile",
+                  overwrite_layer=TRUE)
> writeOGR(Route, dsn="outputs", layer="Route", driver="ESRI Shapefile",
+                  overwrite_layer=TRUE)
> # write KML files (Google Earth)
> writeOGR(Map, dsn="outputs/Juniperus.kml", layer="points", driver="KML",
+                  overwrite_layer=TRUE)
> writeOGR(Route, dsn="outputs/Route.kml", layer="lines", driver="KML",
+                  overwrite_layer=TRUE)
```

3.4 Reading GPX Files

The function `readOGR` have also routines for the import of GPX files, which is an exchange file format used byGarmin GPS devices. Two examples are provided here, one for a tracklog and one for waypoints. The tracklog was collected during a trip from Awash to Harer. The waypoints are sites of interest marked during the trip.

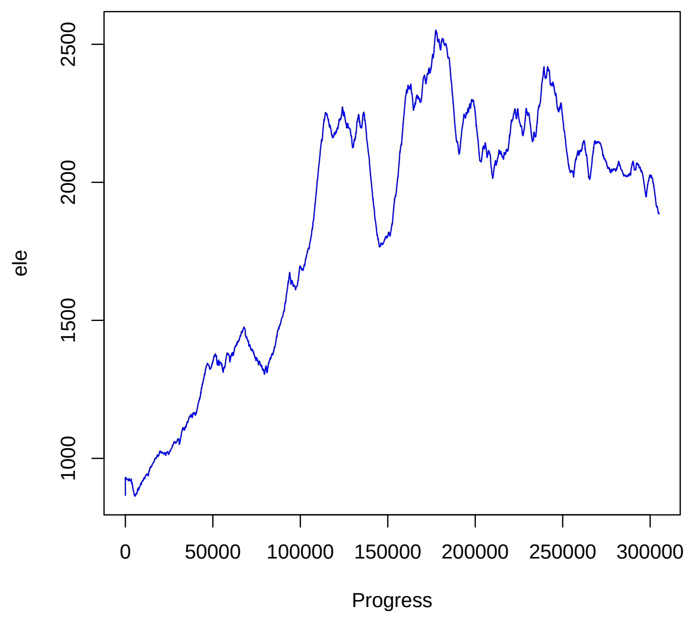
```
> GPStrack <- readOGR(dsn="vectors/track_awash_harar.gpx", layer="tracks")
OGR data source with driver: GPX
Source: "vectors/track_awash_harar.gpx", layer: "tracks"
with 1 features
It has 13 fields
```

```
> GPSwaypoints <- readOGR(dsn="vectors/waypoints_awash_harar.gpx",
+          layer="waypoints")
OGR data source with driver: GPX
Source: "vectors/waypoints_awash_harar.gpx", layer: "waypoints"
with 17 features
It has 23 fields
> plot(GPStrack, col="red")
> points(GPSwaypoints, pch=16, col="blue")
> text(Towns, labels=Towns$TOWN_NAME, cex=0.7)
> # Making map nicier
> box()
> axis(1)
> axis(2)
> grid()
```



In the case of track logs, there is also the possibility to load single points of the tracks with the respective records (altitude and time). In the following exercise we attempt to produce a profile of the trip according to the altitude values recorded by the device along the way.

```
> GPStrack_points <- readOGR(dsn="vectors/track_awash_harar.gpx",
+                 layer="track_points")
OGR data source with driver: GPX
Source: "vectors/track_awash_harar.gpx", layer: "track_points"
with 6521 features
It has 26 fields
> # Calculating progress in m
> GPStrack_points <- spTransform(GPStrack_points,
+                 CRS("+proj=utm +zone=37 +north +datum=WGS84"))
> Coords <- coordinates(GPStrack_points)
> Progress <- sqrt(diff(Coords[,1])^2 + diff(Coords[,2])^2)
> GPStrack_points$Progress <- cumsum(c(0, Progress))
> # Plot track profile
> with(GPStrack_points@data, plot(Progress, ele, type="l", col="blue"))
```



3.5 Exercises

- Calculate proportion of surface of the country covered by water bodies. Does it change by using different coordinate reference systems?
- Carry out similar tasks by using your own data sets.

4 Raster Data Files in R

Raster Data are collections of pixels containing numeric values. Such data sets are recognised as images (e.g. satellite imagery, aerial photographs) but may also content variables other than values for colors such as altitude, climatic conditions, terrain, etc.

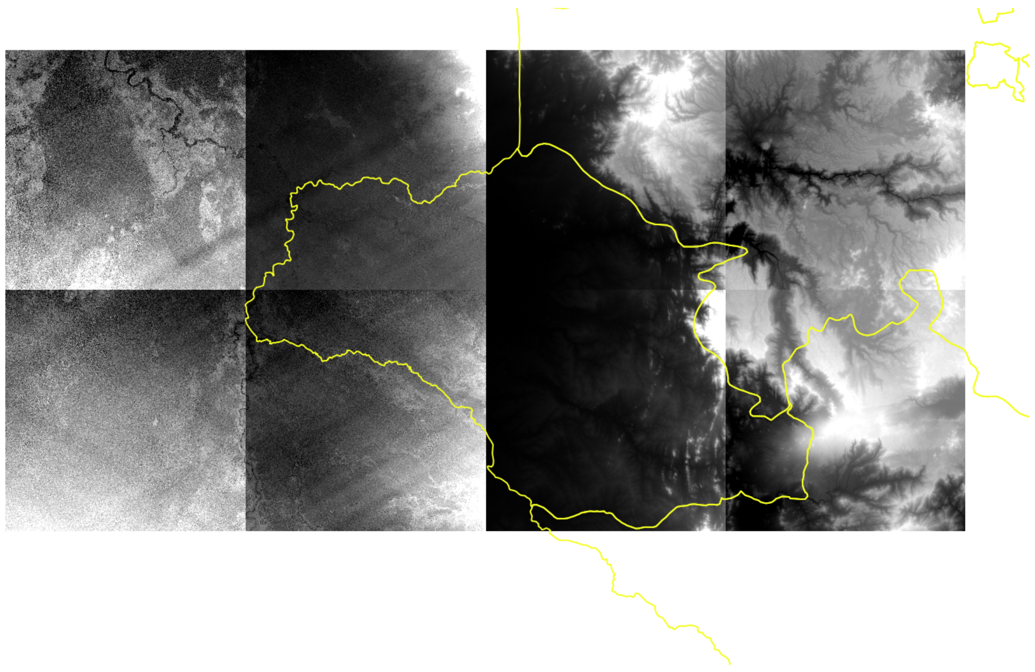
4.1 Import and Basic Process of Rasters

During this workshop we will use the package `raster`, which provides de classes `Raster*` (e.g. `RasterLayer`, `RasterStack`, etc.).

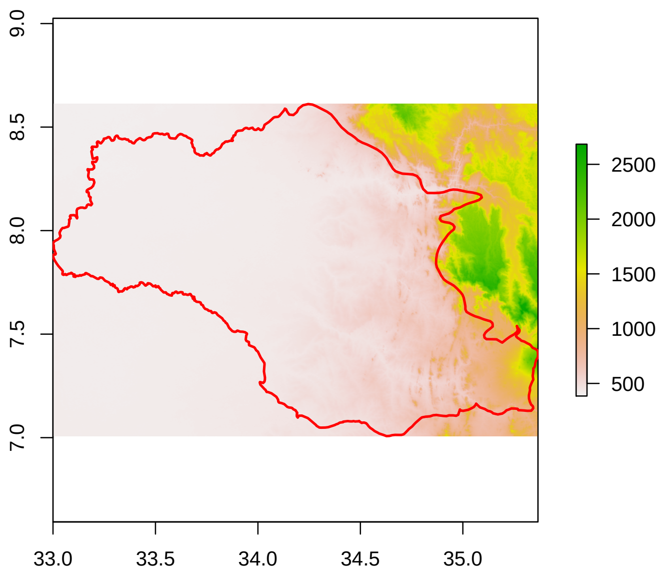
In the first example, we will import elevation models obtained form ASTER GDEM (a product of METI and NASA), which were downloaded from EarthExplorer. Such models have a resolution of 1 arc-second. Since this data set is distributed in 1 by 1 degree tiles, we require for example 8 tiles to display elevation landscape of the Gambela Region (displayed in the next figure).

So, the very first step is to load those tiles in the `workspace` and then to merge them into one data set.

```
> library(raster)
> Gambela <- subset(Regions, NAME_1 == "Gambela") # The region Gambela
> Files <- c(
+                 "N07_E032.tif",
+                 "N07_E033.tif",
+                 "N07_E034.tif",
+                 "N07_E035.tif",
+                 "N08_E032.tif",
+                 "N08_E033.tif",
+                 "N08_E034.tif",
+                 "N08_E035.tif")
> Files <- file.path("rasters", Files) # relative paths of single tiles
> DEM <- lapply(Files, raster) # load using function raster
```



```
> DEM <- do.call(merge, DEM) # merging tiles in one data set
> Ext <- extent(bbox(Gambela))
> DEM <- crop(DEM, Ext, snap="out") # cropping by the extension of Gambela
> # Display result
> plot(DEM)
> plot(Gambela, lwd=2, border="red", add=TRUE)
```

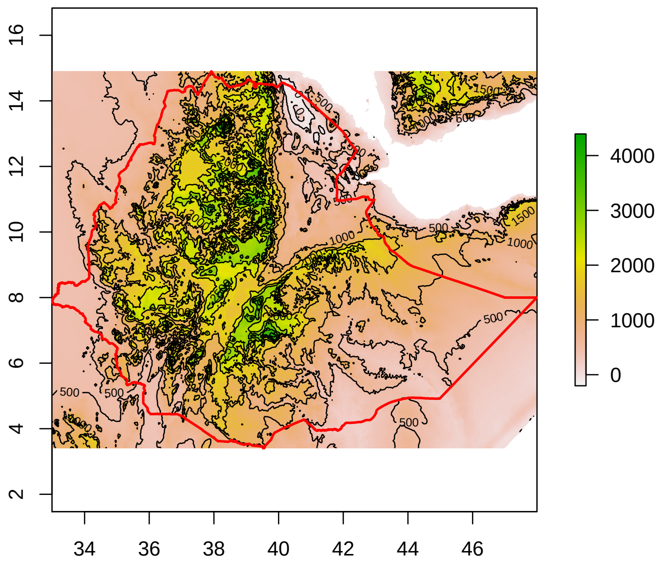


Notice that the object `Files` only contain the relative paths of the GeoTiff files (relative to the `working directory`), while `lapply` executes the function `raster` to import those files and store them as elements of a list (object `DEM`). The function `do.call` apply the method `merge` for the tiles in `DEM`, merging them into one `RasterLayer`. Finally `crop` cuts the resulting raster according to the extension of the Gambela Region.

4.2 Working with Digital Elevation Models

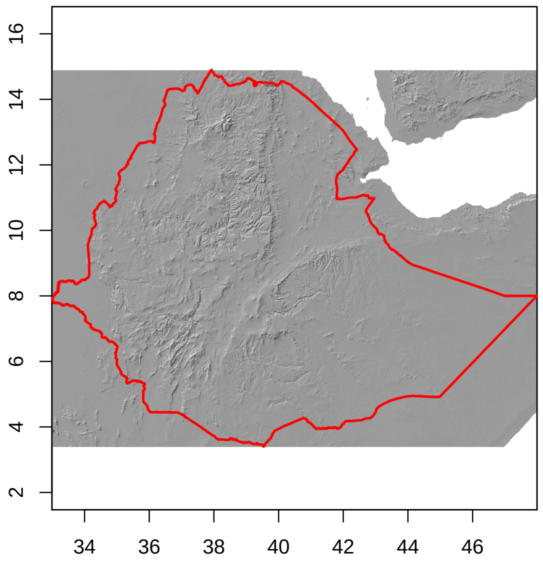
Further exercises use an elevation model downloaded from the WorldClim database (subset for Eastern Africa). We will then display the extention of Ethiopia.

```
> Altitude <- raster("rasters/alt.grd")
> Ext <- extent(bbox(Ethiopia))
> Altitude <- crop(Altitude, Ext, snap="out")
> # Display
> plot(Altitude)
> contour(Altitude, add=TRUE)
> plot(Ethiopia, border="red", lwd=2, add=TRUE)
```



The function `terrain` offers options to calculate the slope and exposition after DEMs and work using "moving windows" (if you are interested on those techniques, look at the help for the function `focal`). In addition we will produce a display of hill shades.

```
> Slope <- terrain(Altitude, opt="slope")
> Aspect <- terrain(Altitude, opt="aspect")
> Hill <- hillShade(Slope, Aspect, 40, 315)
> # Display
> plot(Hill, col=grey(0:100/100), legend=FALSE)
> plot(Ethiopia, border="red", lwd=2, add=TRUE)
```



4.3 Raster Calculation

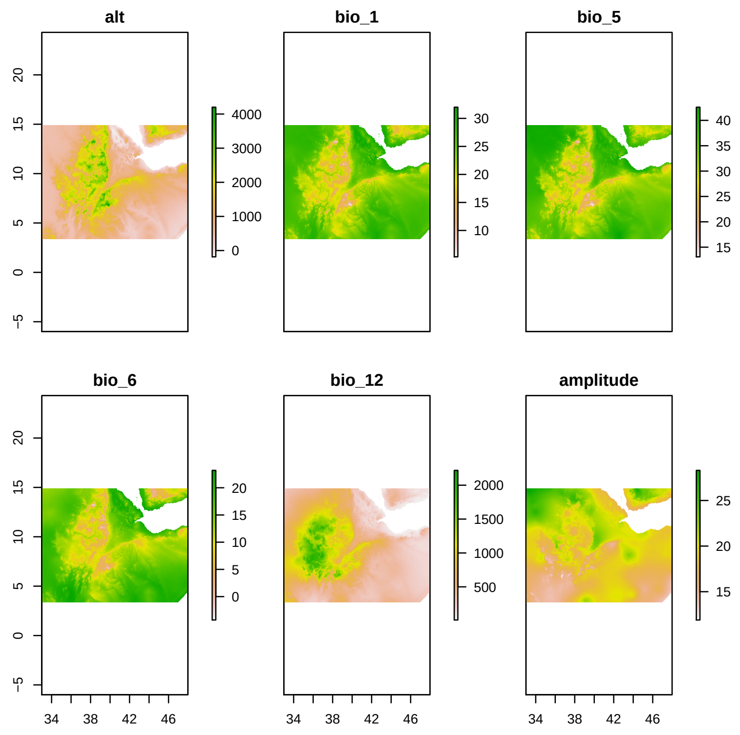
Rasters contain numeric information and resemble matrices in their structure. Therefore calculations can be carried out between matrices, provided that rasters have the very same resolution, the same projection and the same extent. In other words, they may have a perfect spatial matching of pixels. Such kind of files can be contained by objects of the class `RasterBrick` (single multiple-layer file) or `RasterStack` (single files for every layer).

In the following example, we will work with a subset of the WorldClim database. This database provide an elevation model and climatic variables obtained from interpolation of historical records from climatic stations distributed worldwide (for more details, see Hijmans et al. 2005).

```
> Files <- c(
+                 "alt.grd", # Altitude (m asl.)
+                 "bio_1.grd", # Mean temperature (degrees*10)
+                 "bio_5.grd", # Max temperature in warmest month (degrees*10)
+                 "bio_6.grd", # Min temperature in coldest month (degrees*10)
+                 "bio_12.grd") # Annual precipitation (mm)
> Files <- file.path("rasters", Files)
> # Creating stack
> ClimData <- stack(Files)
> ClimData <- crop(ClimData, Ext, snap="out")
>
```

Following import, we may convert temperature values into Celsius degrees for further display. We will additionally calculate the amplitude as the difference between the minimum temperature during the coldest month (variable `bio_6`) and the maximum temperature during the warmest month (varaible `bio_5`).

```
> # Convertion of temperatures into Celsius degrees
> ClimData[["bio_1"]] <- ClimData[["bio_1"]]/10
> ClimData[["bio_5"]] <- ClimData[["bio_5"]]/10
> ClimData[["bio_6"]] <- ClimData[["bio_6"]]/10
> # Calculation of amplitude
> ClimData[["amplitude"]] <- ClimData[["bio_5"]] - ClimData[["bio_6"]]
> # Display
> plot(ClimData)
```



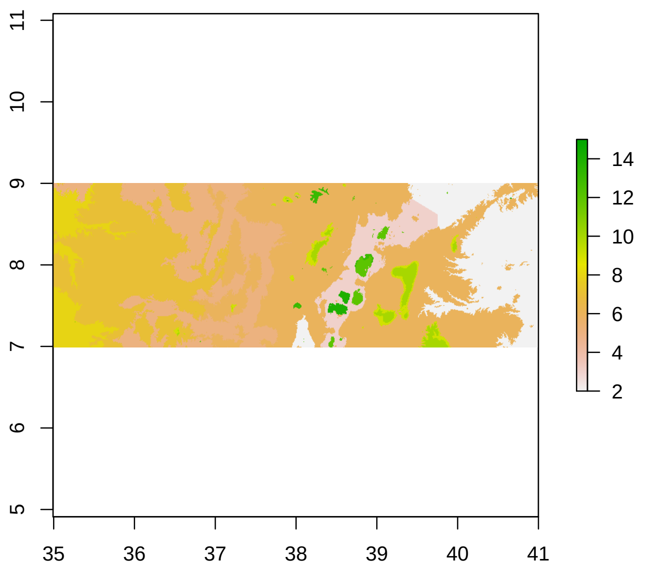
Such operations can be also carried out using the functions `overlay` or the function `calc`. We calculate amplitude again as follows:

```
> Vars <- subset(ClimData, c("bio_6","bio_5")) # subset of variables
> Amplitude <- overlay(Vars, fun=function(x,y) x-y) # alternative 1
> Amplitude <- calc(Vars, fun=function(x) x[1]-x[2]) # alternative 2
```

4.4 Rasterizing and Masking

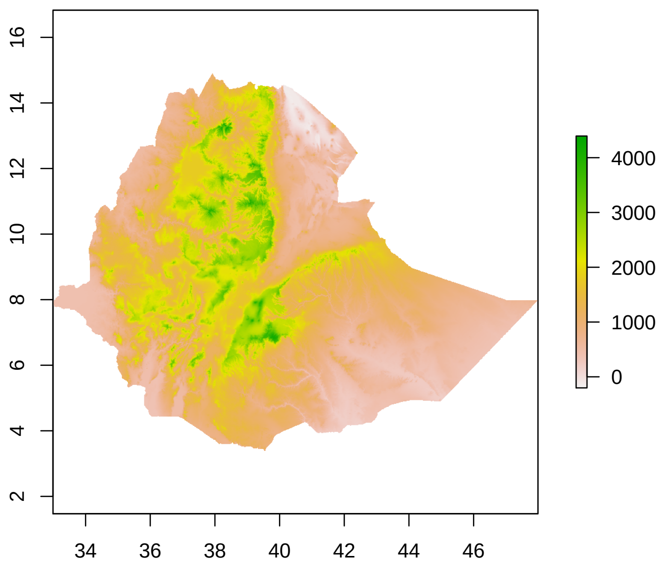
Information contained as attributes of s `SpatialPolygonsDataFrame` (slot `data`) can be also transfered to raster files. In the following example, we will use a shapefile of the potential natural vegetation of Ethiopia (obtained from Friis et al. 2010). In this file ther is a numeric index for vegetation units, included in the attribute table. We will just work with a strip of the country to save some processing time.

```
> Altitude <- crop(ClimData[["alt"]], extent(c(35,41,7,9)), snap="out")
> Vegetation <- readOGR(dsn="vectors", layer="ET_Vegetation")
OGR data source with driver: ESRI Shapefile
Source: "vectors", layer: "ET_Vegetation"
with 4265 features
It has 3 fields
> Vegetation_raster <- rasterize(Vegetation, Altitude, "value")
> plot(Vegetation_raster)
```



While the function `crop` is cutting rasters according to extensions, therefore in rectangular shapes, you may also desire to cut rasters according to shapes of polygons, for example to get all altitude values of pixels included in Ethiopia.

```
> Altitude <- mask(ClimData[["alt"]], Ethiopia)
> plot(Altitude)
```

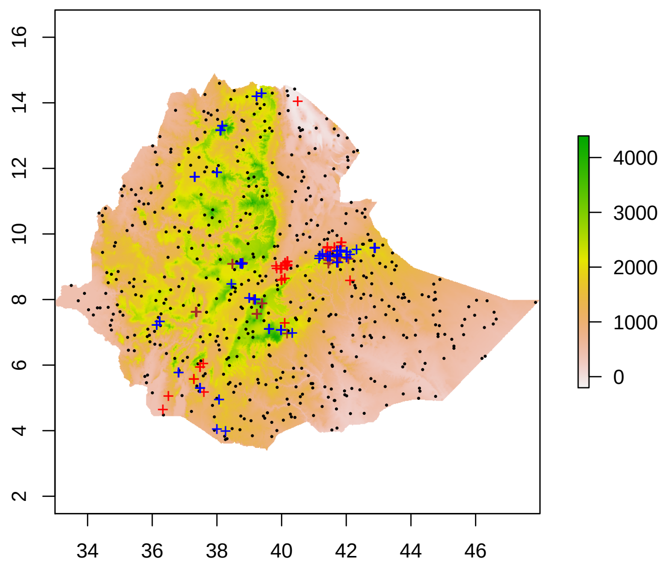


4.5 Extracting Data from Rasters

In the folder `inputs` you will find three files in csv format (comma separated values). Those are tables containing observations of occurrence of three woody species, namely *Acacia mellifera*, *Hagenia abyssinica* and *Juniperus procera*. Such observations were obtained from herbarium vouchers, which data are downloaded from Global Biodiversity Information Facility by using the package `rgbif`. The mentioned data sets contain those observations that are geo-referenced and made in Ethiopia.

In the following `command lines` we will display the location of observations for each species. Additionally, we will select a set of 500 random points distributed across Ethiopia in order to explore climatic conditions in the whole country (the background). Random points will be selected by using the function `randomPoints` of the package `dismo`.

```
> Acacia <- read.csv("inputs/Acacia_mellifera.csv")
> Hagenia <- read.csv("inputs/Hagenia_abyssinica.csv")
> Juniperus <- read.csv("inputs/Juniperus_procera.csv")
> # Selecting random points from background
> library(dismo)
> Background <- randomPoints(Altitude, 500)
> # Display
> plot(Altitude)
> points(Background, pch=20, cex=0.3)
> points(Acacia[,c("decimalLongitude","decimalLatitude")], pch="+", col="red")
> points(Hagenia[,c("decimalLongitude","decimalLatitude")], pch="+", col="brown")
> points(Juniperus[,c("decimalLongitude","decimalLatitude")], pch="+", col="blue")
```

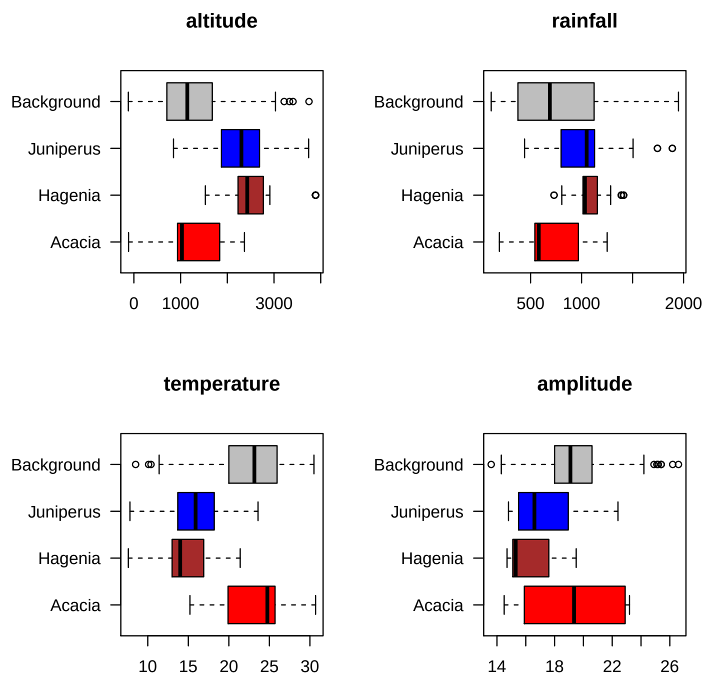


Botanists and plant ecologists will wander, if some climatic conditions may determine the occurrence of those species. In other words, which are the preferences of those species regarding climatic conditions (ecological answer). So, we will create a `SpatialPointsDataFrame` object including all the points shown in the previous map in order to extract the values of climatic variables from the object `ClimData`.

```
> Distrib <- list()
> for(i in c("Acacia","Hagenia","Juniperus")) {
+         Distrib[[i]] <- with(get(i), data.frame(x=decimalLongitude,
+                                          y=decimalLatitude, species=i))
+ }
> Distrib[["Background"]] <- data.frame(x=Background[,1], y=Background[,2],
+                 species="Background")
> Distrib <- do.call(rbind, Distrib)
> # Now convert it to spatial object
> coordinates(Distrib) <- ~ x + y
> proj4string(Distrib) <- CRS("+proj=longlat +datum=WGS84")
```

With this object we can now extract the values of climatic variables using the function `extract`.

```
> Distrib <- extract(ClimData, Distrib, sp=TRUE)
> # Display
> par(mfrow=c(2,2), las=1, mar=c(3,7,5,1))
> boxplot(alt ~ species, data=Distrib, col=c("red","brown","blue","grey"),
+                 main="altitude", horizontal=TRUE)
> boxplot(bio_12 ~ species, data=Distrib, col=c("red","brown","blue","grey"),
+                 main="rainfall", horizontal=TRUE)
> boxplot(bio_1 ~ species, data=Distrib, col=c("red","brown","blue","grey"),
+                 main="temperature", horizontal=TRUE)
> boxplot(amplitude ~ species, data=Distrib, col=c("red","brown","blue","grey"),
+                 main="amplitude", horizontal=TRUE)
```



After a quick view on the resulting boxplots, we can stat that *H. abyssinica* and *J. procera* prefer high altitudes and places with relatively high rainfall, low temperatures and low amplitude. On the other hand, *A. mellifera* is a species growing in lowlands, in places with low rainfall and high temperatures, while it seems to be relatively indifferent to amplitude of temperature.

4.6 Exercises

- Create a `SpatialPointsDataFrame` for the observations of each woody species as explained in chapter 4.4. Plot them in a map with a personalised layout and add a legend (look the help for `legend`).
- Extract altitude (or other environmental variable) for each type of Potential Natural Vegetation from Ethiopia.
- Check in the `rasters` folder for the SRTM files. Process them to get a digital elevation model with the shape of Ethiopia.
- In the tracklog Marigat to Niahururu, extract altitude values from the provided elevation model (`alt.grd`). Compare values from elevation model with those measured by the GPS device.
- Draw an altitude profile for the trip from Addis Ababa to Dire Dawa extracting altitude values from the provided elevation model. Use for the profile alternativelly distance progress and longitude as x axis.
- Check the climatic distribution of towns in Ethiopia, comparing with the background.

5 References

5.1 Cited and Recommended Literature

Adler (2010). *R in a nutshell.* O'Reilly Media. PDF Online

Bivand & Gebhardt (2000). Implementing functions for spatial statistical analysis using the R language. *J Geogr Syst* 2: 307–317.

Bivand et al. (2008). *Applied spatial datan analysis with R.* Springer. PDF Online

Borcard et al. (2011). *Numerical ecology with R.* Springer.

Burns (2011). *The R inferno.* PDF Online

Crawley (2007). *The R book.* Wiley. PDF Online

Friis et al. (2010). Atlas of the potential vegetation of Ethiopia. *Biol Skr Dan Vid Selsk* 58: 307 pp.

Hengl (2009). *A practical guide to geostatistical mapping.* University of Amsterdam. PDF Online

Hijmans et al. (2005). Very high resolution interpolated climate surfaces for global land areas. *Int J Climatol* 25: 1965–1978.

Leipzig & Li (2009). *Data mashups in R.* O'Reilly Media. PDF Online

Murrell (2006). *R graphics.* Chapman & Hall.

Paradis (2005). *R for beginners.* Université Montpellier. PDF Online

Pebesma & Bivand (2005). Classes and methods for spatial data in R. *R News* 5(2): 9–13.

Robinson (2010). *IcebreakR.* University of Melbourne. PDF Online

Venables et al. (2013). *An introduction to R.* R Development Core Team. PDF Online

Verzani (2001). *SimpleR – using R for introductory statistics.* R Development Core Team. PDF Online

5.2 Links

Software:

R

RStudio

QGIS

Additional Material:

R Graphic Gallery

R Reference Card

R Reference Card for Data Mining