Spatial data in R: Using R as a GIS

A tutorial to perform basic operations with spatial data in R, such as importing and exporting data (both vectorial and raster), plotting, analysing and making maps.

Francisco Rodriguez-Sanchez

v 2.1

18-12-2013

Check out code and latest version at GitHub

CONTENTS

1. INTRODUCTION

R is great not only for doing statistics, but also for many other tasks, including GIS analysis and working with spatial data. For instance, R is capable of doing wonderful maps such as this or this. In this tutorial I will show some basic GIS functionality in R.

Basic packages

```
library(sp)  # classes for spatial data
library(raster)  # grids, rasters
library(rasterVis)  # raster visualisation
library(maptools)
library(rgeos)
# and their dependencies
```

There are many other useful packages, e.g. check CRAN Spatial Task View. Some of them will be used below.

Back to Contents

2. GENERIC MAPPING

Retrieving base maps from Google with `gmap` function in package `dismo`

Some examples:

Getting maps for countries:

```
library(dismo)

mymap <- gmap("France")  # choose whatever country
plot(mymap)
```

Choose map type:

```
mymap <- gmap("France", type = "satellite")
plot(mymap)
```

Choose zoom level:

```
mymap <- gmap("France", type = "satellite", exp = 3)
plot(mymap)
```

Save the map as a file in your working directory for future use

```
mymap <- gmap("France", type = "satellite", filename = "France.gmap")
```

Now get a map for a region drawn at hand

```

mymap <- gmap("Europe")
plot(mymap)

select.area <- drawExtent()
# now click 2 times on the map to select your region
mymap <- gmap(select.area)
plot(mymap)
# See ?gmap for many other possibilities
```

`RgoogleMaps`: Map your data onto Google Map tiles

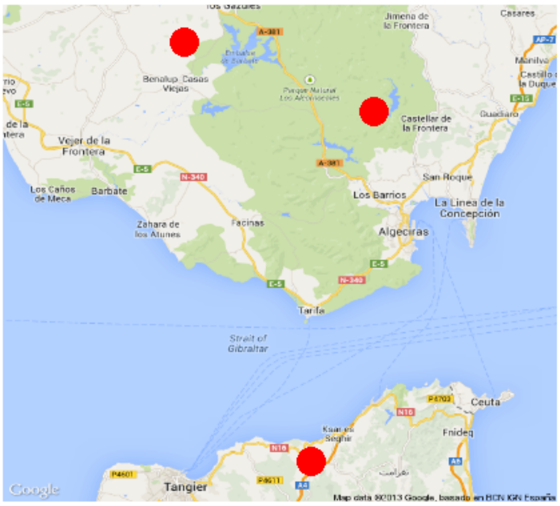
```
library(RgoogleMaps)
```

Get base maps from Google (a file will be saved in your working directory)

```
newmap <- GetMap(center = c(36.7, -5.9), zoom = 10, destfile = "newmap.png",
    maptype = "satellite")

# Now using bounding box instead of center coordinates:
newmap2 <- GetMap.bbox(lonR = c(-5, -6), latR = c(36, 37), destfile = "newmap2.png",
    maptype = "terrain")

# Try different maptypes
newmap3 <- GetMap.bbox(lonR = c(-5, -6), latR = c(36, 37), destfile = "newmap3.png",
    maptype = "satellite")
```

Now plot data onto these maps, e.g. these 3 points

```
PlotOnStaticMap(lat = c(36.3, 35.8, 36.4), lon = c(-5.5, -5.6, -5.8), zoom = 10,
    cex = 4, pch = 19, col = "red", FUN = points, add = F)
```



googleVis: visualise data in a web browser using Google Visualisation API

```
library(googleVis)
```

Run `demo(googleVis)` to see all the possibilities

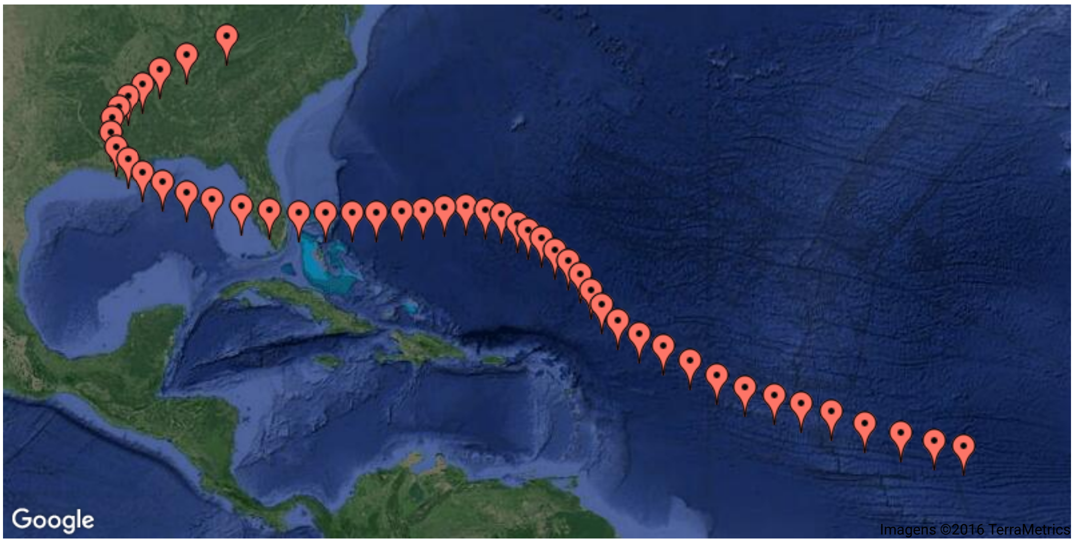
Example: plot country-level data

```
data(Exports)    # a simple data frame
Geo <- gvisGeoMap(Exports, locationvar="Country", numvar="Profit",
                  options=list(height=400, dataMode='regions'))
plot(Geo)
```

Using `print(Geo)` we can get the HTML code to embed the map in a web page!

Example: Plotting point data onto a google map (internet)

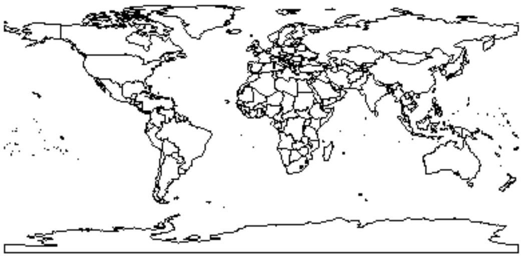
```
data(Andrew)
M1 <- gvisMap(Andrew, "LatLong", "Tip",
              options=list(showTip=TRUE, showLine=F, enableScrollWheel=TRUE,
                           mapType='satellite', useMapTypeControl=TRUE, width=800,height=400))
plot(M1)
```



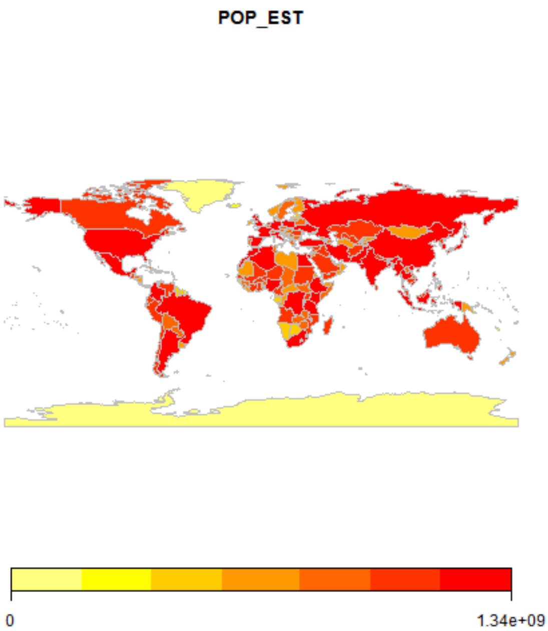
`RWorldMap`: mapping global data

Some examples

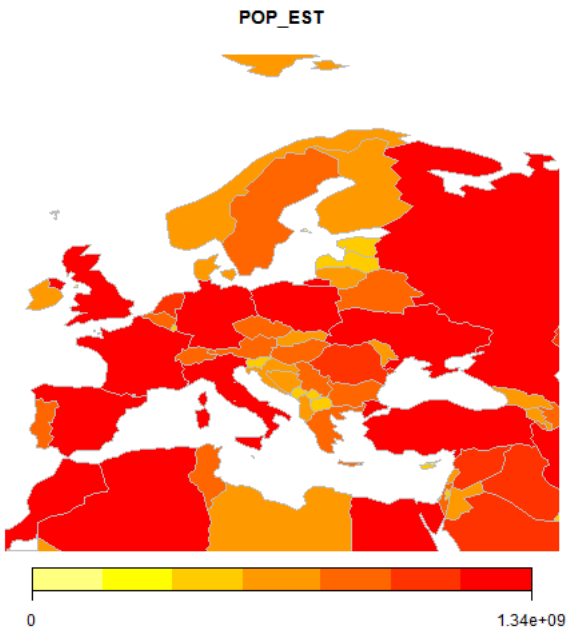
```
library(rworldmap)

newmap <- getMap(resolution = "coarse")  # different resolutions available
plot(newmap)
```



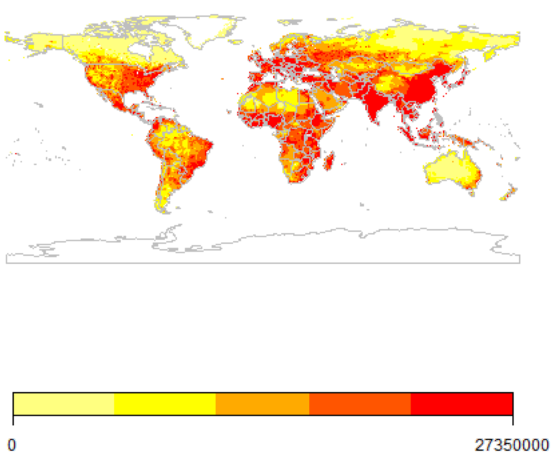
```
mapCountryData()
```



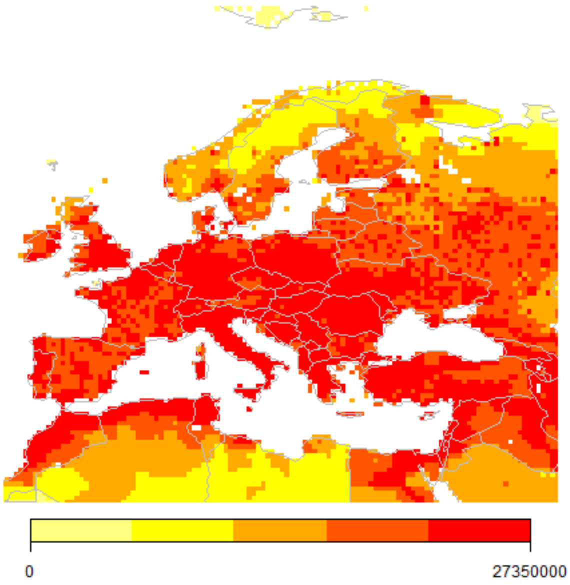

```
mapCountryData(mapRegion = "europe")
```




```
mapGriddedData()
```




```
mapGriddedData(mapRegion = "europe")
```



Back to Contents

3. SPATIAL VECTOR DATA (points, lines, polygons)

Example dataset: retrieve point occurrence data from GBIF

Let's create an example dataset: retrieve occurrence data for the laurel tree (Laurus nobilis) from the Global Biodiversity Information Facility (GBIF)

```
library(dismo)  # check also the nice 'rgbif' package!
laurus <- gbif("Laurus", "nobilis")
```

```
## Laurus nobilis : 2120 occurrences found
## 1-1000-2000-2120
```

```
# get data frame with spatial coordinates (points)
locs <- subset(laurus, select = c("country", "lat", "lon"))
head(locs)  # a simple data frame with coordinates
```

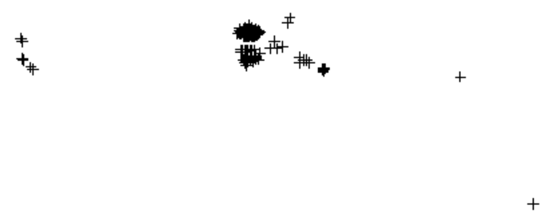
```
##   country   lat    lon
## 1   Spain 36.12 -5.579
## 2   Spain 38.26 -5.207
## 3   Spain 36.11 -5.534
## 4   Spain 36.87 -5.312
## 5   Spain 37.30 -1.918
## 6   Spain 36.10 -5.545
```

```
# Discard data with errors in coordinates:
locs <- subset(locs, locs$lat < 90)
```

Making data 'spatial'

So we have got a simple dataframe containing spatial coordinates. Let's make these data explicitly *spatial*

```
coordinates(locs) <- c("lon", "lat")  # set spatial coordinates
plot(locs)
```



Define spatial projection

Important: define geographical projection. Consult the appropriate PROJ.4 description here: http://www.spatialreference.org/

```
crs.geo <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84")  # geographical, datum WGS84
proj4string(locs) <- crs.geo  # define projection system of our data
summary(locs)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##         min    max
## lon -123.25 145.04
## lat  -37.78  59.84
## Is projected: FALSE
## proj4string :
## [+proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0]
## Number of points: 2109
## Data attributes:
##    Length     Class      Mode
##      2109 character character
```

Quickly plotting point data on a map

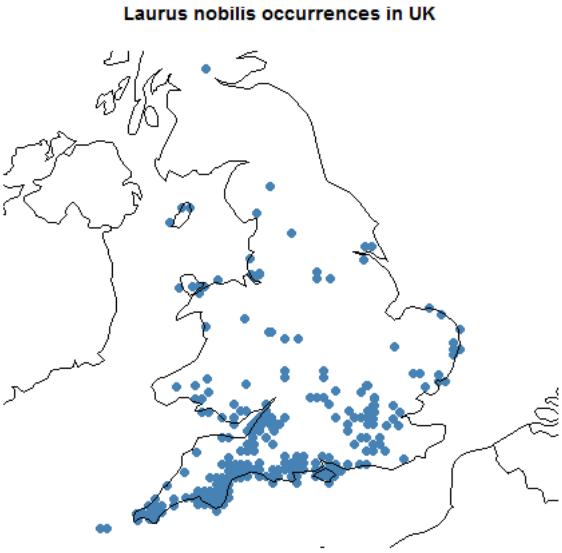
```
plot(locs, pch = 20, col = "steelblue")
library(rworldmap)
# library rworldmap provides different types of global maps, e.g:
data(coastsCoarse)
data(countriesLow)
plot(coastsCoarse, add = T)
```

Subsetting and mapping again

```
table(locs$country)  # see localities of Laurus nobilis by country
```

```
##
##      Australia         Canada        Croatia         France        Germany
##              2              1              1              1              1
##         Greece        Ireland         Israel          Italy          Spain
##              5             69           1231              2            206
##         Sweden United Kingdom  United States
##              2            578             10
```

```
locs.gb <- subset(locs, locs$country == "United Kingdom")  # select only locs in UK
plot(locs.gb, pch = 20, cex = 2, col = "steelblue")
title("Laurus nobilis occurrences in UK")
plot(countriesLow, add = T)
```




```
summary(locs.gb)
```

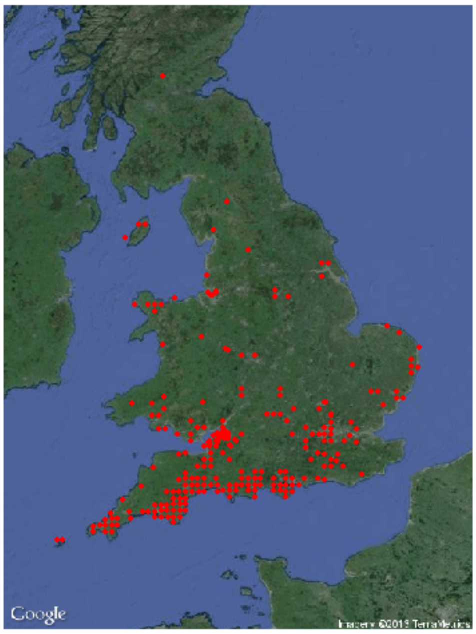
```
## Object of class SpatialPointsDataFrame
## Coordinates:
##        min    max
## lon -6.392  1.772
## lat 49.951 56.221
## Is projected: FALSE
## proj4string :
## [+proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0]
## Number of points: 578
## Data attributes:
##    Length     Class      Mode
##       578 character character
```

Mapping vectorial data (points, polygons, polylines)

Mapping vectorial data using gmap from `dismo`

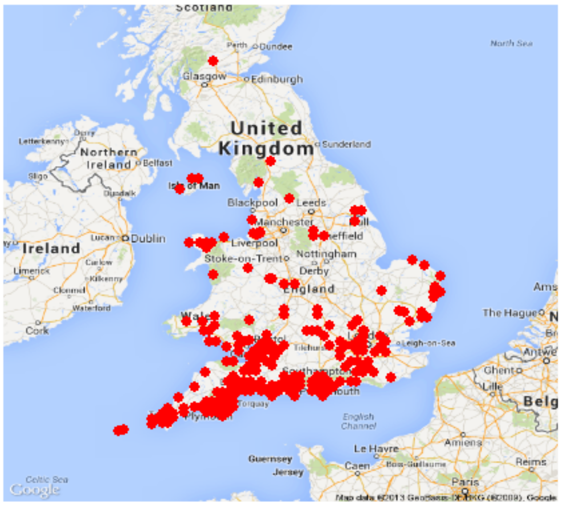
```
gbmap <- gmap(locs.gb, type = "satellite")
locs.gb.merc <- Mercator(locs.gb)  # Google Maps are in Mercator projection.
# This function projects the points to that projection to enable mapping
plot(gbmap)
```

```
points(locs.gb.merc, pch = 20, col = "red")
```



Mapping vectorial data with RgoogleMaps

```
require(RgoogleMaps)

locs.gb.coords <- as.data.frame(coordinates(locs.gb))  # retrieves coordinates
# (1st column for longitude, 2nd column for latitude)
PlotOnStaticMap(lat = locs.gb.coords$lat, lon = locs.gb.coords$lon, zoom = 5,
    cex = 1.4, pch = 19, col = "red", FUN = points, add = F)
```

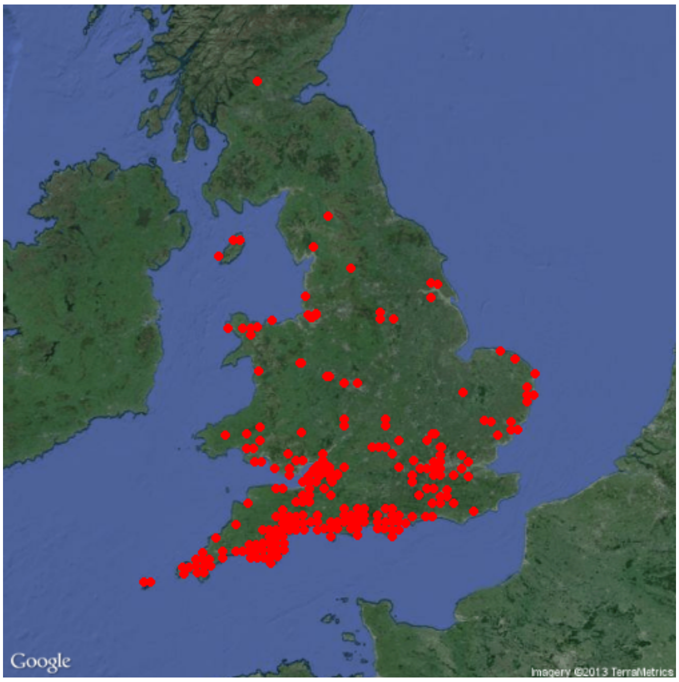


Download base map from Google Maps and plot onto it

```
map.lim <- qbbox(locs.gb.coords$lat, locs.gb.coords$lon, TYPE = "all")  # define region
# of interest (bounding box)
mymap <- GetMap.bbox(map.lim$lonR, map.lim$latR, destfile = "gmap.png", maptype = "satellite")
```

```
## [1] "http://maps.google.com/maps/api/staticmap?
center=53.086237,-2.30987445&zoom=6&size=640x640&maptype=satellite&format=png32&sensor=true"
```

```
# see the file in the wd
PlotOnStaticMap(mymap, lat = locs.gb.coords$lat, lon = locs.gb.coords$lon, zoom = NULL,
    cex = 1.3, pch = 19, col = "red", FUN = points, add = F)
```

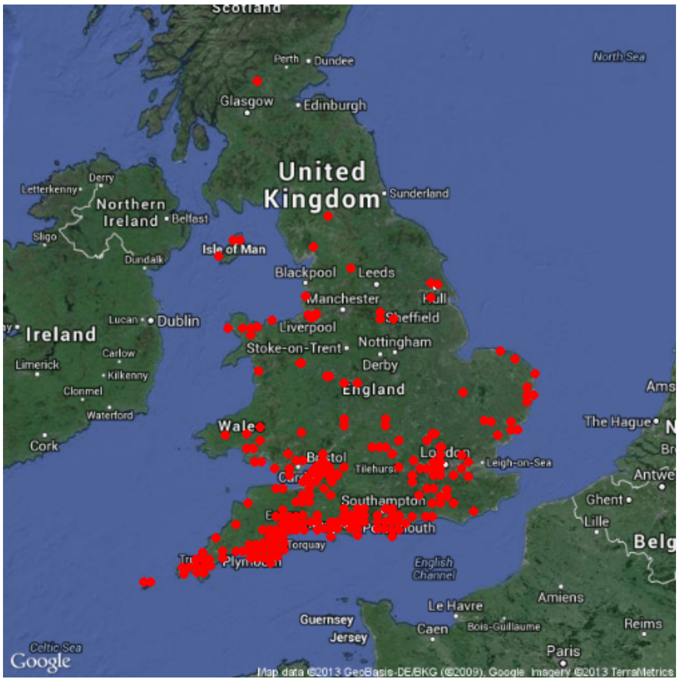


Using different background (base map)

```
mymap <- GetMap.bbox(map.lim$lonR, map.lim$latR, destfile = "gmap.png", maptype = "hybrid")
```

```
## [1] "http://maps.google.com/maps/api/staticmap?
center=53.086237,-2.30987445&zoom=6&size=640x640&maptype=hybrid&format=png32&sensor=true"
```

```
PlotOnStaticMap(mymap, lat = locs.gb.coords$lat, lon = locs.gb.coords$lon, zoom = NULL,
    cex = 1.3, pch = 19, col = "red", FUN = points, add = F)
```



Map vectorial data with `googleVis` (internet)

```
points.gb <- as.data.frame(locs.gb)
points.gb$latlon <- paste(points.gb$lat, points.gb$lon, sep=":")
map.gb <- gvisMap(points.gb, locationvar="latlon", tipvar="country",
                  options = list(showTip=T, showLine=F, enableScrollWheel=TRUE,
                           useMapTypeControl=T, width=1400,height=800))
plot(map.gb)
```

Some of the data rows were truncated ×

Dados cartogr

```
#print(map.gb)    # get HTML suitable for a webpage
```

Drawing polygons and polylines (e.g. for digitising)

```
plot(gbmap)
mypolygon <- drawPoly()  # click on the map to draw a polygon and press ESC when finished
summary(mypolygon)  # now you have a spatial polygon! Easy, isn't it?
```

Converting between formats, reading in, and saving spatial vector data

Exporting KML (Google Earth)

```
writeOGR(locs.gb, dsn = "locsgb.kml", layer = "locs.gb", driver = "KML")
```

Reading KML

```
newmap <- readOGR("locsgb.kml", layer = "locs.gb")
```

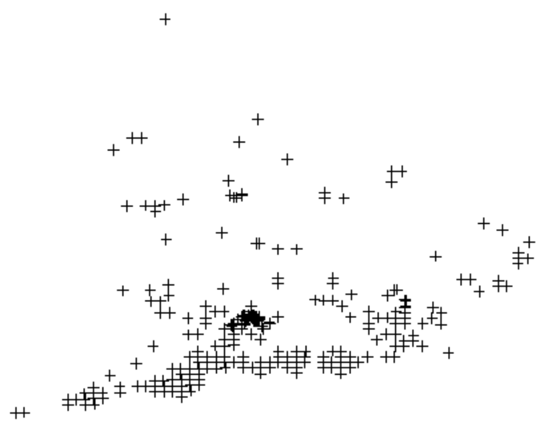
```
## OGR data source with driver: KML
## Source: "locsgb.kml", layer: "locs.gb"
## with 578 features and 2 fields
## Feature type: wkbPoint with 2 dimensions
```

Save as shapefile

```
writePointsShape(locs.gb, "locsgb")
```

Reading shapefiles

```
gb.shape <- readShapePoints("locsgb.shp")
plot(gb.shape)
```



Use `readShapePoly` to read polygon shapefiles, and `readShapeLines` to read polylines. See also `shapefile` in `raster` package.

Changing projection of spatial vector data

`spTransform` (package `sp`) will do the projection as long as the original and new projection are correctly specified.

Projecting point dataset

To illustrate, let's project the dataframe with Laurus nobilis coordinates that we obtained above:

```
summary(locs)
```

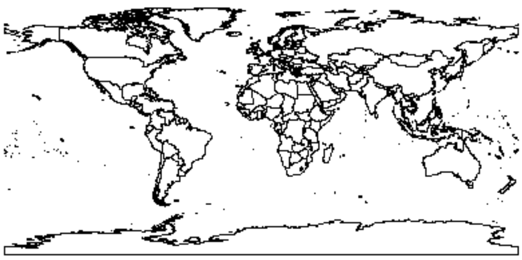
```
## Object of class SpatialPointsDataFrame
## Coordinates:
##         min    max
## lon -123.25 145.04
## lat  -37.78  59.84
## Is projected: FALSE
## proj4string :
## [+proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0]
## Number of points: 2109
## Data attributes:
##    Length     Class      Mode
##      2109 character character
```

The original coordinates are in lat lon format. Let's define the new desired projection: Lambert Azimuthal Equal Area in this case (look up parameters at http://spatialreference.org)

```
crs.laea <- CRS("+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80 +units=m
+no_defs")  # Lambert Azimuthal Equal Area
locs.laea <- spTransform(locs, crs.laea)  # spTransform makes the projection
```

Projecting shapefile of countries

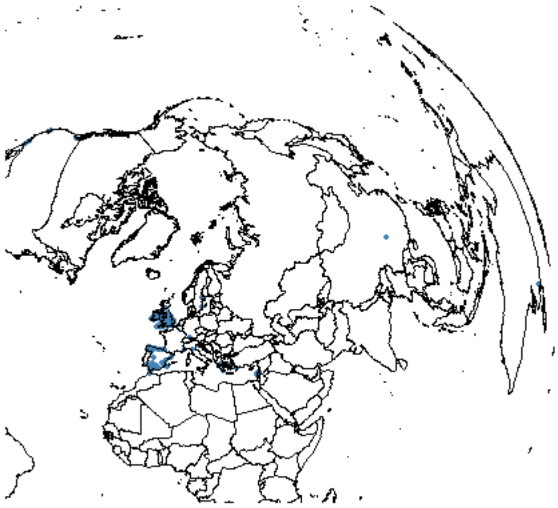
```
plot(countriesLow)  # countries map in geographical projection
```



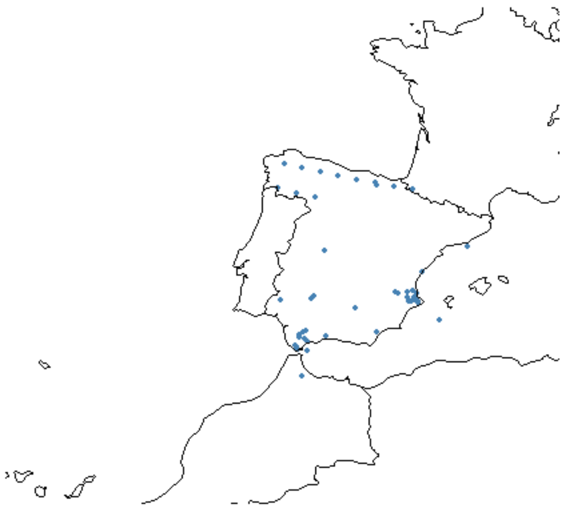
```
country.laea <- spTransform(countriesLow, crs.laea)  # project
```

Let's plot this:

```
plot(locs.laea, pch = 20, col = "steelblue")
plot(country.laea, add = T)
```



```
# define spatial limits for plotting
plot(locs.laea, pch = 20, col = "steelblue", xlim = c(1800000, 3900000), ylim = c(1e+06,
    3e+06))
plot(country.laea, add = T)
```



Back to Contents

4. USING RASTER (GRID) DATA

Downloading raster climate data from internet

The getData function from the dismo package will easily retrieve climate data, elevation, administrative boundaries, etc. Check also the excellent rWBclimate package by rOpenSci with additional functionality.

```
tmin <- getData("worldclim", var = "tmin", res = 10)  # this will download
# global data on minimum temperature at 10' resolution
```

Loading a raster layer

```
tmin1 <- raster(paste(getwd(), "/wc10/tmin1.bil", sep = ""))  # Tmin for January
```

Easy! The raster function reads many different formats, including Arc ASCII grids or netcdf files (see raster help). And values are stored on disk instead of memory! (useful for large rasters)

```
fromDisk(tmin1)
```

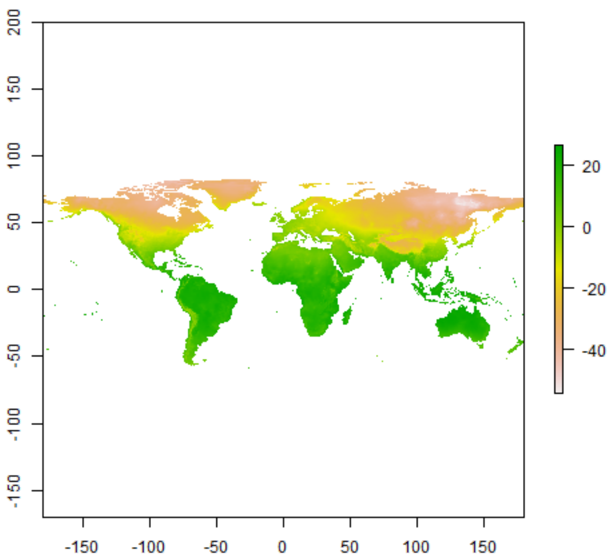
```
## [1] TRUE
```

Let's examine the raster layer:

```
tmin1 <- tmin1/10  # Worldclim temperature data come in decimal degrees
tmin1  # look at the info
```

```
## class       : RasterLayer
## dimensions  : 900, 2160, 1944000  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -180, 180, -60, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +no_defs
## data source : in memory
## names       : tmin1
## values      : -54.7, 26.6  (min, max)
```

```
plot(tmin1)
```



Creating a raster stack

A raster stack is collection of many raster layers with the same projection, spatial extent and resolution. Let's collect several raster files from disk and read them as a single raster stack:

```
library(gtools)
file.remove(paste(getwd(), "/wc10/", "tmin_10m_bil.zip", sep = ""))
```

```
## [1] TRUE
```

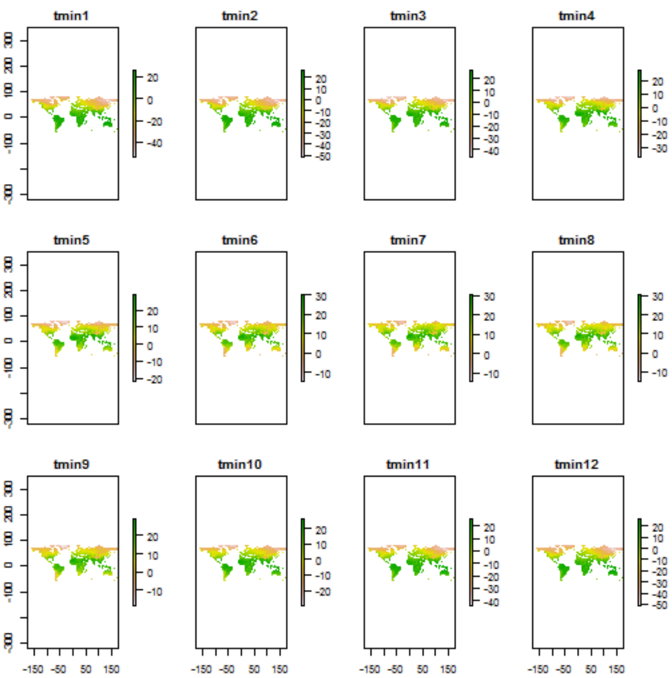
```
list.ras <- mixedsort(list.files(paste(getwd(), "/wc10/", sep = ""), full.names = T,
    pattern = ".bil"))
list.ras  # I have just collected a list of the files containing monthly temperature values
```

```
##  [1] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin1.bil"
##  [2] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin2.bil"
##  [3] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin3.bil"
##  [4] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin4.bil"
##  [5] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin5.bil"
##  [6] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin6.bil"
##  [7] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin7.bil"
##  [8] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin8.bil"
##  [9] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin9.bil"
## [10] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin10.bil"
## [11] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin11.bil"
## [12] "C:/Users/FRS/Dropbox/R.scripts/my.Rcode/R-GIS tutorial/wc10/tmin12.bil"
```

```
tmin.all <- stack(list.ras)
tmin.all
```

```
## class       : RasterStack
## dimensions  : 900, 2160, 1944000, 12  (nrow, ncol, ncell, nlayers)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -180, 180, -60, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +no_defs
## names       : tmin1, tmin2, tmin3, tmin4, tmin5, tmin6, tmin7, tmin8, tmin9, tmin10, tmin11,
tmin12
## min values  :  -547,  -525,  -468,  -379,  -225,  -170,  -171,  -178,  -192,   -302,   -449,
-522
## max values  :   266,   273,   277,   283,   295,   312,   311,   312,   300,    268,    267,
268
```

```
tmin.all <- tmin.all/10
plot(tmin.all)
```



Raster bricks

A rasterbrick is similar to a raster stack (i.e. multiple layers with the same extent and resolution), but all the data must be stored in a single file on disk.

```
tmin.brick <- brick(tmin.all)  # creates rasterbrick
```

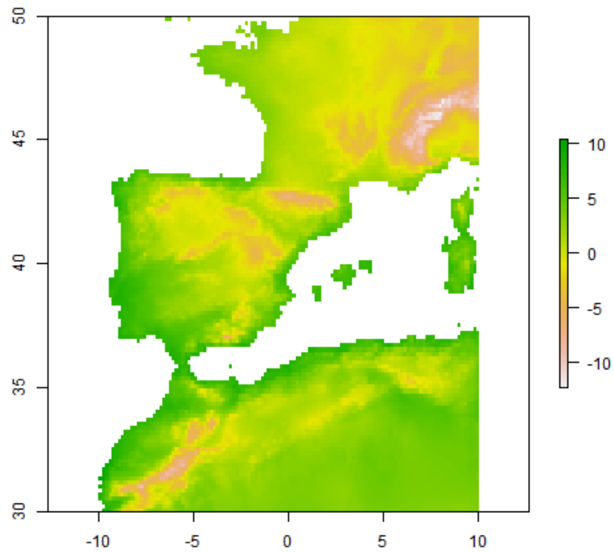
Crop rasters

Crop raster manually (drawing region of interest):

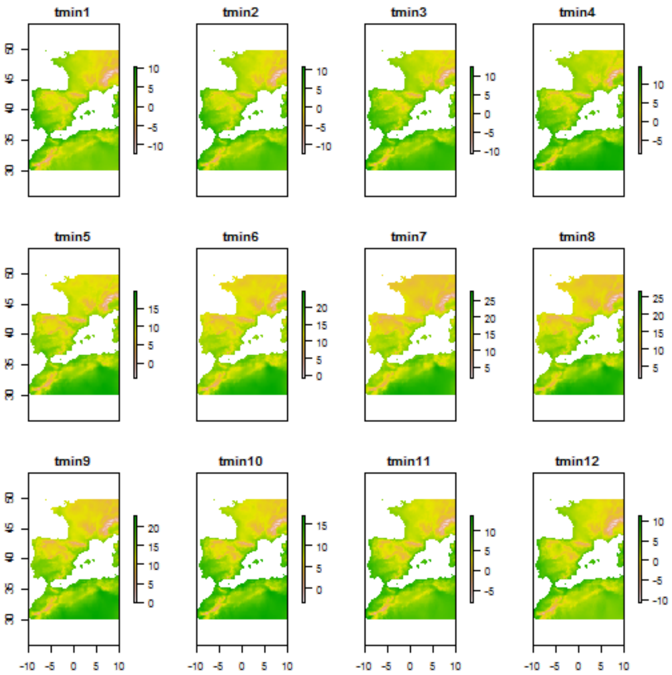
```
plot(tmin1)
newext <- drawExtent()  # click twice on the map to select the region of interest
tmin1.c <- crop(tmin1, newext)
plot(tmin1.c)
```

Alternatively, provide coordinates for the limits of the region of interest:

```
newext <- c(-10, 10, 30, 50)
tmin1.c <- crop(tmin1, newext)
plot(tmin1.c)
```




```
tmin.all.c <- crop(tmin.all, newext)
plot(tmin.all.c)
```



Define spatial projection of the rasters

```
crs.geo  # defined above
```

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

```
projection(tmin1.c) <- crs.geo
projection(tmin.all.c) <- crs.geo
tmin1.c  # notice info at coord.ref.
```

```
## class       : RasterLayer
## dimensions  : 120, 120, 14400  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : tmin1
## values      : -12.3, 10.3  (min, max)
```

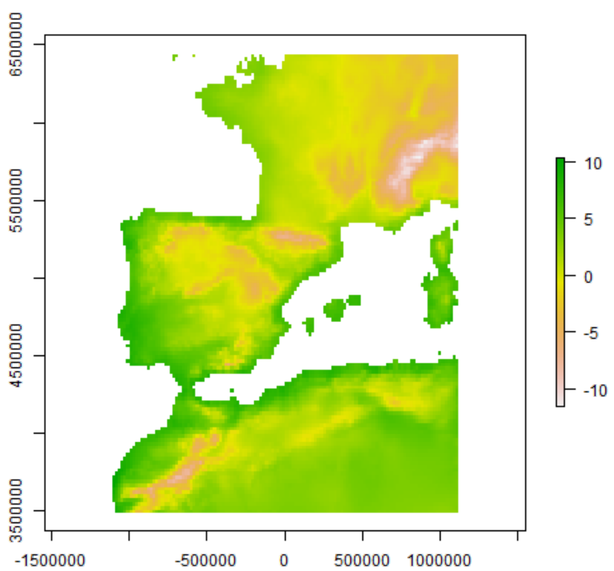
Changing projection

Use `projectRaster` function:

```
tmin1.proj <- projectRaster(tmin1.c, crs = "+proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +a=6378137
+b=6378137 +units=m +no_defs")  # can also use a template raster, see ?projectRaster
tmin1.proj  # notice info at coord.ref.
```

```
## class       : RasterLayer
## dimensions  : 132, 134, 17688  (nrow, ncol, ncell)
## resolution  : 18600, 24200  (x, y)
## extent      : -1243395, 1249005, 3372876, 6567276  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +a=6378137 +b=6378137 +units=m +no_defs
## data source : in memory
## names       : tmin1
## values      : -11.59, 10.3  (min, max)
```

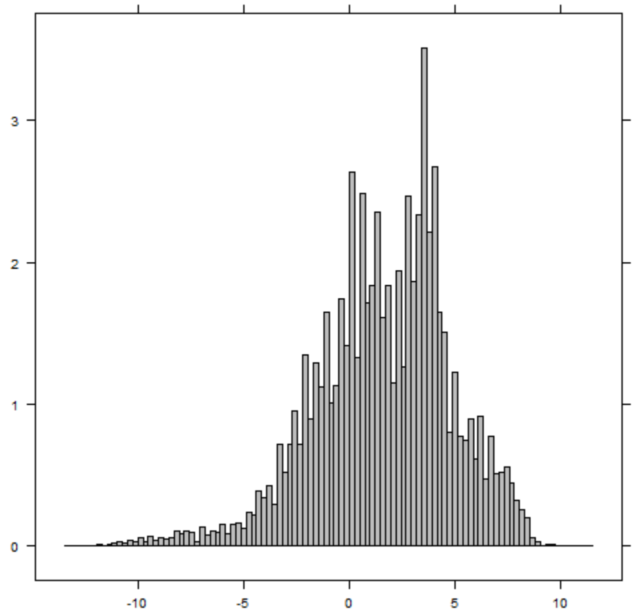
```
plot(tmin1.proj)
```



Plotting raster data

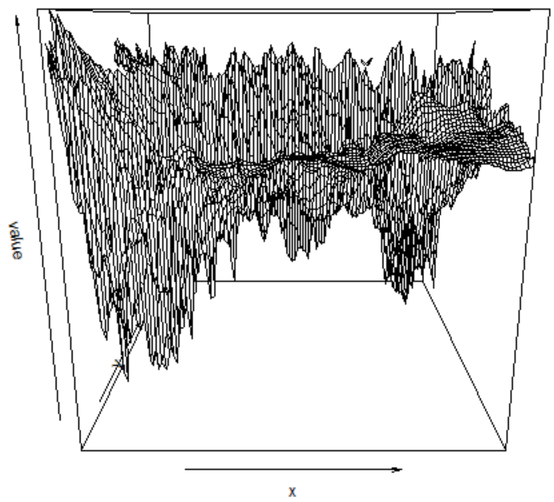
Different plotting functions:

```
histogram(tmin1.c)
```

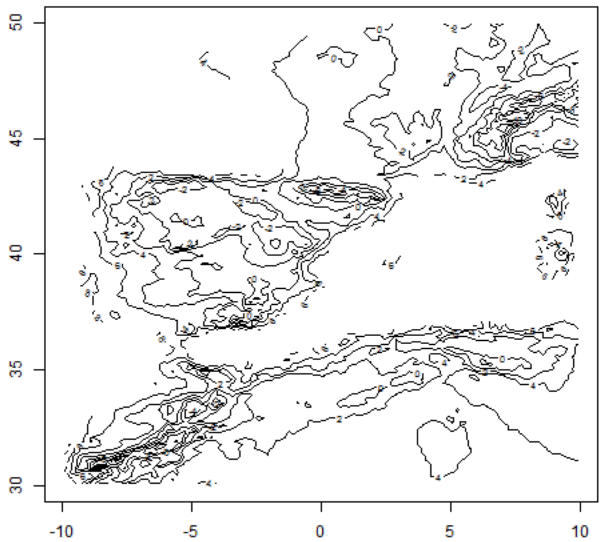



```
pairs(tmin.all.c)
```

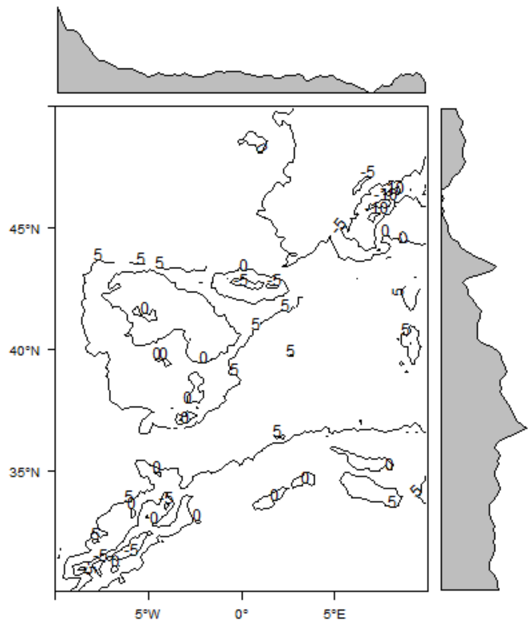
```
persp(tmin1.c)
```



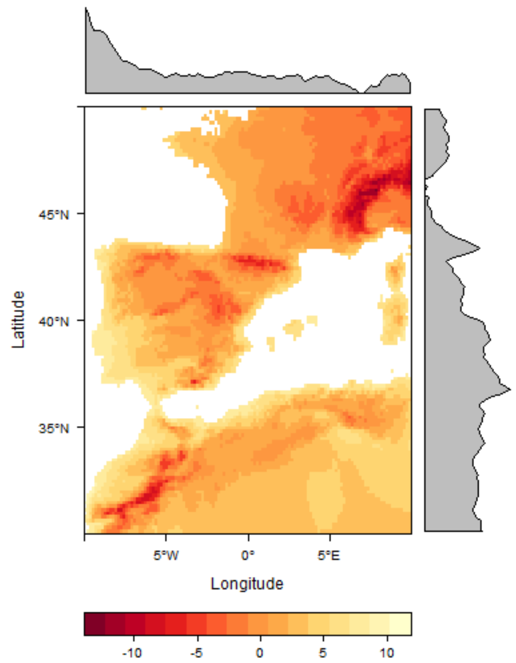

```
contour(tmin1.c)
```



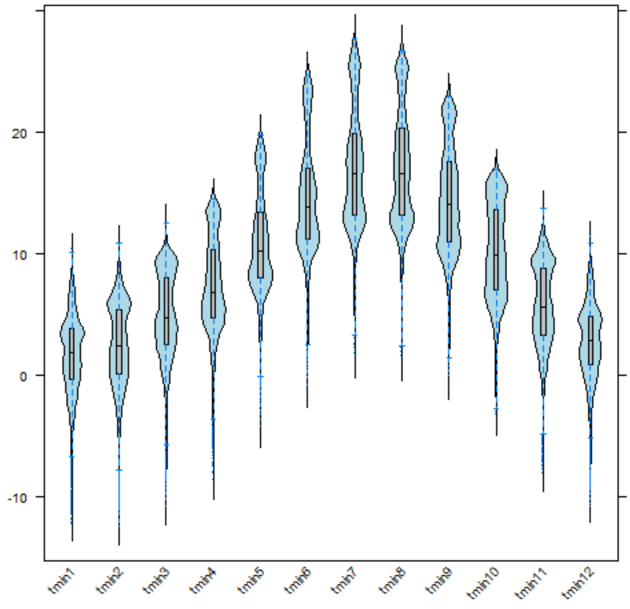

```
contourplot(tmin1.c)
```



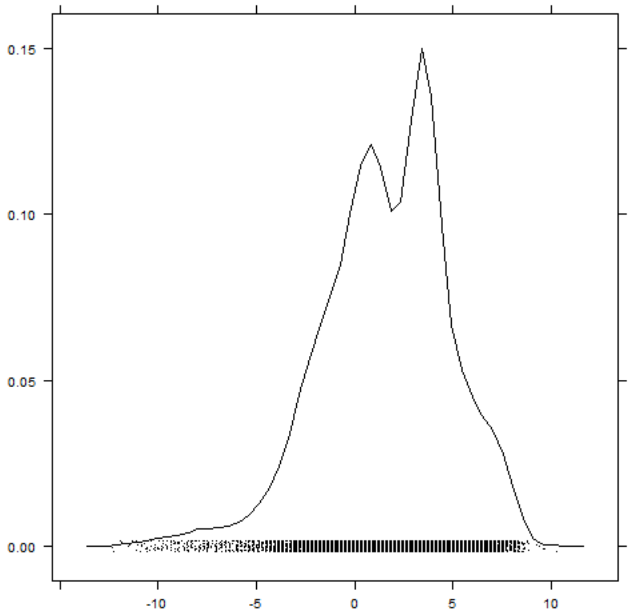

```
levelplot(tmin1.c)
```




```
# plot3D(tmin1.c)
bwplot(tmin.all.c)
```




```
densityplot(tmin1.c)
```

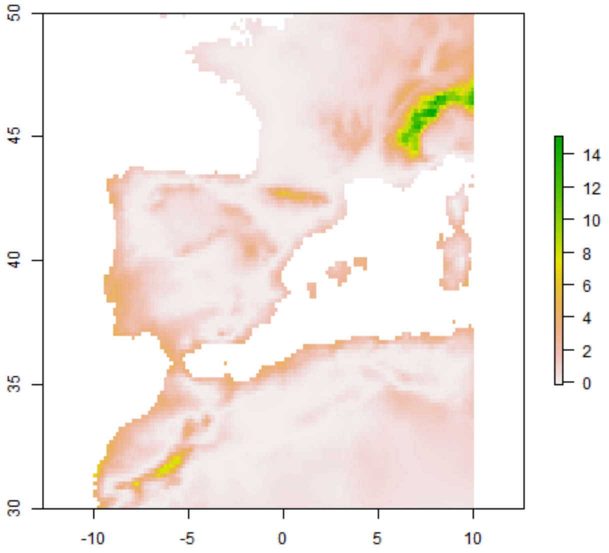


Spatial autocorrelation

```
Moran(tmin1.c)  # global Moran's I
```

```
## [1] 0.9099
```

```
tmin1.Moran <- MoranLocal(tmin1.c)
plot(tmin1.Moran)
```



Extract values from raster

Use `extract` function:

```
head(locs)  # we'll obtain tmin values for our points
```

```
##   country
## 1   Spain
## 2   Spain
## 3   Spain
## 4   Spain
## 5   Spain
## 6   Spain
```

```
projection(tmin1) <- crs.geo
locs$tmin1 <- extract(tmin1, locs)  # raster values
# are incorporated to the dataframe
head(locs)
```

```
##   country tmin1
## 1   Spain   6.7
## 2   Spain   2.1
## 3   Spain   6.7
## 4   Spain   4.2
## 5   Spain   6.2
## 6   Spain   6.7
```

You can also extract values for a given region instead of the whole raster:

```
plot(tmin1.c)
reg.clim <- extract(tmin1.c, drawExtent())  # click twice to
# draw extent of the region of interest
summary(reg.clim)
```

Using `rasterToPoints`:

```
# rasterToPoints
tminvals <- rasterToPoints(tmin1.c)
head(tminvals)
```

```
##            x     y tmin1
## [1,] -6.4167 49.92   4.2
## [2,] -6.2500 49.92   4.2
## [3,] -5.2500 49.92   2.4
## [4,]  0.5833 49.92   1.0
## [5,]  0.7500 49.92   1.0
## [6,]  0.9167 49.92   1.0
```

And also, the `click` function will get values from particular locations in the map

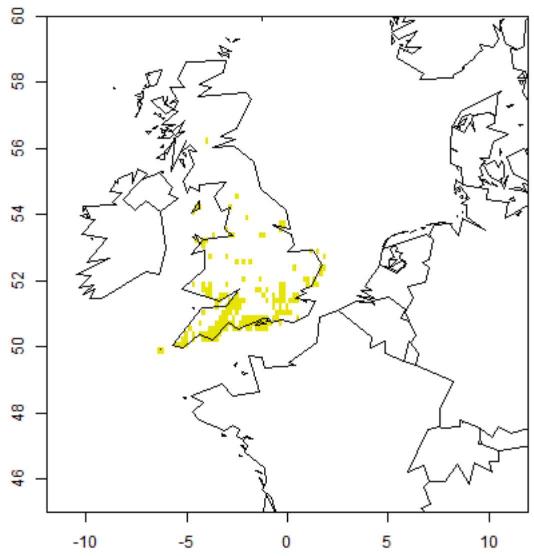
```
plot(tmin1.c)
click(tmin1.c, n = 3)  # click n times in the map to get values
```

Rasterize points, lines or polygons

```
locs2ras <- rasterize(locs.gb, tmin1, field = rep(1, nrow(locs.gb)))
locs2ras
```

```
## class       : RasterLayer
## dimensions  : 900, 2160, 1944000  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -180, 180, -60, 90  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 1, 1  (min, max)
```

```
plot(locs2ras, xlim = c(-10, 10), ylim = c(45, 60), legend = F)
data(wrld_simpl)
plot(wrld_simpl, add = T)
```



Changing raster resolution

Use `aggregate` function:

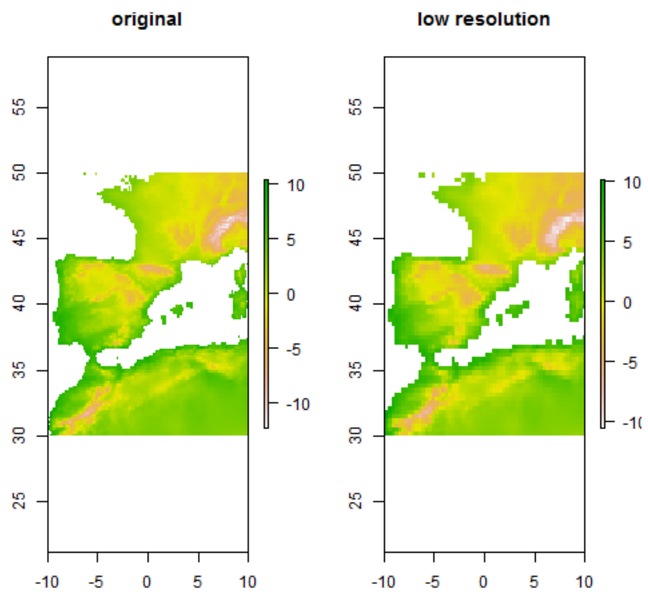
```
tmin1.lowres <- aggregate(tmin1.c, fact = 2, fun = mean)
tmin1.lowres
```

```
## class       : RasterLayer
## dimensions  : 60, 60, 3600  (nrow, ncol, ncell)
## resolution  : 0.3333, 0.3333  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : tmin1
## values      : -10.57, 10.1  (min, max)
```

```
tmin1.c  # compare
```

```
## class       : RasterLayer
## dimensions  : 120, 120, 14400  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : tmin1
## values      : -12.3, 10.3  (min, max)
```

```
par(mfcol = c(1, 2))
plot(tmin1.c, main = "original")
plot(tmin1.lowres, main = "low resolution")
```



Spline interpolation

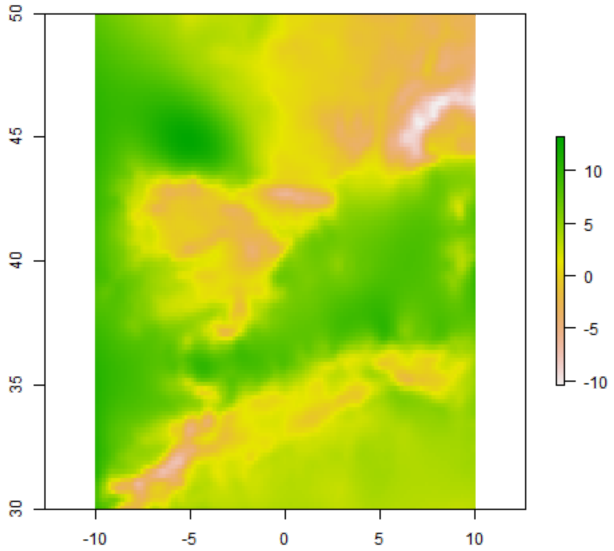
```
xy <- data.frame(xyFromCell(tmin1.lowres, 1:ncell(tmin1.lowres)))  # get raster cell coordinates
head(xy)
```

```
##        x     y
## 1 -9.833 49.83
## 2 -9.500 49.83
## 3 -9.167 49.83
## 4 -8.833 49.83
## 5 -8.500 49.83
## 6 -8.167 49.83
```

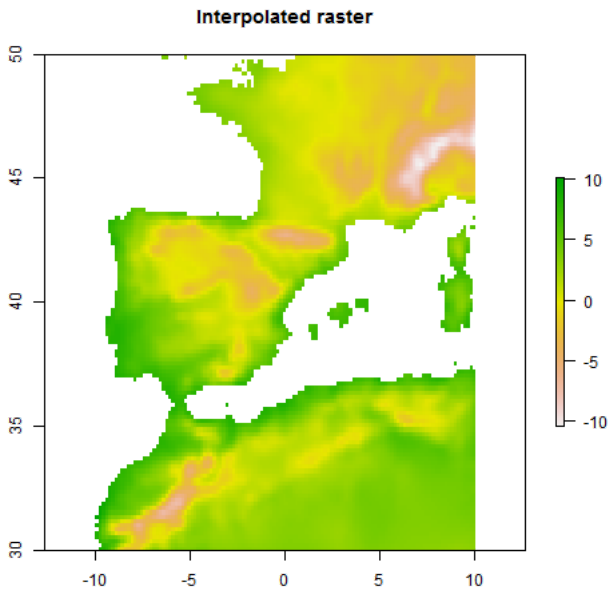
```
vals <- getValues(tmin1.lowres)
library(fields)
spline <- Tps(xy, vals)  # thin plate spline
intras <- interpolate(tmin1.c, spline)
intras  # note new resolution
```

```
## class       : RasterLayer
## dimensions  : 120, 120, 14400  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : -10.43, 13.16  (min, max)
```

```
plot(intras)
```




```
intras <- mask(intras, tmin1.c)  # mask to land areas only
plot(intras)
title("Interpolated raster")
```



Setting all rasters to the same extent, projection and resolution all in one

See `spatial_sync_raster` function from `spatial.tools` package.

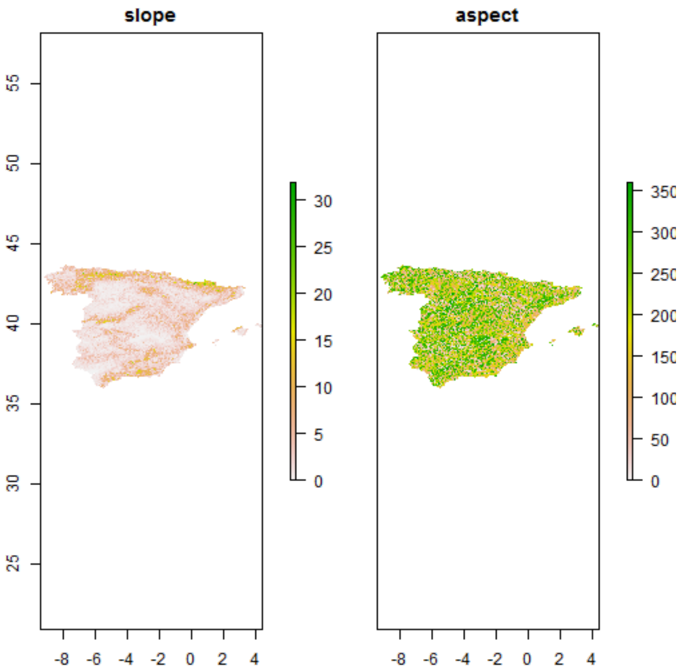
Elevations, slope, aspect, etc

Download elevation data from internet:

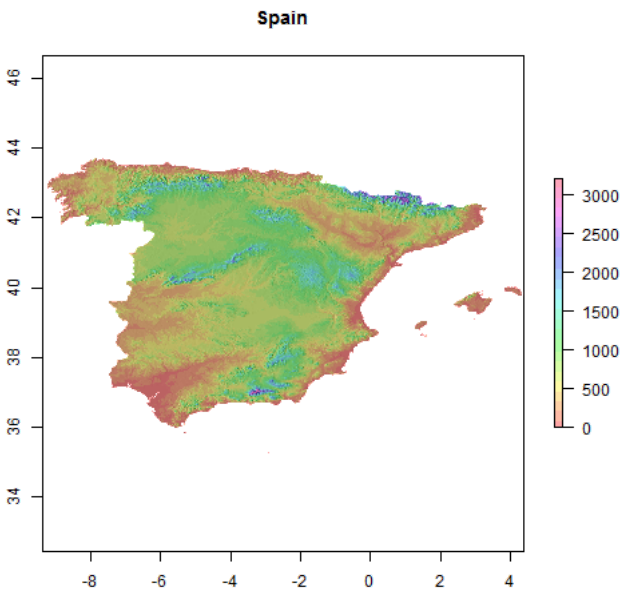
```
elevation <- getData("alt", country = "ESP")
```

Some quick maps:

```
x <- terrain(elevation, opt = c("slope", "aspect"), unit = "degrees")
plot(x)
```




```
slope <- terrain(elevation, opt = "slope")
aspect <- terrain(elevation, opt = "aspect")
hill <- hillShade(slope, aspect, 40, 270)
plot(hill, col = grey(0:100/100), legend = FALSE, main = "Spain")
plot(elevation, col = rainbow(25, alpha = 0.35), add = TRUE)
```



Saving and exporting raster data

Saving raster to file:

```
writeRaster(tmin1.c, filename = "tmin1.c.grd")
```

```
## class       : RasterLayer
## dimensions  : 120, 120, 14400  (nrow, ncol, ncell)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : C:\Users\FRS\Dropbox\R.scripts\my.Rcode\R-GIS tutorial\tmin1.c.grd
## names       : tmin1
## values      : -12.3, 10.3  (min, max)
```

```
writeRaster(tmin.all.c, filename = "tmin.all.grd")
```

```
## class       : RasterBrick
## dimensions  : 120, 120, 14400, 12  (nrow, ncol, ncell, nlayers)
## resolution  : 0.1667, 0.1667  (x, y)
## extent      : -10, 10, 30, 50  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +ellps=WGS84 +datum=WGS84 +towgs84=0,0,0
## data source : C:\Users\FRS\Dropbox\R.scripts\my.Rcode\R-GIS tutorial\tmin.all.grd
## names       : tmin1, tmin2, tmin3, tmin4, tmin5, tmin6, tmin7, tmin8, tmin9, tmin10, tmin11,
tmin12
## min values  : -12.3, -12.5, -10.8,  -8.6,  -4.2,  -0.8,   1.8,   1.6,  -0.1,   -3.3,   -8.1,
-10.8
## max values  :  10.3,  10.8,  12.5,  14.5,  19.7,  24.7,  27.6,  26.7,  22.9,   16.9,   13.7,
11.3
```

`writeRaster` can export to many different file types, see help.

Exporting to KML (Google Earth)

```
tmin1.c <- raster(tmin.all.c, 1)
KML(tmin1.c, file = "tmin1.kml")
KML(tmin.all.c)  # can export multiple layers
```

Back to Contents

5. SPATIAL STATISTICS (just a glance)

Point pattern analysis

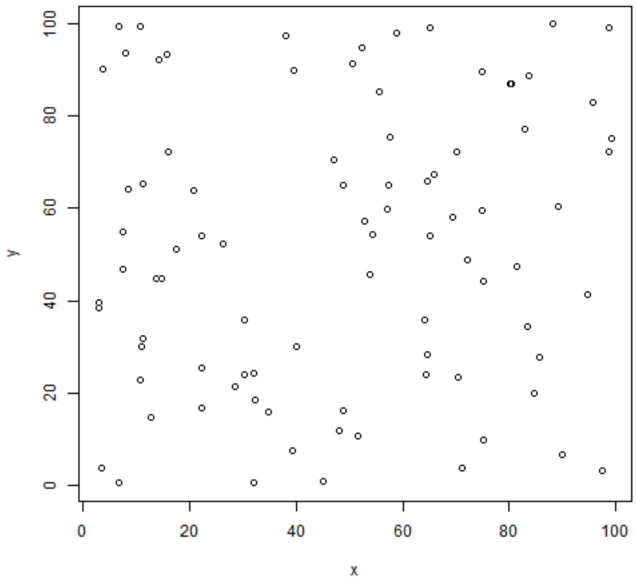
Some useful packages:

```
library(spatial)
library(spatstat)
library(spatgraphs)
library(ecespa)  # ecological focus
```

See CRAN Spatial Task View.

Let's do a quick example with Ripley's K function:

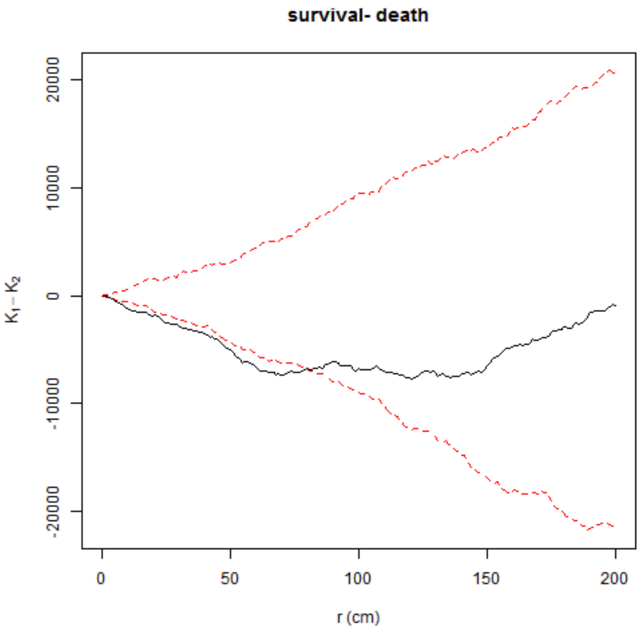
```
data(fig1)
plot(fig1)  # point pattern
```




```
data(Helianthemum)
cosa12 <- K1K2(Helianthemum, j = "deadpl", i = "survpl", r = seq(0, 200, le = 201),
    nsim = 99, nrank = 1, correction = "isotropic")
```

```
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
## 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
## 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
## 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
## 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
## 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
## 91, 92, 93, 94, 95, 96, 97, 98,  99.
```

```
plot(cosa12$k1k2, lty = c(2, 1, 2), col = c(2, 1, 2), xlim = c(0, 200), main = "survival- death",
    ylab = expression(K[1] - K[2]), legend = FALSE)
```



```
##       lty col   key         label
## lo      2   2    lo         lo(r)
## K1-K2   1   1 K1-K2 K1(r) - K2(r)
## hi      2   2    hi         hi(r)
##                                                                  meaning
## lo                                           lower pointwise envelope of simulations
## K1-K2 differences of  Ripley isotropic correction estimate of expression(K[1] - K[2])
## hi                                           upper pointwise envelope of simulations
```

Geostatistics

Some useful packages:

```
library(gstat)
library(geoR)
library(akima)  # for spline interpolation
library(spdep)  # dealing with spatial dependence
```

See CRAN Spatial Task View.

Back to Contents

6. INTERACTING WITH OTHER GIS

```
library(spgrass6)  # GRASS
library(RPyGeo)  # ArcGis (Python)
library(RSAGA)  # SAGA
library(spsextante)  # Sextante
```

Back to Contents

7. OTHER USEFUL PACKAGES

```
library(Metadata)  # automatically collates data from online GIS datasets (land cover, pop
density, etc) for a given set of coordinates

# library(GeoXp) # Interactive exploratory spatial data analysis
example(columbus)
histomap(columbus, "CRIME")

library(maptools)
# readGPS

library(rangeMapper)  # plotting species distributions, richness and traits


# Species Distribution Modelling
library(dismo)
library(biomod2)
library(SDMTools)

library(BioCalc)  # computes 19 bioclimatic variables from monthly climatic values (tmin, tmax,
prec)
```

Back to Contents

8. TO LEARN MORE

- ASDAR book
- Packages help and vignettes, especially

  http://cran.r-project.org/web/packages/raster/vignettes/Raster.pdf
  http://cran.r-project.org/web/packages/dismo/vignettes/sdm.pdf
  http://cran.r-project.org/web/packages/sp/vignettes/sp.pdf
- CRAN Task View: Analysis of Spatial Data
- Introduction to Spatial Data and ggplot2
- R spatial tips
- R wiki: tips for spatial data
- Spatial analysis in R
- Displaying time series, spatial and space-time data with R
- Notes on Spatial Data Operations in R
- Analysing spatial point patterns in R
- Spatial data in R
- NCEAS Geospatial use cases
- Spatial Analyst
- Making maps with R
- The Visual Raster Cheat Sheet
- R-SIG-Geo mailing list

Back to Contents