Spatial interpolation

Roberto Molowny-Horas (CREAF-EMF)

March 16th-17th, 2022

Point data

During our analysis we may need to carry out spatial interpolation from point data in order to show our results as 2D rasters. Rasters or images may be more useful to convey the results of our analysis than figures with clouds of points.

We first retrieve point data for the Eurasian jay, project it to Transversal Mercator and create a SpatVector.

```
suppressPackageStartupMessages(library(sf))
suppressPackageStartupMessages(library(terra))
pxy <-  vect(st_transform(st_read("Eurasian jay.gpkg", quiet = T), "epsg:3109"))
```

We get the two climatic raster and the DEM from worldclim.org and project them too.

```
t_mean <- project(rast("Annual mean temperature.tif"), "epsg:3109")
p_total <- project(rast("Annual total precipitation.tif"), "epsg:3109")
dem <- project(rast("DEM.tif"), "epsg:3109")
```

We will extract their values at the locations of the point pattern.

```
pxy$t_mean <- extract(t_mean, pxy, method="simple")$t_mean
pxy$p_total <- extract(p_total, pxy, method="simple")$p_total
pxy$dem <- extract(dem, pxy, method="simple")$dem
pxy <- na.omit(pxy,field="")
```

Maps are too big for our calculations below. Let's shrink them.

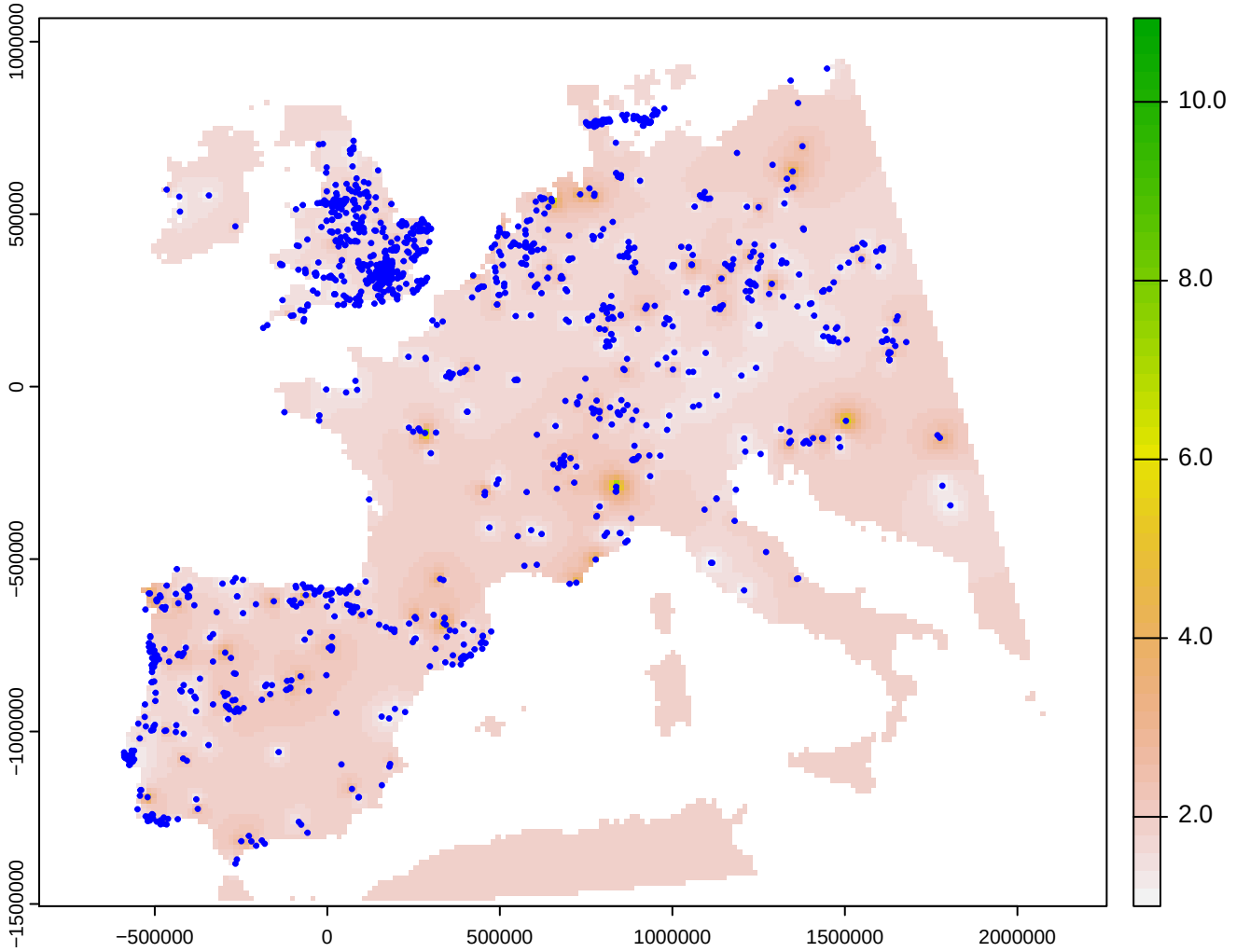
```
t_mean <- aggregate(t_mean,fact=4)
p_total <- aggregate(p_total,fact=4)
dem <- aggregate(dem,fact=4)
```

Inverse-Distance Weighted (IDW) interpolation.

Inverse-distance weighted interpolation is probably one of the simplest deterministic interpolation.

```
suppressPackageStartupMessages(library(gstat))
suppressPackageStartupMessages(library(sp))
df_idw <- data.frame(values(pxy),crds(pxy))
df_idw_new <- data.frame(crds(t_mean,na.rm=F))
coordinates(df_idw) <- ~x+y
coordinates(df_idw_new) <- ~x+y
idw.fit <- idw(individualCount~1,df_idw,df_idw_new)
rast_idw <- rast(t_mean)
values(rast_idw) <- idw.fit$var1.pred
rast_idw[is.na(t_mean)] <- NA
plot(rast_idw)
plot(pxy,add=T,col="blue",cex=.5)
```

```
## [inverse distance weighted interpolation]
```



Trend Surface

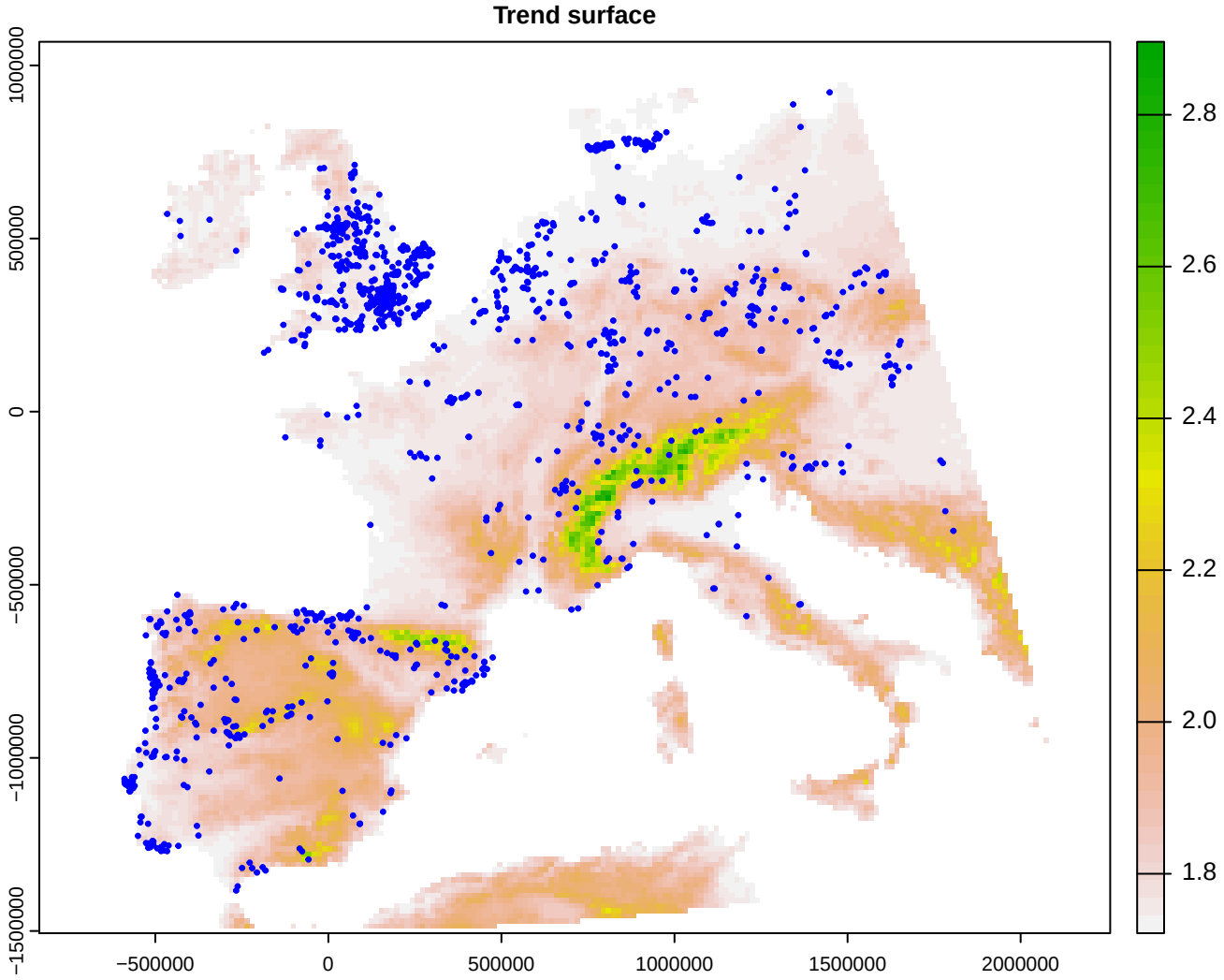
Calculating a trend surface to interpolate our data is simple.

```
re <- glm(individualCount~t_mean++p_total+I(p_total^2)+dem+I(dem^2),data=values(pxy),family=poisson)
suppressPackageStartupMessages(library(MASS))
re <- stepAIC(re,trace=0)
print(summary(re))
```

```
##
## Call:
## glm(formula = individualCount ~ dem, family = poisson, data = values(pxy))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0449  -0.6205  -0.6010   0.1949   5.8401
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 5.442e-01  2.482e-02  21.927  < 2e-16 ***
## dem         1.723e-04  6.329e-05   2.723  0.00647 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1004.18  on 1411  degrees of freedom
## Residual deviance:  997.04  on 1410  degrees of freedom
## AIC: 4350.9
##
## Number of Fisher Scoring iterations: 5
```

Next, we will derive a global map of the individual counts of the *Eurasian jay* species.

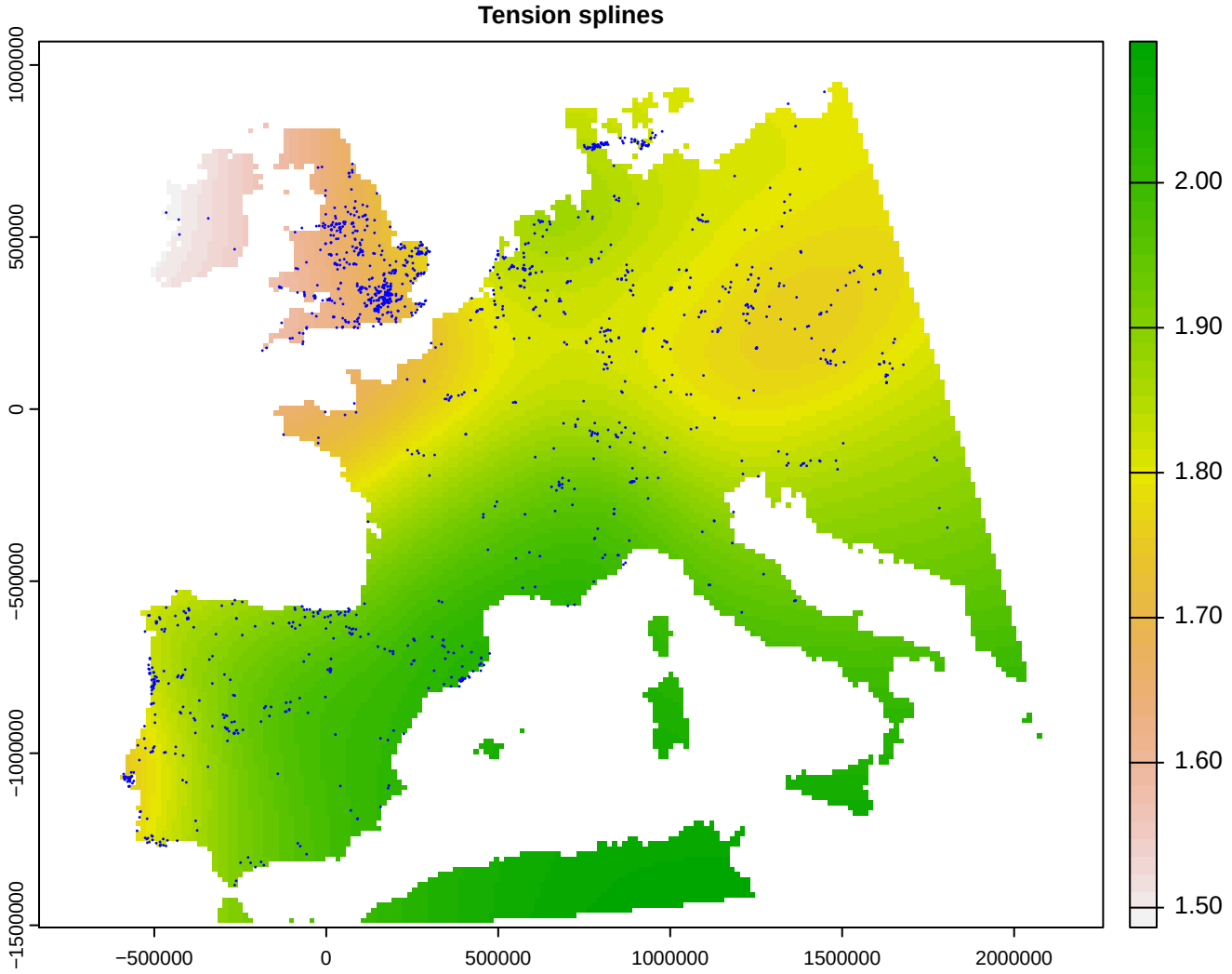
```
df_trend_new <- data.frame(t_mean=values(t_mean,na.rm=F),p_total=values(p_total,na.rm=F),
                           dem=values(dem,na.rm=F))
rast_trend <- rast(t_mean)
names(rast_trend) <- "individual"
values(rast_trend) <- predict(re,newdata=df_trend_new,type="response")
plot(rast_trend,main="Trend surface")
plot(pxy,add=T,col="blue",cex=.5)
```


Trend surface
1000000
500000
0
−500000
−1000000
−1500000
−500000
0
500000
1000000
1500000
2000000
2.8
2.6
2.4
2.2
2.0
1.8

Tension splines

A flexible spline surface is used to interpolate between points.

```
suppressPackageStartupMessages(library(fields))
tps <- Tps(crds(pxy),pxy$individualCount)
rast_tens <- interpolate(rast(t_mean),tps)
rast_tens <- mask(rast_tens,t_mean)
plot(rast_tens,main="Tension splines")
plot(pxy,add=T,pch=16,cex=.2,col="blue")
```


Tension splines
1000000
500000
0
−500000
−1000000
−1500000
−500000
0
500000
1000000
1500000
2000000
2.00
1.90
1.80
1.70
1.60
1.50

Simple and ordinary krigging

First, we prepare the data.

```
df_krig <- st_as_sf(data.frame(values(pxy),crds(pxy)),coords=c("x","y"))
df_krig_new <- st_as_sf(data.frame(crds(t_mean)),coords=c("x","y"))
```

Next, we fit the variogram with a Gaussian model.

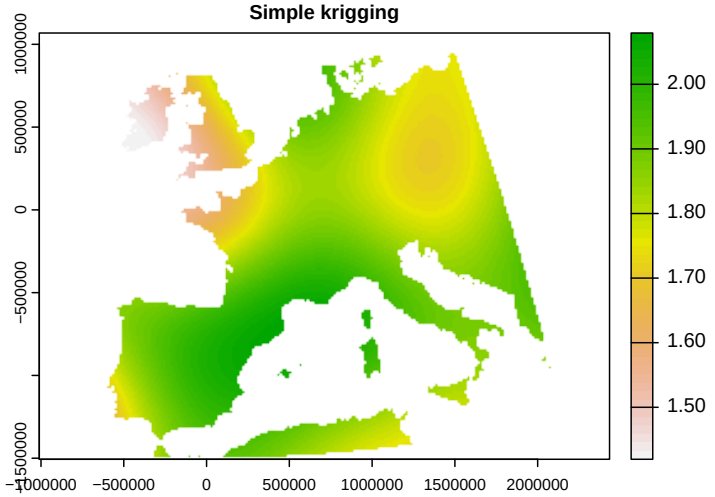
```
v <- variogram(individualCount~1,df_krig)
fitv <- fit.variogram(v, gstat::vgm(model="Gau",nugget=1.5,psill=1.8))
```

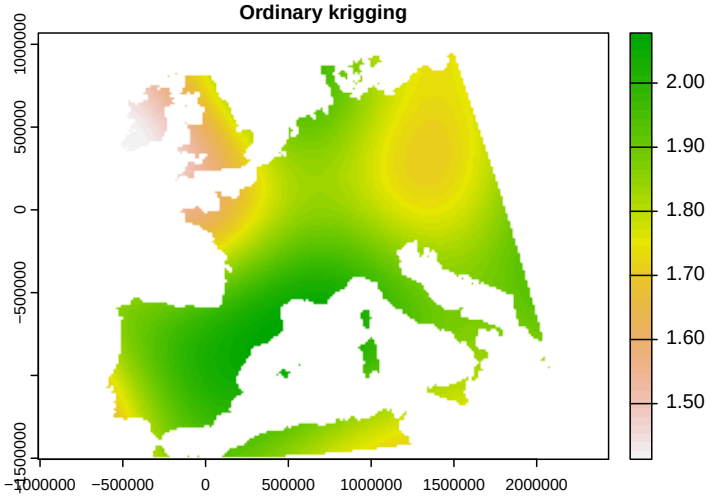
Finally, we calculate a simple and an ordinary krigging and rasterize the result.

```
krig_simple = krige(individualCount~1, df_krig, df_krig_new, model = fitv, beta = mean(pxy$individualCount))
krig_ordin = krige(individualCount~1, df_krig, df_krig_new, model = fitv)
krig_simple <- rasterize(vect(krig_simple["var1.pred"]),t_mean,field="var1.pred",fun=sum)
krig_ordin <- rasterize(vect(krig_ordin["var1.pred"]),t_mean,field="var1.pred",fun=sum)
```

```
## [using simple kriging]
## [using ordinary kriging]
```

```
par(mfrow=c(1,2))
plot(krig_simple,main="Simple krigging")
plot(krig_ordin,main="Ordinary krigging")
par(mfrow=c(1,1))
```


Simple krigging
1000000
500000
0
−500000
−1500000
−1000000
−500000
0
500000
1000000
1500000
2000000
2.00
1.90
1.80
1.70
1.60
1.50


Ordinary krigging
1000000
500000
0
−500000
−1500000
−1000000
−500000
0
500000
1000000
1500000
2000000
2.00
1.90
1.80
1.70
1.60
1.50

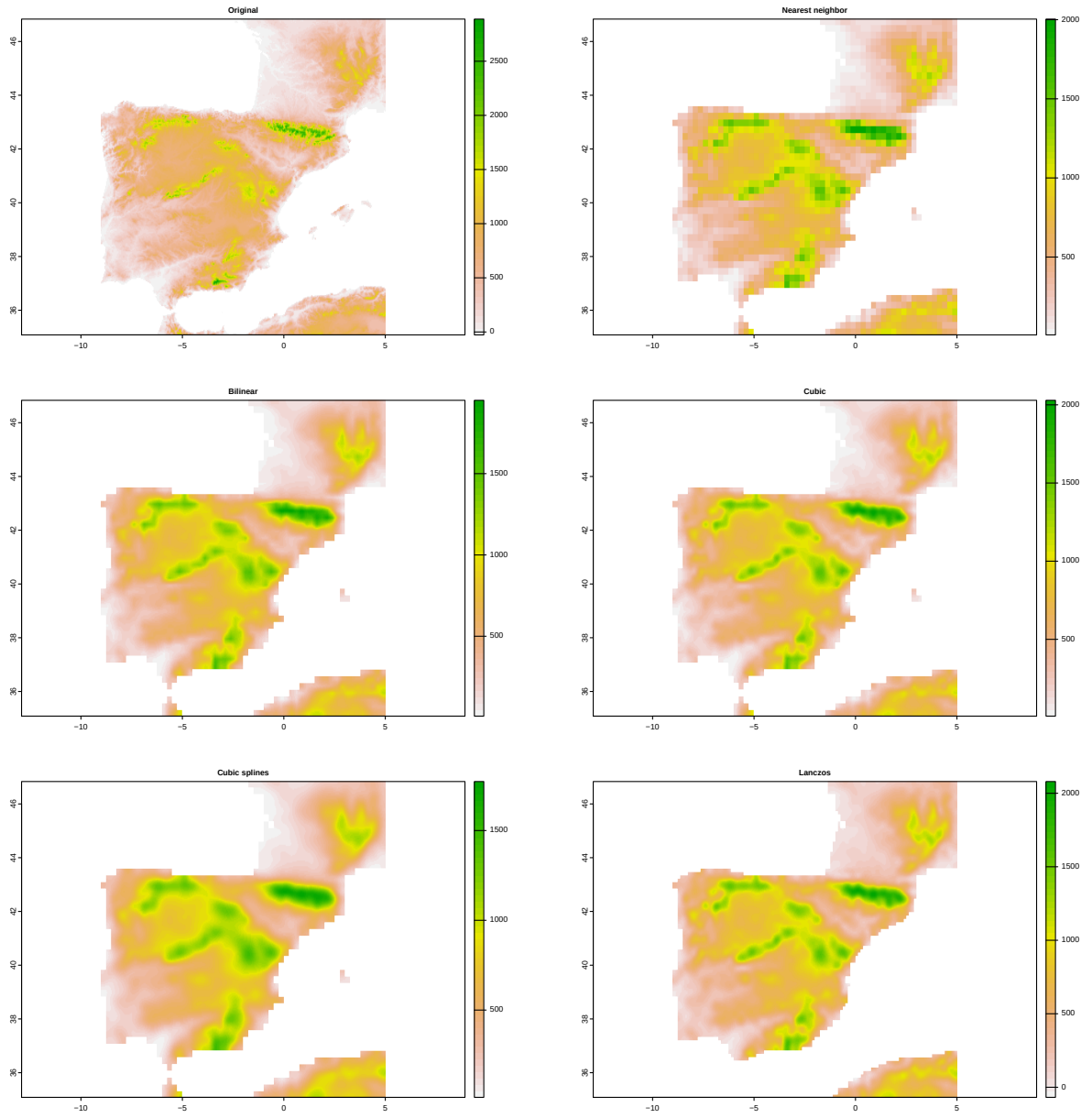
Raster data

We will carry out a quick comparison of the performance of several interpolation algorithms for rasters. Our procedure will be to reduce the dimension of a raster by a factor of 3 and then interpolate the resulting raster back to the size of the original one.

```
dem <- rast("DEM.tif")
ex <- ext(-9,5,34,46.85)
dem <- crop(dem,ex)
dem_agg <- aggregate(dem)
dem_small <- aggregate(dem,fact=6,fun="mean")
dem_nn <- resample(dem_small,dem,method="near")
dem_bi <- resample(dem_small,dem,method="bilinear")
dem_cu <- resample(dem_small,dem,method="cubic")
dem_sp <- resample(dem_small,dem,method="cubicspline")
dem_la <- resample(dem_small,dem,method="lanczos")
```

And the results are as follows:

```
par(mfrow=c(3,2))
plot(dem,main="Original")
plot(dem_nn,main="Nearest neighbor")
plot(dem_bi,main="Bilinear")
plot(dem_cu,main="Cubic")
plot(dem_sp,main="Cubic splines")
plot(dem_la,main="Lanczos")
par(mfcol=c(1,1))
```


Original
Nearest neighbor
Bilinear
Cubic
Cubic splines
Lanczos

We can check the Pearson correlation coefficient between the original image and the interpolated ones.

```
df <- cbind(values(dem),values(dem_nn),values(dem_bi),values(dem_cu),values(dem_sp),values(dem_la))
colnames(df) <- c("dem","dem_nn","dem_bi","dem_cu","dem_sp","dem_la")
df <- na.omit(df)
print(cor(df))
```

```
##              dem    dem_nn    dem_bi    dem_cu    dem_sp    dem_la
## dem    1.0000000 0.9320131 0.9463103 0.9503922 0.9351921 0.9533135
## dem_nn 0.9320131 1.0000000 0.9779913 0.9792574 0.9694331 0.9764687
## dem_bi 0.9463103 0.9779913 1.0000000 0.9987117 0.9967115 0.9959960
## dem_cu 0.9503922 0.9792574 0.9987117 1.0000000 0.9930486 0.9983620
## dem_sp 0.9351921 0.9694331 0.9967115 0.9930486 1.0000000 0.9887779
## dem_la 0.9533135 0.9764687 0.9959960 0.9983620 0.9887779 1.0000000
```