How to rasterize primary biodiversity data

Alejandro Ruete

12 September 2016

This tutorial will help you to work in R with downloaded primary biodiversity data (i.e. species observations) required to calculate Ignorance Scores and Ignorance Maps (Ruete 2015 (http://bdj.pensoft.net/articles.php?id=5361)). There are basically two ways you could retrieve data from GBIF (http://www.gbif.org/). Those are 1) directly querying from their web page (http://www.gbif.org/species#) to later receive a download link pointing to a .CSV file or .ZIP file with the DarwinCore standard format attached to an email, or 2) directly retrieving data using the GBIF API. Thankfully, there are great tutorials on how to retrieve data directly from your R console. Take a look at the great job done by rOpenSci (https://ropensci.org/%3E) in producing (https://cran.r-project.org/web/packages/rgbif/index.html) and documenting (https://cran.r-project.org/web/packages/rgbif/vignettes/rgbif_vignette.html) the `rgbif` package. Check out also Scott Chamberlain's (https://github.com/sckott) and Katie Heineman's (https://sites.google.com/site/mikegahan1/rgbif) blog post for parallelization options. Other methods to retrieve data via the GBIF APIs include the **QGBIF** package (https://github.com/BelgianBiodiversityPlatform/qgis-gbif-api) for QuantumGIS (http://www.qgis.org/).

You may also need to check these other tools, protocols and workflows the will help you curating, searching for duplicates, georeferencing issues, and getting the data fit-for-use:

- Taverna workflow for data refinement at BioVel (https://www.biovel.eu/application-areas/data-refinement-wf)
- Biogeo (http://onlinelibrary.wiley.com/doi/10.1111/ecog.02118/full): an R package for assessing and improving data quality of occurrence record data sets
- OpenRefine (http://openrefine.org/): a powerful tool for working with messy data.
- The report (http://www.gbif.org/resource/82612) of the Task Group on GBIF Data Fitness for Use in Distribution Modelling
- The GBIF position paper (http://www.gbif.org/resource/80623) on future directions and recommendations for enhancing Fitness-for-Use across the GBIF Network and these slides (http://www.gbif.org/resource/81831)

Depending on your questions, explorations of the bias in sampling effort using Ignorance Scores could be performed before or after such curations.

Enough with the prelude

For this tutorial (and for the apps presented at the ENC2016) I downloaded from GBIF all observations for the class Amphibia (taxonKey = 131) within Europe. Then, I quickly removed observations without identification at the species level and kept those with accepted taxonomic status (taxonomicStatus == "ACCEPTED"). I also removed Fossils.

With the following scripts you will create a spatial grid of the resolution needed (alt, use a grid you have), and summarize in two raster layers the number of observations and number of species observed per grid cell. These are the two data sets you would need to produce the ignorance scores. At this point you should have the data as a matrix with one row per observation with at least three important variables: X and Y coordinates, and species name. These columns may change labels depending on the data standard of the file.

You will need to install and load these packages:

```
require(raster)
require(maptools)
require(rgdal)
require(parallel)
```

Create reference raster.

You may already have a raster you want to use as reference. In that case skip to next point.

We use the projected coordinate system EPSG:3857 (WGS84 Web Mercator) because it is used by most of the popular web mapping applications (Google/Bing/OpenStreetMap/Leaflet, etc). However, it is best if rasterize using the coordinate system you plan to use later on to avoid some problems that may appear when re-projecting the rasters. The extent of this coordinate systems is `world.ext <- c(xmn=-20026376.39, xmx=20026376.39, ymn=-20048966.10, ymx=20048966.10)`, that excludes the polar regions. However, keep the extent of the reference raster tight to your data not to waste memory and processing power.

```
res<-25 * 1000 # in metres
r25<-raster(resolution=res,
            xmn=-2.5e+06, xmx=1e+07, ymn=2e+06, ymx=1.5e+07, # extent
            crs="+init=epsg:3857", # CRS used by leaflet
            vals=NA) # make it empty
```

Load a map of Europe plus a buffer of 5 km project it in epsg:3857.We use it later to mask out areas that are not useful to us.

```
Europe<-readShapePoly("EuropeB.shp", proj4string=CRS("+proj=longlat +ellps=WGS84
 +datum=WGS84 +no_defs"))
Europe<-spTransform(Europe, CRS("+init=epsg:3857"))
```

Rasterize Data

Here we transform the data matrix into a SpatialPointsDataFrame. Make sure you know the coordinate system of the data you downloaded. The coordinates are often stored in geographical coordinates (Lat/Long type; i.e. not projected), although some providers include coordinates in two coordinates systems. Here we need to transform them to our system.

```
AmpEur <- read.csv("GBIF Amphibia Europe light spp.csv") # All observations with identification at spp level, no Fossils

# coerce to SpatialPointsDataFrame
coordinates(AmpEur)<- ~ decimallongitude + decimallatitude # use the actual names of the columns
proj4string(AmpEur)<- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
AmpEur <- spTransform(AmpEur, CRS("+init=epsg:3857"))

beginCluster() # parallelization option offered by the raster package

# Count of observations
tmp<- rasterize(AmpEur, r25, field="species", fun="count")
tmp[]<-ifelse(is.na(tmp[]), 0, tmp[]) # Change NAs for 0s
tmp <-mask(tmp, Europe) # Change counts out of the area of interest into NAs
writeRaster(tmp, filename="your preferred name here.tif", format="GTiff")

# Count of Species. You need to provide a custom function in 'fun'
tmpR <- rasterize(AmpEur, r25, field="species", fun=function(x,...) length(unique(na.omit(x))))
tmpR[]<-ifelse(is.na(tmpR[]), 0, tmpR[])
tmpR<-mask(tmpR, Europe)
writeRaster(tmpR, filename="your preferred name here.tif", format="GTiff", overwrite=TRUE)
endCluster()
```

In some biodiversity analysis portals (e.g. Swedish LifeWatch (https://www.analysisportal.se/) or API calls (http://www.gbif.org/developer/maps#preview) you can directly get a file where the data falling within each grid cell (of a desired resolution) is summarized per row. In such cases you do like this (example for Amphibians in Sweden):

```
require(raster)
require(maptools)
Amp <- read.csv("SLW GridStatisticsOnSpeciesCounts (Amphibia).csv")

y<-Amp
names(y)<-c("Id","Observation.count","X","Y","Grid.cell.width")
coordinates(y)<- ~ X + Y # beware Coordinate system EPSG:3006, SWEREF99 TM, only for illustration purposes

# coerce to SpatialPixelsDataFrame
gridded(y) <- TRUE

# coerce to raster
raster <- raster(y, layer=2, values=TRUE) ## S4 method for signature 'SpatialGrid'
```

Time slices

So far we rasterized all observations obtained. If what you want is to compare observations between years (or other time units) you need to run a loop over all time units, and for each of it subset the data set e.g. `WhichYear <- which(AmpEur$year == Years[y])`. If there are observations in that time unit then rasterize the subset `AmpEur[WhichYear,]`, else create an empty raster. Remember that there are data where the year has not been specified; make a raster of that as well. Finally, stack the yearly rasters into either a RasterStack or an array. An array will take less memory and the `raster` package offers commands to match array elements with the position of raster pixels.