“Where the wild things are” A tutorial for the speciesgeocodeR R-package

Alexander Zizka December 29, 2014

What is it? SpecisgeocodeR is an R package for the analysis of species occurrences data linked with geographical areas. It is a tool for data cleaning, data exploration and data analysis. SpeciesgeocodeR is especially suited for bio-geographical and ecological questions on the distribution of specimens and species in geographic areas (e.g. biomes or national parks). The package has particularly been tested for large datasets. Based on a list of point coordinates (specimen occurrences) and a list of polygons (geographical areas) speciesgeocodeR handles the classification of geographic points to geographical areas (point in polygon test) and runs multiple statistical analyses to summarize the results. An easy-to-use wraper function allows the run of a standard analysis with only three commands, making the package suited for R beginners. The standard output includes a nexus file to use the results as direct input for bio-geographic analysis linked with phylogenetic trees, as well as summary tables, -graphs and geographical maps. Furthermore speciesgeocodeR can be used to automatically clean geographic occurrence data, calculate a coexistence matrix, map diversity in geographic areas, batch calculate extent of occurrence (EOO) for conservation assessment, batch download elevation data from coordinates and to include elevation and the WWF ecoregions directly into a standrd analysis.

Contents

1 Introduction

Information on the spatial distribution of species and individuals is an essential basis of ecology, biogeography and conservation biology. Point-occurrences from GPS data, collection specimens or radio-collared animals are nowadays one of the major sources for this kind of data. However, for many analyses the classification of point-occurrences into descrete geographical areas, such as continents, biomes or national parks are of particular interest. The large amount of point-occurrence data available for many taxa (easily reaching tens of thousands up to millions of single data-points) together with the common use of multiple geographical areas and multiple species for large-scale analysis renders a manual classification difficult.

SpeciesgeocodeR is an R package to explore the relation of point (occurrence points) and polygon (geographic areas) data. The package is a tool for data cleaning, data exploration and data analysis. It is especially designed to be easily accessible for R beginners. The package is designed and tested for small-scale (e.g. radio-colar data in combination with national park borders) and large scale (e.g. global distribution of all birds to biomes) analyses. A standard speciesgeocoder analysis classifies each sample point to a polygon, and creates a nexus format output table as well as a set of summary tables, graphs and maps. Furthermore, speciesgeocodeR can be used to:

- clean geographic datasets (chapter 4.1
- calculate a coexistence matrix (chapter 4.2)
- map diversity in geographic areas (chapter 4.3)
- batch calculate extent of occurrence (chapter 4.5)
- batch download elevation data from coordinates (chapter 4.6)
- include elevation in the sample classification (chapter 4.7)

This tutorial provides easy-to-use instructions for the major functionalities of the package for R beginners. Chapter 3 specifies the input-file format and chapter 5 describes the outputfiles of a standard analysis. Chapter 6 shows results of benchmarking tests to give an idea of the computational time needed depending on the size of a dataset. Explanatory text is set in standard font, code examples are set in `teletypefont`. Text set in CAPITALS should be replaced by the user. All example scripts are provided in a seperate text-file with the package, so that the code can be copied directly into the R console. Some basic experience in using R is helpful but no prerequisite. If you have no experience with R you might want to have a look at Crawley (2012). Alternatively you can use the python version of Speciesgeocoder (Töpel et al., 2014) or the web interface (**?**) for some functionalities of the package. SpeciesgeocodeR is mainly based on functions of the maps (Becker et al., 2013), maptools (Bivand and Lewin-Koh, 2013), raster (Hijmans, 2014), rgeos (Bivand and Rundel, 2014) and sp (Pebesma and Bivand, 2005; Bivand et al., 2013) packages. Please report bugs to alexander.zizka@bioenv.gu.se.

2 SpeciesGeoCoder standard: the quick and easy way

You can run a standard speciesgeocodeR analysis with only three commands. The example code shown here is also provided as text file with the package so if you are unfamiliar with the R environment, you can just copy the example code from there to the R console (the CAPITALized words correspond to your filenames and must be changed accordingly). For this tutorial we will use a distribution dataset from the plant genus *Ivesia* in Western North America (Töpel et al., 2012) as example dataset. We will use speciesdata and polygondata from a .txt file. If you use different input formats the command in step five below will be different, have a look at the respective section in chapter 3.

1. Create a new folder in your home directory (the working directory). At the moment speciesgeocodeR is provided as .zip file. Copy the speciesgeocodeR.zip into your working directory. Then copy your input files into the working directory and give them characteristic names (here we will use ”ivesia_coordinates.txt” as speciesdata and ”ivesia_polygons.txt” as polygondata, see the example_data subfolder). If you are insecure about the format of your input files or an error occurs right away, check chapter 3 of this manual or take look at the example files delivered with the package.

2. Start R

3. Tell R where to find the input files and save the outputfiles. You must put the exact path of your working directory in the quotation marks (e.g. ”C:\\Desktop\\data”). If you use mac use / instead of \\.

   ```
   setwd("YOUR WORKING DIRECTORY PATH")
   ```

4. Load the functions of the speciesgeocodeR package into your R session

   ```
   install.packages("speciesgeocodeR.zip", repos = NULL)
   library(speciesgeocodeR)
   ```

5. Execute the `SpeciesGeoCoder()` function with the names of your two input files as ar-

guments (e.g. “ivesia_coordinates.txt”, “ivesia_polygons.txt”). Depending on the size of your dataset this might take a while. The “graphs” argument controls if graphical output is produced (default = T), the “coex” argument defines if a coexistence matrix for all species in each polygon is produced (default = F, this is only recommended for datasets with less then 200 species, as it is computationally intense)

```
SpeciesGeoCoder("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons.txt", graphs = T, coex = F)
```

6. Done! Close R, the output files are in your working directory. Summary tables are saved as tab-delimited .txt files, graphs and maps as .pdf files. If problems occur, check your input-files or try using one of the example datasets delivered with the package.

3 Input files

To run a speciesgeocodeR analysis you need a set of point occurrences (e.g. GPS-coordinates of specimen, hereafter called “speciesdata”) and a set of discrete geographical areas (e.g continents, biomes or national park borders, hereafter called “polygondata”). You can provide these in four different formats

1. species- and polygondata as tables in a text (.txt) file
2. species- and polygondata as R data.frame
3. polygondata as a shape (.shp) file
4. speciesdata from taxon names

3.1 Species- and polygondata as tables in a text file

The two input files are required to be tab-delimited .txt files with three columns each. For both files, column names must be: “identifier”,“XCOOR”,“YCOOR”. In case of the speciesdata the first column contains the species name (or individual name), the second column contains the longitude coordinates (as decimal degrees, e.g. 59.1 for East, or -59.1 for West) and the third column contains latitude coordinates (as decimal degrees, e.g. 59.1 for North, or -59.1 for South). The polygondata should be in the same format: each row represents the coordinates of one polygon corner. The first column contains the polygon identifier, the second one the longitude coordinates, the third one the latitude coordinates. It is important that the coordinates of the first and the last point of each polygon are identical. Example input files are provided with the package (in the example_files folder). You can easily export such a file from any GIS program (e.g. QGIS **?**).

When you prepare the input files please make sure:

1. that the order of the columns and the column headers are correct (“identifier”,“XCOOR”, “YCOOR”)

2. that your coordinates conform with the limits of the coordinate system (XCOOR between -180 and 180; YCOOR between -90 and 90)

3. that for each polygon in the polygondata the first and last point have identical coordinates.

Using to input text files the `SpeciesGeoCoder()` function will be called:

```
SpeciesGeoCoder("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons", graphs = T, coex = F)
```

3.2 Species- and polygondata as R data.frame

If you have loaded your speciesdata and polygondata into R already, you can use two data.frame objects as input for the `SpeciesGeoCoder()` function:

```
spdata <- read.table("example_data\\ivesia_coordinates.txt", sep = "\t", header =
T)
polydata<- read.table("example_data\\ivesia_polygons.txt", sep = "\t", header = T)
SpeciesGeoCoder(spdata, polydata, graphs = T, coex = F)
```

3.3 Polygons from a shape file

Geographic areas are often available as shape files (.shp). You can use shape files loaded into R as a SpatialPolygonsDataframe as input for the `SpeciesGeoCoder()` function. In this case you need to name the column of the SpatialPolygonsDataframe which specifies the name of the included polygons via the "areanames" argument. You can easily find this via the `head()` function after you loaded the shape file into R. If possible avoid polygon names with special characters or polygons named "NA":

```
library(maptools)
polygondata <- readShapeSpatial("example_data\\drawn_ivesia.shp")
head(polygondata)
SpeciesGeoCoder("example_data\\ivesia_coordinates.txt", polygondata, areanames =
"name", graphs = T, coex = F)
```

3.4 Species occurrences from GBIF

If you do not have occurrence data for your species of interest, you can use speciesgeocoder to download speciesdata from the Global Biodiversity Information Facility (GBIF, 2014) based on species names. In this case you can enter a vector of speciesname(s) as speciesdata for the `SpeciesGeoCoder()` function. Please note that no data cleaning is performed at this point an that this feature is based on the `gibf{dismo}` function and therefore uses the old GBIF format:

```
SpeciesGeoCoder(c("Ivesia saxosa", "Ivesia baileyi"), example_data\\ivesia_polygons.txt",
graphs = T, coex = F)
```

4 Further functionalities

4.1 Cleaning geographica data

Species distribution datasets assembled from multiple sources (such as in GBIF) offen suffer from geographic inconsistencies or errors, which raise doubt on the reliability of analyses run on these data. You can use the `geoClean()` function of SpeciesgeocodeR to automatically check your dataset for common problems with geographic coordinates. You can check for general coordinates validity, common simple errors such as zero coordinates, equal latitude and longitude and check for a set of issues that have proven problematic when using sucha datasets in the past (for example if the coordinates fall with in the right country or if they fall on on the capital or centroid of the country. You can type `?geoClean()` to see all possibles cleaning test of the function. You need to provide geoCLean with a data.frame with at least three columns: “species”, “XCOOR”, “YCOOR”. The function has a number of logical arguments to switch on sinlge tests. If you run `geoClean` you will get a vector of the same length as the input data.frame with “FALSE” flags for all coordinates that failed the check. The following example runs general test of coordinate validity.

```
test <- read.table( example_data\\ivesia_dirty, sep = "\t", header = T)
out <- geoClean(test, isna = T, isnumeric = T, coordinatevalidity = T,
containszero = T, zerozero = T, latequallong = T, GBIFhead = T,
countrycentroid = F, capitalcoords = F, countrycheck = F, verbose = F)
cbind(test, out)
```

If you use “verbose = T”, you will get a data.frame instead of a vector, where each column states the results of a single test.

```
test <- read.table( example_data\\ivesia_dirty, sep = "\t", header = T
out <- geoClean(test, isna = T, isnumeric = T, coordinatevalidity = T,
containszero = T, zerozero = T, latequallong = T, GBIFhead = T,
countrycentroid = F, capitalcoords = F, countrycheck = F, verbose = T)
out
```

If you want to check if the coordintes fall into the right country or if they fall on the country centroid or the country capital, your input data.frame needs an additional column indicating the country. Additionally you need to set the respective arguments of the fucntion to TRUE and to specify the area around the centroid coordinates to be check (the default is to 0.5, which checks a one degree box around the country centroid or the country capital). PLease note that the country test can only be run if the coordinates are generally valid:

```
library(maptools)
data(wrld_simpl)
data(countryref)


test <- read.table( example_data\\ivesia_dirty, sep = "\t", header = T)
out <- geoClean(test, isna = T, isnumeric = T, coordinatevalidity = T,
containszero = T, zerozero = T, latequallong = T, GBIFhead = T,
countrycentroid = F, capitalcoords = F, countrycheck = F, verbose = F)
test2 <- test[out == T, ]
out2 <- geoClean(test2, isna = F, isnumeric = F, coordinatevalidity = F,
containszero = F, zerozero = F, latequallong = F, GBIFhead = F,
countrycentroid = T, capitalcoords = T, countrycheck = T, polygons = wrld_simpl,
verbose = F)
out2
```

Alternatively you can also use data directly downloaded from GBIF:

```
library(dismo)
test <- gbif(genus = "Ivesia", geo = T)
test <- read.table( example_data\\ivesia_dirty, sep = "\t", header = T
out <- geoClean(test, isna = T, isnumeric = T, coordinatevalidity = T,
containszero = T, zerozero = T, latequallong = T, GBIFhead = T,
countrycentroid = F, capitalcoords = F, countrycheck = F, verbose = F)
out
```

4.2 Calculating a co-existence matrix

Understanding co-existence patterns is a crucial aspect of ecology and biogeography. You can use speciesgeocodeR to calculate a matrix of species co-existence in each input polygon. Based on our example from the previous chapter you can calculate a co-existence matrix and visualize it as a heatplot. The matrix will have the same dimensions as there are input species and will give the percent of occurrence points of each species (row) with all other species (columns) given the input polygons. This can be done in two different ways:

1. When running a normal analysis, by setting coex = T in the `SpeciesGeoCoder()` function. In this case the matrix is saved as .txt file in the working directory and the heatplot (only for datasets <40 species) is saved as .pdf in the working directory. However, this is only recommended for small datasets, as the computation might take a long time.

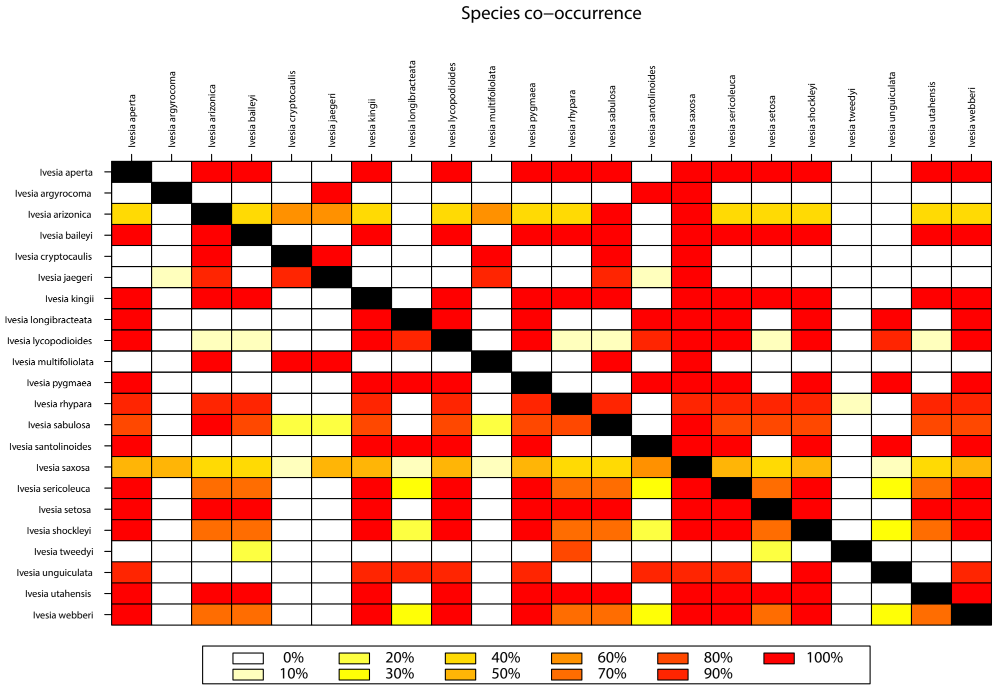
```
setwd("YOUR WORKING DIRECTORY PATH")
install.packages("speciesgeocodeR.zip", repos = NULL)
library(speciesgeocodeR)
SpeciesGeoCoder("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons.txt", graphs = T, coex = T)
```

2. More efficiently, you can also calculate the coexistence matrix without the wrapper function. To do this you first need to load the input data using the `ReadPoints()` function and run the `SpGeoCodH()` function which creates an object of the class “spgeoOUT” (it runs speciesgeocoder without output files). This object can be used with the
texttt CoExClass() function to create the coexistence matrix.

```
inp_ivesia <- ReadPoints("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons.txt")
outp_ivesia <- SpGeoCodH(inp_ivesia)
coex_ivesia <- CoExClass(outp_ivesia)
```

3. For small datasets the matrix can then be plotted as a heatplot using `HeatPlotCoEx()`.

```
HeatPlotCoEx(coex_ivesia)
```

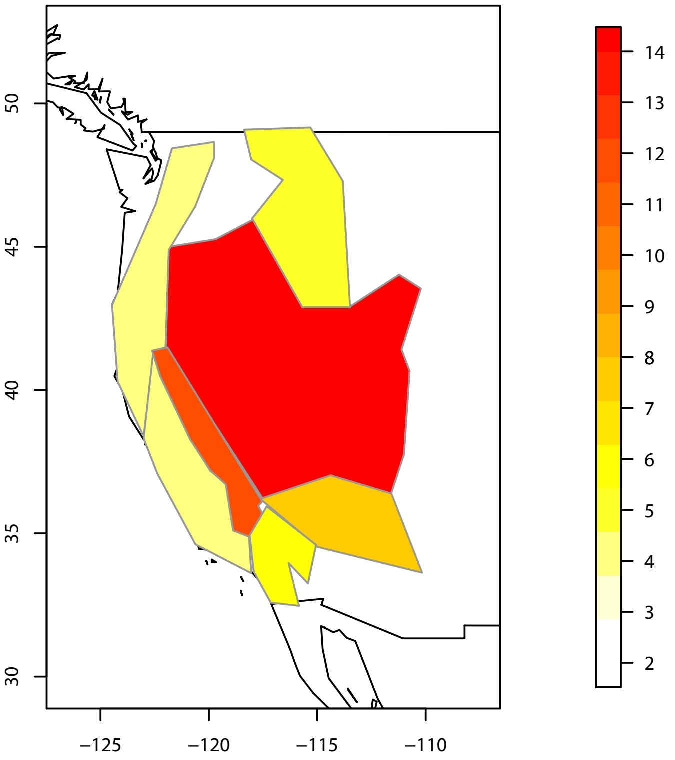


4.3 Mapping diversity in polygons

The `MapDiversity()` function of speciesgeocodeR can be used to color-code diversity in the input areas.

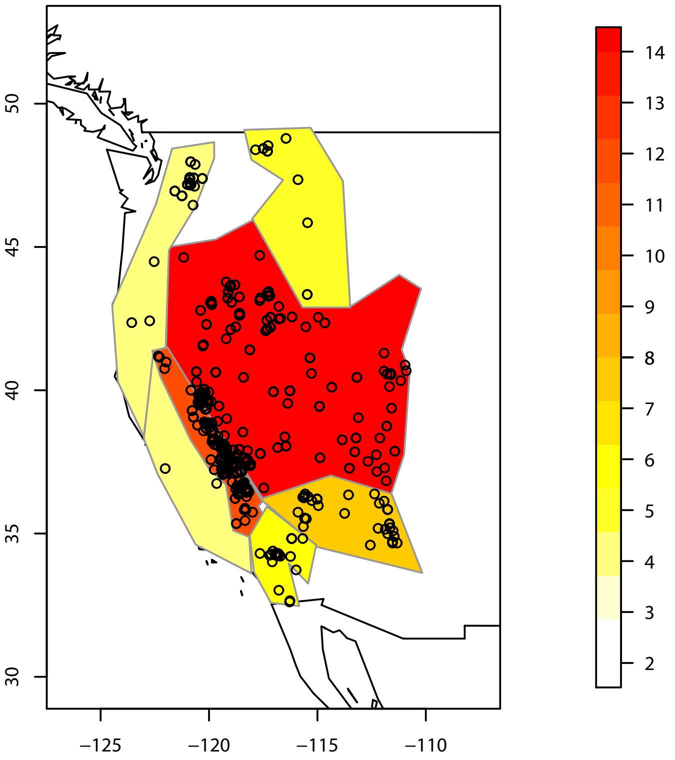
If you want to use polygons imported from a text file, you must first load the data and run `SpeciesGeoCoder()` as in 4.2. Then you can use the `MapDiversity()` function on the output object.

```
inp_ivesia <- ReadPoints("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons.txt")
outp_ivesia <- SpGeoCodH(inp_ivesia)
MapDiversity(outp_ivesia)
```



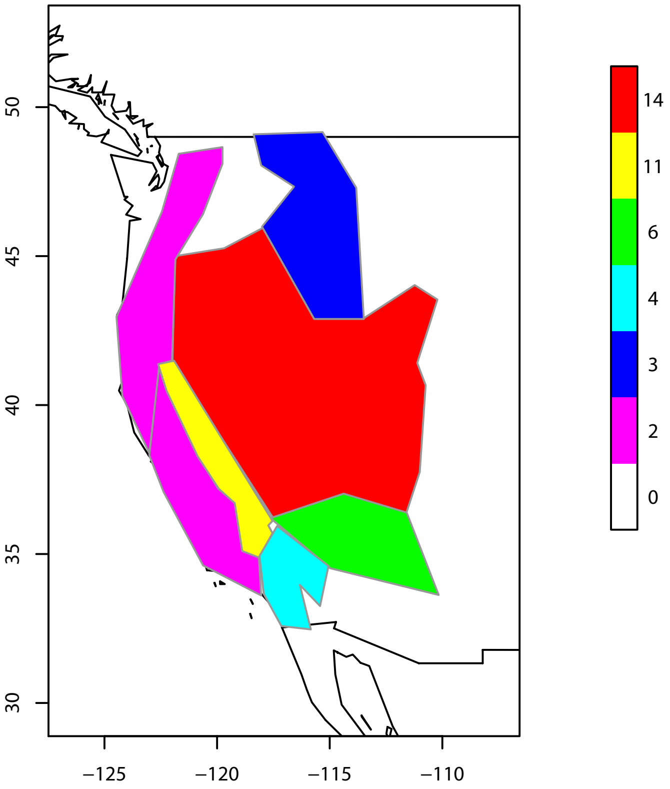
The `MapDiversity()` function has a number of arguments to control the appearance of the plot: leg = ”continuous” or ”discrete” defines the use of a continuous or discrete color scheme.The show.occ = T shows the occurrence points on the map.

```
MapDiversity(outp_ivesia", leg = "continuous", show.occ = T)
```



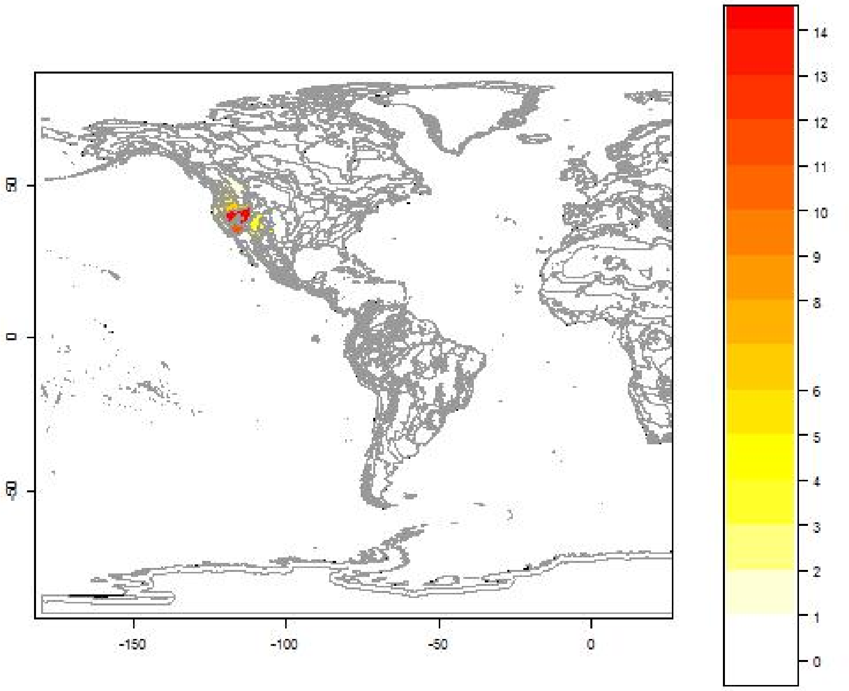
When using neighboring areas with little difference in species number a the discrete coloring might be more appropriate.

```
MapDiversity(outp_ivesia, leg = "descrete")
```



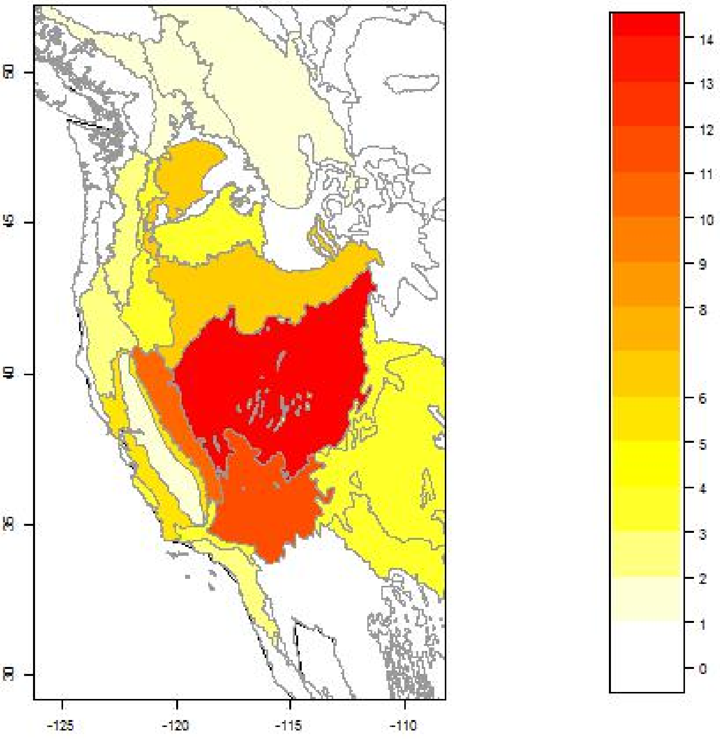
You can use the `MapDiversity()` function with any areas, but speciesgeocoder allows to directly use the WWF ecoregions and biomes. Note that this will download a package of shape files from the wwf webpage and save it in the working directory. If you use complicated polygondataset you need to name the column of the SpatialPolygonsDataframe which specifies the name of the included polygons via the "areanames" argument. You can easily find this via the `head()`. If possible avoid polygon names with special characters or polygons named "NA".

```
wwf <- WWFload()
inp_ivesia <- ReadPoints("example_data\\ivesia_coordinates.txt", wwf)
head(wwf)
outp_ivesia <- SpGeoCodH(inp_ivesia, areanames = "ECO_NAME")
MapDiversity(outp_ivesia, areanames = "ECO_NAME", leg = "continous")
```



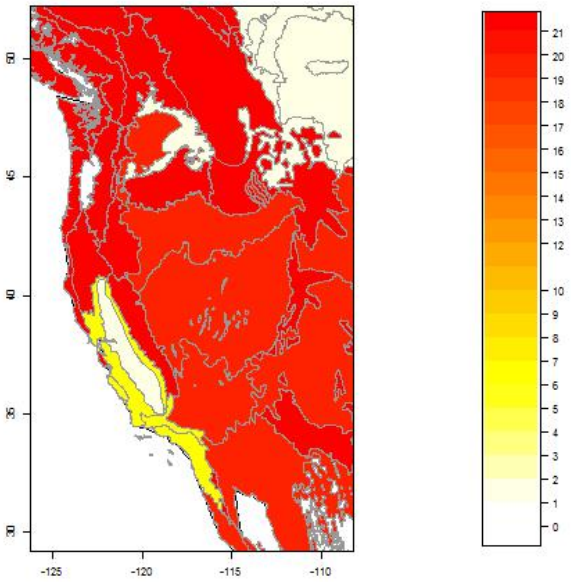
In this case your polygons cover a much larger area than the actual area of interest (as in this example where they stretch over entire North America), the lim argument can be used to adjust the pane shown on the map. lim = “polygons” or “points” defines if the points or polygons are used to define the plot outline.

```
MapDiversity(outp_ivesia, areanames = "ECO_NAME", leg = "continous", lim = "points")
```



If you want to color-code biomes instead of ecoregions, you need to change the “areanames” argument to “BIOME”. Note that in this case all areas with the same biome classification are coloured, eventhough they might be on diffreent parts of the globe, far outside the species range. If you use global classifications and want to avoid this crop the polygons from global to a more regional scale before running speciesgeocoder.

```
wwf <- WWFload()
inp_ivesia <- ReadPoints("example_data\\ivesia_coordinates.txt", wwf)
head(wwf) outp_ivesia <- SpGeoCodH(inp_ivesia, areanames = "BIOME")
MapDiversity(outp_ivesia, areanames = "BIOME", leg = "continuous", lim = "points")
```



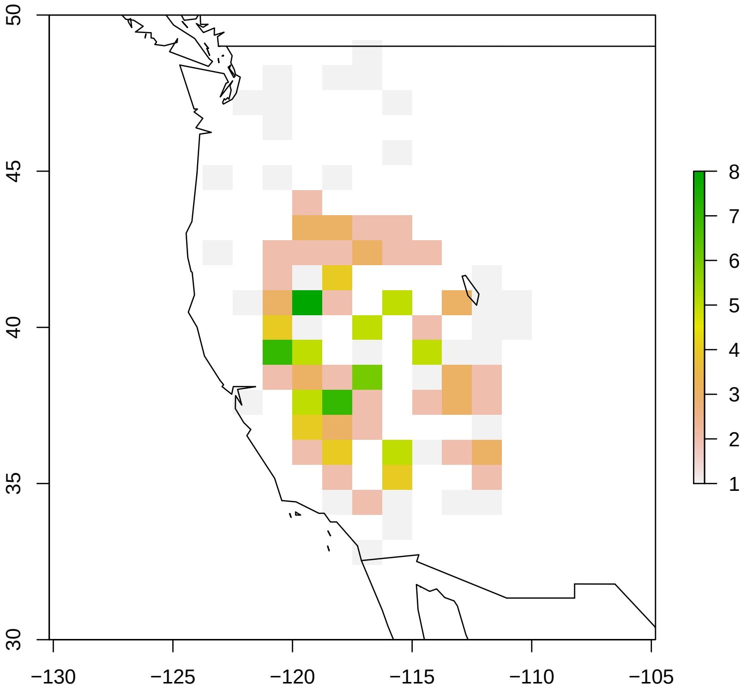
4.4 Deriving diversity and abundance grids

In some situations the visualization of diversity patterns in a raster is of interest. You can use the `DiversityGrid()` function to produce a diversity raster of your data indicating the number of species per raster cell. The “e” argument controls the extent of the raster (xmin, xmax, ymin, ymax), the “reso” argument sets the resolution (in geographical minutes; 60 = 1 degree):

```
limits <- extent(-130,-105, 30, 50)
ivesia_div <- DiversityGrid("example_data\\ivesia_coordinates.txt",
e = limits, reso = 60, type = "div")
```

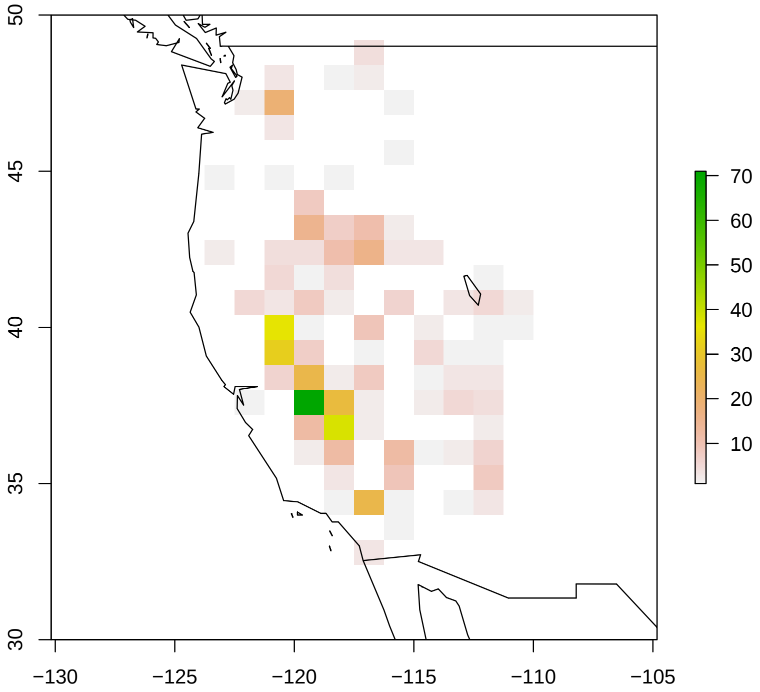
This produces a raster object, which can be handled with all the functions of the raster-package (Hijmans, 2014). As an easy solution to plot the raster in the context of national borders you can use the `MapGrid()` function:

```
MapGrid(ivesia_div)
```



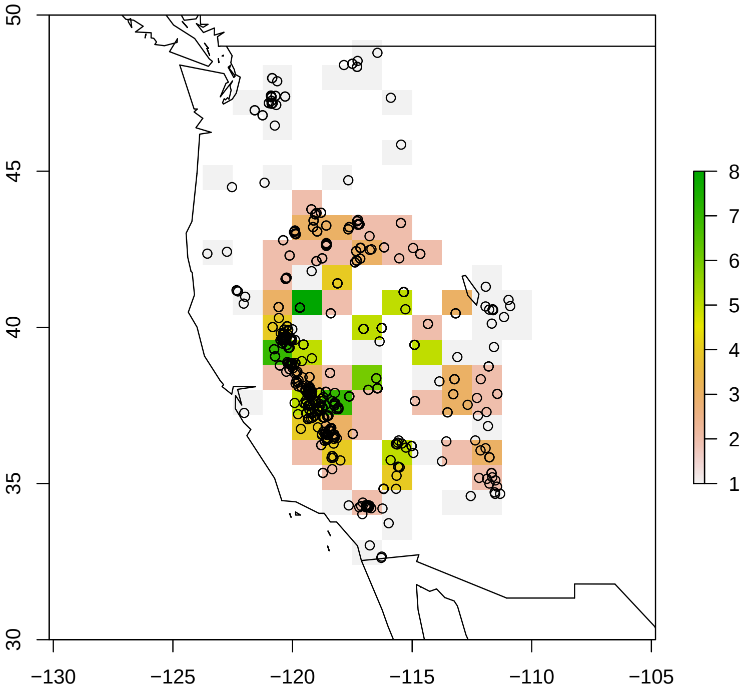
If you want to represent the number of collection points in a grid instead of species numbers you can set the type argument to “abu”.

```
limits <- extent(-130,-105, 30, 50)
ivesia_abu <- DiversityGrid("example_data\\ivesia_coordinates.txt", e = limits, reso
= 60, type = "abu")
MapGrid(ivesia_abu)
```



Alternatively to using a input text file, you can also use `DiversityGrid()` on an object of the class ”spgeoOUT” produced by `SpGeoCodH()` for abundance and diversity grids. In this case you can easily add the input points and/or your polygons to the plot.

```
inp_ivesia <- ReadPoints("example_data\\ivesia_coordinates.txt",
"example_data\\ivesia_polygons.txt")
outp_ivesia <- SpGeoCodH(inp_ivesia)
limits <- extent(-130,-105, 30, 50)
ivesia_div <- DiversityGrid("example_data\\ivesia_coordinates.txt", e = limits, reso
= 1, type = "div")
MapGrid(ivesia_div)
points(outp_ivesia$species_coordinates_in$XCOOR, outp_ivesia$species_coordinates_in$YCOOR)
```



4.5 Calculating extent of occurrence (EOO)

You can use species geocoder to calculate the Extent of occurrence. The EOO can be seen as an approximation of the geographical distribution range of a species is and is recommended as one of the tools for a conservation assessment under criterion B of the IUCN (IUCN Standards and Petitions Subcommitte, 2014). The `CalcRange()` function calculates the EOO for each species in your dataset using a convex hull. This creates a polygon encopassing all occurrence points for each species with the interior angles smaller than 180 degrees. Subsequently the area of this polygon is calculated and given out as the EOO (in square kilometers). As input for the function you need a dataframe with three columns: “identifier”, “XCOOR”, “YCOOR”, as described in chapter 3.

```
ivesia_coordinates <- read.table("example_data\\ivesia_coordinates.txt")
CalcRange(ivesia_coordinates, mode = "EOO")
```

4.6 Deriving elevation based on coordinates

You can use the `getElevation()` function to download elevation data for your point occurrences or just provide the function with a vector of species names. In the latter case occurrence points will automatically be downloaded from GBIF (GBIF, 2014). The elevation informtion will be downloaded from the SRTM 90m Digital Elevation Data (Jarvis et al., 2008). The function returns a vector of elevation values that can be combined with the input data using `cbind()`. Note that `getElevation()` will download tiles of gif files from the internet and save them in the working directory. This might take time and use lots of disk space for large scale analyses.

```
ivesia_coordinates <- read.table("example_data\\ivesia_coordinates.txt")
ele_dat <- getElevation(ivesia_coordinates)
cbind(ivesia_coordinates, ele_dat)

ele_gbif <- getElevation(c("Ivesia jaegeri","Ivesia baileyi", "Ivesia argyrocoma"))
```

4.7 Including elevation into a standard speciesgeocodeR analysis

Elevation is an important proxy for many environmental factors, and often it is of interest, to not only destinguish between different geographic regions, but also between highland and lowland species. You can directly included elevation into the `SpeciesGeoCoder()` function. When you set the “elevation” argument to TRUE you can provide a vector with different elevation thresholds (in m) with the “threshold” argument. A seperate set of outputfiles will then be generated for each elevation class.

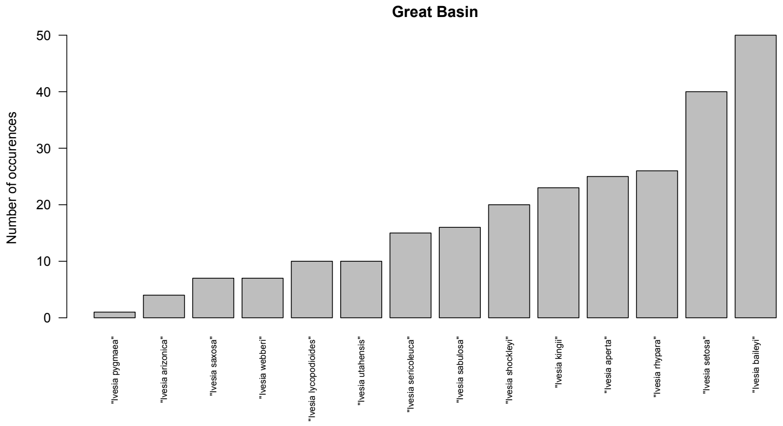
```
spdata <- read.table("example_data\\ivesia_coordinates.txt", sep = "\t", header =
T)
polydata<- read.table("example_data\\ivesia_polygons.txt", sep = "\t", header = T)
SpeciesGeoCoder(spdata, polydata, graphs = F, coex = F, elevation = T, threshold
= c(500, 700, 1000))
```

5 Output-file description

With the “graphs” and “coex” options swiched on, `SpeciesGeoCoder()` produces 13 output files by default (1 nexus file, 5 summary tables, 3 barcharts, 3 maps and 1 heatplot). All files are saved to the working directory. The summary tables are tab delimited .txt files and all graphics and maps are saved as .pdf files. Depending on the size of you dataset, some of the graphical representations might not be adequate.

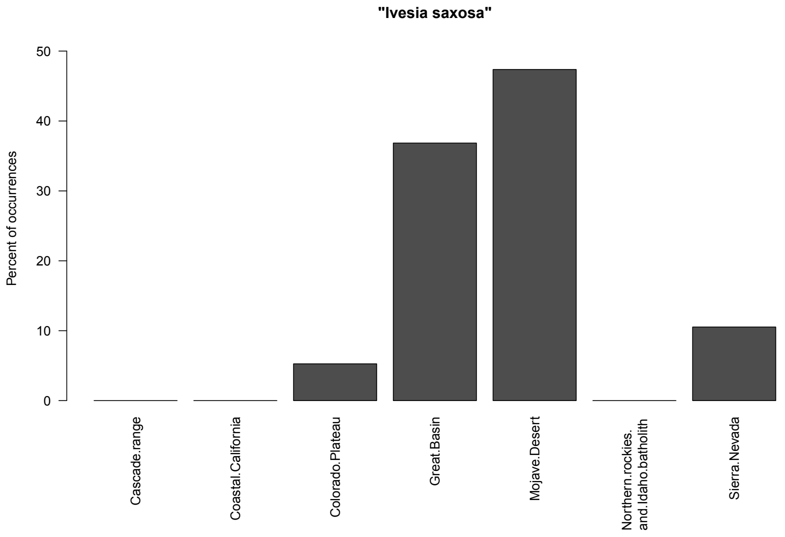
Barchart_per_polygon.pdf

This file contains one barchart for each input polygon showing the total number of occurrences for each species in this polygon.



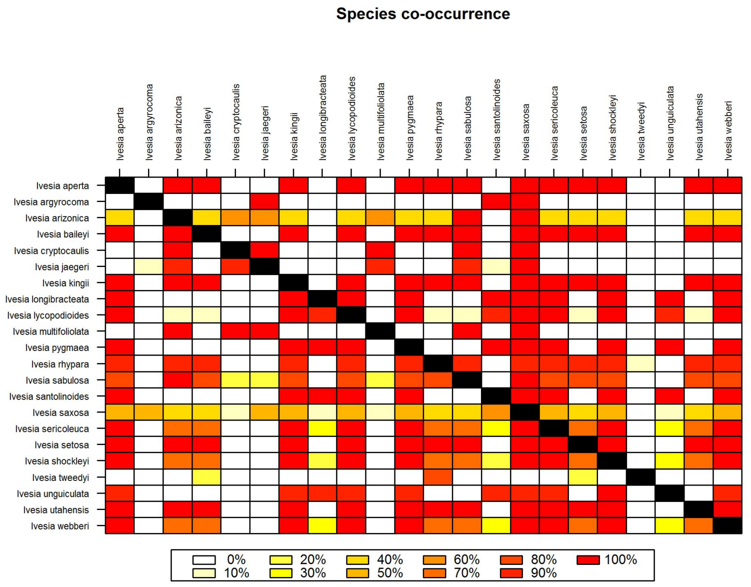
Barchart_per_species.pdf

This file contains one barchart per species, showing the relative number of occurrences in each input polygon.



Heatplot_coexistence.pdf

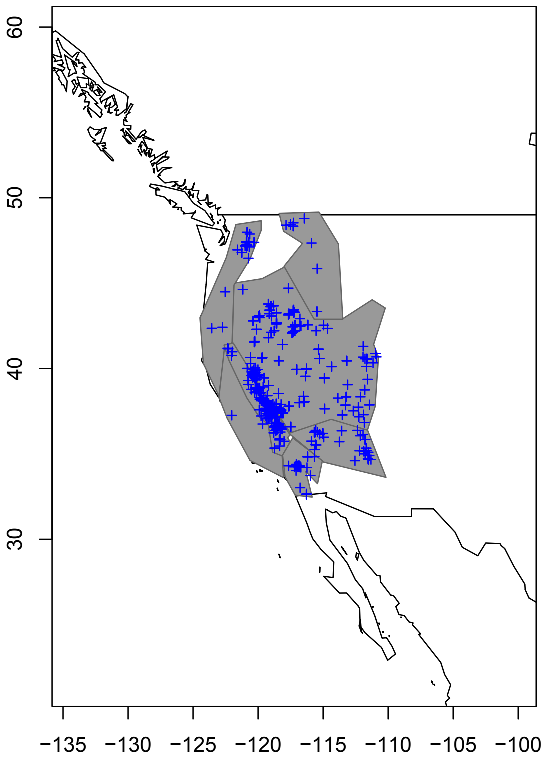
This file contains a heat plot showing the co-existence pattern of all species in the analysis. This output is turned of by default. To turn it on, use ”coex = T” as argument of the `SpeciesGeoCoder()` function.



Map_samples_overview.pdf

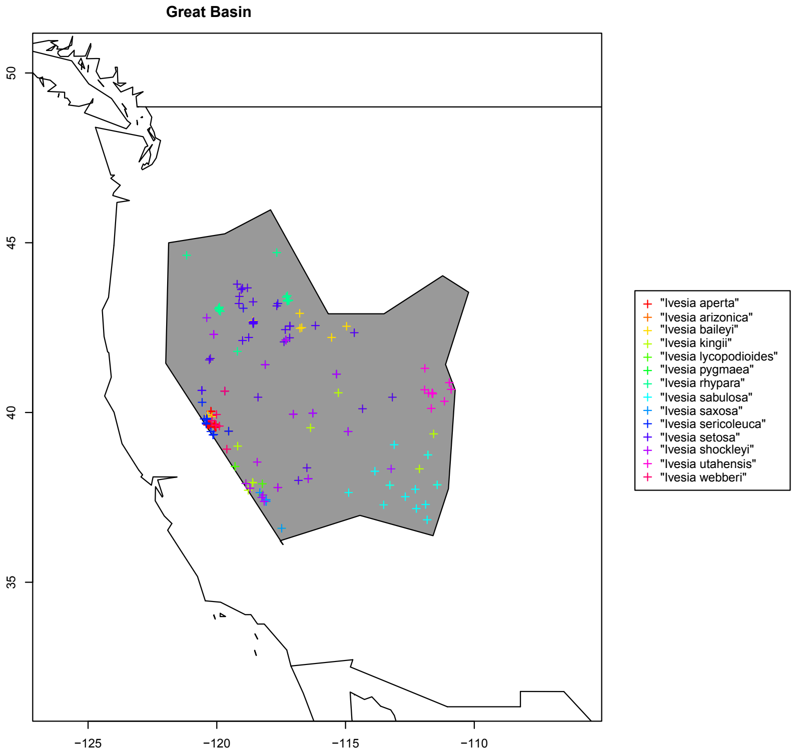
This file contains an overview map showing all input points and polygons.

All samples



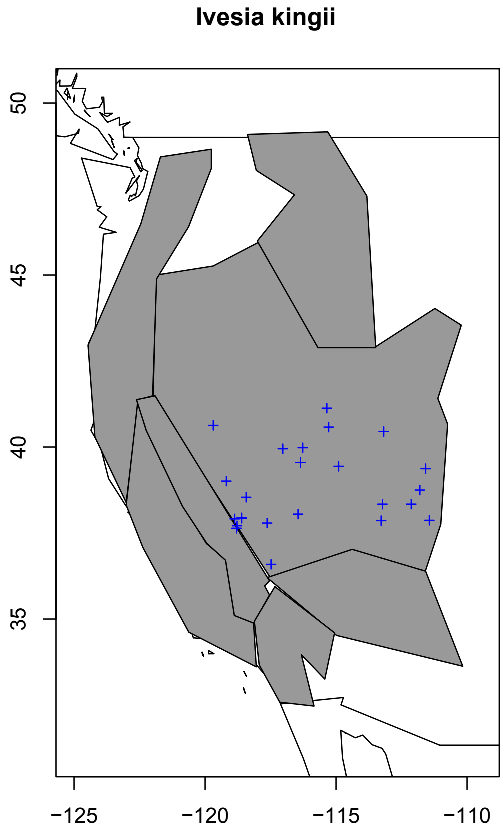
Map_samples_per_polygon.pdf

This file contains one map for each input polygon, showing all occurrence points included in the polygon color-coded per species.



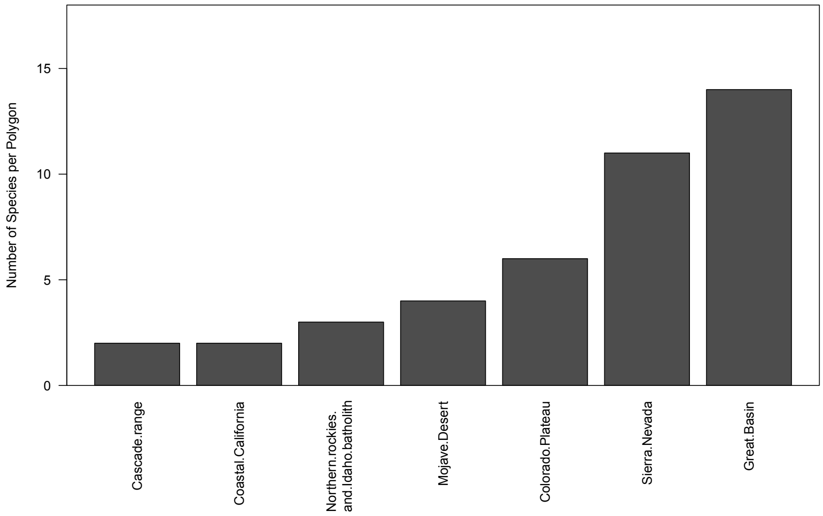
Map_samples_per_species.pdf

This file contains one map for each input species showing all polygons and all occurrence points of this species. Point samples classified to any polygon are shown in blue, unclassified samples are shown in red.



Numer_of_species_per_polygon.pdf

This file contains a barchart indicating the number of species per polygon.



Species_classification.nex

This is a nexus file containing the area coding for each species (1 = occurrence, 0 = absence) for biogeographic analysis in combination with phylogenetic data.

Sample_classification_to_polygon.txt

This is a summary table showing the classification for each sample point.

Species_occurrence_per_polygon.txt

This is a summary table showing the presence or absence of every species in the input polygons.

Species_number_per_polygon.txt

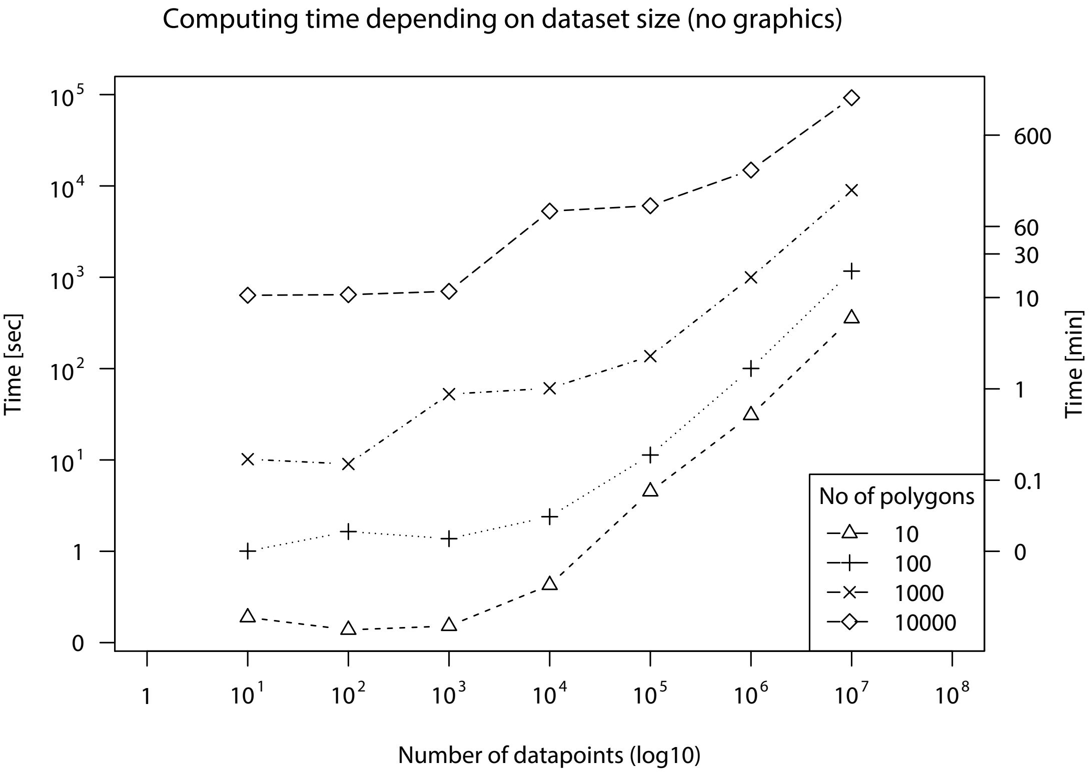
This is a table summarizing the number of species for each polygon.

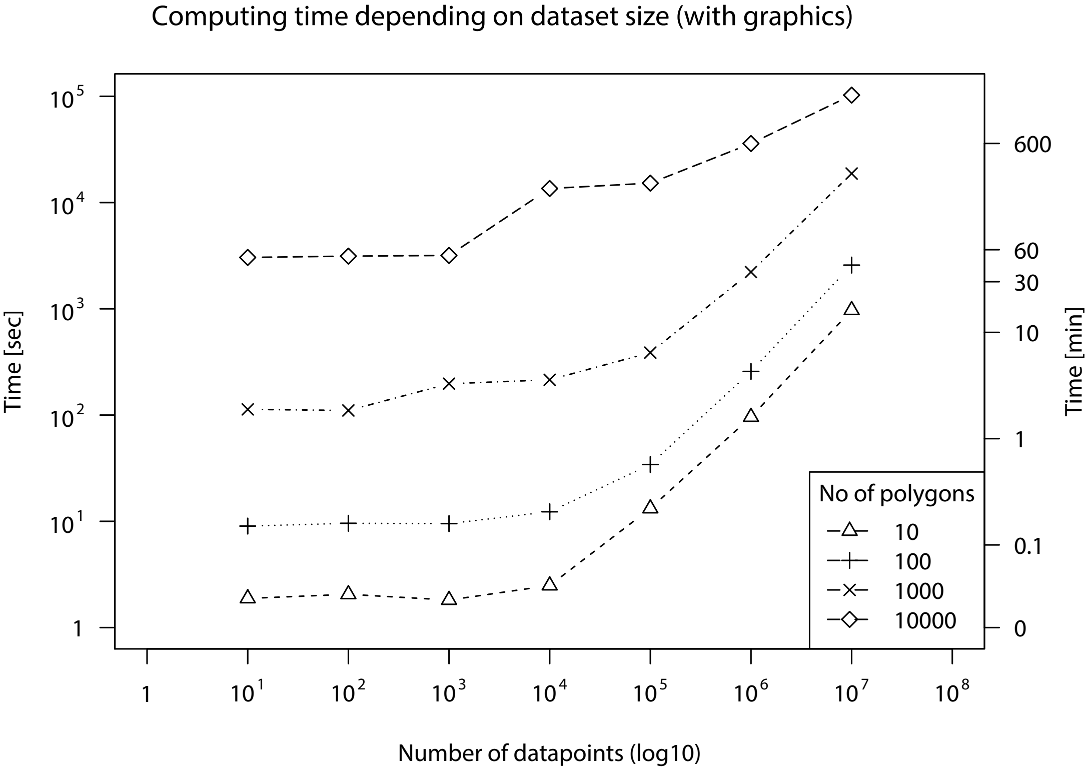
Unclassified_samples.txt

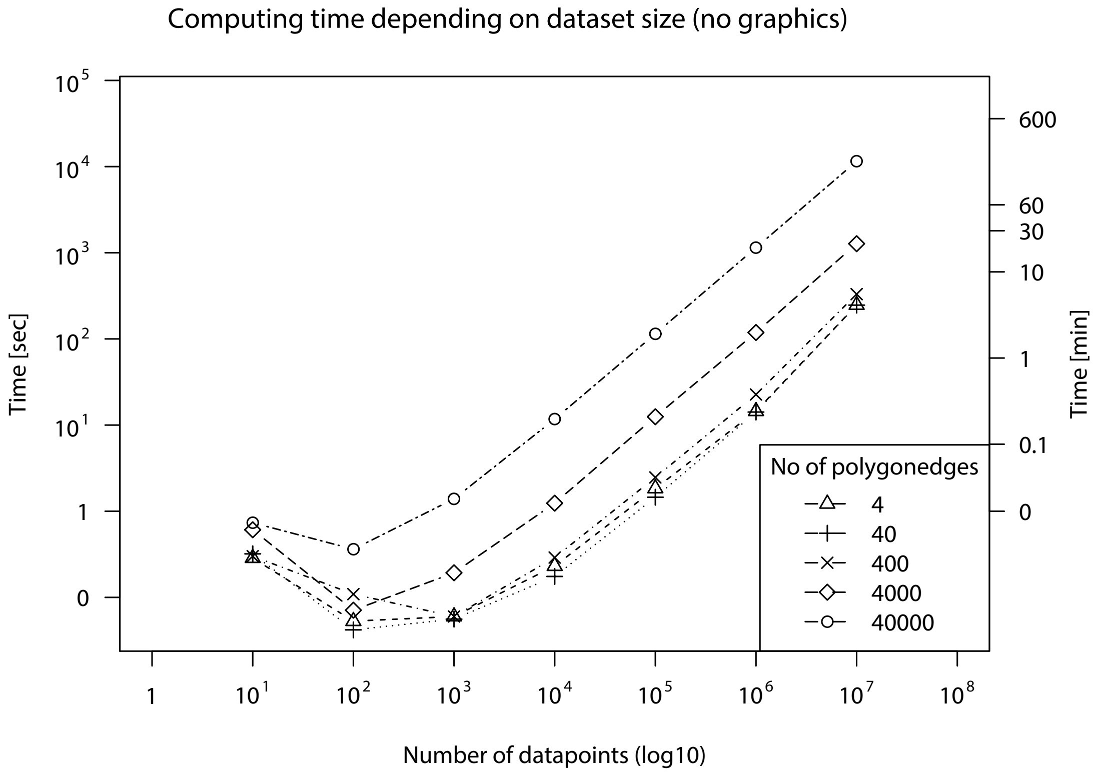
This file contains a map showing all samples that could not be classified to any of the input polygons.

6 Benchmarking

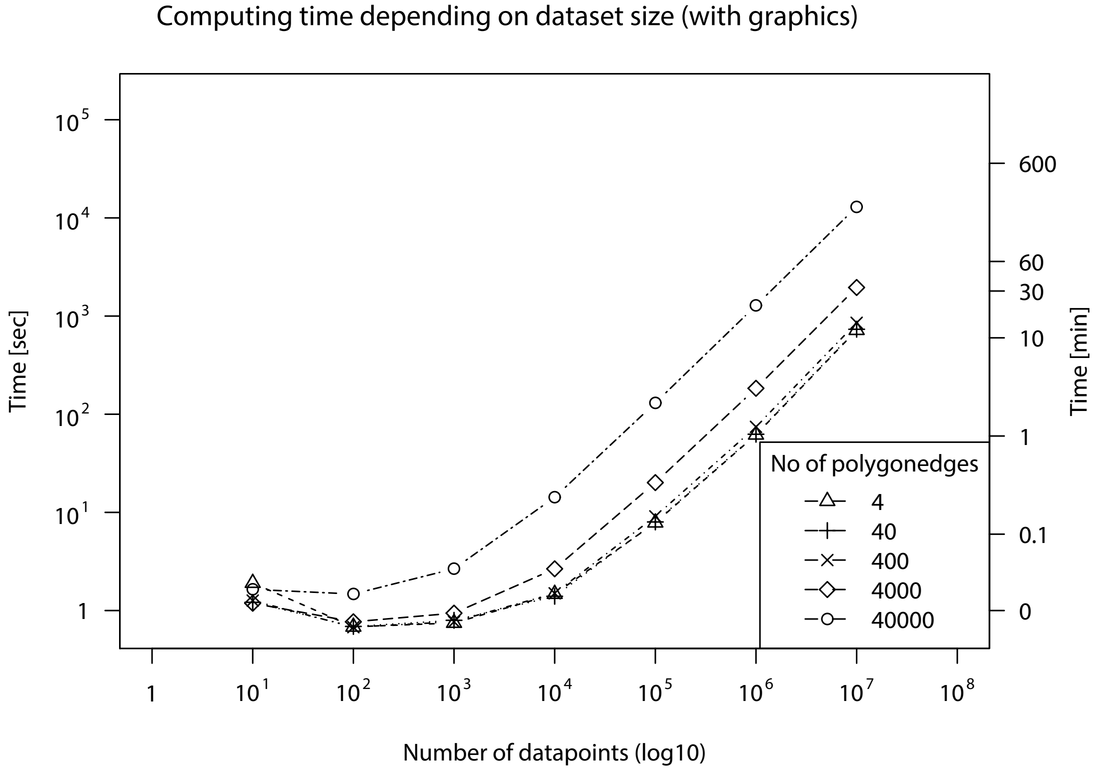
SpeciesgeocodeR was explicitly designed for the analysis of large datasets. Most analyses should run within minutes. However, very large datasets might need longer. To give you an idea of the expected computation time for you dataset using the `SpeciesGeoCoder()` function this chapter shows some result of benchmarking tests. The computation time is shown against the number of data-points in your dataset. The figures also show the impact of polygon number and polygon complexity, as well as the additional time needed when the graphical output is switched on. When using very large datasets, please note that R reads all information into the memory, and that there is a limit of 2 GB that can be allocated to a single vector.


Computing time depending on dataset size (no graphics)
Time [sec]
10^5
10^4
10^3
10^2
10^1
1
0
Time [min]
600
60
30
10
1
0.1
0
1
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
Number of datapoints (log10)
No of polygons
10
100
1000
10000


Computing time depending on dataset size (with graphics)
Time [sec]
10^5
10^4
10^3
10^2
10^1
1
Time [min]
600
60
30
10
1
0.1
0
1
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
Number of datapoints (log10)
No of polygons
10
100
1000
10000



Computing time depending on dataset size (no graphics)
Time [sec]
10^5
10^4
10^3
10^2
10^1
1
0
Time [min]
600
60
30
10
1
0.1
0
Number of datapoints (log10)
1
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
No of polygonedges
4
40
400
4000
40000



Computing time depending on dataset size (with graphics)
Time [sec]
10^5
10^4
10^3
10^2
10^1
1
Time [min]
600
60
30
10
1
0.1
0
Number of datapoints (log10)
1
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
No of polygonedges
4
40
400
4000
40000

Bibliography

Becker, R. A., Wilks, A. R., Brownrigg, R., and Minka, T. P. (2013). *maps: Draw Geographical Maps*.

Bivand, R. and Lewin-Koh, N. (2013). *maptools: Tools for reading and handling spatial objects*.

Bivand, R. and Rundel, C. (2014). *rgeos: Interface to Geometry Engine - Open Source (GEOS)*.

Bivand, R. S., Pebesma, E., and Gomez-Rubio, V. (2013). *Applied spatial data analysis with R, Second edition*. Springer.

Crawley, M. J. (2012). *The R book*. John Wiley and Sons.

GBIF (2014). Global Biodiversity Information Facility - Free and open access to biodiveristy data.

Hijmans, R. J. (2014). *Raster: Geographic data analysis and modeling*.

IUCN Standards and Petitions Subcommitte (2014). *Guidelines for using the IUCN Red List categories and criteria*, volume 11.

Jarvis, A., Reuter, H., Nelson, A., and Guevara, E. (2008). Hole-filled SRTM for the globe Version 4.

Pebesma, E. J. and Bivand, R. S. (2005). *Classes and methods for spatial data in R*.

Töpel, M., Antonelli, A., Yesson, C., and Eriksen, B. (2012). Past Climate Change and Plant Evolution in Western North America: A Case Study in Rosaceae. *PLOS ONE*, 7(12):16pp.

Töpel, M., Calio, M. F., Zizka, A., Scharn, R., Silvestro, D., and Antonelli, A. (2014). SpeciesGeoCoder: Fast categorisation of species occurrences for analyses of biodiversity, biogeography, ecology and evolution. *bioRxiv*.