A simple workflow for IsoriX

The IsoriX core team

2017-06-25

Welcome to **IsoriX**, in this vignette we present the steps required for you to build an isoscape and infer the geographic origin of your favourite animals or plants. This document is a simple introduction, we will thus not explore all the functionalities of the package. Such exploration is the subject of additional existing and future vignettes. Please let us know if you would like us to create vignettes on specific topics. You can also discover advanced functionalities by reading the documentation of the package.

Note that this vignette takes a lot of time to run. It has thus not been compiled by CRAN but by us. We have included the sources of this vignette as a text file which you can find in the folder `IsoriX/doc` within the folder where you usually install your packages (the latter being usually the one that shows up when you type `.libPaths()[1]` in R).

In addition, due to constraints on how big this document could be, we had to reduce a lot the resolution of the figures. If you run it on your computer you will see that figures produced by **IsoriX** are great!

A final remark before we begin is that we are no longer allowed to distribute freely the data used to produce this vignette. Every user must, from now on, compile its own dataset. You can download precipitation isotope data from the Global Networks of Isotopes in Precipitation (GNIP; http://www-naweb.iaea.org/napc/ih/IHS_resources_isohis.html). The data are free to download after the registration process is completed. For the time being, downloads are limited to 10,000 records per batch, which makes the compilation of huge databases (such as the one used here) fastidious. GNIP promised to come up, in the future, with datasets directly prepared for **IsoriX** to save you such trouble.

Step 1 - Loading IsoriX

Before using the package, you need to start R and define your working directory using the function `setwd()`. You also need to have installed **IsoriX** on your system. We assume that you already know R a little bit and that you thus know how to perform such basic operations. If it is not the case, you should read *An Introduction to R* or any other introduction to R before continuing. Once your R session is ready, you should load the package as follows:

```
library(IsoriX)
```

```
##
##  IsoriX version 0.5.1.9008 is loaded!
##
##  Many functions and objects have changed names since the version 0.4.
##  This is to make IsoriX more intuitive for you to use.
##  We will do our best to limit changes in names in the future!!
##
##  Type:
##     * ?IsoriX for a short description.
##     * browseVignettes(package = 'IsoriX') for tutorials.
##     * news(package = 'IsoriX') for news.
```

Step 2 - Prepare the full dataset containing the data required to build an isoscape

You must start by building a dataset containing the data needed for the construction of the isoscape. In this example, we are going to build a large dataset called `GNIPData`. This dataset derives from data kindly provided to us by GNIP. It contains data most of the data GNIP possesses for deuterium.

This is what the raw dataset provided by GNIP looks like:

```
rawGNIP <- read.csv("2016-10-31 Extract_ISORIX.csv")  ## we load the data
t(rawGNIP[1, ]) ## we display the first row for this dataset as a column for visualisation
```

```
##                          1
## Name.of.Site             "NY ALESUND"
## Country                  "NO"
## WMO.Code                 "100400"
## Latitude                 "78.91"
## Longitude                "11.93"
## Altitude                 "7"
## Type.Of.Site             "Precipitation Collectors - not specified"
## Source.of.Information    "[Partners: Sverdrup Station, Norsk Polarinstitutt], [Info: Major revision in 1
## Sample.Name              "199001                    "
## Media.Type               "Water - Precipitation - rain & snow (\"mixed\")"
## Date                     "1990-01-15 00:00:00"
## Begin.Of.Period          "1990-01-01 00:00:00"
## End.of.Period            "1990-01-31 00:00:00"
## Comment                  ""
## O18                      "-9.83"
## O18.Laboratory           "International Atomic Energy Agency, Vienna, AT"
## H2                       "-72.8"
## H2.Laboratory            "International Atomic Energy Agency, Vienna, AT"
## H3                       "6.9"
## H3.Error                 "0.5"
## H3.Laboratory            "International Atomic Energy Agency, Vienna, AT"
## Precipitation            "27.6"
## Air.Temperature          "-6.4"
## Vapour.Pressure          "2.6"
```

We are now going to reshape these data to make them ready for **IsoriX**.

First, we reformat some temporal information:

```
rawGNIP$year.begin  <- as.numeric(format(as.Date(rawGNIP$Begin.Of.Period), "%Y"))
rawGNIP$year.end    <- as.numeric(format(as.Date(rawGNIP$End.of.Period), "%Y"))
rawGNIP$year.span   <- rawGNIP$year.begin - rawGNIP$year.end
rawGNIP$month.begin <- as.numeric(format(as.Date(rawGNIP$Begin.Of.Period), "%m"))
rawGNIP$month.end   <- as.numeric(format(as.Date(rawGNIP$End.of.Period), "%m"))
rawGNIP$day.span    <- as.Date(rawGNIP$Begin.Of.Period) - as.Date(rawGNIP$End.of.Period)
rawGNIP$Year        <- as.numeric(format(as.Date(rawGNIP$Date), "%Y"))
rawGNIP$Month       <- as.numeric(format(as.Date(rawGNIP$Date), "%m"))
```

Second, we identify the rows for which crucial information is missing. Because we are going to make an isoscape based on deuterium measurements, we only check for the availability of data for this isotope in particular. If you work on oxygen, you will have to adjust the script accordingly. We also check that the intervals during which precipitation water has been collected for each measurement correspond roughly to one month:

```
rows_missing_info <- is.na(rawGNIP$H2) | is.na(rawGNIP$day.span) |
                     rawGNIP$day.span > -25 | rawGNIP$day.span < -35
```

Third, we only keep the rows and columns we are interested in:

```
columns_to_keep <- c("Name.of.Site", "Latitude", "Longitude", "Altitude",
                     "Year", "Month", "H2")
GNIPData <- rawGNIP[!rows_missing_info, columns_to_keep]
```

Fourth, we turn the variable `Name.of.Site` into a factor:

```
GNIPData$Name.of.Site <- as.factor(GNIPData$Name.of.Site)
```

Last, we rename the columns to conform to the general **IsoriX** format and we check that the data seem correct:

```
colnames(GNIPData) <- c("stationID", "lat", "long", "elev", "year", "month", "isoscape.value")
str(GNIPData)
```

```
## 'data.frame':    62279 obs. of  7 variables:
##  $ stationID     : Factor w/ 1095 levels "ABISKO","ACQUA ROSSA",..: 693 693 693 693 693 693 693 693 6
##  $ lat           : num  78.9 78.9 78.9 78.9 78.9 ...
##  $ long          : num  11.9 11.9 11.9 11.9 11.9 ...
##  $ elev          : num  7 7 7 7 7 7 7 7 7 7 ...
##  $ year          : num  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
##  $ month         : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ isoscape.value: num  -72.8 -67.6 -107 -77.8 -77.1 -67.3 -80 -97.1 -68.4 -79.2 ...
```

This is what the `GNIPData` dataset now looks like:

```
head(GNIPData) ## we display the first six rows of the dataset
```

```
##    stationID   lat  long elev year month isoscape.value
## 1 NY ALESUND 78.91 11.93    7 1990     1          -72.8
## 2 NY ALESUND 78.91 11.93    7 1990     2          -67.6
## 3 NY ALESUND 78.91 11.93    7 1990     3         -107.0
## 4 NY ALESUND 78.91 11.93    7 1990     4          -77.8
## 5 NY ALESUND 78.91 11.93    7 1990     5          -77.1
## 6 NY ALESUND 78.91 11.93    7 1990     6          -67.3
```

If you want to practice using the package, we have included a small dataset containing deuterium precipitation data for Germany that is directly ready to use. It is called `GNIPDataDE`. You can access this dataset by simply typing its name:

```
head(GNIPDataDE)
```

```
##   stationID   lat long elev year month isoscape.value
## 1 SCHLESWIG 54.52 9.54   43 1997     6          -59.0
## 2 SCHLESWIG 54.52 9.54   43 1997     7          -56.0
## 3 SCHLESWIG 54.52 9.54   43 1997     8          -60.8
## 4 SCHLESWIG 54.52 9.54   43 1997     9          -51.0
## 5 SCHLESWIG 54.52 9.54   43 1997    10          -58.7
## 6 SCHLESWIG 54.52 9.54   43 1997    11          -74.6
```

Step 3 - Filter and aggregate the dataset required to build the isoscape

In order to build your isoscape with **IsoriX**, your dataset must be aggregated in such a way that each observation corresponds to the mean and variance of isotope values collected in one location, and potentially over a fixed period of time. You can choose to aggregate your dataset on your own, or to use our function `queryGNIP()`. This function allows you to aggregate the data over different time intervals. It also allows you to apply some restriction on the data, such as selecting only particular months, years, or locations.

Here we will use the function `queryGNIP()` to select from the dataset `GNIPData` the observations available within an extent of latitude and longitude that covers roughly Europe. We will use the default aggregation scheme, which simply produces an overall average per location.

```
GNIPDataEU <- queryGNIP(data = GNIPData,
                        long.min = -30, long.max = 60,
                        lat.min = 30, lat.max = 70)
```

Let us now visualise the first 6 rows of the dataset:

```
head(GNIPDataEU)
```

```
##   stationID isoscape.value var.isoscape.value n.isoscape.value   lat
## 1     ADANA      -23.86009          273.27022              336 36.98
## 2   AGUALVA      -16.21667           48.86015               12 38.77
## 3  AL BAQAA      -26.89000          128.79878               10 32.09
## 4   AL-JAFR        0.48000          721.87200                5 30.33
## 5    ALAGOA      -14.96667           41.24697               12 38.79
## 6    ALEPPO      -32.13900          383.20099               10 36.18
##     long elev
## 1  35.30   73
## 2 -27.19  260
## 3  35.77  700
## 4  36.14  870
## 5 -27.19   70
## 6  37.21  410
```

Important: before continuing to the next step, you must make sure that your final dataset contains the same type of information and column names than the dataset we have just created!

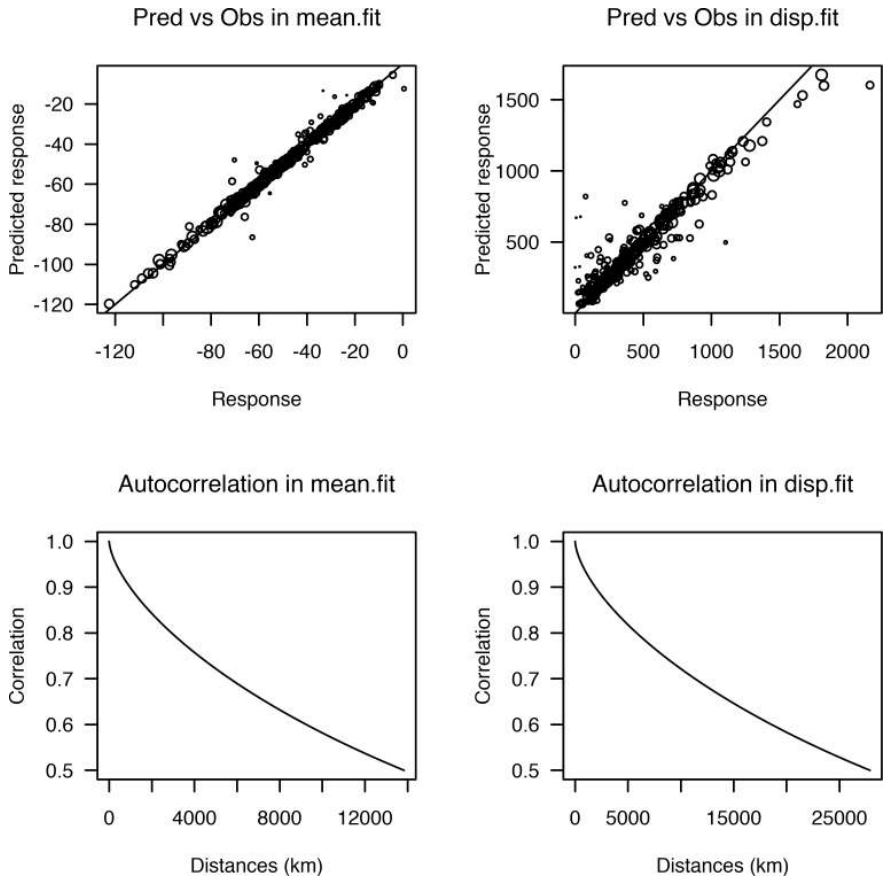
Step 4 - Fit the geostatistical model

We are now going to use the function `isofit()` to fit the geostatisical model required to build isoscapes. This function has several parameters that can be adjusted to fit different models, but we will here use its default settings.

```
Europefit <- isofit(iso.data = GNIPDataEU)
```

You can check the output of the model fit by typing its name (not shown), and you can also use the function `plot()` from our package to check visually some properties of the model fit.

```
plot(Europefit)
```



In the panel produced, the left column shows the relationship between the observed and predicted response (top) and the variation in spatial autocorrelation with the distance between location (bottom) captured by the model for the subfit called **mean.fit**, which corresponds to the fit of the mean isotopic values. The right column shows the same information for the subfit called **disp.fit**, which corresponds to the fit of the residual dispersion variance in the isotope values. On the first row you can see points distributed practically along the 1:1 diagonal. A different slope would suggest a high residual variance of the data. We do not expect the points to fall exactly on the line because the model fit does not attempt to predict perfectly the observations used during the fit. Instead, the model fit produces a smooth surface that aims at reaching a maximum predictive power for locations not considered during the fit. The second row gives you an idea of the strength of the spatial autocorrelation captured by the models. Here the results suggest that the autocorrelation is very strong. Not considering this autocorrelation would thus lead here to a poor approximation of the isoscape and resulting assignments.

Step 5 - Prepare the elevation raster

To build the isoscape using the geostatistical model fitted above, you need to provide the function `isoscape()` with an elevation raster. We will thus start by preparing such raster. You first need to download a high resolution raster from Internet. We are storing the *Global Multi-resolution Terrain Elevation Data 2010*, such high resolution raster, on our server, and you can easily download it from R using our function `getelev`:

```
getelev()
```

We then import the high resolution elevation raster and transform it into an R object of class `RasterLayer` object using the package **raster**:

```
## we load the package raster
library(raster)

## we import the high resolution raster
elevationraster <- raster("gmted2010_30mn.tif")

## we check that the import worked
elevationraster
```

```
## class       : RasterLayer
## dimensions  : 20880, 43200, 902016000  (nrow, ncol, ncell)
## resolution  : 0.008333333, 0.008333333  (x, y)
## extent      : -180.0001, 179.9999, -90.00014, 83.99986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : /Users/alex/Dropbox/Boulot/Mes_projets_de_recherche/R_packages/IsoriX_project/sources_
## names       : gmted2010_30mn
## values      : -430, 8625  (min, max)
```

The information for the field data source shows where the elevation data are located on your computer. The content will be thus be different for you. The other fields should however be identical.

Then, you may need to crop and resize this raster to the appropriate size. Here, this is justified because the elevation raster covers the entire planet but we are only interested in Europe. You can do this easily by using our function `relevate()`. To crop the elevation raster to a particular size, you can either provide a set of coordinates to the function by means of the argument `manual.crop`. Alternatively, you can simply provide a fitted model to the function `relevate()` by means of the argument `isofit`. In this latte case, the function will determine the coordinates required for the cropping automatically, from the coordinates of the weather stations used during the fitting procedure. We choose here to use this latter alternative.

We also choose here to reduce the resolution of the elevation raster by choosing an aggregation factor of 10. The lower the aggregation factor, the higher the resolution of the elevation raster will be, but the longest the computation time and the largest the map produced will also be. We recommend to use the lowest aggregation factor that you hardware and patience can handle.

```
elev <- relevate(elevation.raster = elevationraster,
                 isofit = Europefit,
                 aggregation.factor = 10)
```

```
## class       : RasterLayer
## dimensions  : 514, 1115, 573110  (nrow, ncol, ncell)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : /private/var/folders/r4/nmf9vqyj7pz968sk2vrtmbvm0000gn/T/RtmpFKXgAc/raster/r_tmp_2017-
## names       : gmted2010_30mn
## values      : -412, 4060.89  (min, max)
```

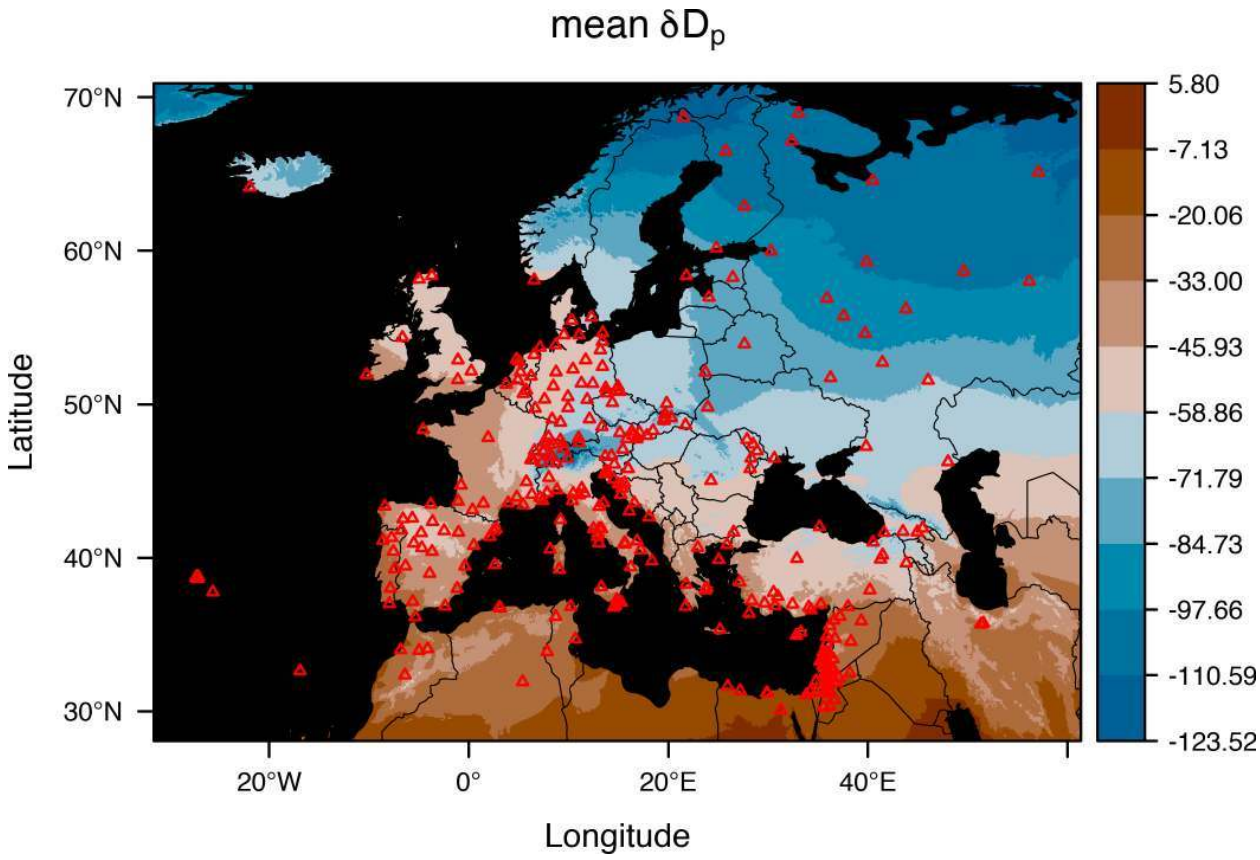
Step 6 - Build the isoscape

We will now build the isoscape using the function `isoscape()`. The function simply takes the elevation raster and the fitted model as arguments:

```
isoscape <- isoscape(elevation.raster = elev, isofit = Europefit)
```

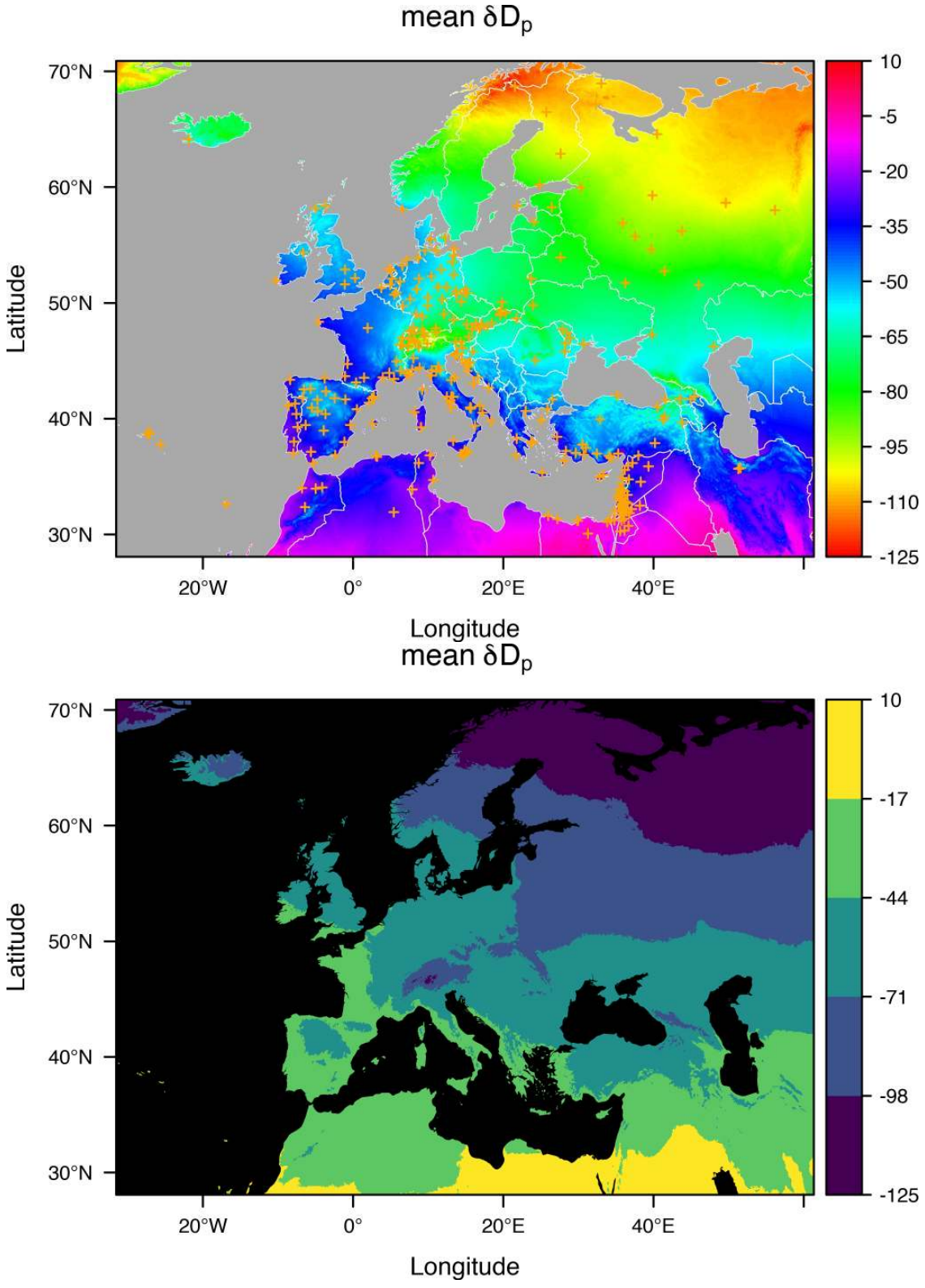
The function `isoscape()` creates 8 different rasters that correspond to different aspects of the isoscape (see `?isoscape()` for details). Let us start by plotting the usual isoscape, that is the one showing the prediction of the average isotope value in space (i.e. the point predictions):

```
plot(x = isoscape)
```



The plotting functions we created for **IsoriX** are quite flexible and allows you to create quite different looking isoscapes. We will now redraw the previous figure couple of times with different settings to illustrate some of the things you can change (see `?plot.isocape` for details):

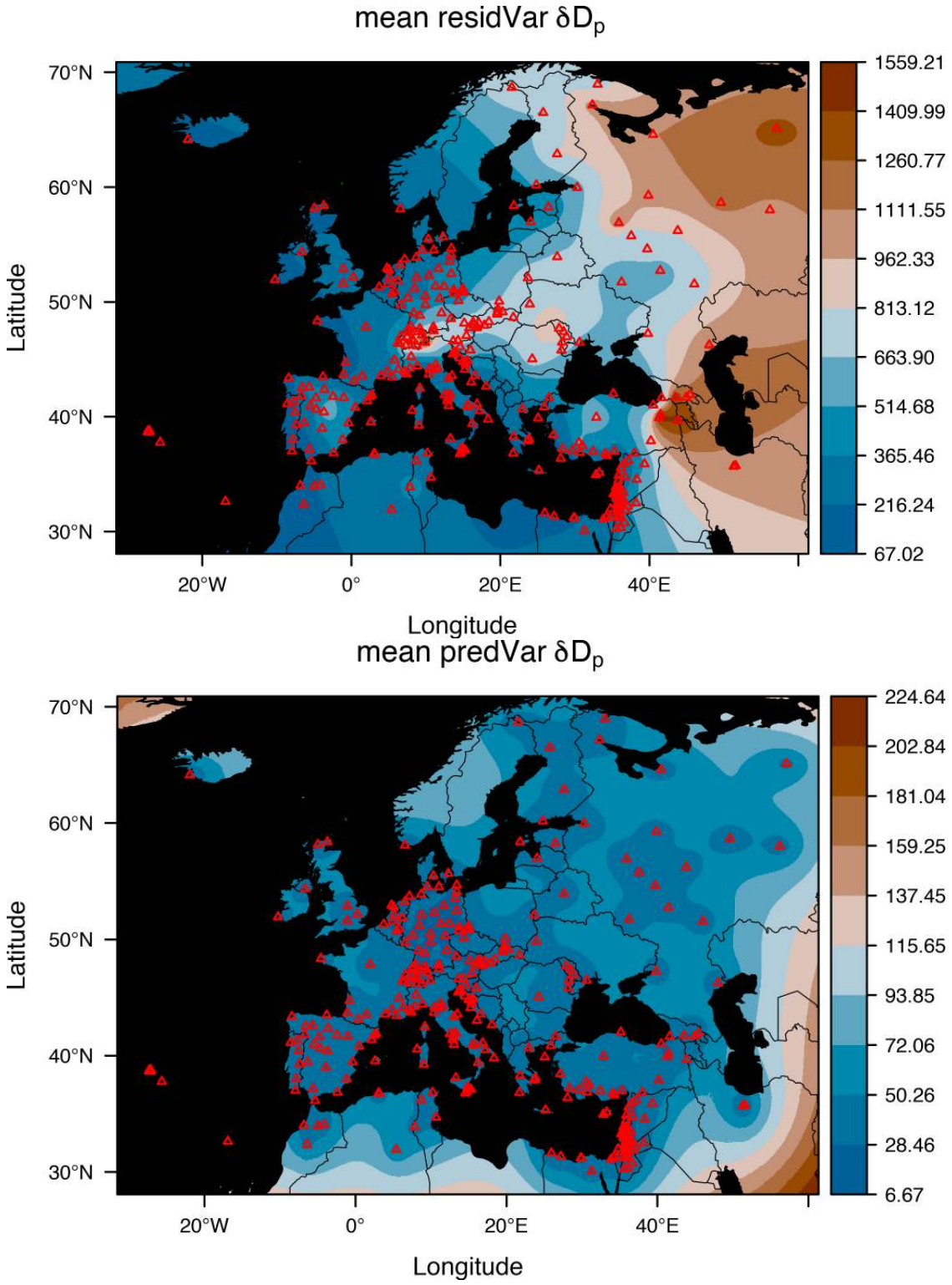
```
plot(x = isoscape,
     sources = list(pch = 3, col = "orange"),
     borders = list(col = "white"),
     mask    = list(fill = "darkgrey"),
     palette = list(range = c(-125, 10), step = 1, n.labels = 10, fn = "rainbow"))

plot(x = isoscape,
     sources = list(draw = FALSE),
     borders = list(borders = NULL),
     mask    = list(fill = "black"),
     palette = list(range = c(-125, 10), step = 27, fn = NULL))
```



As you can see, it is possible to provide any function providing colours to the argument `fn` from the list `palette`. When it is set to `NULL`, the palette viridis is used instead of our default palette. Notice also that we have changed the way values are plotted along the colorkey.

As examples of complementary plot you could draw, we are now going to plot the residual variation in isotope values predicted at each location, and the prediction variance, which quantifies the uncertainty of our point predictions:

```
plot(x = isoscape, which = "mean.residVar")
plot(x = isoscape, which = "mean.predVar")
```



If your goal was to produce isoscapes, you should be done by now. The following steps of this tutorial are for users interested in inferring the origin of organisms based on their isotopic signature.

Step 7 - Fit the calibration function

In this example, we will use the calibration dataset `CalibDataBat` that is provided with **IsoriX**. This dataset concerns sedentary bats inhabiting Europe (see `?CalibDataBat` for details). We now use the function `calibfit()` to fit the relationship between the isotope values in the environment and the isotope values in our calibration organisms:

```
calib <- calibfit(calib.data = CalibDataBat, isofit = Europefit)
```

```
## Warning in .calc_logdisp_cov_new(object = object, dvdloglamMat =
## dvdloglamMat, : phi dispVar component not yet available for phi model !=
## ~1.
```

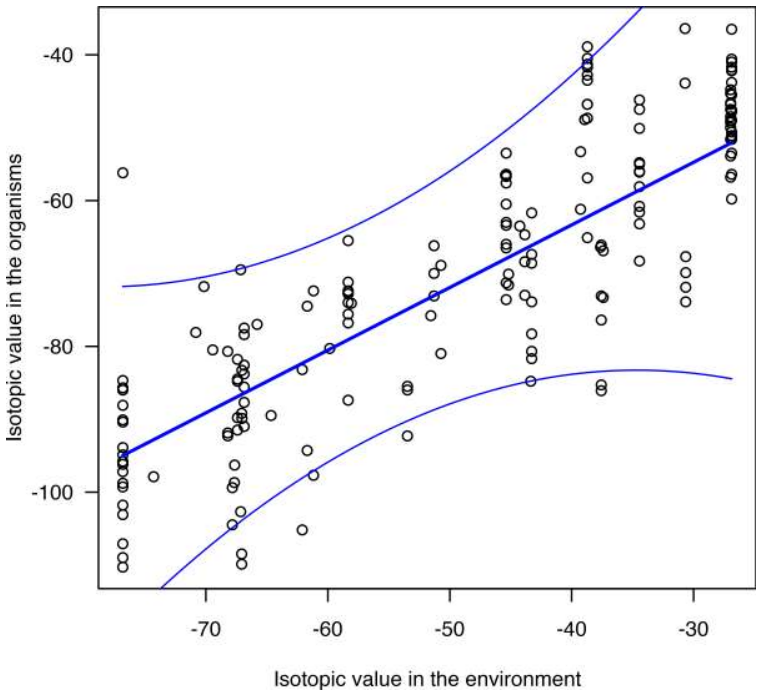
These warnings should disappear in the future. They reflect limitations of the current version of the package **spaMM**, which we use to fit our models.

To get the result of the calibration fit, you can simply type the name of the object we just created, or plot it:

```
calib
```

```
##
## Fixed effect estimates of the calibration fit
## tissue.value = intercept + slope * mean.iso +   corrMatrix(1|siteID) + slope^2 * (1|siteID) + Error
##
##            intercept (+/- SE) = -28.96 +/- 6.30
##            slope     (+/- SE) =   0.86 +/- 0.10
##
## [for more information, use summary()]
```

```
plot(calib)
```



Step 8 - Perform the assignment

We will now perform the assignment procedure that aims at identifying areas with the same isotopic signature than the area where individuals originate. To put this into practice, we will use a dataset containing isotopic measurements on other bats, which is also provided in **IsoriX**. We will select from this dataset the bats of the species "Myotis_bechsteinii":

```
M_bechsteinii <- subset(AssignDataBat, species == "Myotis_bechsteinii")
M_bechsteinii
```

```
##    animalID            species tissue.value
## 1     Mbe_1 Myotis_bechsteinii        -68.8
## 2     Mbe_2 Myotis_bechsteinii        -64.3
## 3     Mbe_3 Myotis_bechsteinii        -72.9
## 4     Mbe_4 Myotis_bechsteinii        -76.1
## 5     Mbe_5 Myotis_bechsteinii        -69.9
## 6     Mbe_6 Myotis_bechsteinii        -60.9
## 7     Mbe_7 Myotis_bechsteinii        -63.4
## 8     Mbe_8 Myotis_bechsteinii        -68.0
## 9     Mbe_9 Myotis_bechsteinii        -72.9
## 10   Mbe_10 Myotis_bechsteinii        -60.4
```

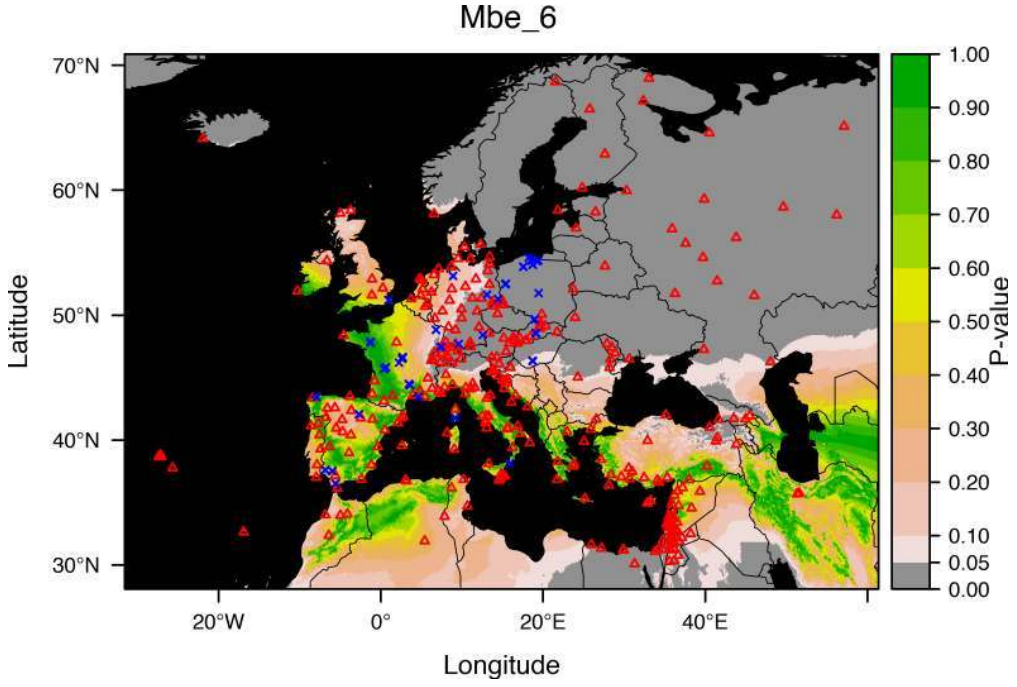
We now perform the assignment using our function `isofind()`:

```
assignment <- isofind(assign.data = M_bechsteinii, isoscape = isoscape, calibfit = calib)
```

Note that by default the function `isofind()` exclude with certainty that the origin of the bats could come from the water.

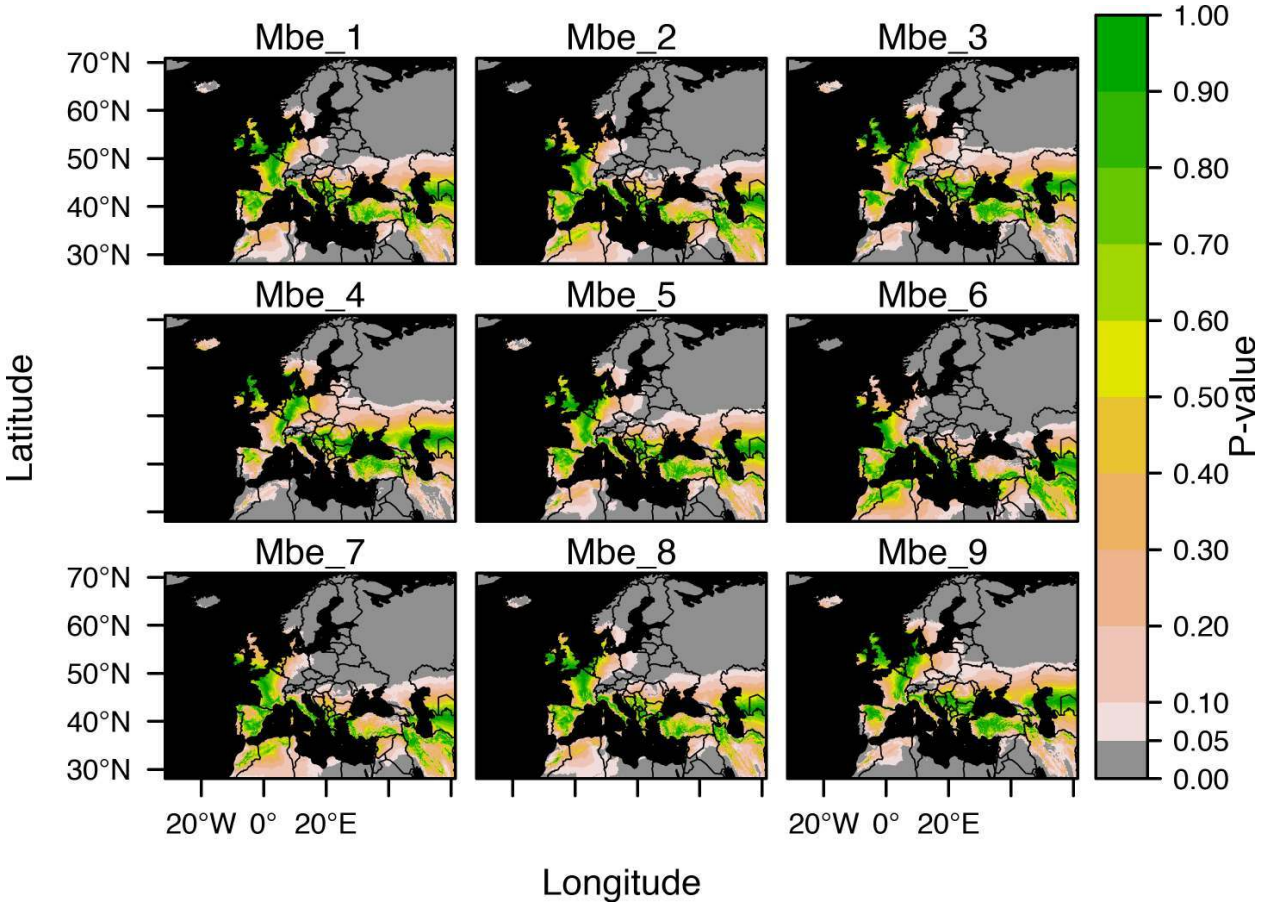
Once the assignment done, you can draw several assignment maps. Let us start by plotting the assignment map for a specific bat by telling the function `plot` that `who` we want the plot for the bat with the `animalID` "Mbe_6" for example:

```
plot(x = assignment, who = "Mbe_6")
```



We can also use number for individuals and plot several at once:

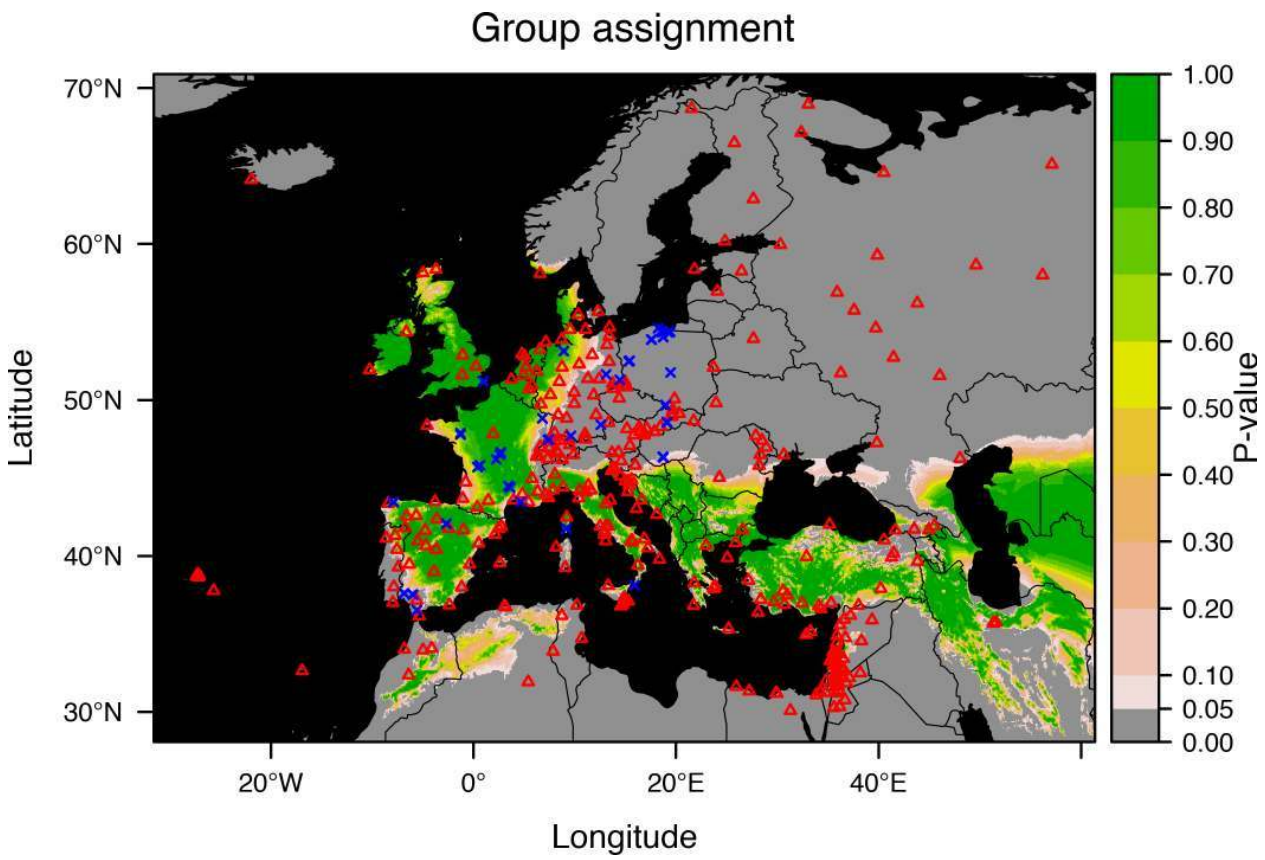
```
plot(x = assignment,
     who = 1:9,
     sources = list(draw = FALSE),
     calib = list(draw = FALSE))
```



Note that we removed the plotting of sources and calibration points in the previous plot by setting using `draw = FALSE`.

Assuming that all bats originate from the same place we can also plot the global assignment for the group by setting `who = "group"`:

```
plot(x = assignment, who = "group")
```



We can easily add information on these plots. As an example, we will add the point that is the most compatible with the unknown origin of these bats. We start by extracting its location using functions from the package **raster**:

```
coord <- coordinates(assignment$group$pv)
max.location <- coord[which.max(values(assignment$group$pv)), ]
maximum <- data.frame(long = max.location[1], lat = max.location[2])
maximum
```

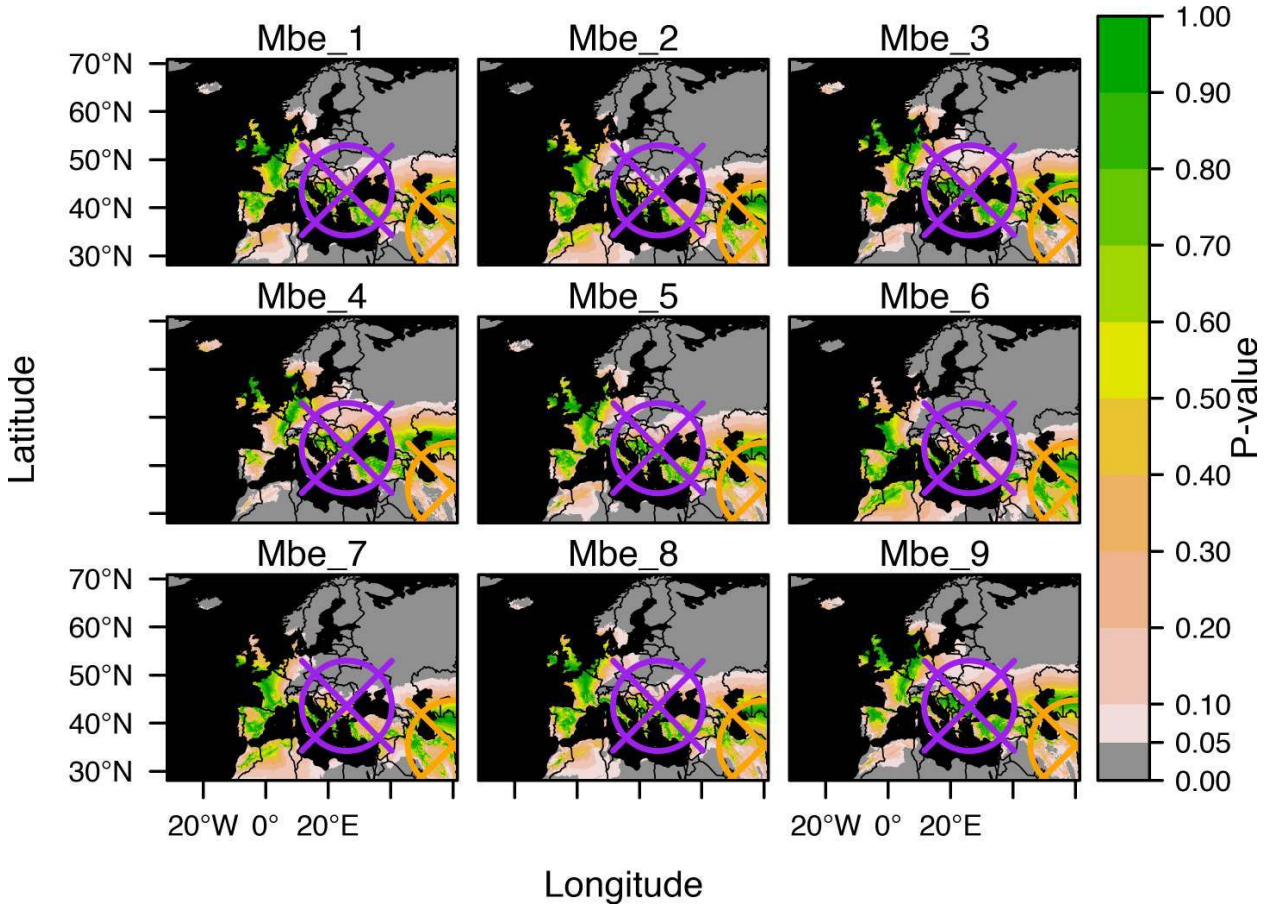
```
##       long      lat
## x 59.98319 35.35819
```

Let us also compute for comparison the point that corresponds to the real origin of bats. We know it because they were caught in a cave in the North of Bulgaria:

```
origin <- data.frame(long = 25.982122, lat = 43.611536)
```

You can plot these information on top of the last plot we created by means of the package **lattice**:

```
library(lattice)
trellis.last.object() +
  xyplot(origin$lat~origin$long, pch = 13, col = "purple", cex = 5, lwd = 2,
      panel = panel.points) +
  xyplot(maximum$lat~maximum$long, pch = 13, col = "orange", cex = 5, lwd = 2,
      panel = panel.points)
```



The function `trellis.last.object()` is very handy: it retrieves the last plot created using **lattice** and our plot functions for isoscapes and assignments use **lattice** since they use **rasterVis**, which is a package itself based on **lattice**.

As you can see the location of origin (purple) and the location for the best assignment (orange) differ a bit. The flexibility of **IsoriX** helps to explore potential sources of problems and limitation. We can for example compare our isoscape prediction in these two locations:

```
print(paste("mean isoscape value at origin (+/-SE) =",
            round(extract(isoscape$isoscape$mean, origin), 2), "+/-",
            round(sqrt(extract(isoscape$isoscape$mean.predVar, origin)), 2)))
```

```
## [1] "mean isoscape value at origin (+/-SE) = -53.5 +/- 6.93"
```

```
print(paste("mean isoscape value at maximum (+/-SE) =",
            round(extract(isoscape$isoscape$mean, maximum),2), "+/-",
            round(sqrt(extract(isoscape$isoscape$mean.predVar, maximum)), 2)))
```

```
## [1] "mean isoscape value at maximum (+/-SE) = -45.68 +/- 12.39"
```

Note that the estimate for the mean isotopic value where the maximum is has a large standard error as this area is not well covered by sources. You can also observe this effect by simply looking at the map for the prediction isoscape variance which we have plotted above. In this context, it is thus expected that such area will be difficult to rule out as a possible source of origin. This example is a good reminder that one must always think about the quality of the isoscape before drawing definitive biological conclusion.

Another way to look at our assignment is to check if for all bats the predicted assignment region does include the real location of origin (which is possible because here we do know what the real origin of the bats is). To

do so, we start by extracting the p-values of the assignment at the location of origin for all individuals:

```
pvs <- c(extract(assignment$indiv$pv, origin))
```

We can then count for how many bats the true origin location is not rejected by our assignment test:

```
table(pvs > 0.05)
```

```
##
## TRUE
##   10
```

All bats are in! That is great but it will not necessarily happen all the time, especially if as here the calibration is performed on a different species than the individuals you want to allocate. As an exercise you can now try to plot the bats we assigned into the calibration fit to see if they do behave in the same way as other species.

Next?

For other tutorials on **IsoriX**, check the other vignettes we created for you by simply typing:

```
browseVignettes(package ='IsoriX')
```

Please note that the number and the quality of the vignettes keeps growing with updates of **IsoriX**. So keep updating your packages in R and check what has been changing between two versions by typing:

```
news(package = 'IsoriX')
```

The End

That is all for now! Here are the information of the R session we used:

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] lattice_0.20-35   raster_2.5-8      sp_1.2-4          IsoriX_0.5.1.9008
## [5] knitr_1.16
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.11    digest_0.6.12   rprojroot_1.2   grid_3.4.0
##  [5] backports_1.1.0 magrittr_1.5    evaluate_0.10   stringi_1.1.5
##  [9] rmarkdown_1.6   tools_3.4.0     stringr_1.2.0   yaml_2.1.14
## [13] compiler_3.4.0  htmltools_0.3.6
```