A simple workflow for IsoriX

The IsoriX core team

2017-06-01

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 will be the subject of additional vignettes that will be available with future versions of the package. For the time being, users can 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 your computer. This file is 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 in fact figures produced by **IsoriX** are great!

Step 0 - 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 can load the package:

```
library(IsoriX)
```

```
##
##  IsoriX version 0.5 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 1 - Select the isoscape data

Start by selecting the GNIP data needed for us to build an isoscape. In this example, we will consider all the data available in `GNIPdata` within an extent of latitude and longitude that covers roughly Europe.

```
## load all the GNIP data
data(GNIPdata)

## select the relevant data
GNIPdataEU <- queryGNIP(
    data=GNIPdata,
```

```
    long.min = -30,
    long.max = 60,
    lat.min = 30,
    lat.max = 70)
```

A warning appears because repeated measurements at one location are associated with different elevations. This is a mismatch in the original data, and the function `queryGNIP()` will only select the first elevation found for this location in the dataset.

The function `queryGNIP` could also be used to select data from particular years or months. The function aggregates data so that there are as many rows in `GNIPdataEU` as the number of weather stations considered. If you explore the dataset we created, you will notice that it has 322 rows and 7 columns. Typing `head(GNIPdataEU)` will show you the first 6 rows of the dataset. We should get the following output:

stationID	isoscape.value	var.isoscape.value	n.isoscape.value	lat	long	elev
142700	-46.19083	195.0560	109	58.10	6.57	13
206000	-122.47571	919.5862	70	68.68	21.53	403
284500	-104.19625	618.5105	80	66.49	25.75	107
291701	-101.16667	877.2270	69	62.89	27.62	116
297401	-83.32869	559.0186	122	60.18	24.83	30
304401	-50.63818	139.2143	22	58.15	-4.97	73

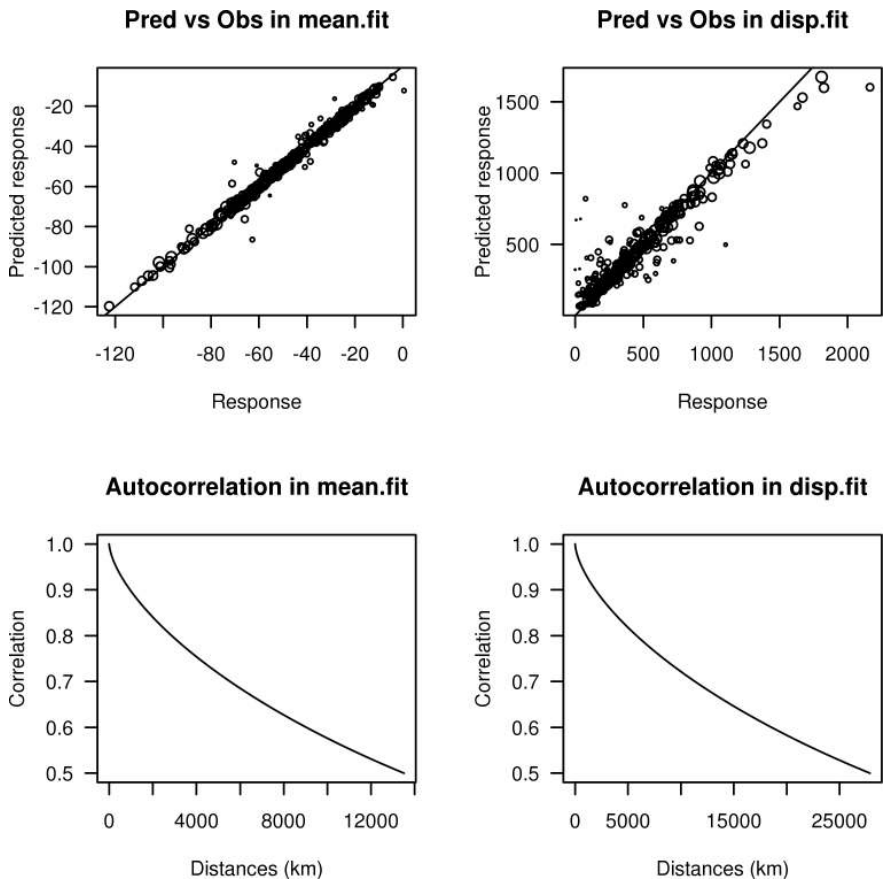
Step 2 - Fit the geostatistical model

The second thing you need to do is to fit the geostatisical model using the function `isofit()`. This function has many parameters that can be ajusted to fit different models, but we will here use its default settings.

```
Europefit <- isofit(iso.data=GNIPdataEU,
    mean.model.fix=list(elev=TRUE, lat.abs=TRUE),
    mean.model.rand=list("uncorr"=TRUE),
    disp.model.rand=list("uncorr"=TRUE))
```

We should check the output of the model fits 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 fits.

```
plot(Europefit)
```



The left column of this panel 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 model fit called **mean.fit**, which corresponds to the fit of the mean isotopic values. The right column shows the same information for the model fit called **disp.fit**, which corresponds to the fit of the residual dispersion variance in the isotopic values. On the first row you observe 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 observation 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 poor approximation of the isoscape and resulting assignments.

Step 3 - Prepare the elevation raster

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

```
getelev()
```

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

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

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

## we always 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=longat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : /media/alex/Data/Dropbox/Boulot/Mes_projets_de_recherche/R_packages/IsoriX_project/sou
## names       : gmted2010_30mn
## values      : -430, 8625  (min, max)
```

Then, we crop and resize it using our function `relevate()`. The crop is automatically set by default to the extent corresponding to the dataset we used above (`GNIPdataEU`), which the function access through the fitted `Europefit`. To resize the raster we chose here 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  : 467, 1014, 473538  (nrow, ncol, ncell)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -27.34181, 57.15819, 30.04153, 68.95819  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : /tmp/Rtmp30b2wv/raster/r_tmp_2017-04-11_184234_17946_58549.grd
## names       : gmted2010_30mn
## values      : -412, 4318.32  (min, max)
```

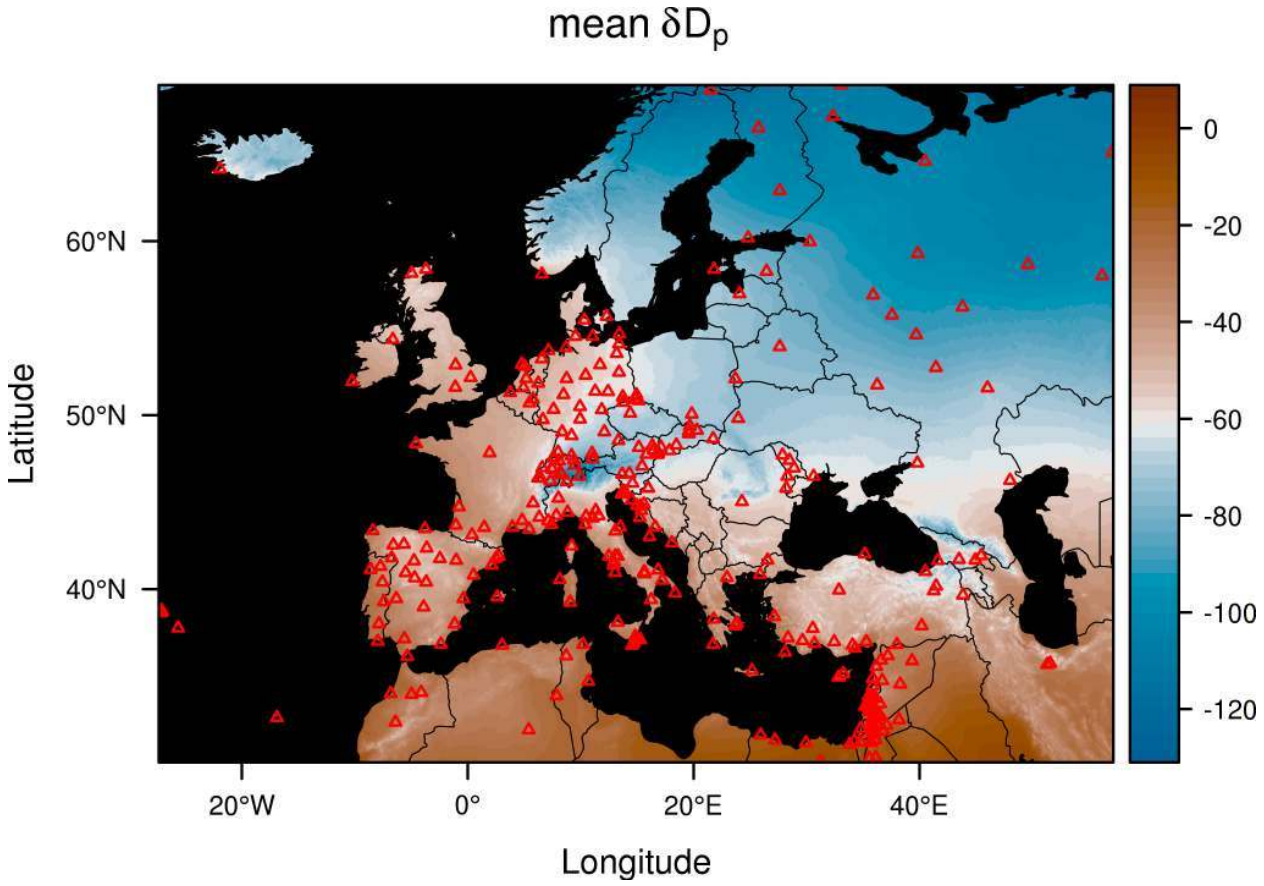
Step 4 - Build the isoscape

We will now build the isocape using the function `isoscape()`.

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

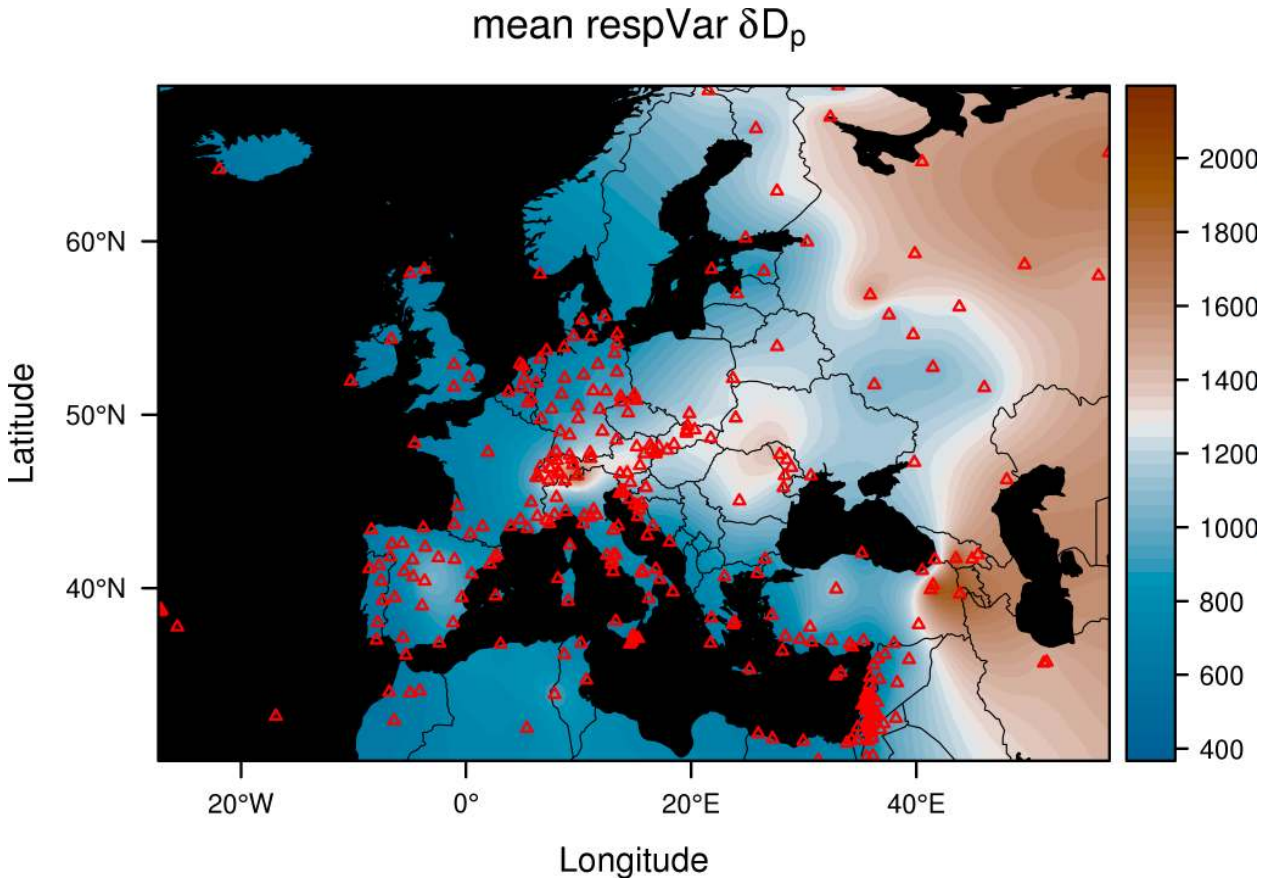
```
## we load the objects that are used to decorate the maps
data(countries)
data(oceanmask)
data(isopalette1)

## we plot the prediction for the mean isotopic values
plot(
    x=isoscape,
    which="mean",
    borders=list(borders=countries),
    mask=list(mask=oceanmask),
    palette=isopalette1
)
```



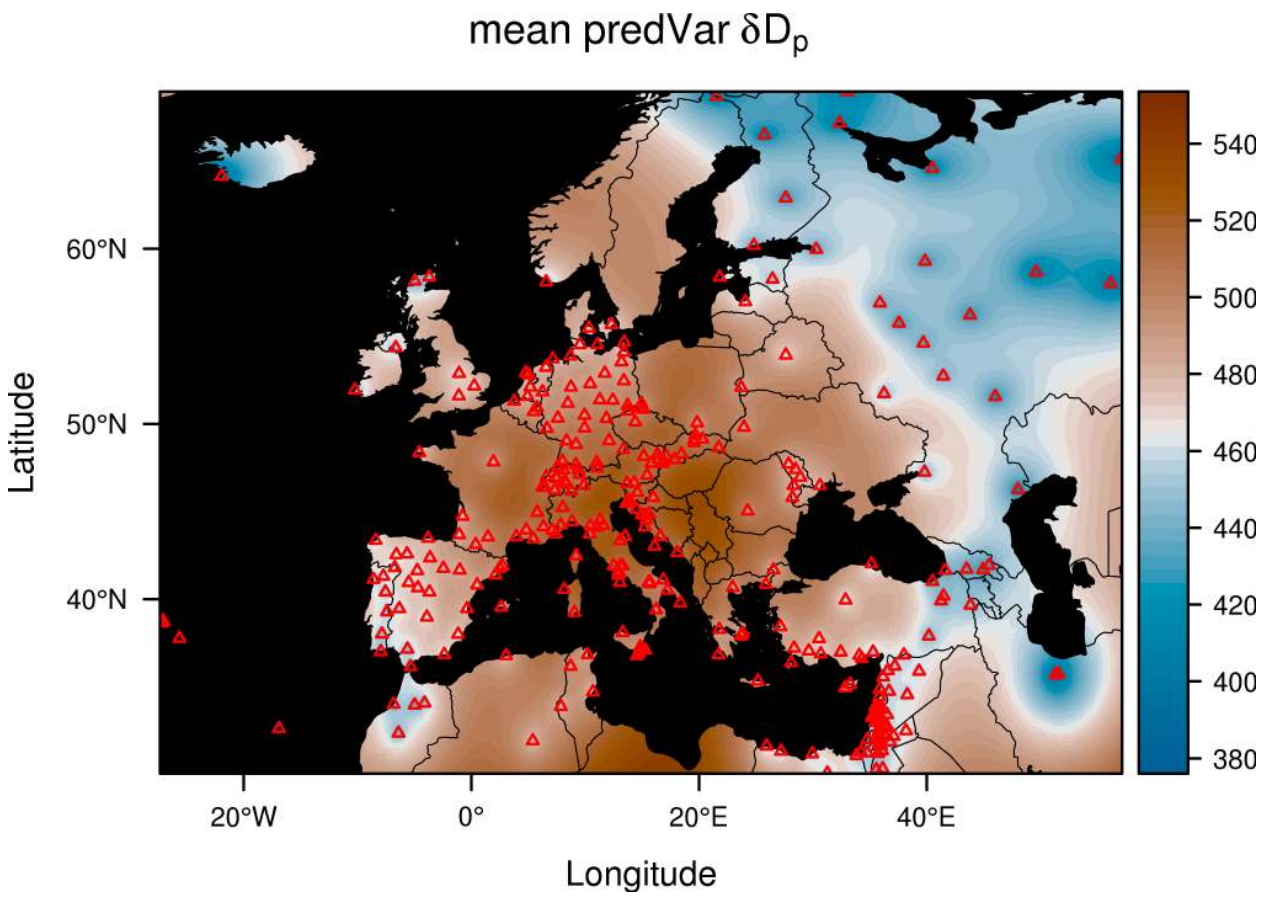
The function `isoscape()` creates 8 different rasters that correspond to different aspects of the isoscape (see `?isoscape()` for details). For example, we will now plot the variation in isotopic values predicted at each location.

```
plot(
    x=isoscape,
    which="mean.respVar",
    borders=list(borders=countries),
    mask=list(mask=oceanmask),
    palette=isopalette1
)
```



We can also plot the prediction variance, which will be useful to discuss the quality of our assignment:

```
plot(
    x=isoscape,
    which="mean.predVar",
    borders=list(borders=countries),
    mask=list(mask=oceanmask),
    palette=isopalette1
)
```



Step 5 - Fit the calibration function

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. In this example, we will use the calibration dataset `calibdata` that is provided with **IsoriX**. This dataset concerns sedentary bats inhabiting Europe (see `?calibdata` for details).

```
## load the calibration dataset
data(calibdata)

## fit the calibration model
calib <- calibfit(
    calib.data=calibdata,
    isofit=Europefit
    )
```

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.

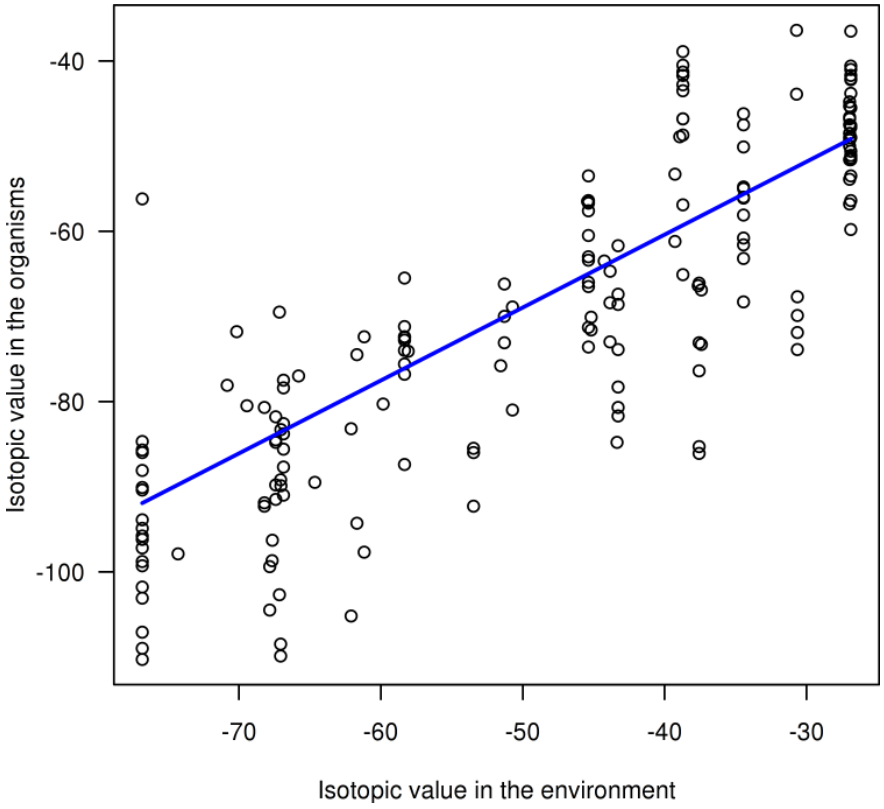
To get the result of the calibration fit, you can simply type the name of the object we just created, or plot it:

```
calib
```

```
##
## Fixed effect estimates of the calibration fit
## tissue.value = intercept + slope * mean.iso +    corrMatrix(1|siteID) + slope^2 * (1|siteID) + Error
##
```

```
##             intercept (+/- SE) = -26.13 +/- 18.61
##             slope     (+/- SE) =   0.86 +/- 0.10
##
## [for more information, use summary()]
```

```
plot(calib)
```



Step 6 - 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":

```
## load the assignment dataset
data(assigndata)

## keep only one bat species
M_bechsteinii <- subset(assigndata, species=="Myotis_bechsteinii")

## first 6 rows of the dataset
head(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
```

```
## number of bats to assign
nrow(M_bechsteinii)
```

```
## [1] 10
```

We now perform the assignment:

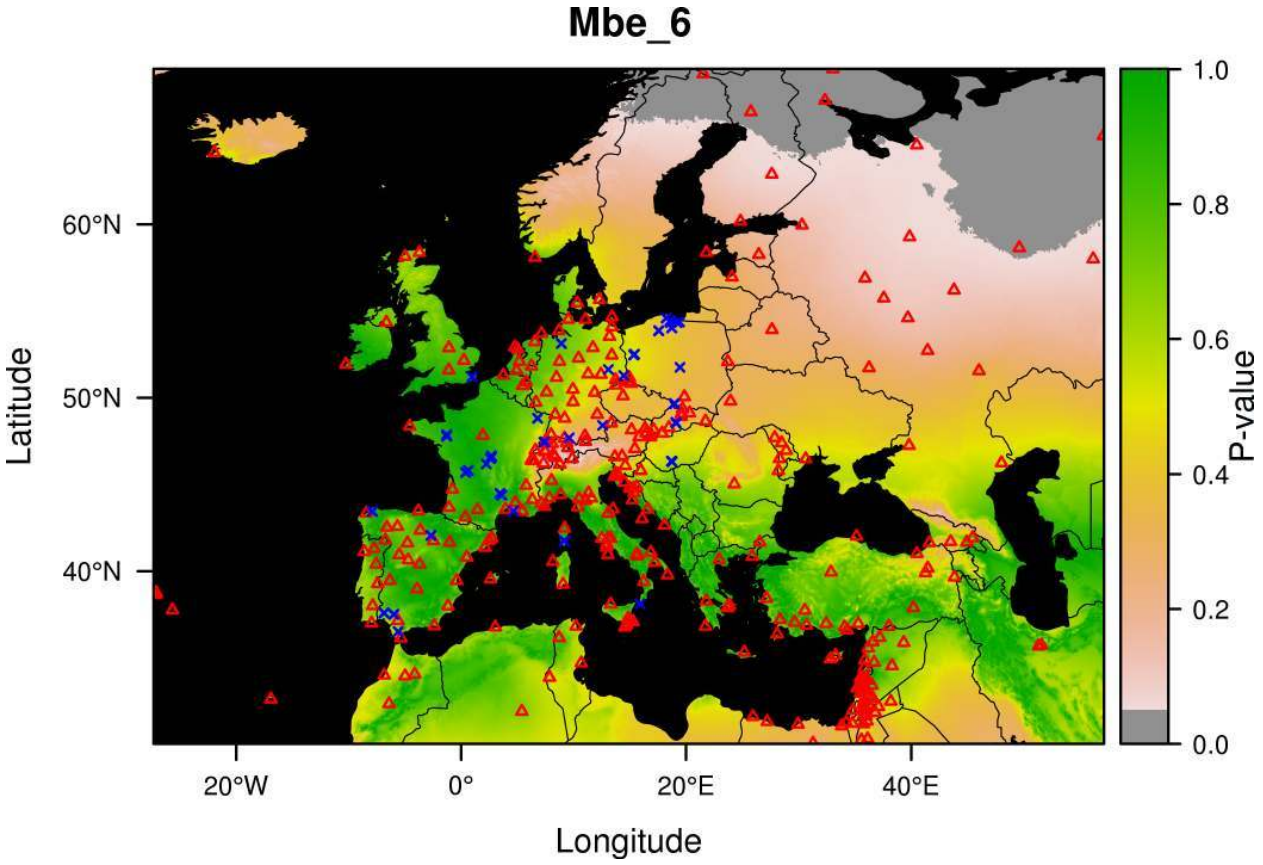
```
assignment <- isofind(
    assign.data=M_bechsteinii,
    isoscape=isoscape,
    calibfit=calib,
    mask=oceanmask
    )
```

Note that we used the argument `mask=oceanmask` so to exclude with certainty that the origin of the bats comes from the water.

Once the assignment done, we 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:

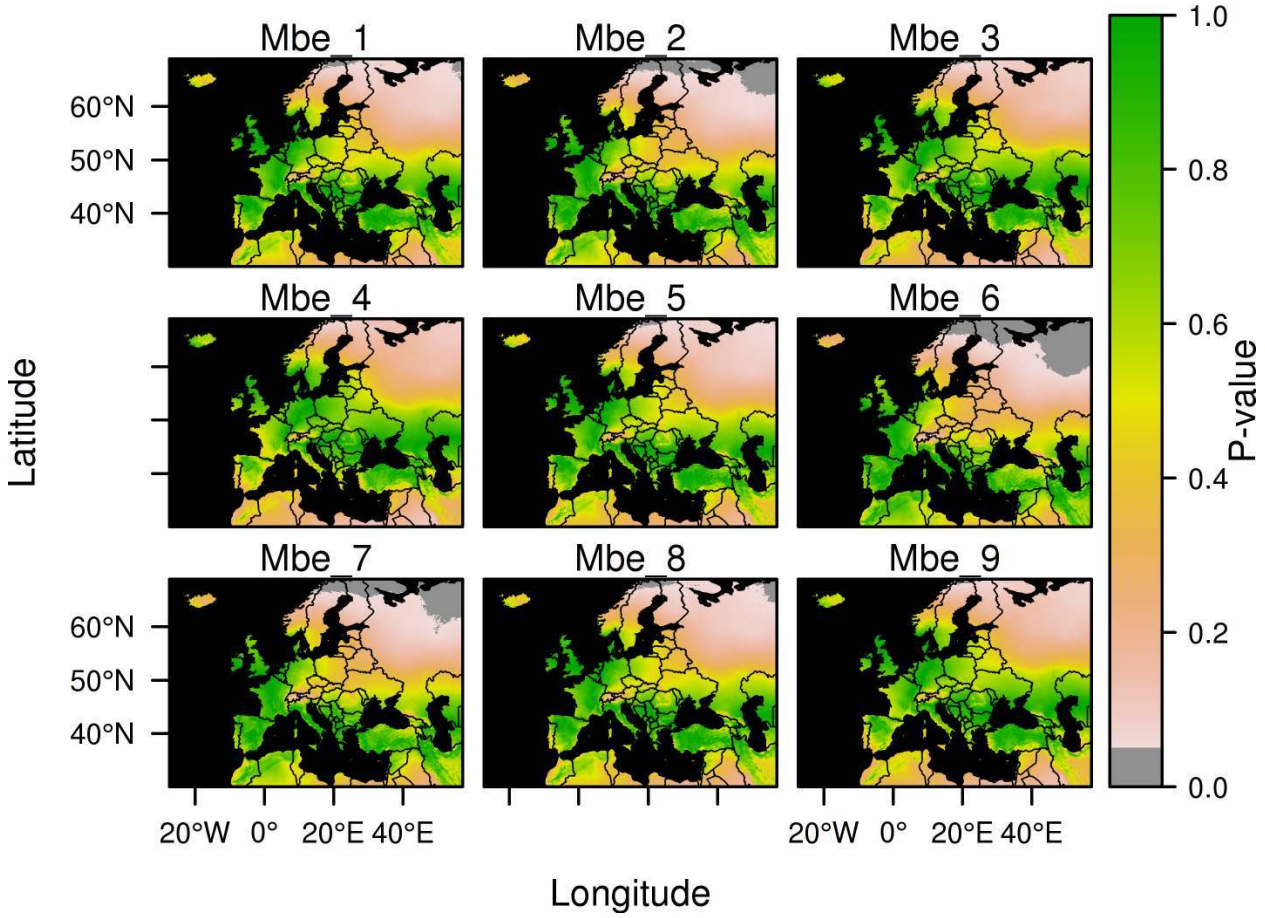
```
## we load new colors for the maps
data(isopalette2)

## we plot
plot(x=assignment,
     who="Mbe_6",
     borders=list(borders=countries),
     mask=list(mask=oceanmask),
     palette=isopalette2)
```



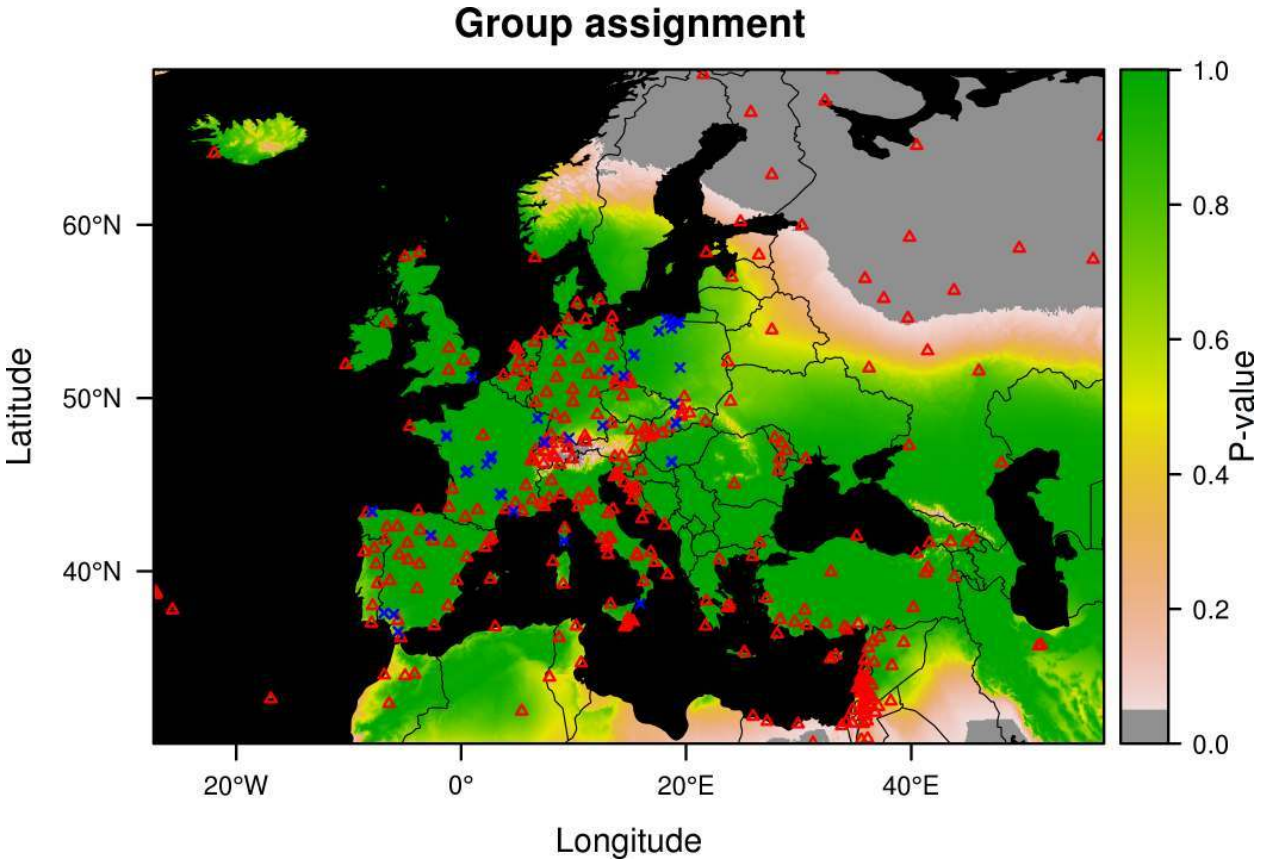
We can also use number for individuals and plot several at once:

```
## plot the first 9 bats
plot(x=assignment,
     who=1:9,
     borders=list(borders=countries),
     mask=list(mask=oceanmask),
     sources=list(draw=FALSE),
     calib=  list(draw=FALSE),
     palette=isopalette2)
```



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",
     borders=list(borders=countries),
     mask=list(mask=oceanmask),
     palette=isopalette2)
```



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
```

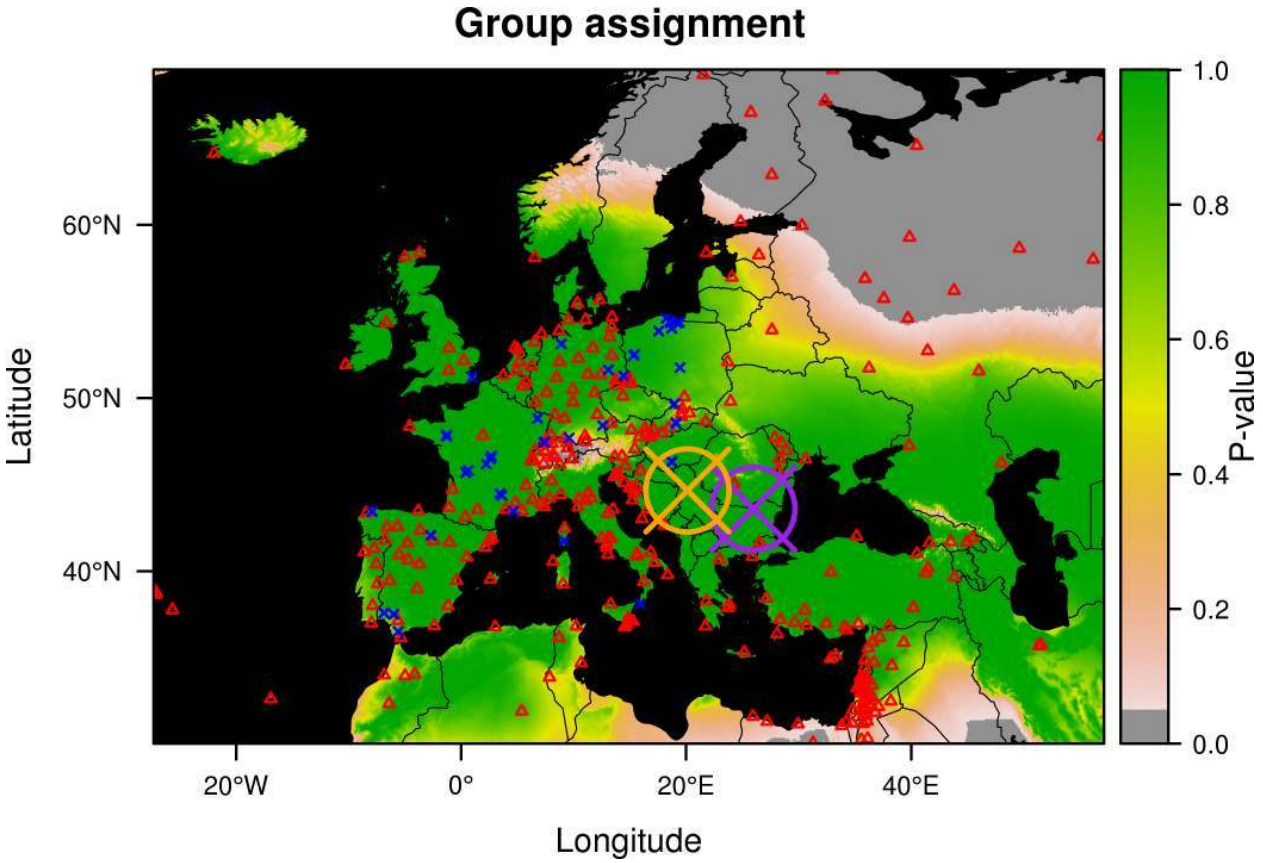
```
##       long      lat
## x 20.11653 44.66653
```

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:

```
## the real origin of these bats (as known by us)
origin <- data.frame(long=25.982122, lat=43.611536)
```

We add these plots to the last plot we created using 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) = -53.49 +/- 22.61"
```

```
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) = -48.94 +/- 23.07"
```

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 making 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 by simply typing:

```
## extract the p-values for all individuals at the location of origin
pvs <- c(extract(assignment$indiv$pv, origin))

## count how many individuals are in
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. In one of the next version of this vignette we will show you how in case you have difficulties.

More?

To export these nice plots on your hard drive, please check another vignette:

```
vignette("Saveplot", package="IsoriX")
```

The number and the quality of the vignettes should grow in the future, so keep updating your packages in R and check what has been changing between version by typing:

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