Package 'IsoriX'

September 23, 2022

Version 0.9.0

Encoding UTF-8

Title Isoscape Computation and Inference of Spatial Origins using Mixed Models

Depends R (>= 3.5.0)

Imports elevatr, graphics, grDevices, grid, lattice, latticeExtra, numDeriv, raster, rasterVis (>= 0.30), sp, spaMM (>= 3.13), stats, tools, utils, viridisLite

Description Building isoscapes using mixed models and inferring the geographic origin of samples based on their isotopic ratios. This package is essentially a simplified interface to several other packages which implements a new statistical framework based on mixed models. It uses 'spaMM' for fitting and predicting isoscapes, and assigning an organism's origin depending on its isotopic ratio. 'IsoriX' also relies heavily on the package 'rasterVis' for plotting the maps produced with 'raster' using 'lattice'.

License GPL (>= 2)

Suggests colorspace, gmp, magick, maps, maptools, rgeos, rgdal, rgl, testthat

LazyData true

URL https://github.com/courtiol/IsoriX/

BugReports https://github.com/courtiol/IsoriX/issues/

RoxygenNote 7.2.1

NeedsCompilation no

Author Alexandre Courtiol [aut, cre] (<https://orcid.org/0000-0003-0637-2959>), François Rousset [aut] (<https://orcid.org/0000-0003-4670-0371>), Marie-Sophie Rohwaeder [aut], Stephanie Kramer-Schadt [aut] (<https://orcid.org/0000-0002-9269-4446>)

Maintainer Alexandre Courtiol <alexandre.courtiol@gmail.com>

Repository CRAN

Date/Publication 2022-09-22 22:10:03 UTC

R topics documented:

`IsoriX-package` *Isoscape Computation and Inference of Spatial Origins using Mixed Models*

Description

IsoriX can be used for building isoscapes using mixed models and inferring the geographic origin of organisms based on their isotopic signature. This package is essentially a simplified interface combining several other packages which implements the statistical framework proposed by Courtiol & Rousset 2017. It uses the package **spaMM** for fitting and predicting isoscapes, and for performing the assignment. **IsoriX** also heavily relies on the package **rasterVis** for plotting the maps produced with the package **raster** using the powerful package **lattice** visualization system.

Details

Below, we describe briefly the main steps of the workflow that aims at performing the construction of an isoscape and the assignment of organisms of unknown geographic origin(s) based on their isotopic signature. We advise you to also read the detailed book chapter we wrote (in press), as well as our online documentation, which essentially cover the same material in a more detailed manner. You should also read the dedicated help pages of the functions you are using.

The statistical methods will not be detailed in this document but information on the computation of isoscapes is available in Courtiol & Rousset 2017, and information on the calibration and assignment in the appendix of Courtiol et al. 2019.

1. Fitting the isoscape model with isofit:
   The function isofit fits a geostatistical model, which approximates the relationship between the topographic features of a location and its isotopic signature (see isofit for details). The model fits observations of isotopic delta values at several geographic locations (hereafter, called *sources*). One common type of sources used in ecology is the delta values for hydrogen in precipitation water collected at weather stations, but one may also use measurements performed on sedentary organisms. In either case, the accuracy of the isoscape (and thereby the accuracy of assignments) increases with the number and spatial coverage of the sources. The function isofit is designed to fit the model on data aggregated per location across all measurements. If instead you want to fit the model on measurements split per time intervals (e.g. per month), within each location, you should use the alternative function isomultifit. Either way the data must be prepared using the function prepsources.
2. Preparing the structural raster with prepraster:
   Building isoscapes and assigning organisms to their origin requires an adequate structural raster, i.e. a matrix representing a spatial grid. The function prepraster allows restricting the extent of the raster to the area covered by isoscape data (in order to avoid extrapolation) and to reduce the resolution of the original structural raster (in order to speed up computation in all following steps). Note that aggregating the raster may lead to different results for the assignment, if the structural raster is used to define a covariate. This is because the values of raster cells changes depending on the aggregation function, which in turn will affect model predictions.
   We provide the function getelev to download an elevation raster for the entire world at a resolution of one altitude per square-km, and other rasters may be used. Such an elevation raster can be used as a structural raster. We have also stored a low resolution raster for Germany in our package (see ElevRasterDE) for users to try things out, but we do not encourage its use for real application.
3. Predicting the isoscape across the area covered by the elevation raster with isoscape:
   The function isoscape generates the isoscapes: it uses the fitted geostatistical models to predict the isotopic values (and several variances associated to those) for each raster cell defined by the structural raster. If the model has been fitted with isomultifit, you should use the alternative function isomultiscape to generate the isoscape.
   Our package allows the production of fine-tuned isoscape figures (using the function plot.ISOSCAPE). Alternatively, the isoscape rasters can be exported as ascii raster and edited in any Geographic Information System (GIS) software (see isoscape and the online documentation for details).
4. Fitting the calibration model with calibfit:
   In most cases, organisms are of another kind than the sources used to build the isoscape (i.e. the isoscape is built on precipitation isotopic values and organisms are not water drops, but e.g.

the fur of some bats). In such a case, the hydrogen delta values of the sampled organisms were modulated by their distinct physiology and do not directly correspond to the isotopic signature of the sources. In this situation, one must use sedentary organisms to study the relationship between the isotopic values in organisms and that of their environment. The function calibfit fits a statistical model on such a calibration dataset.

If the isoscape is directly built from isotopic values of organisms, there is no need to fit a calibration model.

5. Inferring spatial origins of samples with isofind:

   The function isofind tests for each location across the isoscape if it presents a similar isotopic signature than the unknown origin of a given individual(s). This assignment procedure considered the some (but not all, see Courtiol et al. 2019) uncertainty stemming from the model fits (geostatistical models and calibration model). The function plot.ISOFIND then draws such assignment by plotting the most likely origin with the prediction region around it. When several organisms are being assigned, both assignments at the level of each sample and a single assignment for the whole group can be performed.

Note

Please note that the geographic coordinates (latitude, longitude) of any spatial data (locations, rasters) must be given in decimal degrees following the WGS84 spheroid standard.

Author(s)

Alexandre Courtiol <alexandre.courtiol@gmail.com>,

François Rousset,

Marie-Sophie Rohwaeder,

Stephanie Kramer-Schadt <kramer@izw-berlin.de>

References

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). "Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX." In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

Courtiol A, Rousset F (2017). "Modelling isoscapes using mixed models." bioRxiv. doi: 10.1101/207662, link.

AssignDataAlien *Simulated assignment dataset*

Description

This dataset contains simulated hydrogen delta values. The data can be used as an example to perform assignments using the function [isofind].

Format

A *dataframe* with 10 observations on 2 variables:

[, 1]	sample_ID	(*factor*)	Identification of the sample
[, 2]	sample_value	(*numeric*)	Hydrogen delta value of the tissue

See Also

[isofind] to perform assignments

Examples

```
head(AssignDataAlien)
str(AssignDataAlien)

## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## The following describes how we created such dataset

### We prepare the precipitation data
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

### We fit the models for Germany
GermanFit <- isofit(data = GNIPDataDEagg)

### We build the isoscape
GermanScape <- isoscape(raster = ElevRasterDE, isofit = GermanFit)

### We create a simulated dataset with 1 site and 10 observations
set.seed(1L)
Aliens <- create_aliens(calib_fn = list(intercept = 3, slope = 0.5, resid_var = 5),
                        isoscape = GermanScape,
                        raster = ElevRasterDE,
                        coordinates = data.frame(site_ID = "Berlin",
                                                 long   = 13.52134,
                                                 lat    = 52.50598),
                        n_sites = 1,
                        min_n_samples = 10,
                        max_n_samples = 10)
AssignDataAlien <- Aliens[, c("sample_ID", "sample_value")]

### Uncomment the following to store the file as we did
#save(AssignDataAlien, file = "AssignDataAlien.rda", compress = "xz")

}
```

AssignDataBat *Assignment dataset for bat species*

Description

This dataset contains data from Voigt & Lenhert (2019). It contains hydrogen delta values of fur keratin from common noctule bats (*Nyctalus noctula*) killed at wind turbines in northern Germany. The data can be used as an example to perform assignments using the function [isofind].

Format

A *dataframe* with 14 observations on 4 variables:

[, 1]	sample_ID	(*factor*)	Identification of the animal
[, 2]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 2]	lomg	(*numeric*)	Longitude coordinate (decimal degrees)
[, 4]	sample_value	(*numeric*)	Hydrogen delta value of the tissue

Source

data directly provided by the authors of the following publication

References

Voigt CC & Lehnert L (2019). Tracking of movements of terrestrial mammals using stable isotopes. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

See Also

[isofind] to perform assignments

Examples

```
head(AssignDataBat)
str(AssignDataBat)
```

AssignDataBat2 *Assignment dataset for bat species*

Description

This dataset contains data from Voigt, Lehmann and Greif (2015). It contains hydrogen delta values of fur keratin from bats captured in 2008, 2009 and 2013 from their roosting sites in Bulgaria. We only retained the bats of the genus Myotis from the original study. The data can be used as an example to perform assignments using the function [isofind].

Format

A *dataframe* with 244 observations on 3 variables:

[, 1]	sample_ID	(*factor*)	Identification of the animal
[, 2]	species	(*factor*)	Animal species name
[, 3]	sample_value	(*numeric*)	Hydrogen delta value of the tissue

Source

data directly provided by the authors of the following publication

References

Voigt, C.C., Lehmann, D., Greif, S. (2015). Stable isotope ratios of hydrogen separate mammals of aquatic and terrestrial food webs. Methods in Ecology and Evolution 6(11).

See Also

[isofind] to perform assignments

Examples

```
head(AssignDataBat2)
str(AssignDataBat2)
```

`CalibDataAlien` *Simulated calibration dataset*

Description

This dataset contains simulated hydrogen delta values for corresponding locations based on an assumed linear relationship between the animal tissue value and the hydrogen delta values in the environment. The data can be used as an example to fit a calibration model using the function [calibfit].

Format

A *dataframe* with x observations on 6 variables:

[, 1]	site_ID	(*factor*)	Identification of the sampling site
[, 2]	long	(*numeric*)	Longitude coordinate (decimal degrees)
[, 3]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 4]	elev	(*numeric*)	Elevation asl (m)
[, 5]	sample_ID	(*factor*)	Identification of the sampled animal
[, 6]	tissue.value	(*numeric*)	Hydrogen delta value of the tissue

Details

Users who wish to use their own dataset for calibration should create a *dataframe* of similar structure than this one. The columns should possess the same names as the ones described above. If the elevation is unknown at the sampling sites, elevation information can be extracted from a high resolution elevation raster using the function [raster::extract]. In this dataset, we retrieved elevations from the Global Multi-resolution Terrain Elevation Data 2010.

See Also

[calibfit] to fit a calibration model

Examples

```
head(CalibDataAlien)
str(CalibDataAlien)

## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## We prepare the precipitation data
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

## We fit the models for Germany
GermanFit <- isofit(data = GNIPDataDEagg)

## We build the isoscape
GermanScape <- isoscape(raster = ElevRasterDE, isofit = GermanFit)

## We create a simulated dataset with 50 site and 10 observations per site
set.seed(2L)
CalibDataAlien <- create_aliens(calib_fn = list(intercept = 3, slope = 0.5, resid_var = 5),
                                isoscape = GermanScape,
                                raster = ElevRasterDE,
                                n_sites = 50,
                                min_n_samples = 10,
                                max_n_samples = 10)
plot(sample_value ~ source_value, data = CalibDataAlien)
```

```
abline(3, 0.5)

CalibDataAlien$source_value <- NULL

## Uncomment the following to store the file as we did
#save(CalibDataAlien, file = "CalibDataAlien.rda", compress = "xz")

}
```

`CalibDataBat` *Calibration dataset for bat species*

Description

This dataset contains hydrogen delta values of fur keratin from 6 sedentary bat species. It corresponds to the combination of several studies as detailed in Voigt & Lenhert 2019. This is the dataset used in Courtiol et al. 2019. The data can be used as an example to fit a calibration model using the function [calibfit].

Format

A *dataframe* with 335 observations on 7 variables:

[, 1]	site_ID	(*factor*)	Identification of the sampling site
[, 2]	long	(*numeric*)	Longitude coordinate (decimal degrees)
[, 3]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 4]	elev	(*numeric*)	Elevation asl (m)
[, 5]	sample_ID	(*factor*)	Identification of the sampled animal
[, 6]	species	(*factor*)	A code for the species
[, 7]	sample_value	(*numeric*)	Hydrogen delta value of the tissue

Details

Users who wish to use their own dataset for calibration should create a *dataframe* of similar structure than this one (only the column 'species' can be dropped). The columns should possess the same names as the ones described above. If the elevation is unknown at the sampling sites, elevation information can be extracted from a high resolution elevation raster using the function [raster::extract] (see **Examples** in [CalibDataBat2]).

References

Voigt CC & Lehnert L (2019). Tracking of movements of terrestrial mammals using stable isotopes. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

See Also

[CalibDataBat2] for another (related) calibration dataset

[calibfit] to fit a calibration model

Examples

```
head(CalibDataBat)
str(CalibDataBat)
```

`CalibDataBat2` *Calibration dataset for bat species*

Description

This dataset contains hydrogen delta values of fur keratin from sedentary bat species captured between 2005 and 2009 from Popa-Lisseanu et al. (2012). The data can be used as an example to fit a calibration model using the function [calibfit].

Format

A *dataframe* with 178 observations on 6 variables:

[, 1]	site_ID	(*factor*)	Identification of the sampling site
[, 2]	long	(*numeric*)	Longitude coordinate (decimal degrees)
[, 3]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 4]	elev	(*numeric*)	Elevation asl (m)
[, 5]	sample_ID	(*factor*)	Identification of the sampled animal
[, 6]	sample_value	(*numeric*)	Hydrogen delta value of the tissue

Details

Users who wish to use their own dataset for calibration should create a *dataframe* of similar structure than this one (only the column 'species' can be dropped). The columns should possess the same names as the ones described above. If the elevation is unknown at the sampling sites, elevation information can be extracted from a high resolution elevation raster using the function [raster::extract] (see **Examples**). Note that the original study used a different source of elevation data.

Source

data directly provided by the authors of the following publication

References

Popa-Lisseanu, A. G., Soergel, K., Luckner, A., Wassenaar, L. I., Ibanez, C., Kramer-Schadt, S., Ciechanowski, M., Goerfoel, T., Niermann, I., Beuneux, G., Myslajek, R. W., Juste, J., Fonderflick, J., Kelm, D., Voigt, C. C. (2012). A triple isotope approach to predict the breeding origins of European bats. PLoS ONE 7(1):e30388.

See Also

[CalibDataBat] for another (related) calibration dataset

[calibfit] to fit a calibration model

Examples

```
head(CalibDataBat2)
str(CalibDataBat2)

## The following example require to have downloaded
## an elevation raster with the function getelev()
## and will therefore not run unless you uncomment it

#if (require(raster)){
#    ## We delete the elevation data
#    CalibDataBat2$elev <- NULL
#
#    ## We reconstruct the elevation data using an elevation raster
#    getelev(file = "elevBats.tif", z = 6,
#            lat_min = min(CalibDataBat2$lat),
#            lat_max = max(CalibDataBat2$lat),
#            long_min = min(CalibDataBat2$long),
#            long_max = max(CalibDataBat2$long))
#    ElevationRasterBig <- raster("elevBats.tif")
#    CalibDataBat2$elev <- extract(
#        ElevationRasterBig,
#        cbind(CalibDataBat2$long, CalibDataBat2$lat))
#    head(CalibDataBat2)
#}
```

---

calibfit — *Fit the calibration model (or load parameters from calibration done outside IsoriX)*

Description

This function establishes the relationship between the isotopic values of organisms (e.g. tissues such as hair, horn, ivory or feathers; referred in code as *sample_value*) and the isotopic values of their environment (e.g. precipitation water; referred in code as *source_value*). This function is only

needed when the assignment of organisms has to be performed within an isoscape that was not built using the organisms themselves, but that was instead built using another source of isotopic values (e.g., precipitation). If the isoscape had been fitted using isotopic ratios from sedentary animals directly, this calibration step is not needed (e.g. isoscape fitted using sedentary butterflies and migratory butterflies to assign). In other cases, this calibration step is usually needed since organisms may not directly reflect the isotopic values of their environment. Depending on the calibration data to be used (provided via the argument `data`), one of four possible calibration methods must be selected (via the argument `method`). Each method considers a different statistical model and requires particular data that are organised in a specific way (see **Details** for explanations and **Examples** for use cases).

Usage

```
calibfit(
  data,
  isofit = NULL,
  method = c("wild", "lab", "desk", "desk_inverse"),
  verbose = interactive(),
  control_optim = list()
)
```

Arguments

`data`	A *dataframe* containing the calibration data (see note below)
`isofit`	The fitted isoscape created by isofit
`method`	A *string* indicating the method used to generate the data used for the calibration. By default method is `"wild"`, but the other `"lab"`, `"desk"` and `"desk_inverse"`. See **Details** for the difference between these three methods.
`verbose`	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if you run an interactive R session and `FALSE` otherwise.
`control_optim`	A *list* to pass information to the argument control in the call to optim (only effective when `method = "wild"`; for advanced users only).

Details

The `method` argument can take one of the four values "wild" (default), "lab", "desk" and "desk_inverse" corresponding to the four calibration methods. It is crucial for you to select the method that is most appropriate for your workflow, as the choice of method can impact the most likely assignment locations during the assignment test performed in isofind.

Method "wild":

This calibration method is the one to be used when the calibration data to be used correspond to isotopic values measured on sedentary organisms and when no direct measurement of isotopic values in the environment are available at the locations where sedentary organisms have been collected. In such a case, the isotopic values in the environment of sedentary organisms are predicted internally using an isoscape fitted with isofit. This calibration method thus aims at

estimating and accounting for the uncertainty associated with these predicted values. Such uncertainty is accounted for when fitting the calibration fit so as to produce an unbiased estimation of the calibration relationship and it is also then accounted for by isofind when inferring the possible locations of origin. Before we added the argument `method` in calibfit (i.e. before releasing the version 0.8.3), this method was the only one available in IsoriX.

- **Statistical model**: in this case, the calibration model to be fitted is a linear mixed-effects model (LMM) that fits the isotopic values of sedentary organisms as a linear function of the isotopic values in their environment (e.g. precipitation). The function considers that the isotopic values from the environment (e.g. from precipitation) at the locations at which organisms were sampled are not known. The function therefore predicts these isotopic values from the geostatistical model fitted by the function isofit, which is provided to calibfit using the argument `isofit`. The LMM used to fit the calibration function has a simple fixed-effect structure: an intercept and a slope. The random effect is more complex: it is normally distributed with mean zero, a certain variance between locations proportional to the squared (fixed) slope, and a covariance structure defined by the prediction covariance matrix of the isoscape model between the calibration locations. See appendix in Courtiol et al. 2019 for more details.
- **Required calibration data**: the calibration data to be used here must be a dataframe (or a tibble) containing at least the following columns:
  - `sample_value`: the isotopic value of the calibration sample
  - `long`: the longitude coordinate (decimal degrees)
  - `lat`: the latitude coordinate (decimal degrees)
  - `site_ID`: the sample site

  The column name must be identical to those indicated here. Other columns can be present in the data but won't be used. Each row must correspond to a different calibration sample (i.e. a single isotopic measurement). See CalibDataAlien, CalibDataBat, or CalibDataBat2 for examples of such a dataset.

Method "lab":

This calibration method is the one to be used when the calibration data to be used correspond to isotopic values recorded for both organisms and their environment. We can foresee three main situations in which the "lab" method is the one to be used:

1. the data are generated by growing organisms in a controlled environment where they are fed and/or given water with a specific (known) isotopic value.
2. sedentary organisms are sampled in the wild together with a sample from their environment and that isotopic values have been measured for both.
3. you want to use a calibration made by others based on a plot of that calibration showing the datapoints. In such a case, you should use an R package (e.g. metaDigitse or digitize) or software (e.g. graphClick or dataThief) to extract the coordinates on the plots so as to obtain the isotopic values of the sample and the environment behind each point.

Note that the use cases 1 and 2 will allow for the propagation of all relevant sources of uncertainty during the assignment. In contrast, the third use case implies to neglect uncertainty in the isotopic values in the environment if those were initially predicted using an isoscape. It also neglects the covariances involving such predicted values. That being said, if you want to use someone else calibration relationship, using this method is generally preferable to using the method "desk" described below (less error prone and de facto accounting for all five parameters mentioned for the method "desk").

- **Statistical model**: in this case, the calibration model to be fitted is a simple linear model (LM) or a simple linear mixed-effects model (LMM) that fits the isotopic values of sedentary organisms as a linear function of the isotopic values in their environment (e.g. precipitation). Whether it is a LM or a LMM depends on the presence of a column `site_ID` in the dataset as well as on the number of unique values for such a column. If the column is present and the number of unique values is larger than 4, a LMM is fitted. Otherwise, a LM is fitted. In both cases, the function considers that the isotopic values from the environment (e.g. from precipitation) at the locations at which organisms were sampled are known. Contrary to the method "wild", the environment values are thus considered as observed and not predicted from an isoscape. The argument `isofit` should thus remain `NULL` in this case (since no isoscape is used, no isoscape fit is required to perform the calibration). The model used to fit the calibration function has a simple fixed effect structure: an intercept and a slope.
- **Required calibration data**: the calibration data to be used here must be a dataframe (or a tibble) containing at least the following columns:
  - `sample_value`: the isotopic value of the calibration sample
  - `source_value`: the isotopic value of the environment
  - `site_ID` (optional): the sample site

  The column name must be identical to those indicated here. Other columns can be present in the data but won't be used. Each row must correspond to a different calibration sample (i.e. a single sample-environment pair of isotopic measurements).

Methods "desk" and "desk_inverse":

These calibration methods must only be used as a last resource! They are unlikely to yield robust inference during the assignment step. These calibration methods are the ones to be used when no calibration data is directly available, when you cannot either extract the data from a plot, and thus when you must rely solely on published metrics (including intercept and slope) to represent a calibration relationship. They work by making crude assumptions that various uncertainty components are null.

The method "desk" is the one to be used when the published calibration relationship is of the form `lm(sample_value ~ source_value)` and the method "desk_inverse" is the one to be used when the published calibration relationship is of the form `lm(source_value ~ sample_value)`. Do make sure you are using the correct alternative. Note that the model used for the published calibration must be a linear regression (LM) and not a reduced major axis regression (RMA). If you use parameter values stemming from a RMA, the assignment will most likely be biased.

Both methods require five metrics to work at their best: the intercept and slope of a calibration relationship, the standard errors (SE) associated to them, and the residual variance (not SD). For statistical reasons, the method "desk" is more flexible than the method "desk_inverse" and can still work (in the sense of running, but the reliability of the assignments will get worse) if the SEs and/or the residual variance is not provided. For the method "desk_inverse" all metrics are unfortunately necessary.

Don't expect miracles: even if the "desk" method is used together with its five parameters, the assignment will still suffer from the same limitations as those impacting the method "lab" usage number 3. If less than five parameters are provided, further assumptions are made and this comes with a cost: again, it can bias the assignment and bias the confidence region. For these reasons, we were tempted to use `method = "dirty"` instead of `method = "desk"`... but we chickened out since we predicted that users would then refrain from mentioning the method they used in publications...

Note that if the provided slope is set to 0 and an intercept is considered, the calibration methods actually corresponds to the simple consideration of a fractionation factor.

- **Statistical model**: none!
- **Required calibration data** for method "desk": the calibration data to be used here must be a dataframe (or a tibble) containing a single row with the following columns:
  - `intercept`: the estimated slope of a LM calibration fit
  - `slope`: the estimated slope of a LM calibration fit
  - `intercept_se` (optional): the standard error around the intercept
  - `slope_se` (optional): the standard error around the slope
  - `resid_var` (optional): the residual variance (not SD) of a LM calibration fit
- **Required calibration data** for method "desk_inverse": the calibration data to be used here must be a dataframe (or a tibble) containing a single row with the following columns:
  - `intercept`: the estimated slope of a LM calibration fit
  - `slope`: the estimated slope of a LM calibration fit
  - `intercept_se`: the standard error around the intercept
  - `slope_se`: the standard error around the slope
  - `resid_var`: the residual variance (not SD) of a LM calibration fit
  - `sign_mean_Y`: a *numeric* indicating the sign of the mean value of the isotopes in the environment in the format returned by sign; that is either `1` (if positive) or `-1` (if negative). This is required for pivoting the regression from "desk_inverse" to "desk".
  - `N`: a *numeric* indicating the sample size of the data used for the calibration fit. This is required for pivoting the regression from "desk_inverse" to "desk".

Value

This function returns a *list* of class *CALIBFIT* containing the name of the calibration method used, whether a species_ID random effect was estimated, whether a site_ID random effect was estimated, the fixed-effect estimates of the calibration function, the covariance of the fixed effects, the residual variance of the calibration fit, the fitted calibration model (if applicable), the fitted isoscape model (if applicable), the original calibration data set with additional information added during the fit, and the location of the calibration points as spatial points.

References

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

See Also

see plot for the help on how to plot the calibration relationship.

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)
```

```
if (getOption_IsoriX("example_maxtime") > 30) {

#####################################################
## 1 Example of calibration using the method "wild" #
#####################################################

## 1.1 We prepare the data to fit the isoscape:
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

## 1.2 We fit the isoscape models for Germany:
GermanFit <- isofit(data = GNIPDataDEagg,
                    mean_model_fix = list(elev = TRUE, lat_abs = TRUE))

## 1.3 We fit the calibration model using the method "wild" (the default):
CalibAlien <- calibfit(data = CalibDataAlien, isofit = GermanFit)

## 1.4 We explore the outcome of the calibration:
CalibAlien
summary(CalibAlien)
plot(CalibAlien)

## Note 1: you can plot several calibrations at once (using bats this time):
CalibBat1 <- calibfit(data = CalibDataBat, isofit = GermanFit)
CalibBat2 <- calibfit(data = CalibDataBat2, isofit = GermanFit)
plot(CalibBat1)
points(CalibBat2, pch = 3, col = "red", CI = list(col = "green"))

## Note 2: you can extract data created by plot() for plotting things yourself:
dataplot <- plot(CalibAlien, plot = FALSE)
plot(sample_fitted ~ source_value, data = dataplot,
     xlim = range(dataplot$source_value),
     ylim = range(dataplot$sample_lwr, dataplot$sample_upr), col = NULL)
polygon(x = c(dataplot$source_value, rev(dataplot$source_value)),
        y = c(dataplot$sample_lwr, rev(dataplot$sample_upr)),
        col = 3)
points(sample_fitted ~ source_value, dataplot, type = "l", lty = 2)


#####################################################
## 2 Example of calibration using the method "lab" #
#####################################################

## 2.0 We create made up data here because we don't have yet a good dataset
## for this case, but you should use your own data instead:
GermanScape <- isoscape(raster = ElevRasterDE, isofit = GermanFit)
set.seed(123)
CalibDataAlien2 <- create_aliens(calib_fn = list(intercept = 3, slope = 0.5,
                                                 resid_var = 5),
                                 isoscape = GermanScape,
                                 raster = ElevRasterDE,
                                 n_sites = 25,
                                 min_n_samples = 5,
```

```
                                  max_n_samples = 5)
CalibDataAlien2 <- CalibDataAlien2[, c("site_ID", "sample_ID", "source_value",
                                       "sample_value")]
head(CalibDataAlien2) ## your data should have this structure

## 2.1 We fit the calibration model using the method "lab":
CalibAlien2 <- calibfit(data = CalibDataAlien2, method = "lab")

## 2.2 We explore the outcome of the calibration:
CalibAlien2
summary(CalibAlien2)
plot(CalibAlien2)


#####################################################
## 3 Example of calibration using the method "desk" #
#####################################################

## 3.1 We format the information about the calibration function to be used
## as a dataframe:
CalibDataAlien3 <- data.frame(intercept = 1.67, slope = 0.48,
                              intercept_se = 1.65, slope_se = 0.03,
                              resid_var = 3.96)
CalibDataAlien3

## 3.2 We fit the calibration model using the method "desk":
CalibAlien3 <- calibfit(data = CalibDataAlien3, method = "desk")

## 3.3 We explore the outcome of the calibration:
CalibAlien3
summary(CalibAlien3)
plot(CalibAlien3, xlim = c(-100, 100), ylim = c(-50, 50))

## Note: the desk function also work with just intercept and slope:
CalibDataAlien4 <- CalibDataAlien3[, c("intercept", "slope")]
CalibAlien4 <- calibfit(data = CalibDataAlien4, method = "desk")
CalibAlien4
summary(CalibAlien4)
plot(CalibAlien3, xlim = c(-100, 100), ylim = c(-50, 50))
points(CalibAlien4, line = list(col = "orange"))
## Regression lines are the same, but the new calibration does not have a
## confidence intervals since we provided no uncertainty measure in
## CalibDataAlien4, which will make a difference during assignments...


############################################################
## 4 Example of calibration using the method "desk_inverse" #
############################################################

## 4.1 We format the information about the calibration function to be used
## as a dataframe:
CalibDataAlien4 <- data.frame(intercept = -16.98822, slope = 1.588885,
                              intercept_se = 2.200435, slope_se = 0.08106032,
```

```
                                resid_var = 13.15102, N = 125, sign_mean_Y = -1)
CalibDataAlien4

## 4.2 We fit the calibration model using the method "desk_inverse":
CalibAlien4 <- calibfit(data = CalibDataAlien4, method = "desk_inverse")

## 4.3 We explore the outcome of the calibration:
CalibAlien4
summary(CalibAlien4)
plot(CalibAlien4, xlim = c(-100, 100), ylim = c(-50, 50))
}
```

`CountryBorders` *Borders of world CountryBorders*

Description

This dataset contains a polygon shapefile that can be used to plot the borders of CountryBorders.

Format

A *SpatialPolygons*

Source

This *SpatialPolygons* is derived from the maps::world of the package **maps**. Please refer to this other package for description and sources of this dataset. See example for details on how we created the dataset.

See Also

OceanMask for another polygon shapefile used to embellish the plots

Examples

```
if(require(sp))
  plot(CountryBorders, border="red", col="darkgrey")

## How did we create this file?

## Uncomment the following to create the file as we did
#if(require(maps) & require(maptools) & require(rgeos)){
#    worldmap <- map("world", fill = TRUE, plot = FALSE)
#    CountryBorders <- map2SpatialPolygons(worldmap, IDs = worldmap$names)
#    CountryBorders <- gBuffer(CountryBorders, byid = TRUE, width = 0)
#    proj4string(CountryBorders) <- CRS(as.character(NA))
#    CountryBorders
#    save(CountryBorders, file = "CountryBorders.rda", compress = "xz")
```

```
#}
```

`create_aliens` *Simulate datasets for calibrations or assignments*

Description

This function allows to simulate data so to provide examples for the calibration and for the assignment procedure. We name the simulated individuals 'Aliens' so to make it clear that the data we use to illustrate our package are not real data.

Usage

```
create_aliens(
  calib_fn = list(intercept = 3, slope = 0.5, resid_var = 5),
  isoscape = NULL,
  coordinates = NA,
  raster = NULL,
  n_sites = NA,
  min_n_samples = 1,
  max_n_samples = 10
)
```

Arguments

`calib_fn`	A *list* containing the parameter values describing the relationship between the isotope values in the environment and those in the simulated organisms. This list must contain three parameters: the intercept, the slope, and the residual variance.
`isoscape`	The output of the function isoscape
`coordinates`	An optional *data.frame* with columns `site_ID`, `long` and `lat`
`raster`	A *RasterLayer* containing an elevation raster
`n_sites`	The number of sites from which the simulated organisms originate (*integer*)
`min_n_samples`	The minimal number of observations (*integer*) per site
`max_n_samples`	The maximal number of observations (*integer*) per site

Details

The isostopic values for the organisms are assumed to be linearly related to the one from the environment. The linear function can be parametrized using the first argument of the function (`calib_fn`). With this function the user can simulate data for different sites.

The number and locations of sites can be controlled in two ways. A first possibility is to use the argument `n_sites`. The sites will then be selected randomly among the locations present in the isoscape (argument `isoscape`) provided to this function. An alternative possibility is to provide

a data frame containing three columns (`site_ID`, `long` and `lat`) to input the coordinate of the sampling site manually.

Irrespectively of how locations are chosen, a random number of observations will be drawn, at each site, according to a uniform distribution bounded by the values of the argument `min_n_samples` and `max_n_samples`.

From the selected coordinates, the isotope values for the environment are directly extracted from the corresponding point predictions stored in the isoscape object. No uncertainty is considered during this process. Then the linear calibration defines the means of the isotope values for the simulated organisms. The actual values is then drawn from a Gaussian distribution centred around such mean and a variance defined by the residual variance (`resid_var`) input within the list `calib_fn`.

Value

This functions returns a *data.frame* (see example for column names)

See Also

calibfit for a calibration based on simulated data

isofind for an assignment based on simulated data

IsoriX for the complete work-flow of our package

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## We fit the models for Germany
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

GermanFit <- isofit(data = GNIPDataDEagg)

## We build the isoscapes
GermanScape <- isoscape(raster = ElevRasterDE, isofit = GermanFit)

## We create a simulated dataset with 25 sites and 5 observations per site
Aliens <- create_aliens(calib_fn = list(intercept = 3, slope = 0.5, resid_var = 5),
                        isoscape = GermanScape,
                        raster = ElevRasterDE,
                        n_sites = 25,
                        min_n_samples = 5,
                        max_n_samples = 5)

## We display the simulated dataset
Aliens
```

```
## We plot the relationship between the environmental isotope values
## and those from the simulated organisms
plot(sample_value ~ source_value, data = Aliens, ylab = "Tissue", xlab = "Environment")
abline(3, 0.5, col = "blue") ## the true relationship

## We create a simulated dataset with 2 sites imputing coordinates manually
Aliens2 <- create_aliens(calib_fn = list(intercept = 3, slope = 0.5, resid_var = 5),
                         isoscape = GermanScape,
                         coordinates = data.frame(site_ID = c("Berlin", "Bielefeld"),
                                                  long   = c(13.52134, 8.49914),
                                                  lat    = c(52.50598, 52.03485)),
                         raster = ElevRasterDE,
                         min_n_samples = 5,
                         max_n_samples = 5)

Aliens2

}
```

`downloadfile` *Download files and check their binary integrity*

Description

This function is the internal function used in IsoriX to download the large files from internet and it could be useful to download anything from within R. We created this function to make sure that the downloaded files are valid. Downloads can indeed result in files that are corrupted, so we tweaked the options to reduce this possibility and the function runs a check if the signature of the file is provided to the argument `md5sum`.

Usage

```
downloadfile(
  address = NULL,
  filename = NULL,
  path = NULL,
  overwrite = FALSE,
  md5sum = NULL,
  verbose = interactive()
)
```

Arguments

`address`	A *string* indicating the address of the file on internet
`filename`	A *string* indicating the name under which the file must be stored
`path`	A *string* indicating where to store the file on the hard drive (without the file name!)

`overwrite`	A *logical* indicating if an existing file should be re-downloaded
`md5sum`	A *string* indicating the md5 signature of the valid file as created with `tools::md5sum()`
`verbose`	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session and `FALSE` otherwise. If a *numeric* is provided instead, additional information about the download will be provided if the number is greater than 1.

Value

The complete path of the downloaded file (invisibly)

Note

Users should directly use the function `getelev()` and `getprecip()`.

See Also

`getelev()`, `getprecip()`

`ElevRasterDE` *The raster of elevation for Germany*

Description

This raster contains the elevation of the surface of Germany (meters above sea level) with a resolution of approximately 30 square-km.

Format

A *RasterLayer*

Details

This raster contains elevation data of Germany in a highly aggregated form corresponding to a resolution of approximately one elevation value per 50 square-km. This is only for the purpose of having a small and easy-to-handle file to practice, but it should not be used to perform real assignments!

In the example below, we show how we generated this small raster from a large original *DEM* (digital elevation model) of the entire world. The original raster has a resolution of approximately one elevation value per square-km (cell size of 30 arcseconds, i.e. 0.0083 decimal degrees). Although working on large rasters is technically problematic (memory and CPU greedy), we highly recommend to rely on high-resolution rasters with small to moderate aggregation levels in order to perform reliable assignment analyses. Indeed, large aggregation of raster cells can bias assignments due to the transformation of all elevations into a single value per aggregated raster cell.

We downloaded "Global Multi-resolution Terrain Elevation Data 2010" from:
`https://topotools.cr.usgs.gov/gmted_viewer/viewer.htm`
and converted it into a *tif* file. Because the original file is very large, we directly provide the url link of the *tif* file in the example below.

Source

https://topotools.cr.usgs.gov/gmted_viewer/viewer.htm

See Also

prepraster to crop and/or aggregate this raster

Examples

```
## The following example require to download
## a large elevation raster with the function getelev()
## and will therefore not run unless you uncomment it

#### Creating the object ElevRasterDE
#
### Download the tif file (ca. 700 Mb)
### (see ?getelev for details on how to get the tif file)
#getelev()
#
### Convert the tif into R raster format
#  ElevationRasterBig <- raster("gmted2010_30mn.tif")
#
#  ## Create the highly agregated elevation raster
#  ElevRasterDE <- prepraster(ElevationRasterBig,
#                             aggregation_factor = 10,
#                             manual_crop = c(5.5, 15.5, 47, 55.5))
#
#  ## Plot the elevation
#  levelplot(ElevRasterDE, margin = FALSE, par.settings=RdBuTheme()) +
#    layer(sp.polygons(CountryBorders, col = "white"))
#
#  ## Compute crudely the resolution:
#  median(values(area(ElevRasterDE)))  ## approximative size of cells in km2
```

`getelev` *Download an elevation raster from internet*

Description

The function `getelev` downloads an elevation raster from internet. It is a wrapper that 1) calls the function elevatr::get_elev_raster to download the data and 2) saves the downloaded raster on the hard drive (so that you don't have to keep downloading the same file over and over again). The file saved on the disk is a *.tif file which you can directly read using the function raster::raster.

Usage

```
getelev(
  file = "~/elevation_world_z5.tif",
  z = 5,
  long_min = -180,
  long_max = 180,
  lat_min = -90,
  lat_max = 90,
  margin_pct = 5,
  override_size_check = FALSE,
  overwrite = FALSE,
  Ncpu = getOption_IsoriX("Ncpu"),
  verbose = interactive(),
  ...
)
```

Arguments

`file`	A *string* indicating where to store the file on the hard drive (default = `~/elevation_world_z5.tif`)
`z`	An *integer* between 1 and 14 indicating the resolution of the file do be downloaded (1 = lowest, 14 = highest; default = 5)
`long_min`	A *numeric* indicating the minimum longitude to select from. Should be a number between -180 and 180 (default = -180).
`long_max`	A *numeric* indicating the maximal longitude to select from. Should be a number between -180 and 180 (default = 180).
`lat_min`	A *numeric* indicating the minimum latitude to select from. Should be a number between -90 and 90 (default = -90).
`lat_max`	A *numeric* indicating the maximal latitude to select from (default = 90).
`margin_pct`	The percentage representing by how much the area should extend outside the area used for cropping (default = 5, corresponding to 5%). Set to 0 if you want exact cropping.
`override_size_check`	A *logical* indicating whether or not to override size checks (default = `FALSE`)
`overwrite`	A *logical* indicating if an existing file should be re-downloaded
`Ncpu`	An *integer* specifying the number of CPU's to use when downloading AWS tiles (default set by global package options).
`verbose`	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session and `FALSE` otherwise.
`...`	Other parameters to be passed to the function elevatr::get_elev_raster

Details

By default (and to keep with the spirit of the former implementations of `getelev` in IsoriX, which did not rely on elevatr::elevatr), an elevation raster of the whole world is downloaded with a resolution correspond to ca. 0.6 km2 per raster cell. You can increase the resolution by increasing

the value of the argument z. You can also restrict the area to be downloaded using the arguments long_min, long_max, lat_min & lat_max.

Note that when using prepraster you will be able to reduce the resolution and restrict the boundaries of this elevation raster, but you won't be able to increase the resolution or expend the boundaries. As a consequence, it is probably a good idea to overshoot a little when using getelev and download data at a resolution slightly higher than you need and for a extent larger than your data.

You can customise further what you download by using other parameters of elevatr::get_elev_raster (via the elipsis ...).

Please refer to the documentation of elevatr::get_elev_raster for information on the sources and follows link in there to know how to cite them.

Value

This function returns the full path where the file has been stored

Examples

```
## To download the high resolution
## raster in your current working
## directory, just type:
## getelev()
```

---

getprecip *Download rasters of monthly precipitation from internet*

Description

The function getprecip allows for the download of rasters of monthly precipitation from internet. It downloads the "precipitation (mm) WorldClim Version 2.1" at a spatial resolution of 30 seconds (~1 km2). After download, the function also unzip the file. The function getprecip uses the generic function downloadfile that can also be used to download directly other files. This raster needs further processing with the function prepcipitate. It can then be used to predict annual averages precipitation weighted isoscapes with the function isomultiscape.

Usage

```
getprecip(path = NULL, overwrite = FALSE, verbose = interactive())
```

Arguments

path	A *string* indicating where to store the file on the hard drive (without the file name!)
overwrite	A *logical* indicating if an existing file should be re-downloaded

verbose — A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is TRUE if users use an interactive R session and FALSE otherwise. If a *numeric* is provided instead, additional information about the download will be provided if the number is greater than 1.

Details

In the argument "path" is not provided, the file will be stored in the current working directory. The functions can create new directories, so you can also indicate a new path. The integrity of the elevation raster is tested after a call to `getprecip`. In case of corruption, try downloading the file again, specifying overwrite = TRUE to overwrite the corrupted file.

Source

https://worldclim.org/data/worldclim21.html

Examples

```
## To download the monthly precipitation
## in your current working
## directory, just type:
## getprecip()
## Mind that the file weights ca. 1GB!
```

GNIPDataDE — *Hydrogen delta values in precipitation water, Germany*

Description

This dataset contains the hydrogen delta value from precipitation water sampled at weather stations between 1961 and 2013 in Germany. These data have been kindly provided by Christine Stumpp and processed by the International Atomic Energy Agency IAEA in Vienna (GNIP Project: Global Network of Isotopes in Precipitation). These data are free to reuse provided the relevant citations (see references). These data represent a small sample of the much larger dataset compiled by the GNIP. We no longer provide larger GNIP dataset in the package as those are not free to reuse (but we do provide aggregated verisons of it; see [GNIPDataEUagg]). You can still download the complete GNIP dataset for free, but you will have to proceed to a registration process with GNIP and use their downloading interface WISER (http://www-naweb.iaea.org/napc/ih/IHS_resources_isohis.html).

Format

The *dataframe* includes 8591 observations on the following variables:

[, 1]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 2]	long	(*numeric*)	Longitude coordinate (decimal degrees)

[, 3]	elev	(*numeric*)	Elevation asl (m)
[, 4]	source_value	(*numeric*)	hydrogen delta value (per thousand)
[, 5]	year	(*numeric*)	Year of sampling
[, 6]	month	(*numeric*)	Month of sampling
[, 7]	source_ID	(*factor*)	The unique identifier of the weather station

Details

The dataset contains non-aggregated data for 27 weather stations across Germany.

This dataset is the raw data source and should not be directly used for fitting isoscapes.

Please use [prepsources] to filter the dataset by time and location.

If you want to use your own dataset, you must format your data as those produced by the function [prepsources].

Source

Data provided by the IAEA.

References

GNIP Project IAEA Global Network of Isotopes in Precipitation: https://www.iaea.org

Stumpp, C., Klaus, J., & Stichler, W. (2014). Analysis of long-term stable isotopic composition in German precipitation. Journal of hydrology, 517, 351-361.

Klaus, J., Chun, K. P., & Stumpp, C. (2015). Temporal trends in d18O composition of precipitation in Germany: insights from time series modelling and trend analysis. Hydrological Processes, 29(12), 2668-2680.

See Also

[prepsources] to prepare the dataset for the analyses and to filter by time and location.

Examples

```
head(GNIPDataDE)
```

`GNIPDataEUagg` *Hydrogen delta values in precipitation water (aggregated per location)*

Description

These datasets contain the mean and variance of hydrogen delta value from precipitation water sampled at weather stations between 1953 and 2015 in Europe ('GNIPDataEUagg') and in the entire world ('GNIPDataALLagg'). These data have been extracted from the International Atomic Energy Agency IAEA in Vienna (GNIP Project: Global Network of Isotopes in Precipitation) and processed by us using the function [prepsources]. The data are aggregated per location. We no longer provide the full non-aggregate GNIP dataset in the package as it is not free to reuse. You can still download the complete GNIP dataset for free, but you will have to proceed to a registration process with GNIP and use their downloading interface WISER (http://www-naweb.iaea.org/napc/ih/IHS_resources_isohis.html).

Format

The *dataframe*s include many observations on the following variables:

[, 1]	lat	(*numeric*)	Latitude coordinate (decimal degrees)
[, 2]	long	(*numeric*)	Longitude coordinate (decimal degrees)
[, 3]	elev	(*numeric*)	Elevation asl (m)
[, 4]	source_value	(*numeric*)	hydrogen delta value (per thousand)
[, 5]	year	(*numeric*)	Year of sampling
[, 6]	month	(*numeric*)	Month of sampling
[, 7]	source_ID	(*factor*)	The unique identifier of the weather station

Details

These datasets have been aggregated and can thus be directly used for fitting isoscapes.

If you want to use your own dataset, you must format your data as these datasets.

Source

Data provided by the IAEA and processed by us.

References

GNIP Project IAEA Global Network of Isotopes in Precipitation: https://www.iaea.org

See Also

[GNIPDataDE] for a non-aggregated dataset.

Examples

```
head(GNIPDataALLagg)
dim(GNIPDataALLagg)
head(GNIPDataEUagg)
dim(GNIPDataEUagg)
```

`isofind` *Infer spatial origins*

Description

This function performs the assignment of samples of unknown origins.

Usage

```
isofind(
  data,
  isoscape,
  calibfit = NULL,
  mask = NA,
  neglect_covPredCalib = TRUE,
  verbose = interactive()
)
```

Arguments

`data`	A *dataframe* containing the assignment data (see note below)
`isoscape`	The output of the function isoscape
`calibfit`	The output of the function calibfit (This argument is not needed if the isoscape had been fitted using isotopic ratios from sedentary animals.)
`mask`	A *SpatialPolygons* of a mask to replace values on all rasters by NA inside polygons (see details)
`neglect_covPredCalib`	A *logical* indicating whether to neglect the covariance between the uncertainty of predictions from the isoscape mean fit and the uncertainty in predictions from the calibration fit (default = `TRUE`). See **Details**.
`verbose`	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session and `FALSE` otherwise.

Details

An assignment is a comparison, for a given organism, of the predicted isotopic source value at its location of origin and the predicted isotopic source value at each location of the `isoscape`. The difference between these two values constitute the statistic of the assignment test. Under the null hypothesis (the organism is at a location with the same isotopic value than its original location), the test statistics follows a normal distribution with mean zero and a certain variance that stems from both the isoscape model fits and the calibration fit. The function isofind computes the map of p-value for such an assignment test (i.e. the p-values in all locations of the isoscape) for all samples in the dataframe `data`. The function also performs a single assignment for the entire group by combining the p-value maps of all samples using the Fisher's method (Fisher 1925). Significant p-values are strong evidence that the sample do NOT come from the candidate location (and not the

opposite!). For statistical details about this procedure as well as a discussion of which uncertainties are captured and which are not, please refer to Courtiol et al. 2019.

Details on parameters:

- *neglect_covPredCalib*: as long as the calibration method used in calibfit is "wild", a covariance is expected between the uncertainty of predictions from the isoscape mean fit and the uncertainty in predictions from the calibration fit. This is because both the isoscape and the calibration use in part the same data. By default this term is omitted (i.e. the value for the argument `neglect_covPredCalib` is `TRUE`) since in practice it seems to affect the results only negligibly in our trials and the computation of this term can be quite computer intensive. We nonetheless recommend to set `neglect_covPredCalib` to `FALSE` in your final analysis. If the calibration method used in calibfit is not "wild", this parameter has no effect.
- *mask*: a mask can be used so to remove all values falling in the mask. This can be useful for performing for example assignments on lands only and discard anything falling in large bodies of water (see example). By default our OceanMask is considered. Setting `mask` to NULL allows to prevent this automatic behaviour.

Value

This function returns a *list* of class *ISOFIND* containing itself three lists (`sample`, `group`, and `sp_points`) storing all rasters built during assignment and the spatial points for sources, calibration and assignments. The *list* `sample` contains three set of raster layers: one storing the value of the test statistic ("stat"), one storing the value of the variance of the test statistic ("var") and one storing the p-value of the test ("pv"). The *list* `group` contains one raster storing the p-values of the assignment for the group. The *list* `sp_points` contains two spatial point objects: `sources` and `calibs`.

Note

See AssignDataAlien to know which variables are needed to perform the assignment and their names.

References

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

Fisher, R.A. (1925). Statistical Methods for Research Workers. Oliver and Boyd (Edinburgh). ISBN 0-05-002170-2.

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 200) {
```

```
## We fit the models for Germany
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

GermanFit <- isofit(data = GNIPDataDEagg,
                    mean_model_fix = list(elev = TRUE, lat_abs = TRUE))


## We build the isoscape
GermanScape <- isoscape(raster = ElevRasterDE,
                        isofit = GermanFit)


## We fit the calibration model
CalibAlien <- calibfit(data = CalibDataAlien,
                       isofit = GermanFit)

## We perform the assignment on land only
AssignmentDry <- isofind(data = AssignDataAlien,
                         isoscape = GermanScape,
                         calibfit = CalibAlien)

## perform the assignment on land and water
Assignment <- isofind(data = AssignDataAlien,
                      isoscape = GermanScape,
                      calibfit = CalibAlien,
                      mask = NULL)

## We plot the group assignment
plot(Assignment, who = "group", mask = list(mask = NULL))

plot(AssignmentDry, who = "group", mask = list(mask = NULL))

## We plot the assignment for the 8 first samples
plot(AssignmentDry, who = 1:8,
     sources = list(draw = FALSE),
     calibs = list(draw = FALSE))

## We plot the assignment for the sample "Alien_10"
plot(AssignmentDry, who = "Alien_10")


### Other example without calibration:
### We will try to assign a weather station
### in the water isoscape

## We create the assignment data taking
## GARMISCH-PARTENKIRCHEN as the station to assign
GPIso <- GNIPDataDEagg[GNIPDataDEagg$source_ID == "GARMISCH-PARTENKIRCHEN", "mean_source_value"]
AssignDataGP <- data.frame(sample_value = GPIso,
                           sample_ID = "GARMISCH-PARTENKIRCHEN")

## We perform the assignment
```

```
AssignedGP <- isofind(data = AssignDataGP,
                      isoscape = GermanScape,
                      calibfit = NULL)
## We plot the assignment and
## show where the station really is (using lattice)
plot(AssignedGP) +
  xyplot(47.48~11.06,
         panel = panel.points,
         cex = 5, pch = 13, lwd = 2, col = "black")


}
```

`isofit` *Fit the isoscape models*

Description

This function fits the aggregated source data using mixed models. The fitting procedures are done by the package spaMM::spaMM which we use to jointly fit the mean isotopic values and their associated residual dispersion variance in a spatially explicit manner.

Usage

```
isofit(
  data,
  mean_model_fix = list(elev = FALSE, lat_abs = FALSE, lat_2 = FALSE, long = FALSE,
    long_2 = FALSE),
  disp_model_fix = list(elev = FALSE, lat_abs = FALSE, lat_2 = FALSE, long = FALSE,
    long_2 = FALSE),
  mean_model_rand = list(uncorr = TRUE, spatial = TRUE),
  disp_model_rand = list(uncorr = TRUE, spatial = TRUE),
  uncorr_terms = list(mean_model = "lambda", disp_model = "lambda"),
  spaMM_method = list(mean_model = "fitme", disp_model = "fitme"),
  dist_method = "Earth",
  control_mean = list(),
  control_disp = list(),
  verbose = interactive()
)
```

Arguments

`data`	The *dataframe* containing the data used for fitting the isoscape model
`mean_model_fix`	A *list* of *logical* indicating which fixed effects to consider in mean_fit
`disp_model_fix`	A *list* of *logical* indicating which fixed effects to consider in disp_fit
`mean_model_rand`	A *list* of *logical* indicating which random effects to consider in mean_fit

`disp_model_rand`
: A *list* of *logical* indicating which random effects to consider in disp_fit

`uncorr_terms`
: A *list* of two strings defining the parametrization used to model the uncorrelated random effects for mean_fit and disp_fit

`spaMM_method`
: A *list* of two strings defining the spaMM functions used for mean_fit and disp_fit

`dist_method`
: A *string* indicating the distance method

`control_mean`
: A *list* of additional arguments to be passed to the call of mean_fit

`control_disp`
: A *list* of additional arguments to be passed to the call of disp_fit

`verbose`
: A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session and `FALSE` otherwise.

Details

The detailed statistical definition of the isoscape model is described in Courtiol & Rousset 2017 and summarized in Courtiol et al. 2019.

Briefly, the fitting procedure of the isoscape model is divided into two fits: `mean_fit` and `disp_fit`. `mean_fit` corresponds to the fit of the "mean model", which we will use to predict the mean isotopic values at any location in other functions of the package. `disp_fit` corresponds to the fit of the "residual dispersion model", which we will use to predict the residual dispersion variance associated to the mean predictions. `mean_fit` is a linear mixed-effects model (LMM) with fixed effects, an optional spatial random effect with a Matérn correlation structure and an optional uncorrelated random effect accounting for variation between sources unrelated to their location. `disp_fit` is a Gamma Generalized LMM (Gamma GLMM) that also has fixed effects, an optional spatial random effect with a Matérn correlation structure and an optional uncorrelated random effect. For the GLMM the residual variance is fixed to its theoretical expectation.

The *dataframe* `data` must contain a single row per source location with the following columns: `mean_source_value` (the mean isotopic value), `var_source_value` (the unbiased variance estimate of the isotopic value at the location), `n_source_value` (the number of measurements performed at the location, could be 1) and `source_ID` (a factor defining the identity of the sources at a given location).

The arguments `mean_model_fix` and `disp_model_fix` allow the user to choose among different fixed-effect structures for each model. These arguments are lists of booleans (`TRUE` or `FALSE`), which define which of the following fixed effects must be considered: the elevation (`elev`), the absolute value of the latitude (`lat_abs`), the squared latitude (`lat_2`), the longitude (`long`) and the squared longitude (`long_2`). An intercept is always considered in both models.

In the models, the mean (for the mean model) or the log residual variance (for the residual dispersion model) follow a Gaussian distribution around a constant value. The arguments `mean_model_rand` and `disp_model_rand` allow to choose among different random effects for each model influencing the realizations of these Gaussian random processes. For each model one can choose not to include random effects or to include an uncorrelated random effect, a spatial random effect, or both (default). Setting `"uncorr" = TRUE` implies that the realizations of the random effect differ between sources for reasons that have nothing to do with the relative geographic location (e.g. some microclimate or some measurement errors trigger a shift in all measurements (mean model) or a shift in the variance between measurements (residual dispersion model) performed at a given source by the same amount). Setting `"spatial" = TRUE` (default) implies that the random realizations of the

Gaussian process follow a Matérn correlation structure. Put simply, this implies that the closer two locations are, the more similar the means (or the log residual variance) in isotopic values are (e.g. because they are likely to be traversed by the same air masses).

The arguments `uncorr_terms` allow the choice between two alternative parametrizations for the uncorrelated random effect in the fits: `"lambda"` or `"nugget"` for each model. When using `"lambda"`, the variance of the uncorrelated random terms is classically modelled by a variance. When a spatial random effect is considered, one can alternatively choose `"nugget"`, which modifies the Matérn correlation value when distance between location tends to zero. If no random effect is considered, one should stick to the default setting and it will be ignored by the function. The choice of the parametrization is a matter of personal preferences and it does not change the underlying models, so the estimations for all the other parameters of the models should not be impacted by whether one chooses `"lambda"` or `"nugget"`. However, only uncertainty in the estimation of `"lambda"` can be accounted for while computing prediction variances, which is why we chose this alternative as the default. Depending on the data one parametrization may lead to faster fit than the other.

The argument `spaMM_method` is also a list of two *strings* where the first element defines the spaMM functions used for fitting the mean model and the second element defines the spaMM method used for fitting the residual dispersion model. The possible options are `"HLfit"`, `"corrHLfit"` and `"fitme"`. Note that `"HLfit"` shall only be used in the absence of a Matérn correlation structure and `"corrHLfit"` shall only be used in the presence of it. In contrast, `"fitme"` should work in all situations. Which method is best remains to be determined and it is good practice to try different methods (if applicable) to check for the robustness of the results. If all is well one should obtain very similar results with the different methods. If this is not the case, carefully check the model output to see if one model fit did not get stuck at a local minimum during optimization (which would translate in a lower likelihood, or weird isoscapes looking flat with high peaks at very localised locations).

The argument `dist_method` allows modifying how the distance between locations is computed to estimate the spatial correlation structure. By default, we consider the so-called "Earth" distances which are technically called orthodromic distances. They account for earth curvature. The alternative "Euclidean" distances do not. For studies performed on a small geographic scale, both distance methods should lead to similar results.

The arguments `control_mean` and `control_dist` are lists that are transmitted to the spaMM::spaMM fitting functions (defined by `spaMM_method`). These lists can be used to finely control the fitting procedure, so advanced knowledge of the package spaMM::spaMM is required before messing around with these inputs.

We highly recommend users to examine the output produced by isofit. Sometimes, poor fit may occur and such models should therefore not be used for building isoscapes or performing assignments.

Value

This function returns a *list* of class *ISOFIT* containing two inter-related fits: `mean_fit` and `disp_fit`. The returned *list* also contains the object `info_fit` that contains all the call arguments.

Note

There is no reason to restrict `mean_fit` and `disp_fit` to using the same parametrization for fixed and random effects.

Never use a mean_fit object to draw predictions without considering a disp_fit object: mean_fit is not fitted independently from disp_fit.

For all methods, fixed effects are being estimated by Maximum Likelihood (ML) and dispersion parameters (i.e. random effects and Matern correlation parameters) are estimated by Restricted Maximum Likelihood (REML). Using REML provides more accurate prediction intervals but impedes the accuracy of Likelihood Ratio Tests (LRT). Our choice for REML was motivated by the fact that our package is more likely to be used for drawing inferences than null hypothesis testing. Users interested in model comparisons may rely on the conditional AIC values that can be extracted from fitted models using the function spaMM::AIC from **spaMM**.

Variable names for `data` must be respected to ensure a correct utilization of this package. Alteration to the fixed effect structure is not implemented so far (beyond the different options proposed) to avoid misuse of the package. Users that would require more flexibility should consider using spaMM directly (see Courtiol & Rousset 2017) or let us know which other covariates would be useful to add in IsoriX.

Source

https://kimura.univ-montp2.fr/~rousset/spaMM.htm

References

Courtiol, A., Rousset, F. (2017). Modelling isoscapes using mixed models. https://www.biorxiv.org/content/10.1101/207662v1

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

Rousset, F., Ferdy, J. B. (2014). Testing environmental and genetic effects in the presence of spatial autocorrelation. Ecography, 37(8):781-790.

Bowen, G. J., Wassenaar, L. I., Hobson, K. A. (2005). Global application of stable hydrogen and oxygen isotopes to wildlife forensics. Oecologia, 143(3):337-348.

See Also

spaMM::spaMM for an overview of the **spaMM** package

spaMM::fitme and spaMM::corrHLfit for information about the two possible fitting procedures that can be used here

spaMM::atern.corr] for information about the Matérn correlation structure

prepsources for the function preparing the data for isofit

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 10) {
```

```
## Fitting the models for Germany
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

GermanFit <- isofit(data = GNIPDataDEagg, mean_model_fix = list(elev = TRUE, lat_abs = TRUE))

GermanFit

## Diagnostics for the fits
plot(GermanFit)

## Exploration of the fitted models
GermanFit$mean_fit
GermanFit$disp_fit
AIC(GermanFit$disp_fit)

}
```

`isomultifit` *Fit isoscape models per strata (typically time interval such as months)*

Description

This function fits several set of isoscapes (e.g. one per strata). It can thus be used to predict annual averages precipitation weighted isoscapes.

Usage

```
isomultifit(
  data,
  split_by = "month",
  mean_model_fix = list(elev = FALSE, lat_abs = FALSE, lat_2 = FALSE, long = FALSE,
    long_2 = FALSE),
  disp_model_fix = list(elev = FALSE, lat_abs = FALSE, lat_2 = FALSE, long = FALSE,
    long_2 = FALSE),
  mean_model_rand = list(uncorr = TRUE, spatial = TRUE),
  disp_model_rand = list(uncorr = TRUE, spatial = TRUE),
  uncorr_terms = list(mean_model = "lambda", disp_model = "lambda"),
  spaMM_method = list(mean_model = "fitme", disp_model = "fitme"),
  dist_method = "Earth",
  control_mean = list(),
  control_disp = list(),
  verbose = interactive()
)
```

Arguments

`data`	The *dataframe* containing the data used for fitting the isoscape model

split_by	A *string* indicating the name of the column of data used to split the dataset. The function isofit will then be called on each of these sub-datasets. The default behaviour is to consider that the dataset should be split per months (split_by = "month").
mean_model_fix	A *list* of *logical* indicating which fixed effects to consider in mean_fit
disp_model_fix	A *list* of *logical* indicating which fixed effects to consider in disp_fit
mean_model_rand	A *list* of *logical* indicating which random effects to consider in mean_fit
disp_model_rand	A *list* of *logical* indicating which random effects to consider in disp_fit
uncorr_terms	A *list* of two strings defining the parametrization used to model the uncorrelated random effects for mean_fit and disp_fit
spaMM_method	A *list* of two strings defining the spaMM functions used for mean_fit and disp_fit
dist_method	A *string* indicating the distance method
control_mean	A *list* of additional arguments to be passed to the call of mean_fit
control_disp	A *list* of additional arguments to be passed to the call of disp_fit
verbose	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is TRUE if users use an interactive R session and FALSE otherwise.

Details

This function is a wrapper around the function isofit.

Value

This function returns a *list* of class *MULTIISOFIT* containing all pairs of inter-related fits (stored under multi_fits). The returned *list* also contains the object info_fit that contains all the call arguments.

See Also

isofit for information about the fitting procedure of each isoscape.

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## We prepare the GNIP monthly data between January and June for Germany

GNIPDataDEmonthly <- prepsources(data = GNIPDataDE,
```

```
                                month = 1:6,
                                split_by = "month")

head(GNIPDataDEmonthly)

## We fit the isoscapes

GermanMonthlyFit <- isomultifit(data = GNIPDataDEmonthly)

GermanMonthlyFit

plot(GermanMonthlyFit)
}
```

`isomultiscape` *Predicts the average spatial distribution of isotopic values over months, years...*

Description

This function is the counterpart of isoscape for the objects created with isomultifit. It creates the isoscapes for each strata (e.g. month) defined by `split_by` during the call to isomultifit and the aggregate them. The function can handle weighting for the aggregation process and can thus be used to predict annual averages precipitation weighted isoscapes.

Usage

```
isomultiscape(raster, isofit, weighting = NULL, verbose = interactive())
```

Arguments

`raster`	The structural raster (*RasterLayer*) such as an elevation raster created using prepelev
`isofit`	The fitted isoscape created by isofit
`weighting`	An optional RasterBrick containing the weights
`verbose`	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session and `FALSE` otherwise.

Value

This function returns a *list* of class *ISOSCAPE* containing a set of all 8 raster layers mentioned above (all being of class *RasterLayer*), and the location of the sources as spatial points.

See Also

isoscape for details on the function used to compute the isoscapes for each strata isomultifit for the function fitting the isoscape

plot.ISOSCAPE for the function plotting the isoscape model

IsoriX for the complete work-flow

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 180) {

## We prepare the data and split them by month:

GNIPDataDEmonthly <- prepsources(data = GNIPDataDE,
                                 split_by = "month")

dim(GNIPDataDEmonthly)

## We fit the isoscapes:#'
GermanMultiFit <- isomultifit(data = GNIPDataDEmonthly,
                              mean_model_fix = list(elev = TRUE, lat.abs = TRUE))

## We build the annual isoscapes by simple averaging (equal weighting):
GermanMultiscape <- isomultiscape(raster = ElevRasterDE,
                                  isofit = GermanMultiFit)

## We build the annual isoscapes with a weighing based on precipitation amount:
GermanMultiscapeWeighted <- isomultiscape(raster = ElevRasterDE,
                                          isofit = GermanMultiFit,
                                          weighting = PrecipBrickDE)

## We plot the mean isoscape of the averaging with equal weighting:
plot(x = GermanMultiscape, which = "mean")

## We plot the mean isoscape of the averaging with precipitation weighting:
plot(x = GermanMultiscapeWeighted, which = "mean")

## We build the isoscapes for a given month (here January):
GermanScapeJan <- isoscape(raster = ElevRasterDE,
                           isofit = GermanMultiFit$multi_fits[["month_1"]])

## We plot the mean isoscape for January:
plot(x = GermanScapeJan, which = "mean")

}
```

`isopalette2` *Colour palettes for plotting*

Description

These datasets contain colour vectors that can be used for plotting. In our examples, we use the `isopalette1` for plotting the isoscape using plot.ISOSCAPE and `isopalette2` for plotting the assignment outcome using plot.ISOFIND.

Format

A vector of colours

Details

Colour palettes can be created by using the function colorRamp that interpolates colours between a set of given colours. One can also use colorRampPalette to create functions providing colours. Also interesting, the function colorspace::choose_palette offers a GUI interface allowing to create and save a palette in a hexadecimal format (which can later on be imported into R). This latter function is however limited to a maximum of 50 colours. You can also use R colour palettes already available such as terrain.colors or others available (see examples below). Alternatively, you can design your own colour palette by writing standard hexadecimal code of colours into a vector.

Note

We use the package **rasterVis** for plotting. Instead of using colour palettes directly, one can also use any "Theme" designed for the lattice graphic environment (see source for details).

Source

For information on how to use themes, check:

https://oscarperpinan.github.io/rastervis/#themes

See Also

grDevices::rainbow for information about R colour palettes,#' grDevices::colorRamp and colorspace::choose_palette to create your own palettes

Examples

```
## A comparison of some colour palette

par(mfrow = c(2, 3))
pie(rep(1, length(isopalette1)), col = isopalette1,
    border = NA, labels = NA, clockwise = TRUE, main = "isopalette1")
pie(rep(1, length(isopalette2)), col = isopalette2,
    border = NA, labels = NA, clockwise = TRUE, main = "isopalette2")
```

```
pie(rep(1, 100), col = terrain.colors(100), border = NA, labels = NA,
    clockwise = TRUE, main = "terrain.colors")
pie(rep(1, 100), col = rainbow(100), border = NA, labels = NA,
    clockwise = TRUE, main = "rainbow")
pie(rep(1, 100), col = topo.colors(100), border = NA, labels = NA,
    clockwise = TRUE, main = "topo.colors")
pie(rep(1, 100), col = heat.colors(100), border = NA, labels = NA,
    clockwise = TRUE, main = "heat.colors")

## Creating your own colour palette
MyPalette <- colorRampPalette(c("blue", "green", "red"), bias = 0.7)
par(mfrow = c(1, 1))
pie(1:100, col = MyPalette(100), border = NA, labels = NA,
    clockwise = TRUE, main = "a home-made palette")

## Turing palettes into functions for use in IsoriX
Isopalette1Fn <- colorRampPalette(isopalette1, bias = 0.5)
Isopalette2Fn <- colorRampPalette(isopalette2, bias = 0.5)
par(mfrow = c(1, 2))
pie(1:100, col = Isopalette1Fn(100), border = NA, labels = NA,
    clockwise = TRUE, main = "isopalette1")
pie(1:100, col = Isopalette2Fn(100), border = NA, labels = NA,
    clockwise = TRUE, main = "isopalette2")
```

IsoriX-defunct *Defunct and deprecated functions*

Description

The function you asked help for has been defunct (i.e. it does not longer exists) or deprecated (i.e. it will disappear soon). A new function with a different name is surely doing the old job.

Arguments

...	The call of the defunct or deprecated function

---

isoscape *Predicts the spatial distribution of source isotopic values*

Description

This function produces the set of isoscapes, i.e. the spatial prediction (i.e. maps) of the distribution of source isotopic values, as well as several variances around such predictions. The predictions are computed using the fitted geostatistical models for each raster cell of a structural raster. All shape files can be exported and loaded into any Geographic Information System (GIS) if needed (see online tutorials).

Usage

```
isoscape(raster, isofit, verbose = interactive())
```

Arguments

raster	The structural raster (*RasterLayer*) such as an elevation raster created using prepelev
isofit	The fitted isoscape created by isofit
verbose	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is TRUE if users use an interactive R session and FALSE otherwise.

Details

This function computes the predictions (mean), prediction variances (mean_predVar), residual variances (mean_residVar) and response variances (mean_respVar) for the isotopic values at a resolution equal to the one of the structural raster. It also computes the same information for the residual dispersion variance (disp_pred, disp_predVar, disp_residVar, or disp_respVar).

The predictions of isotopic values across the landscape are performed by calling the function spaMM::predict from the package **spaMM** on the fitted isoscape produced by isofit.

Let us summarize the meaning of mean, mean_predVar, mean_residVar and mean_respVar (see Courtiol & Rousset 2017 and Curtiol et al. 2019 for more details):

Our model assumes that that there is a single true unknown isoscape, which is fixed but which is represented by the mixed-effect model as a random draw from possible realizations of isoscapes (random draws of the Matérn-correlated process and of the uncorrelated random effects if considered). We infer this realized isoscape by fitting the model to a limited amount of data, with some uncertainty since different random draws of the unknown isoscape may give the same observed data. There is thus a conditional distribution of possible true isoscapes given the data. For linear mixed-effects models, the mean prediction is the mean of this conditional distribution. The prediction variance is ideally the mean square difference between the true unknown value of the linear predictor and the mean prediction at a given location. The residual variance is simply the prediction of the variance in isotopic value at a given location. Its exact meaning depends on the aggregation scheme used in prepsources, but by default, it would correspond to the temporal variation between months and across years. The response variance estimates the variance of new observations drawn from the true unknown isoscape at a given location. The response variance is simply equal to the sum of the prediction variance and the residual variance (note that the residual variance considered assume that a single observation is being observed per location).

The isoscape can be plotted using the function plot.ISOSCAPE (see examples).

Value

This function returns a *list* of class *ISOSCAPE* containing a set of all 8 raster layers mentioned above (all being of class *RasterLayer*), and the location of the sources as spatial points.

References

Courtiol, A., Rousset, F. (2017). Modelling isoscapes using mixed models. https://www.biorxiv.org/content/10.1101/207662v1

Courtiol A, Rousset F, Rohwäder M, Soto DX, Lehnert L, Voigt CC, Hobson KA, Wassenaar LI, Kramer-Schadt S (2019). Isoscape computation and inference of spatial origins with mixed models using the R package IsoriX. In Hobson KA, Wassenaar LI (eds.), Tracking Animal Migration with Stable Isotopes, second edition. Academic Press, London.

See Also

isofit for the function fitting the isoscape

plot.ISOSCAPE for the function plotting the isoscape model

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## We prepare the data
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

## We fit the models
GermanFit <- isofit(data = GNIPDataDEagg,
                    mean_model_fix = list(elev = TRUE, lat_abs = TRUE))

## We build the isoscapes
GermanScape <- isoscape(raster = ElevRasterDE, isofit = GermanFit)

GermanScape
plot(GermanScape)

## We build more plots
PlotMean <- plot(x = GermanScape, which = "mean", plot = FALSE)

PlotMeanPredVar <- plot(x = GermanScape, which = "mean_predVar", plot = FALSE)

PlotMeanResidVar <- plot(x = GermanScape, which = "mean_residVar", plot = FALSE)

PlotMeanRespVar <- plot(x = GermanScape, which = "mean_respVar", plot = FALSE)

## We display the plots
print(PlotMean, split = c(1, 1, 2, 2), more = TRUE)
print(PlotMeanPredVar,  split = c(2, 1, 2, 2), more = TRUE)
print(PlotMeanResidVar, split = c(1, 2, 2, 2), more = TRUE)
print(PlotMeanRespVar,  split = c(2, 2, 2, 2), more = FALSE)
```

```
## We build a sphere with our isoscape
plot(x = GermanScape, which = "mean", plot = FALSE, sphere = list(build = TRUE))

## We can save a rotating sphere with the isoscape as a .gif-file.
## This file will be located inside your working directory.
## Make sure your current rgl device (from the previous step) is still open
## and that you have both the packages 'rgl' and 'magick' installed.
## The building of the .gif implies to create temporarily many .png
## but those will be removed automatically once the .gif is done.
## Uncomment to proceed (after making sure you have rgl and magick installed)
#if(require("rgl") & require("magick")) {
#   movie3d(spin3d(axis = c(0, 0, 1), rpm = 2), duration = 30, dir = getwd())
#}

}
```

OceanMask *Mask of world oceans*

Description

This dataset contains a polygon shapefile that can be used to mask large bodies of water.

Format

A *SpatialPolygons* object

Source

This *SpatialPolygons* is derived from the CountryBorders. See example for details on how we created the dataset.

See Also

CountryBorders for another polygon shapefile used to embellish the plots

Examples

```
if(require(sp)) {
  plot(OceanMask, col='blue')
}

## How did we create this file?

## Uncomment the following to create the file as we did
#if(require(raster) & require(rgeos)){
#    worldlimit <- as(extent(CountryBorders), "SpatialPolygons")
#    proj4string(worldlimit) <- crs(CountryBorders)
```

```
#    OceanMask <- gDifference(worldlimit, CountryBorders)
#    OceanMask
#    save(OceanMask, file = "OceanMask.rda", compress = "xz")
#}
```

`options` *Setting and displaying the options of the package*

Description

Setting and displaying the options of the package

Usage

```
options_IsoriX(...)

getOption_IsoriX(x = NULL)
```

Arguments

`...`	A named value or a list of named values. The following values, with their defaults, are used: **example_maxtime** The number of seconds allowed for a given example to run. It is used to control whether the longer examples should be run or not based on the comparison between this option and the approximate running time of the example on our computers. **Ncpu** An *integer* corresponding to the number of cores to be used (in functions that can handle parallel processing) **dont_ask** A *logical* indicating if the user prompt during interactive session during plotting must be inactivated (for development purposes only)
`x`	A character string holding an option name.

Value

The options are invisibly returned in an object called `IsoriX:::.data_IsoriX$options`

Examples

```
OldOptions <- options_IsoriX()
OldOptions
getOption_IsoriX("example_maxtime")
options_IsoriX(example_maxtime = 30)
options_IsoriX()
options_IsoriX(example_maxtime = OldOptions$example_maxtime)
options_IsoriX()
```

plots *Plotting functions for IsoriX*

Description

These functions plot objects created by IsoriX (with the exception of plot method for RasterLayer created using raster.

Usage

```
## S3 method for class 'ISOSCAPE'
plot(
  x,
  which = "mean",
  y_title = list(which = TRUE, title = bquote(delta^2 * H)),
  sources = list(draw = TRUE, cex = 0.5, pch = 2, lwd = 1, col = "red"),
  borders = list(borders = NA, lwd = 0.5, col = "black"),
  mask = list(mask = NA, lwd = 0, col = "black", fill = "black"),
  palette = list(step = NA, range = c(NA, NA), n_labels = 11, digits = 2, fn = NA),
  plot = TRUE,
  sphere = list(build = FALSE, keep_image = FALSE),
  ...
)

## S3 method for class 'ISOFIND'
plot(
  x,
  who = "group",
  cutoff = list(draw = TRUE, level = 0.05, col = "#909090"),
  sources = list(draw = TRUE, cex = 0.5, pch = 2, lwd = 1, col = "red"),
  calibs = list(draw = TRUE, cex = 0.5, pch = 4, lwd = 1, col = "blue"),
  assigns = list(draw = TRUE, cex = 0.5, pch = 5, lwd = 1, col = "white"),
  borders = list(borders = NA, lwd = 0.5, col = "black"),
  mask = list(mask = NA, lwd = 0, col = "black", fill = "black"),
  mask2 = list(mask = NA, lwd = 0, col = "purple", fill = "purple"),
  palette = list(step = NA, range = c(0, 1), n_labels = 11, digits = 2, fn = NA),
  plot = TRUE,
  sphere = list(build = FALSE, keep_image = FALSE),
  ...
)

## S3 method for class 'ISOFIT'
plot(x, cex_scale = 0.2, ...)

## S3 method for class 'CALIBFIT'
plot(
  x,
```

```
  pch = 1,
  col = "black",
  xlab = "Isotopic value in the environment",
  ylab = "Isotopic value in the calibration sample",
  xlim = NULL,
  ylim = NULL,
  line = list(show = TRUE, col = "blue"),
  CI = list(show = TRUE, col = "blue"),
  plot = TRUE,
  ...
)

## S3 method for class 'CALIBFIT'
points(
  x,
  pch = 2,
  col = "red",
  line = list(show = TRUE, col = "red"),
  CI = list(show = TRUE, col = "red"),
  plot = TRUE,
  ...
)

## S3 method for class 'RasterLayer'
plot(x, ...)
```

Arguments

x	The return object of a call to isofit, isoscape, calibfit, isofind, or raster::raster]
which	A *string* indicating the name of the raster to be plotted (see details)
y_title	A *list* containing information for the display of the title (see details)
sources	A *list* containing information for the display of the location of the sources (see details)
borders	A *list* containing information for the display of borders (e.g. country borders) (see details)
mask	A *list* containing information for the display of a mask (e.g. an ocean mask) (see details)
palette	A *list* containing information for the display of the colours for the isoscape (see details)
plot	A *logical* indicating whether the plot shall be plotted or just returned
sphere	A *list* containing information whether the raster should be returned as a rotating sphere and if the image created during the process should be saved in your current working directory. The default settings are FALSE.
...	Additional arguments (only in use in plot.CALIBFIT and plot.RasterLayer)
who	Either "group", or a vector of indices (e.g. 1:3) or names of the individuals (e.g. c("Mbe_1", "Mbe_3")) to be considered in assignment plots

`cutoff`	A *list* containing information for the display of the region outside the prediction interval (see details)
`calibs`	A *list* containing information for the display of the location of the calibration sampling location (see details)
`assigns`	A *list* containing information for the display of the location of the assingment sampling location (see details)
`mask2`	A *list* containing information for the display of a mask (e.g. a distribution mask) (see details)
`cex_scale`	A *numeric* giving a scaling factor for the points in the plots
`pch`	The argument pch as in par for plot.CALIBFIT and points.CALIBFIT
`col`	The argument col as in par for plot.CALIBFIT and points.CALIBFIT
`xlab`	A *string* the x-axis label in plot.CALIBFIT
`ylab`	A *string* the y-axis label in plot.CALIBFIT
`xlim`	A range defining the extreme coordinates for the the x-axis in plot.CALIBFIT
`ylim`	A range defining the extreme coordinates for the the y-axis in plot.CALIBFIT
`line`	A *list* containing two elements: `show`, a *logical* indicating whether to show the regression line or not; and `col`, a *string* or *integer* indicating the colour for plotting the regression line
`CI`	A *list* containing two elements: `show`, a *logical* indicating whether to show the confidence interval or not; and `col`, a *string* or *integer* indicating the colour for plotting the confidence interval

Details

General

When called upon an object of class *ISOFIT*, the plot function draws diagnostic information for the fits of the isoscape geostatistical model.

When called upon an object of class *CALIBFIT*, the plot function draws the fitted calibration function.

When called upon an object of class *ISOSCAPE*, the plot function draws a fine-tuned plot of the isoscape.

When called upon an object of class *RasterLayer*, the plot function displays the raster (just for checking things fast and dirty). In this case, the function is a simple shortcut to rasterVis::levelplot.

Plotting isoscapes

When used on a fitted isoscape, the user can choose between plotting the predictions (`which = "mean"`; default), the prediction variance (`which = "mean_predVar"`), the residual variance (`which = "mean_residVar"`), or the response variance (`which = "mean_respVar"`) for the mean model; or the corresponding information for the residual dispersion variance model (`"disp"`, `"disp_predVar"`, `"disp_residVar"`, or `"disp_respVar"`).

When used on a simulated isoscape produced with the function `isosim` (currently dropped due to the package RandomFields being temporarily retired from CRAN), the user can choose between plotting the mean isotopic value (`which = "mean"`) or the residual dispersion (`which = "disp"`).

Plotting assignments

When called upon an object of class *ISOFIND*, the plot function draws a fine-tuned plot of the assignment. You can use the argument `who` to choose between plotting the assignment for the group or for some individuals (check the online tutorial for examples).

Info on parameters influencing the rendering of maps

The argument `y_title` is a list that can be tweaked to customise the title of isoscapes. Within this list, the element `which` is a logical indicating if the name of the layer should be displayed or not. The element `title` is a string or a call used to define the rest of the title. By default it draws the delta value for hydrogen. Check the syntax of this default before trying to modify it.

The arguments `cutoff`, `sources`, `calibs`, `assigns`, `borders`, `mask`, and `mask2` are used to fine-tune additional layers that can be added to the main plot to embellish it. These arguments must be lists that provide details on how to draw, respectively, the area outside the prediction interval (for assignment plots), the locations of sources (for both isoscape and assignment plots), the locations of the calibration samples (for assignment plots), the locations of the assignment samples (for assignment plots), the borders (for both types of plots), and the mask (again, for both). For assignment maps, an extra mask can be used (mask2), as one may want to add a mask covering the area outside the biological range of the species. Within these lists, the elements `lwd`, `col`, `cex`, `pch` and `fill` influences their respective objects as in traditional R plotting functions (see par for details). The element `draw` should be a *logical* that indicates whether the layer must be created or not. The argument `borders` (within the list borders) expects an object of the class *SpatialPolygons* such as the object CountryBorders provided with this package. The argument `mask` (within the list mask) expects an object of the class *SpatialPolygons* such as the object OceanMask provided with this package (see examples).

The argument `palette` is used to define how to colour the isoscape and assignment plot. Within this list, `step` defines the number of units on the z-scale that shares a given colour; `range` can be used to constrain the minimum and/or maximum values to be drawn (e.g. range = c(0, 1)) (this latter argument is useful if one wants to create several plots with the same z-scale); `n_labels` allows for the user to approximatively define the maximum number of numbers plotted on the z-scale; `digits` defines the number of digits displayed for the numbers used as labels; and `fn` is used to specify the function that is used to sample the colours. If `fn` is NULL (default) the palette functions derived from isopalette1 and isopalette2 are used when plotting isoscape and assignments, respectively. If `fn` is NA the function used is the palette viridisLite::viridis.

Default symbols used on maps

Under the default settings, we chose to represent:

- the source data by little red triangles.
- the calibration data by little blue crosses.
- the locations where the samples to assign were collected by white diamonds.

These symbols can be changed as explained above.

See Also

isofit for the function fitting the isoscape

isoscape for the function building the isoscape

calibfit for the function fitting the calibration function

isofind for the function performing the assignment

Examples

```
## See ?isoscape or ?isofind for examples
```

PrecipBrickDE *The precipitation monthly amounts for Germany*

Description

This brick of rasters contains the monthly precipitation amounts (in mm) for Germany with a resolution of approximately 30 square-km.

Format

A *RasterBrick*

Details

The data are derived from "precipitation (mm) WorldClim Version2" which can be downloaded using the function getprecip.

Source

https://www.worldclim.org/data/worldclim21.html

See Also

prepcipitate to prepare this raster

Examples

```
## The following example requires to download
## a large precipitation rasters with the function getprecip()
## and will therefore not run unless you uncomment it

## How did we create this file?

## Uncomment the following to create the file as we did
#getprecip() ## Download the tif files (~ 1 Gb compressed)
#PrecipBrickDE <- prepcipitate(raster = ElevRasterDE)
#save(PrecipBrickDE, file = "PrecipBrickDE", compress = "xz")
```

prepcipitate *Prepare the raster brick containing the precipitation data*

Description

This functions turns the WorldClim data downloaded using the function getprecip into a *RasterBrick* of same resolution and extent as the structural raster. This function is designed to be used with isomultiscape.

Usage

```
prepcipitate(path = NULL, raster, verbose = interactive())
```

Arguments

path	A *string* indicating the path where the WorldClim data have been downloaded. If the path is null (the default) the function will assume that the folder containing the precipitation data is in the current directory
raster	A *raster* containing the structural raster
verbose	A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is TRUE if users use an interactive R session, and FALSE otherwise.

See Also

getprecip to download the relevant precipitation data

PrecipBrickDE for the stored precipitation data for Germany

prepelev to prepare an elevation raster

Examples

```
## The following example takes some time and download a large amount of data (~ 1 Gb).
## It will therefore not be run unless you uncomment it

### We fit the models for Germany:
#GNIPDataDEagg <- prepsources(data = GNIPDataDE)
#
#GermanFit <- isofit(data = GNIPDataDEagg,
#                    mean_model_fix = list(elev = TRUE, lat.abs = TRUE))
#
### We prepare the structural raster:
#StrRaster <- prepraster(raster = ElevRasterDE,
#                        isofit = GermanFit,
#                        aggregation_factor = 0)
#
### We download the precipitation data:
```

```
#getprecip(path = "~/Downloads/")
#
### We prepare the raster brick with all the precipitation data:
#PrecipitationBrick <- prepcipitate(path = "~/Downloads/",
#                                   raster = StrRaster)
#
### We plot the precipitation data:
#levelplot(PrecipitationBrick)
```

`prepraster` *Prepare the structural raster*

Description

This function prepares the structural raster for the follow-up analyses. The size and extent of the structural raster defines the resolution at which the isoscapes and the assignments are defined.

Usage

```
prepraster(
  raster,
  isofit = NULL,
  margin_pct = 5,
  aggregation_factor = 0L,
  aggregation_fn = mean,
  manual_crop = NULL,
  values_to_zero = c(-Inf, 0),
  verbose = interactive()
)
```

Arguments

`raster`	The structural raster (*RasterLayer*)
`isofit`	The fitted isoscape model returned by the function isofit
`margin_pct`	The percentage representing by how much the area should extend outside the area used for cropping (default = 5, corresponding to 5%). Set to 0 if you want exact cropping.
`aggregation_factor`	The number of neighbouring cells (*integer*) to merge during aggregation
`aggregation_fn`	The *function* used to aggregate cells
`manual_crop`	A vector of four coordinates (*numeric*) for manual cropping, e.g. the spatial extent
`values_to_zero`	A *numeric vector* of length two specifying the range of values for the structural raster that must be turned into 0. Default is `c(-Inf, 0)` which for an elevation raster brings all seas to an elevation of zero. For using IsoriX for marine organisms, you should use `c(0, Inf)` instead.

`verbose` A *logical* indicating whether information about the progress of the procedure should be displayed or not while the function is running. By default verbose is `TRUE` if users use an interactive R session, and `FALSE` otherwise.

Details

This functions allows the user to crop a raster according to either the extent of the isoscape or manually. If a fitted isoscape object is provided (see isofit), the function extracts the observed locations of isotopic sources from the model object and crops the structural raster accordingly. Alternatively, `manual_crop` allows you to crop the structural raster to a desired extent. If no model and no coordinates for manual cropping are provided, no crop will be performed. Importantly, cropping is recommended as it prevents extrapolations outside the latitude/longitude range of the source data. Predicting outside the range of the source data may lead to highly unreliable predictions.

Aggregation changes the spatial resolution of the raster, making computation faster and using less memory (this can affect the assignment; see note below). An aggregation factor of zero (or one) keeps the resolution constant (default).

This function relies on calls to the functions raster::aggregate and raster::crop from the package **raster**. It thus share the limitations of these functions. In particular, raster::crop expects extents with increasing longitudes and latitudes. We have tried to partially relax this constrains for longitude and you can use the argument `manual_crop` to provide longitudes in decreasing order, which is useful to centre a isoscape around the pacific for instance. But this fix does not solve all the limitations as plotting polygons or points on top of that remains problematic (see example bellow). We will work on this on the future but we have other priorities for now (let us know if you really need this feature).

Value

The prepared structural raster of class *RasterLayer*

Note

Aggregating the raster may lead to different results for the assignment, because the values of raster cells changes depending on the aggregation function (see example below), which in turn affects model predictions.

See Also

ElevRasterDE for information on elevation rasters, which can be used as structural rasters.

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 30) {

## We fit the models for Germany
```

```
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

GermanFit <- isofit(data = GNIPDataDEagg,
                    mean_model_fix = list(elev = TRUE, lat_abs = TRUE))

### Let's explore the difference between aggregation schemes

## We aggregate and crop using different settings
ElevationRaster1 <- prepraster(
    raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 0,
    aggregation_factor = 0)

ElevationRaster2 <- prepraster(
    raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 5,
    aggregation_factor = 5)

ElevationRaster3 <- prepraster(
    raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 10,
    aggregation_factor = 5, aggregation_fn = max)

## We build the plots of the outcome of the 3 different aggregation schemes
PlotAggregation1 <- levelplot(ElevationRaster1,
        margin = FALSE, main = "Original small raster") +
    layer(sp.polygons(CountryBorders)) +
    layer(sp.polygons(OceanMask, fill = "blue"))
PlotAggregation2 <- levelplot(ElevationRaster2,
        margin = FALSE, main = "Small raster aggregated (by mean)") +
    layer(sp.polygons(CountryBorders)) +
    layer(sp.polygons(OceanMask, fill = "blue"))
PlotAggregation3 <- levelplot(ElevationRaster3,
        margin = FALSE, main = "Small raster aggregated (by max)") +
    layer(sp.polygons(CountryBorders)) +
    layer(sp.polygons(OceanMask, fill = "blue"))

## We plot as a panel using lattice syntax:
print(PlotAggregation1, split = c(1, 1, 3, 1), more = TRUE)
print(PlotAggregation2, split = c(2, 1, 3, 1), more = TRUE)
print(PlotAggregation3, split = c(3, 1, 3, 1))
}

## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. options_IsoriX(example_maxtime = XX)
## if you want to allow for examples taking up to ca. XX seconds to run
## (so don't write XX but put a number instead!)

if(getOption_IsoriX("example_maxtime") > 10) {
```

```
### Let's create a raster centered around the pacific

## We first create an empty raster
EmptyRaster <- raster(matrix(0, ncol = 360, nrow = 180))
extent(EmptyRaster) <- c(-180, 180, -90, 90)
projection(EmptyRaster) <- CRS("+proj=longlat +datum=WGS84")

## We crop it around the pacific
PacificA <- prepraster(EmptyRaster, manual_crop = c(110, -70, -90, 90))
extent(PacificA) # note that the extent has changed!

## We plot (note the use of the function shift()!)
levelplot(PacificA, margin = FALSE, colorkey = FALSE) +
  layer(sp.polygons(CountryBorders, fill = "black")) +
  layer(sp.polygons(shift(CountryBorders, dx = 360), fill = "black"))

}
```

`prepsources` *Filter and aggregate the raw source dataset*

Description

This function prepares the available dataset to be used for creating the isoscape (e.g. GNIPDataDE). This function allows the trimming of data by months, years and location, and for the aggregation of selected data per location, location:month combination or location:year combination. The function can also be used to randomly exclude some observations.

Usage

```
prepsources(
  data,
  month = 1:12,
  year,
  long_min = -180,
  long_max = 180,
  lat_min = -90,
  lat_max = 90,
  split_by = NULL,
  prop_random = 0,
  random_level = "source",
  col_source_value = "source_value",
  col_source_ID = "source_ID",
  col_lat = "lat",
  col_long = "long",
  col_elev = "elev",
  col_month = "month",
  col_year = "year"
)
```

Arguments

`data`	A *dataframe* containing raw isotopic measurements of sources
`month`	A *numeric vector* indicating the months to select from. Should be a vector of round numbers between 1 and 12. The default is 1:12 selecting all months.
`year`	A *numeric vector* indicating the years to select from. Should be a vector of round numbers. The default is to select all years available.
`long_min`	A *numeric* indicating the minimum longitude to select from. Should be a number between -180 and 180 (default = -180).
`long_max`	A *numeric* indicating the maximal longitude to select from. Should be a number between -180 and 180 (default = 180).
`lat_min`	A *numeric* indicating the minimum latitude to select from. Should be a number between -90 and 90 (default = -90).
`lat_max`	A *numeric* indicating the maximal latitude to select from (default = 90).
`split_by`	A *string* indicating whether data should be aggregated per location (`split_by = NULL`, the default), per location:month combination (`split_by = "month"`), or per location:year combination (`split_by = "year"`).
`prop_random`	A *numeric* indicating the proportion of observations or sampling locations (depending on the argument for `random_level`) that will be kept. If `prop_random` is greater than 0, then the function will return a list containing two dataframes: one containing the selected data, called `selected_data`, and one containing the remaining data, called `remaining_data`.
`random_level`	A *string* indicating the level at which random draws can be performed. The two possibilities are `"obs"`, which indicates that observations are randomly drawn taken independently of their location, or "source" (default), which indicates that observations are randomly drawn at the level of sampling locations.
`col_source_value`	A *string* indicating the column containing the isotopic measurements
`col_source_ID`	A *string* indicating the column containing the ID of each sampling location
`col_lat`	A *string* indicating the column containing the latitude of each sampling location
`col_long`	A *string* indicating the column containing the longitude of each sampling location
`col_elev`	A *string* indicating the column containing the elevation of each sampling location
`col_month`	A *string* indicating the column containing the month of sampling
`col_year`	A *string* indicating the column containing the year of sampling

Details

This function aggregates the data as required for the IsoriX workflow. Three aggregation schemes are possible for now. The most simple one, used as default, aggregates the data so to obtained a single row per sampling location. Datasets prepared in this way can be readily fitted with the function isofit to build an isoscape. It is also possible to aggregate data in a different way in order to build sub-isoscapes representing temporal variation in isotope composition, or in order to produce

isoscapes weighted by the amount of precipitation (for isoscapes on precipitation data only). The two possible options are to either split the data from each location by month or to split them by year. This is set with the `split_by` argument of the function. Datasets prepared in this way should be fitted with the function isomultifit.

The function also allows the user to filter the sampling locations based on time (years and/ or months) and space (locations given in geographic coordinates, i.e. longitude and latitude) to calculate tailored isoscapes matching e.g. the time of sampling and speeding up the model fit by cropping/clipping a certain area. The dataframe produced by this function can be used as input to fit the isoscape (see isofit and isomultifit).

Value

This function returns a *dataframe* containing the filtered data aggregated by sampling location, or a *list*, see above argument `prop_random`. For each sampling location the mean and variance sample estimates are computed.

Examples

```
## Create a processed dataset for Germany
GNIPDataDEagg <- prepsources(data = GNIPDataDE)

head(GNIPDataDEagg)

## Create a processed dataset for Germany per month
GNIPDataDEmonthly <-prepsources(data = GNIPDataDE,
                                split_by = "month")

head(GNIPDataDEmonthly)

## Create a processed dataset for Germany per year
GNIPDataDEyearly <- prepsources(data = GNIPDataDE,
                                split_by = "year")

head(GNIPDataDEyearly)

## Create isoscape-dataset for warm months in germany between 1995 and 1996
GNIPDataDEwarm <- prepsources(data = GNIPDataDE,
                              month = 5:8,
                              year = 1995:1996)

head(GNIPDataDEwarm)


## Create a dataset with 90% of obs
GNIPDataDE90pct <- prepsources(data = GNIPDataDE,
                               prop_random = 0.9,
                               random_level = "obs")

lapply(GNIPDataDE90pct, head) # show beginning of both datasets

## Create a dataset with half the weather sources
```

```
GNIPDataDE50pctsources <- prepsources(data = GNIPDataDE,
                                      prop_random = 0.5,
                                      random_level = "source")

lapply(GNIPDataDE50pctsources, head)


## Create a dataset with half the weather sources split per month
GNIPDataDE50pctsourcesMonthly <- prepsources(data = GNIPDataDE,
                                             split_by = "month",
                                             prop_random = 0.5,
                                             random_level = "source")

lapply(GNIPDataDE50pctsourcesMonthly, head)
```

Index