Package 'IsoriX'

August 14, 2017

Version 0.7

Date 2017-06-29

Encoding UTF-8

Title Isoscape Computation and Inference of Spatial Origins using Mixed Models

Author Alexandre Courtiol [aut, cre],
François Rousset [aut],
Marie-Sophie Rohwaeder [aut],
Stephanie Kramer-Schadt [aut, cre]

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

Depends R (>= 3.1.0)

Imports graphics, grDevices, grid, latticeExtra, numDeriv, raster, rasterVis (>= 0.30), sp, spaMM (>= 2.1.3), stats, tools, utils, viridisLite

Description Building isoscapes using mixed models and inferring the geographic origin of organisms based on their isotopic ratios. This package is essentially a simplified interface to several other packages. 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 using lattice.

License GPL (>= 2)

Suggests Cairo, GISTools, lattice, maps, maptools, RandomFields, R.rsp, rgdal, rgeos

VignetteBuilder R.rsp

LazyData true

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

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

RoxygenNote 6.0.1

NeedsCompilation no

Repository CRAN

Date/Publication 2017-08-14 09:53:16 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. 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 using the powerful **lattice** visualization system.

Details

We describe below, step-by-step, the general work-flow 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 vignettes provided with this package, which essentially cover the same material in a more practical and 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 should soon be available as publications (we are currently working on it).

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 deuterium 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`.

2. Preparing the elevation raster with `relevate`:

   Building isoscapes and assigning organisms to their origin requires an adequate elevation raster, i.e. a matrix representing altitude data on a spatial grid. The function `relevate` 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 elevation raster (in order to speed up computation in all following steps). Note that aggregating the raster may lead to different results for the assignment, because the elevation 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. 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 isoscape: it uses the fitted geostatistical model to predict the isotopic values for each raster cell defined by the elevation 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 vignette `exportGIS` 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 (e.g. the isoscape is built on precipitation isotopic values and organisms are not water drops, i.e. the deuterium values of their keratin structures were modulated by their distinct physiology). 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 organisms 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 uncertainty stemming from the model fits (geostatistical model and calibration model). The function `plot.isorix` then draws such assignment by plotting the most likely origin with the prediction region around it. When several organisms are being assigned, both individual assignments and a single assignment for the whole group can be performed.

Note

Please note that for now, 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>,

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

References

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.

Examples

```
### A simple workflow for IsoriX
### is provided in a vignette:
### vignette("Workflow", "IsoriX")
```

AssignDataAlien *Simulated assignment dataset*

Description

This dataset contains simulated deuterium 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]	animalID	(*Factor*)	Identification of the animal
[, 2]	tissue.value	(*numeric*)	Deuterium 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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

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

## We build the isoscape
isoscape <- isoscape(elevation.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 = isoscape,
                        elevation_raster = ElevRasterDE,
                        coordinates = data.frame(siteID = "Berlin",
                                                 long   = 13.52134,
                                                 lat    = 52.50598),
                        n_sites = 1,
                        min_n_samples = 10,
                        max_n_samples = 10)
AssignDataAlien <- Aliens[, c("animalID", "tissue.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, Lehmann and Greif (2015). It contains deuterium 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]	animalID	(*Factor*)	Identification of the animal
[, 2]	species	(*Factor*)	Animal species name
[, 3]	tissue.value	(*numeric*)	Deuterium 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(AssignDataBat)
str(AssignDataBat)
```

`CalibDataAlien` *Simulated calibration dataset*

Description

This dataset contains simulated deuterium delta values for corresponding locations based on an assumed linear relationship between the animal tissue value and the deuterium 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]	siteID	(*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]	animalID	(*Factor*)	Identification of the sampled animal
[, 6]	tissue.value	(*numeric*)	Deuterium 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 `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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

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

## We build the isoscape
isoscape <- isoscape(elevation.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 = isoscape,
                        elevation_raster = ElevRasterDE,
                        n_sites = 50,
                        min_n_samples = 10,
                        max_n_samples = 10)
CalibDataAlien$env.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 deuterium 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 7 variables:

[, 1]	siteID	(*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]	animalID	(*Factor*)	Identification of the sampled animal
[, 6]	species	(*Factor*)	Species name
[, 7]	tissue.value	(*numeric*)	Deuterium 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 `extract`. In this dataset, we retrieved elevations from the Global Multi-resolution Terrain Elevation Data 2010. 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

calibfit to fit a calibration model

Examples

```
head(CalibDataBat)
str(CalibDataBat)

## The following example require to have downloaded
## a large elevation raster with the function getelev()
## and will therefore not run unless you type:
## example(CalibDataBat, run.dontrun=TRUE)

## Not run:
if(require(raster)){
    ## We delete the elevation data
    CalibDataBat$elev <- NULL

    ## We reconstruct the elevation data using an elevation raster
    ## (see ?getelev for details on how to get the tif file)
    elevationrasterbig <- raster("gmted2010_30mn.tif")
    CalibDataBat$elev <- extract(
        elevationrasterbig,
        cbind(CalibDataBat$long, CalibDataBat$lat))
    head(CalibDataBat)
}


## End(Not run)
```

Calibfit *Deprecated functions*

Description

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

Usage

```
Calibfit(...)

GetElev(...)
```

```
Isorix(...)

Isofit(...)

Isoscape(...)

Isosim(...)

queryGNIP(...)

QueryGNIP(...)

RElevate(...)
```

Arguments

...	The call of the deprecated function

`calibfit` *Fit the calibration model*

Description

This function fits a model that establishes the relationship between the isotopic values of organisms (e.g. tissues such as hair, horn, ivory or feathers) and the isotopic values of their environment (e.g. precipitation).

Usage

```
calibfit(calib.data, isofit, verbose = interactive(),
  control.optim = list())
```

Arguments

`calib.data`	A *dataframe* containing the calibration data (see note below)
`isofit`	The fitted isoscape model 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.
`control.optim`	A *list* to pass information to the argument control of the optim call (for advanced users only).

Details

The calibration model 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).

This 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. All models used in **IsoriX** will be soon detailed in an additional document.

This function is only needed in the case for which the assignment of organisms has to be performed within an isoscape that was built using another source of isotopic values (e.g., precipitation). This implies that if the isoscape had been fitted using isotopic ratios from sedentary animals, then this calibration step is not needed.

If source isotopic values are known at the locations where sedentary organisms were sampled, users should calibrate their data directly using the function `lm` by fitting tissue isotopic values as a function of source isotopic values.

Value

This function returns a *list* of class *calibfit* containing the fixed-effect estimates of the calibration function, the covariance of the fixed effects, the fitted calibration model, the original calibration data set with additional information added during the fit, and the location of the calibration points as spatial points.

Note

See `CalibDataAlien` to know which variables are needed to perform the calibration fit and their names.

See Also

`IsoriX` for the complete workflow

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

```
GermanFit <- isofit(iso.data = GNIPDataDEagg)

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

## We display minimal information:
calib

## We display more information:
summary(calib)

## We plot the calibration function:
plot(calib)

}
```

`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 `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?

if(require(maps) & require(maptools) & require(raster) & 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("+proj=longlat +datum=WGS84")
    CountryBorders
    ## Uncomment the following to store the file as we did
    #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, elevation_raster = NULL,
  n_sites = 1, 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 `siteID`, `long` and `lat`
`elevation_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 environement. The linear function can be parameterized 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 controled in two ways. A first possiblity 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 possiblity is to provide a

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

Irrespectivelly of how locations are choosen, 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 environement 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 organims. The actual values is then drawn from a Gaussian distribution centered 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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

GermanFit <- isofit(iso.data = GNIPDataDEagg)

## We build the isoscapes
isoscape <- isoscape(elevation.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 = isoscape,
                        elevation_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(tissue.value ~ env.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 = isoscape,
                         coordinates = data.frame(siteID = c("Berlin", "Bielefeld"),
                                                  long   = c(13.52134, 8.49914),
                                                  lat    = c(52.50598, 52.03485)),
                         elevation_raster = ElevRasterDE,
                         n_sites = 25,
                         min_n_samples = 5,
                         max_n_samples = 5)

head(Aliens2)

}
```

`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 50 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:
http://topotools.cr.usgs.gov/gmted_viewer/

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

http://topotools.cr.usgs.gov/gmted_viewer/

See Also

relevate to crop and/or aggregate the elevation raster

Examples

```
## The following example require to have downloaded
## a large elevation raster with the function getelev()
## and will therefore not run unless you type:
## example(ElevRasterDE, run.dontrun=TRUE)

## Not run:
### 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
if(require(raster)) {
  elevationrasterbig <- raster("gmted2010_30mn.tif")

  ## Create the highly agregated elevation raster
  ElevRasterDE <- relevate(elevationrasterbig,
                           aggregation.factor = 10,
                           manual.crop = c(5.5, 15.5, 47, 55.5))

  ## Plot the elevation
  if (require("sp") & require("rasterVis")) {
    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
}

## End(Not run)
```

getelev *Download an elevation raster from internet*

Description

The function getelev allows for the download of an elevation raster from internet. It downloads the "Global Multi-resolution Terrain Elevation Data 2010" from our server. The file was orifginally downloaded from:
http://topotools.cr.usgs.gov/gmted_viewer/
and converted into a *tif* file by us. The function getelev uses the generic function downloadfile that can also be used to download directly other files. This raster needs further processing with the function relevate and can then be passed to isoscape.

Usage

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

Arguments

path	A *string* indicating where to store the file on the hard drive
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

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

Source

http://topotools.cr.usgs.gov/gmted_viewer/

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 Version2" 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
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

precipitation weighted isoscapes 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

http://worldclim.org/version2

Examples

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

GNIPDataDE *Deuterium in precipitation water, Germany*

Description

This dataset contains the mean and variance of Deuterium 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 relevent 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. 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]	isoscape.value	(*numeric*)	Deuterium stable hydrogen delta value [per thousand]
[, 5]	year	(*numeric*)	Year of sampling
[, 6]	month	(*numeric*)	Month of sampling
[, 7]	stationID	(*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 prepdata 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 prepdata.

Source

Data provided by the IAEA.

References

GNIP Project IAEA Global Network of Isotopes in Precipitation: http://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

prepdata to prepare the dataset for the analyses and to filter by time and location.

Examples

```
head(GNIPDataDE)
```

`isofind` *Infer spatial origins*

Description

This function performs the assignment of organims of unknown origins.

Usage

```
isofind(assign.data, isoscape, calibfit = NULL, mask = NA,
  verbose = interactive())
```

Arguments

`assign.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)
`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 value at its location of origin and the predicted isotopic value at each location of the `isoscape`. The difference between these two values constitute the statistic of the assingment test. Under the null hypothesis (when 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 individuals in the

dataframe assign.data. The function also performs a single assignment for the entire group by combining the individual p-value maps using the Fisher's method (Fisher 1925).

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 *isorix* containing itself three lists (indiv, group, and sp.points) storing all rasters built during assignment and the spatial points for sources and calibration. The *list* indiv contains three stack 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: source.points and calib.points.

Note

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

References

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

See Also

IsoriX for the complete workflow

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 200) {

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

GermanFit <- isofit(iso.data = GNIPDataDEagg)


## We build the isoscape
isoscape <- isoscape(elevation.raster = ElevRasterDE,
                     isofit = GermanFit)
```

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

## We perform the assignment on land only
assignment.dry <- isofind(assign.data = AssignDataAlien,
                          isoscape = isoscape,
                          calibfit = calib)

## perform the assignment on land and water
assignment <- isofind(assign.data = AssignDataAlien,
                      isoscape = isoscape,
                      calibfit = calib,
                      mask = NULL)

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

plot(assignment.dry, who = "group", mask = list(mask = NULL))

## We plot the assignment for the 8 first individuals
plot(assignment.dry, who = 1:8,
     sources = list(draw = FALSE),
     calib = list(draw = FALSE))

## We plot the assignment for the individual "Alien_10"
plot(assignment.dry, 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$stationID == "GARMISCH-PARTENKIRCHEN", "isoscape.value"]
AssignDataGP <- data.frame(tissue.value = GPIso,
                           animalID = "GARMISCH-PARTENKIRCHEN")

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


}
```

`isofit` *Fit the isoscape model*

Description

This function fits the isocape as a mixed model. The fitting procedures are done by the package **spaMM** which we use to jointly fit the mean isotopic values and their associated residual dispersion variance in a spatially explicit manner.

Usage

```
isofit(iso.data, mean.model.fix = list(elev = TRUE, lat.abs = TRUE, 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

`iso.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 will be soon available in another document. 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 Matern correlation structure and an optional uncorrelated random effect accounting for variation between weather station 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 Matern correlation structure and an optional uncorrelated random effect. For the GLMM the residual variance is fixed to its theoretical expectation.

The *dataframe* `iso.data` must contain a single row per source location with the following columns: `isoscape.value` (the isotopic value), `var.isoscape.value` (the unbiased variance estimate of the isotopic value at the location), `n.isoscape.value` (the number of measurements performed at the location, could be 1) and `stationID` (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. By default, only intercept are being fitted.

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 different realizations are identical for a given weather station (e.g. some micro-climate or some measurement errors trigger a shift in all measurements (mean model) or a shift in the variance between measurements (residual dispersion model) performed on a given weather station by the same amount). Setting `"spatial" = TRUE` (default) implies that the random realizations of the Gaussian process follow a Matern 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 parameterizations 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 modeled by a variance. When a spatial random effect is considered, one can alternatively choose `"nugget"`, which modifies the Matern 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 will not be impacted by whether one chooses `lambda` or `nugget`. 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 Matern 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).

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** fitting functions (defined by `spaMM.method`). These lists can be used to finely control the fitting procedure, so advanced knowledge of the package **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 AIC values that can be extracted from fitted models using the function `AIC` from the **spaMM**.

Variable names for `iso.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 (at their own risks). We will soon provide a document explaining how to do so.

Source

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

References

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 for an overview of the **spaMM** package

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

`Matern.corr` for information about the Matern correlation structure

`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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 10) {

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

GermanFit <- isofit(iso.data = GNIPDataDEagg)

GermanFit

## Diagnostics for the fits
plot(GermanFit)

}
```

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

Description

This function fits several bunch of isocapes (e.g. one per strata), which we call sub-isoscape. It can thus be used to predict annual averages precipitation weighted isoscapes.

Usage

```
isomultifit(iso.data, split.by = "month", mean.model.fix = list(elev = TRUE,
  lat.abs = TRUE, 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

`iso.data`	The *dataframe* containing the data used for fitting the isoscape model
`split.by`	A *string* indicating the name of the column of `iso.data` used to split the dataset. The function `isofit` will then be called on each of these sub-datasets. The default split the dataset 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: `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 sub-isoscape.

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

GNIPDataDEmonthly <- prepdata(data = GNIPDataDE,
                              month = 1:6,
                              split.by = "month")

dim(GNIPDataDEmonthly)

## We fit the isoscapes

isoscapemodels <- isomultifit(iso.data = GNIPDataDEmonthly)

isoscapemodels
}
```

`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(elevation.raster, isofit, weighting = NULL,
  verbose = interactive())
```

Arguments

`elevation.raster`	The elevation raster (*RasterLayer*) created by `relevate`
`isofit`	The fitted isoscape created by `isofit`
`weighting`	An optional RasterBrick containing the weights #' @return This function returns a *list* of class *isoscape* containing a stack of all 8 raster layers mentioned above (all being of class *RasterLayer*), and the location of the sources as spatial points.

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

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

plot.isoscape for the function plotting the isoscape model

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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 180) {

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

GNIPDataDEmonthly <- prepdata(data = GNIPDataDE,
                              split.by = "month")

dim(GNIPDataDEmonthly)

## We fit the isoscapes:#'
isoscapemodels <- isomultifit(iso.data = GNIPDataDEmonthly,
                              mean.model.fix = list(elev = TRUE, lat.abs = TRUE))

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

## We plot the mean isoscape of the averaging:
plot(x = isoscapes, which = "mean")

## We build the isoscapes for a given month (here January):
isoscape.jan <- isoscape(elevation.raster = ElevRasterDE,
                         isofit = isoscapemodels$multi.fits[["month_1"]])

## We plot the mean isoscape for January:
plot(x = isoscape.jan, 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 outcoume using `plot.isorix`.

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

`rainbow` for information about R colour palettes

`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
my.palette  <- colorRampPalette(c("blue", "green", "red"), bias = 0.7)
par(mfrow = c(1, 1))
pie(1:100, col = my.palette(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")
```

`isoscape` *Predicts the spatial distribution of isotopic values*

Description

This function produces the isoscape, i.e. a spatial prediction (i.e. map) of the distribution of isotopic delta values. Predictions are computed using the fitted isoscape for each raster cell of the elevation raster. All shape files can be exported and loaded into any Geographic Information System (GIS) if needed.

Usage

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

Arguments

`elevation.raster`	The elevation raster (*RasterLayer*) created by `relevate`
`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 elevation 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 `predict` from the package **spaMM** on the fitted isoscape produced by `isofit`.

Let us detail the meaning of `mean`, `mean.predVar` and `mean.respVar`:

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 Matern-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, technically called the best linear unbiased prediction (BLUP), 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 BLUP at a given location. The response variance has a different meaning. It 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 stack of all 8 raster layers mentioned above (all being of class *RasterLayer*), and the location of the sources as spatial points.

See Also

`isofit` 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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

## We fit the models
```

```
GermanFit <- isofit(iso.data = GNIPDataDEagg,
                    mean.model.fix = list(elev = TRUE, lat.abs = TRUE))

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

isoscape

## We build the plots
plot.mean <- plot(x = isoscape, which = "mean", plot = FALSE)

plot.mean.predVar <- plot(x = isoscape, which = "mean.predVar", plot = FALSE)

plot.mean.residVar <- plot(x = isoscape, which = "mean.residVar", plot = FALSE)

plot.mean.respVar <- plot(x = isoscape, which = "mean.respVar", plot = FALSE)

## We display the plots
if(require(lattice)) {
 print(plot.mean, split = c(1, 1, 2, 2), more = TRUE)
 print(plot.mean.predVar,   split = c(2, 1, 2, 2), more = TRUE)
 print(plot.mean.residVar,  split = c(1, 2, 2, 2), more = TRUE)
 print(plot.mean.respVar,   split = c(2, 2, 2, 2), more = FALSE)
}

}
```

`isosim` *Simulate isotopic values*

Description

This function allows for the simulation of isoscapes. Both partial or complete (i.e. maps) isoscapes can be simulated.

Usage

```
isosim(simu.data, mean.model.fix.coef = c(intercept = 64, elev = -0.01,
  lat.abs = -2.3, lat.2 = 0, long = 0, long.2 = 0),
  disp.model.fix.coef = c(intercept = 5.8, elev = 0, lat.abs = 0, lat.2 = 0,
  long = 0, long.2 = 0), mean.model.matern.coef = c(nu = 0.35, rho = 5e-05,
  lambda = 899), disp.model.matern.coef = c(nu = 0.32, rho = 1.5e-05, lambda =
  5), mean.model.uncorr.coef = c(nugget = 0, lambda = 0),
  disp.model.uncorr.coef = c(nugget = 0, lambda = 0), dist.method = "Earth",
  seed = NULL, save.dataframe = FALSE, verbose = interactive())
```

Arguments

`simu.data`	A *data.frame* containing the covariates needed for the simulation, or alternatively an elevation raster of class *RasterLayer*
`mean.model.fix.coef`	A *vector* of coefficients for fixed-effects
`disp.model.fix.coef`	A *vector* of coefficients for fixed-effects
`mean.model.matern.coef`	A *vector* of coefficients for the spatial random effect
`disp.model.matern.coef`	A *vector* of coefficients for the spatial random effect
`mean.model.uncorr.coef`	A *vector* of coefficients for the uncorrelated random effect
`disp.model.uncorr.coef`	A *vector* of coefficients for the uncorrelated random effect
`dist.method`	A *string* indicating the distance method
`seed`	An *integer* used to set the seed of the random generator
`save.dataframe`	A *logical* indicating whether the detailed *data.frame* containing the simulated data should be saved
`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 takes as inputs the values for all covariates matching a series of locations (which can be provided as an elevation raster or as a *data.frame*), as well as the parameters of the isoscape model. The function is not required to fit an isoscape, nor to perform assignments. It is an additional function that can be useful to test the method, and to study the effect of different parameters on isoscapes. We chose default values for parameters inspired to model fits in order to simulate a relatively realistic isoscape. We precised 'relatively' because, in the simulations, the Matern realizations for the mean and the dispersion are drawn independently, which is not the case in nature. Note that extra parameters present in the input lists will not make the function crash but won't be considered during computations either.

Value

This function returns a *list* of class *isoscape* containing a stack of raster and an optional *data.frame*. The stack contains the raster `mean.raster` storing the mean isotopic value, and the raster `disp.raster` storing the residual dispersion variance. The optional *data.frame* contains the simulated data and details of the computation in an object called `data`.

Note

The spatial autocorrelation and the Nugget are computed by the functions `RMwhittle` and `RMnugget`, respectively. These two functions are part of the powerful package **RandomFields**.

See Also

isofit for the function fitting 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. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 60) {

## We simulate data under default settings
simu <- isosim(simu.data = ElevRasterDE,
    save.dataframe = TRUE, seed = 1)

simu

## We build the plots of the outcome using IsoriX
plot.mean.simu <- plot(x = simu, which = "mean")

plot.disp.simu <- plot(x = simu, which = "disp")


## We fit the simulated data by sampling 50 locations

set.seed(123)
newdat <- simu$data[sample(1:nrow(simu$data), 50), ]

isoscapemodel <- isofit(iso.data = newdat,
    mean.model.fix = list(elev = TRUE, lat.abs = TRUE))

isoscape <- isoscape(ElevRasterDE, isoscapemodel)

plot.mean.fitted <- plot(x = isoscape, which = "mean", sources = list(draw = FALSE))

plot.disp.fitted <- plot(x = isoscape, which = "disp", sources = list(draw = FALSE))

## We compare simulated and fitted data visually
if(require(lattice)){
    print(plot.mean.simu, split = c(1, 1, 2, 2), more = TRUE)
    print(plot.disp.simu, split = c(2, 1, 2, 2), more = TRUE)
    print(plot.mean.fitted, split = c(1, 2, 2, 2), more = TRUE)
    print(plot.disp.fitted, split = c(2, 2, 2, 2), more = FALSE)
}

## We compare simulated and fitted data numerically
## Note that differences are expected, as the geographic
##   area is small compared to the scale at which
```

```
##   spatial auto-correlation occurs
round(cbind(
    simulated = c(
        intercept = 64,
        lat.abs = -2.3,
        elev = -0.01,
        nu = 0.35,
        rho = 5e-5,
        rho_div_nu  =  5e-5/0.35,
        lambda.ID = 0,
        lambda.matern = 899,
        intercept.disp = 5.8,
        nu.disp = 3.2e-01,
        rho.disp = 1.5e-05,
        lambda.matern.stationID = 0,
        lambda.matern.disp = 5),
    fitted = c(
        intercept = isoscapemodel$mean.fit$fixef[1],
        lat.abs = isoscapemodel$mean.fit$fixef[2],
        elev = isoscapemodel$mean.fit$fixef[3],
        nu = isoscapemodel$mean.fit$ranFix$nu,
        rho = isoscapemodel$mean.fit$ranFix$rho,
        rho_div_nu = with(isoscapemodel$mean.fit$ranFix,rho/nu),
        lambda.matern = isoscapemodel$mean.fit$lambda,
        intercept.disp = isoscapemodel$disp.fit$fixef[1],
        nu.disp = isoscapemodel$disp.fit$ranFix$nu,
        rho.disp = isoscapemodel$disp.fit$ranFix$rho,
        lambda.matern.disp = isoscapemodel$disp.fit$lambda)), 4)

}
```

`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?

if(require(raster) & require(rgeos)){
    worldlimit <- as(extent(CountryBorders), "SpatialPolygons")
    proj4string(worldlimit) <- crs(CountryBorders)
    OceanMask <- gDifference(worldlimit, CountryBorders)
    OceanMask

## Uncomment the following to store the file as we did
#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

```
IsoriX.options(...)

IsoriX.getOption(x)
```

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 (for future version) **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 invisibly in an object called `.IsoriX.data$options`

Examples

```
IsoriX.options()
IsoriX.getOption("example_maxtime")

## Not run:
IsoriX.options(example_maxtime = 30)
IsoriX.options()

## End(Not run)
```

`plots` *Plotting functions for IsoriX*

Description

These functions plot objects created by **IsoriX**.

Usage

```
## S3 method for class 'isoscape'
plot(x, which = "mean", 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, ...)

## S3 method for class 'isorix'
plot(x, who = "group", what = "pv", cutoff = list(draw =
  TRUE, level = 0.05, col = "#909090"), sources = list(draw = TRUE, cex = 0.5,
  pch = 2, lwd = 1, col = "red"), calib = list(draw = TRUE, cex = 0.5, pch =
  4, lwd = 1, col = "blue"), 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, ...)

## S3 method for class 'isofit'
plot(x, cex.scale = 0.2, ...)

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

Arguments

x	The return object of an isofit, isoscape, calibfit, or isofind call
which	A *string* indicating the name of the raster to be plotted (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
...	Additional arguments (not in use)
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
what	A *string* indicating the name of the raster to be plotted (should remain "pv" if who = "group", otherwise could be "stat", "stat.var", or "pv")
cutoff	A *list* containing information for the display of the region outside the prediction interval (see details)
calib	A *list* containing information for the display of the location of the calibration sampling area (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 scalling factor for the points in the plots

Details

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 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, the user can choose between plotting the mean isotopic value (which = "mean") or the dispersion (which = "disp").

When called upon an object of class *isorix*, 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 vignette "Workflow" for examples).

The arguments `cutoff`, `sources`, `calib`, `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 sampling area (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 maks) 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 usefull 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 ploting isoscape and assignments, respectivelly.

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

`IsoriX` for the complete work-flow

Examples

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

---

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 elevation raster. This function is designed to be used with `isomultiscape`.

Usage

```
prepcipitate(path = NULL, elevation.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
elevation.raster	A *raster* containing the elevation 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

relevate to prepare the elevation raster

Examples

```
## The following example takes some time and download heavy data.
## It will therefore not be run unless you type:
## example(prepcipitate, run.dontrun = TRUE)

## Not run:

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

GermanFit <- isofit(iso.data = GNIPDataDEagg,
                    mean.model.fix = list(elev = TRUE, lat.abs = TRUE))

elevation.raster <- relevate(
    elevation.raster = ElevRasterDE,
    isofit = GermanFit,
    aggregation.factor = 0)

getprecip(path = "~/Desktop/")

precipitation.brick <- prepcipitate(path = "~/Desktop/",
                                    elevation.raster = ElevRaster
                                    )

 if (require(rasterVis)) {
   levelplot(precipitation.brick)
 }


## End(Not run)
```

prepdata *Filter the dataset to create an isoscape*

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

```
prepdata(data, month = 1:12, year, long.min, long.max, lat.min, lat.max,
  split.by = NULL, prop.random = 0, random.level = "station",
  col.isoscape.value = "isoscape.value", col.stationID = "stationID",
  col.lat = "lat", col.long = "long", col.elev = "elev",
  col.month = "month", col.year = "year")
```

Arguments

`data`	A *dataframe* containing original isotopic measurements
`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. If not provided, -180 will be considered.
`long.max`	A *numeric* indicating the maximal longitude to select from. Should be a number between -180 and 180. If not provided, 180 will be considered.
`lat.min`	A *numeric* indicating the minimum latitude to select from. Should be a number between -90 and 90. If not provided, -90 will be considered.
`lat.max`	A *numeric* indicating the maximal latitude to select from. Should be a number between -90 and 90. If not provided, 90 will be considered.
`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 "station" (default), which indicates that observations are randomly drawn at the level of sampling locations.

col.isoscape.value
: A *string* indicating the column containing the isotopic measurements

col.stationID
: 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. 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.

See Also

`IsoriX` for the complete workflow

Examples

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

head(GNIPDataDEagg)

## Create a processed dataset for Germany per month
GNIPDataDEmonthly <-prepdata(data = GNIPDataDE,
                             split.by = "month")
```

```
head(GNIPDataDEmonthly)

## Create a processed dataset for Germany per year
GNIPDataDEyearly <- prepdata(data = GNIPDataDE,
                             split.by = "year")

head(GNIPDataDEyearly)

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

head(GNIPDataDEwarm)


## Create a dataset with 90% of obs
GNIPDataDE90pct <- prepdata(data = GNIPDataDE,
                            prop.random = 0.9,
                            random.level = "obs")

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

## Create a dataset with half the weather stations
GNIPDataDE50pctStations <- prepdata(data = GNIPDataDE,
                                    prop.random = 0.5,
                                    random.level = "station")

lapply(GNIPDataDE50pctStations, head)


## Create a dataset with half the weather stations split per month
GNIPDataDE50pctStationsMonthly <- prepdata(data = GNIPDataDE,
                                           split.by = "month",
                                           prop.random = 0.5,
                                           random.level = "station")

lapply(GNIPDataDE50pctStationsMonthly, head)
```

`relevate` *Prepare the elevation raster*

Description

This function prepares the elevation raster for the follow-up analyses. The size and extent of the elevation raster defines the resolution at which the isoscape and the origin assignment are defined.

Usage

```
relevate(elevation.raster, isofit = NULL, margin_pct = 5,
  aggregation.factor = 0L, aggregation.fun = mean, manual.crop = NULL,
  verbose = interactive())
```

Arguments

`elevation.raster`	The elevation raster (*RasterLayer*)
`isofit`	The fitted isoscape model returned by the function `isofit`
`margin_pct`	The percentage representing by how much the space should extend outside the range of the coordinates of the weather stations (default = 5).
`aggregation.factor`	The number of neighbouring cells (*integer*) to merge during aggregation
`aggregation.fun`	The *function* used to aggregate cells
`manual.crop`	A vector of four coordinates (*numeric*) for manual cropping, e.g. the spatial extent
`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 an elevation 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 elevation raster accordingly. Alternatively, `manual.crop` allows you to crop the elevation 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 `aggregate` and `crop` from the package **raster**. It thus share the limitations of these functions. In particular, `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 center 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 fine-tuned elevation raster of class *RasterLayer*

Note

Aggregating the raster may lead to different results for the assignment, because the elevation 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

IsoriX for the complete workflow

Examples

```
## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 30) {

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

GermanFit <- isofit(iso.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
elevation.raster1 <- relevate(
    elevation.raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 0,
    aggregation.factor = 0)

elevation.raster2 <- relevate(
    elevation.raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 5,
    aggregation.factor = 5)

elevation.raster3 <- relevate(
    elevation.raster = ElevRasterDE,
    isofit = GermanFit,
    margin_pct = 10,
    aggregation.factor = 5, aggregation.fun = max)

## We build the plots of the outcome of the 3 different aggregation schemes
if(require(rasterVis)) {
    plot.aggregation1 <- levelplot(elevation.raster1,
            margin = FALSE, main = "Original small raster") +
```

```
        layer(sp.polygons(CountryBorders)) +
        layer(sp.polygons(OceanMask, fill = "blue"))
    plot.aggregation2 <- levelplot(elevation.raster2,
            margin = FALSE, main = "Small raster aggregated (by mean)") +
        layer(sp.polygons(CountryBorders)) +
        layer(sp.polygons(OceanMask, fill = "blue"))
    plot.aggregation3 <- levelplot(elevation.raster3,
            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(plot.aggregation1, split = c(1, 1, 1, 3), more = TRUE)
    print(plot.aggregation2, split = c(1, 2, 1, 3), more = TRUE)
    print(plot.aggregation3, split = c(1, 3, 1, 3))
}
}

#' ## The examples below will only be run if sufficient time is allowed
## You can change that by typing e.g. IsoriX.options(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(IsoriX.getOption("example_maxtime") > 10) {

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

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

## We crop it around the pacific
pacificA <- relevate(empty.raster, manual.crop = c(110, -70, -90, 90))
extent(pacificA) # note that the extent has changed!

## We plot (note the use of the function shift()!)
if(require(rasterVis)) {
  levelplot(pacificA, margin = FALSE, colorkey = FALSE, col = "blue")+
    layer(sp.polygons(CountryBorders, fill = "black"))+
    layer(sp.polygons(shift(CountryBorders, x = 360), fill = "black"))
  }

}
```

Index