Package ‘vetools’

February 20, 2015

Encoding UTF-8

Depends R (>= 2.10), sp

Imports stringr, tis, lubridate, maptools, plyr, xts, scales

Maintainer Andrew Sajo-Castelli <asajo@usb.ve>

License GPL

Title Tools for Venezuelan Environmental Data

LazyData false

Type Package

Description Integrated data management library that offers a variety of tools concerning the loading and manipulation of environmental data available from different Venezuelan governmental sources. Facilities are provided to plot temporal and spatial data as well as understand the health of a collection of meteorological data.

Version 1.3-28

Revision $Rev: 4 $

BugReports https://github.com/talassio/vetools/issues

Author Andrew Sajo-Castelli [aut, cre], Desiree Villalta [ctb], Lelys Bravo [ctb]

NeedsCompilation no

Repository CRAN

Date 2013-08-01 14:10:01

Date/Publication 2014-10-15 08:47:53

R topics documented:

`vetools-package` *Some tools for Venezuelan environmental data*

Description

This toolset provides a bundle of functions to handle and unify the diverse data formats of distinct government agencies and military bodies: Ministerio del Ambiente, Marina Venezolana, etc. It also provides all necessary tools to load these data sets. In order to standarize the structure of the data provided and/or processed, a **vetools** Catalog Convention is presented.

Details

Package:	vetools
Type:	Package
Version:	1.x series
Initial Release Date:	2013-08-01
License: GPL	

Input functions:
`read.HIDROX`
`read.MINAMB`
`read.MARN`

Preprocessing functions:
`disaggregate.ts`
`disaggregate.MARN`
`complete.series`
`fill.small.missing`

EST family functions:
`est.cut`
`est.fill`
`est.rm`
`est.union`
`est.sort`

Descriptive functions:
`panorama`
`panomapa`
`summary.Catalog`
`print.Catalog`
`plot.Catalog`

SHAPE family functions:
`get.shape.state`
`get.shape.venezuela`
`get.shape.range`
`get.Grid.size`

Class `Catalog`
`Catalog`
`is.Catalog`
`as.Catalog`

For a complete list of functions, use `library(help = "vetools")`.

Convention Sheet:
Catalog Convention White Sheet

Datasets:
CuencaCaroni
Vargas
Vargas2

Author(s)

A. M. Sajo-Castelli. Centro de Estadística y Matemática Aplicada (CEsMA). Universidad Simón Bolívar. Venezuela.

Contributors: D. E. Villalta, L. Bravo. CEsMA, Universidad Simón Bolívar. Venezuela.

R. Ramírez. Parque Tecnológico Sartenejas, Universidad Simón Bolívar. Venezuela.

Maintainer: A. M. Sajo-Castelli <asajo@usb.ve>

References

L. Bravo, S. Abad, I. Llatas, A. Salcedo, L. Delgado, S. Ramos, K. Cordova. Hidrox: Repositorio de Datos Hidroclimáticos para la Gestión de Riesgos Epidemiológicos y Ambientales. 2012. ISBN:987-9-8012596-2-6.

`Catalog` *Collection of class* `Catalog`

Description

Constructs or tests for collections

Usage

```
Catalog(catalog, data, ...)
is.Catalog(x, ignore.class = FALSE)
as.Catalog(x)
```

Arguments

`catalog`	A list of exactly ten elements and zero or more optional elements. See details in Catalog Convention
`data`	A list of any kind of the same length as `catalog`
`x`	Possibly a collection of class `Catalog`
`ignore.class`	Test if `x` is a collection ignoring its class
`...`	zero or more R objects to include in the construction of the collection

Value

`Catalog` and `as.Catalog` return a collection of class `Catalog`, fully qualified and following the Catalog Convention. If "`...`" is not missing then all its objects are included in the collection.

`is.Catalog` returns a logical value.

Author(s)

A.M. Sajo-Castelli

See Also

Catalog Convention, **vetools**.

`CatalogConvention` **vetools** *Catalog Convention White Sheet (Revision 3)*

Description

This white sheet describes the Catalog Convention of **vetools** package.

Catalog Convention White Sheet (Revision 3)
The purpose of this convention is to standarize the data structure used to store the environmental data and associated meta-data. All data-sets provided and processed by the **vetools** package that follow this convention are of class `"Catalog"` and are referred as `"Collection"`s. Each collection provides the following structure of class `list`:

- A list of class `"list"` with name `"catalog"` where each element is composed of exactly ten standard elements and zero or more optional/extra elements. The required standard elements are:

  `Name` Station's name, class `"character"`

  `Altitude` Stations's altitude in metres. Some data sources lack this information an is taken to be NA

  `Latitude` Latitude in degrees

  `Longitude` Longitude in degrees, some data sources use West direction

  `Measure.code` Measured variable code, sometimes indicates MKS unit

  `Measure.unit` Measured data variable description

  `Install` Date of station's installation

  `Start` Date of start of operation of the station. Some data sources lack this information and is taken to be the same as `Instalacion`

  `State` The state of the country to which the station belongs, some sources lack this information and can be taken to be NA

  `Avble.yrs` A vector containing the years on which the station allegedly operated. Some source lack this information and is extracted from the measured variable and reflects those years that have at least one measurement.
- One or more lists of measurement data variables, generally of class `"ts"`. It is required that at least one be present under the name of `"data"`.

This pair of lists (`"catalog"` and `"data"`) form the `collection`. The two (or more) items are always of class `"list"` and are in direct correspondence, *i.e.* item *n* of the `"catalog"` corresponds to the measured variable item *n* in `"data"`.

For example, suppose *collection* is a collection of 30 stations, then `collection$catalog[[4]]` element describes the measurement of `collection$data[[4]]`.

Functions
Functions provided to read data sources are

`read.HIDROX` imports Argus 1.0 data source files

`read.MARN` imports M.A.R.N. files

read.MINAMB imports EDELCA source files

These functions all return a list class "Catalog". Generally are parsed as follows:

```
file = system.file('tests/test-HIDROX.csv',package='vetools')
collection <- read.HIDROX(file)
names(collection$catalog[[1]])
summary(collection)
print(collection)
plot(collection)
```

Author(s)

A.M. Sajo Castelli

See Also

vetools, summary.Catalog, read.HIDROX, read.MARN, read.MINAMB.

Examples

```
## Not run: # This collection has only one station
Collection <- read.MARN(system.file("tests/test-MARN.dat", package="vetools"))
summary(Collection)
plot(Collection$data)
# This collection has many stations
Collection.H <- read.HIDROX(system.file("tests/test-HIDROX.csv", package="vetools"))
summary(Collection.H)
plot(Collection.H$data[[4]])
## End(Not run)
```

---

complete.series *Complete relatively large holes in data-sets*

Description

This functions completes relatively large holes in monthly time-series objects.

Usage

```
complete.series(collection, model, k.ubic = NA, centers = 3, nstart = 3,
weps = 0.05, MAX.ITER = 100, AEM.debug = T)
```

Arguments

`collection` A list of class `Catalog` that contains the objects to complete.

`model` A list of fixed-effects models related to `collection$data`.

`k.ubic` A data.frame of exactly one member `k.ubic$cluster` which is a scalar vector of length equal to `collection$data` and specifying to which cluster belongs to each element of the list `collection$data`.

`centers` If `k.ubic` is unavailable, this sets the quantity of clusters to build.

`nstart` If `k.ubic` is unavailable, then this parametre sets the initial quantity of center with which to start the k-means algorithm.

`weps` Tolerance for the E-M Algorithm.

`MAX.ITER` Maximum number of iterations for the E-M Algorithm.

`AEM.debug` Logical flag indicating if verbosity is required.

Details

The main idea behind this functions is to complete the time-series of the list by first clustering similar stations and then applying to each cluster the E-M Algorithm in order to complete the series. The E-M Algorithms is an iterative method that in each iteration performs two tasks: fist estimates the expected values and then maximizes their likelyhood. This goes on util some stopping criteria is meat.

Value

Returns a completed version of `collection` (`collection$data`).

Note

The current implementation is known to have problems. The iterative proccess not always converges. It is also known that the E-M has been surpassed by other methods and it would be desireble to replace it.

Author(s)

A. Jhan, fixed-up by A.M. Sajo-Castelli.

See Also

`fill.small.missing`

Examples

```
## Not run:
for (k in 1:15) {
        fit[[k]] = lm(collection$data[[k]] ~ ZZ - 1, singular.ok=T, na.action=na.omit)
}
collection.completed = complete.series(collection, fit)
## End(Not run)
```

CuencaCaroni *NA*

Description

Monthly precipitation values for meteorological stations located in the Cuenca del Caroní, Bolívar state, Venezuela.

Data set of precipitation for 91 meteorological stations located in the Bolívar state, Venezuela. Data set follows **vetools** Catalog Convention. See Catalog Convention.

Usage

```
data(CuencaCaroni)
```

Format

This data set contains a `collection` of two members of class `list`, each of 91 elements:

CuencaCaroni$catalog list of each station's meta data. Follows **vetools** Catalog Convention. To see meta data `summary(CuencaCaroni)`.

CuencaCaroni$data list containing each station's time-series of class `"ts"`.

Details

This data set provides monthly precipitation for 91 meteorological stations located in the Cuenca del Caroní region of the Bolívar state of Venezuela. The region is delimited between -63.88083, -60.60722 degrees and 3.895833 and 8.333333 degrees (North). Time-series for statios vary between 1949 and 2011.

The data set was imported into R using `read.MINAMB` function.

Source

Ministerio de Industrías básicas y Minería. CVG, EDELCA (Electrificación del Caroní) C. A. Gerencia de Gestion Ambiental. (`http://www.cvg.gob.ve/espanol/cvgedelca.html`).

See Also

Vargas, Catalog Convention, `read.MINAMB`.

Examples

```
## Not run:
data(CuencaCaroni)
summary(CuencaCaroni)
plot(CuencaCaroni$data[[2]])
start(CuencaCaroni$data[[80]])
end(CuencaCaroni$data[[80]])
frequency(CuencaCaroni$data[[80]])
```

```
cat(cc.cat[[1]]$Nombre)
## End(Not run)
```

`diasdelmes` *Sum of days*

Description

This function returns the number of days in a sequence of months. Takes into account leap years.

Usage

```
diasdelmes(y, meses)
```

Arguments

`y`	integer, year from which to reference the months specified by `meses`
`meses`	a vector of length greater or equal to 1 specifying the months to sum

Value

returns the number of days specified in the months `meses` of the year `y`.

Author(s)

A.M. Sajo-Castelli

See Also

vetools, `diffmonths`, `tssum`, `m12`, `time2ym`, `ym2time`, `xts2ts`.

`diffmonths` *Difference between two time-series*

Description

Calculates the difference in months of two time-series objects.

Usage

```
diffmonths(date1, date2)
```

Arguments

`date1`	objects of class `"ts"`
`date2`	objects of class `"ts"`

Value

Returns the number of months between the start of two class "ts" objects

Author(s)

A.M. Sajo-Castelli

See Also

vetools, `diasdelmes`, `tssum`, `m12`, `time2ym`, `ym2time`, `xts2ts`.

`disaggregate.MARN` *Disaggregates a time-series using a reference (surrogate) serie*

Description

For a brief introduction on disaggregation see `disaggregate.ts`. In order to disaggregate, a distribution of the asterisks is required. In this implementation, the distribution is estimated using a surrogate serie. In general terms the surrogate serie is very carefully drafted.

Usage

```
disaggregate.MARN(stream = NULL, reference = NULL,
na.action = "error", asterisk = -9999, date.eps = 0.004,
float.eps = 1e-04, return.incomplete = TRUE)
```

Arguments

`stream` An aggregated `ts` object.

`reference` A reference or surrogate `ts` object.

`na.action` Action to take if the sample distribution has NAs present. Can be `"mean"` (`"average"`, `"warning"`, `"continue"`) or `"error"`. In the first case the sampled distribution is the average. On the second, the process is stoped, if *return.incomplete* is true then the progress of disaggregation is returned.

`asterisk` Scalar denoting values to complete.

`date.eps` Tolerance in date/time matching.

`float.eps` Smallest mass to distribute along the aggregated elements.

`return.incomplete` Boolean value to interrupt the process and return the incompletely disaggregated series. See details.

Details

The parametre *return.incomplete* is very usefull to build surrogate series, as follows. Say there is a list of 15 aggregated series, then in order to build a reference series for all of them, the following hueristic can help. Suppose these series are ordered by least NAs and asterisks present.

```
reference <- pr[[1]]
k = 1
restart:
for ( station in 1:k ) {
        reference <- desagregate.MARN(pr[[k]],
        reference, return.incomplete=TRUE)
}
if ( reference is not yet fully desagregated ) { k <- k + 1 }
goto restart
```

The main feature of this procedure is that it always tries to use the best serie first then the second best, etc. It may not complete the task if the sample distribution contains NAs for *all* 15 stations. Under this precarious condition, artificial or external information can be used.

Value

Returns a disaggregated series. If the switch *return.incomplete* is true, then it returns a series that was disaggregated until NAs where found on the sample distribution.

Author(s)

A.M. Sajo-Castelli

See Also

`disaggregate.ts`

`disaggregate.ts` *Desagregates a time-series*

Description

This function disaggregates pilled-up data. Agregation points are denoted by the scalar following one or more *asterisks*. The job of this function is to distribute the mass accumulated in the first non asterisk measurment between the previous points marked with asterisks.

Usage

```
disaggregate.ts(x, ...)
```

Arguments

`x` An aggregated `ts` object.

`...` defaults to `asterisk` = -9999 and `fun` = `median`.

Where `asterisk` is a scalar that denotes values to complete, defaults to -9999, and `fun` is the name of the function to use to build the sampled distributions. Defaults to `median`.

Details

Say a time-series is of weekly frequency and is

```
Week   Mon  Tue  Wen  Thu  Fri  Sat  Sun
...
k     14.5 19.0 25.5 25.2 19.8 12.3 13.7
k+1   NA   18.7    *    *    * 83.2 14.2
...
```

The task is to distribute 83.2 between Wen and Sat of week k+1 using the sampled distribution of Wen, Thu, Fri and Sat of *all* available weeks. Sometime this is not possible and in this case all days get the equal mass distribution.

Value

Returns a disaggregated `ts` object.

Author(s)

A.M. Sajo-Castelli

See Also

`disaggregate.MARN`

`est.cut` *Crops a list of time-series*

Description

Given a time window, this function crops all the stations in a collection of data/catalog pair. If a given station starts after the end date (end) or ends before the inital date (`start`), it is removed from the catalog.

Usage

```
est.cut(collection, start = c(1960, 1), end = c(2020, 12))
```

Arguments

`collection` A list of class `Catalog`.

`start` The start of the window to crop. A vector of two elements (year, month).

`end` The end of the windows to crop.

Value

Returns a list with the updated (cropped) `collection`. Note that the information regarding the stations meta-data is NOT modified. (!)

Author(s)

A.M. Sajo-Castelli.

See Also

The other `est.*` family members: `est.rm`, `est.fill`, `est.sort`, `est.union`.

`est.fill` *For each member of a collection call the function* `fill.small.missing`

Description

Given a list of class `Catalog`, it completes each station's `data` in such a manner that *all* stations either start or end at the same time. Missing values for each station are estimated by calling the function `fill.small.missing`.

Usage

```
est.fill(collection, cut = c(1968, 3), at.start = T)
```

Arguments

`collection` A list of class `Catalog` with member `data` of class `ts` with frequency 365.25.

`cut` A vector designating the (year, month) that all stations will start.

`at.start` Boolean value indicating whether the stations data should be completed from the start or the end.

Details

The purpose of this function is to have a common start and/or end dates for a given collection of stations. Suppose there are three stations in a collection, with span

```
range(col$data[[1]]) -> c(1981,4) to c(2013,3)
range(col$data[[2]]) -> c(1981,2) to c(2013,4)
range(col$data[[3]]) -> c(1981,3) to c(2013,5)
```

and would like to have them all start on (1981,2) and end on (2013,5). This funcion can achieve this task.

Value

Returns a list of class Catalog with member data completed.

Author(s)

A.M. Sajo-Castelli

See Also

The other est.* family members: est.rm, est.sort, est.cut, est.union.

---

est.rm *Removes stations from a collection of class* Catalog

Description

Given a list of indexes (list), this function removes stations from a collection of data/catalog pair.

Usage

```
est.rm(collection, list)
```

Arguments

collection	A list of class Catalog.
list	A vector of scalars indicating the stations to remove. The elements of this vector must be between 1 and length(collection$data).

Value

Returns a list of class Catalog with the updated collection.

Note

By specifying a negative list of elements, it is possible to select only those stations:

```
# Remove first 3 stations:
col <- est.rm(collection, list=1:3)
# Select only the first 3 stations:
col <- est.rm(collection, list=-(1:3))
```

Author(s)

A.M. Sajo-Castelli

See Also

The other `est.*` family members: `est.cut`, `est.fill`, `est.sort`, `est.union`.

`est.sort` *Sort a data/catalog pair*

Description

Given a collection of data/catalog pairs, this function orders them by the start time.
Sorts by `start(collection$data[[k]])`, provided that the member `data` is of class `ts`.

Usage

```
est.sort(collection, ascending = T, by.year.only = F)
```

Arguments

`collection`	A list of class `Catalog` objects.
`ascending`	Boolean value indicating wheather it is ordered by earliest or latest starting station.
`by.year.only`	Use only the year to sort instead of year/month. Defaults to FALSE.

Value

Returns a sorted list of class `Catalog`, sorted by `start` date of the objects in `collection$data`.

Author(s)

A.M. Sajo-Castelli

See Also

The other `est.*` family members: `est.rm`, `est.fill`, `est.cut`, `est.union`.

est.union *Unites data from a collection of data/catalog pair*

Description

This function merges a list of ts objects into a single time-series. It does it by taking the mean (meadian, fun) of the common elements for each time.

Usage

```
est.union(collection, fun = mean, return.matrix=FALSE)
```

Arguments

collection	A list of class Catalog.
fun	The function by which to unite the common elements. Defaults to mean.
return.matrix	Returns a matrix where each collumn is a time-series (of each station), synchronized in time.

Value

Returns an enhanced Catalog object with an additional member called union of class ts that contains the union of all stations described in collection. If *return.matrix* is true, then it returns a matrix time stamped where each collumn is a station data.

Author(s)

A.M. Sajo-Castelli

See Also

The other est.* family members: est.rm, est.sort, est.cut, est.fill.

Examples

```
## Not run:
names(collection)
collection = est.union(collection)
names(collection)
plot(collection$union)
abline(h = 250, v = 1997:2000)
## End(Not run)
```

fill.small.missing *Complete daily-frequency time-series*

Description

This routine completes a series of frequency 365.25. Each NA is estimated using the function fun (median) of the same day of all other years (where available).

Usage

```
fill.small.missing(serie, max.len = 3 * 30, func = median)
```

Arguments

serie	A ts object.
max.len	Largest gap (in days) to complete using this method. Defaults to 3 months.
func	Function to use in order to estimate an NA. Defaults to median.

Details

This function completes *small* gaps of NA, it is not intended to complete long periods of NAs. If required to complete large sets of NAs, see complete.series.

Value

Returns a ts object with gaps of NA greater than *max.len* days (if present).

Note

This function is verbose, some information of its running tasks is presented.

Author(s)

A.M. Sajo-Castelli, Desiree Villalta.

See Also

complete.series

get.Grid.size *Build a grid around an object of class* "SpatialPolygonsDataFrame"

Description

Construct a grid that *contains* the shape (object of class "SpatialPolygonsDataFrame") and is spaced by a given distance.

Usage

```
get.Grid.size(shape, origin.grid, x.res = 0.05, y.res = 0.05, plot = FALSE)
```

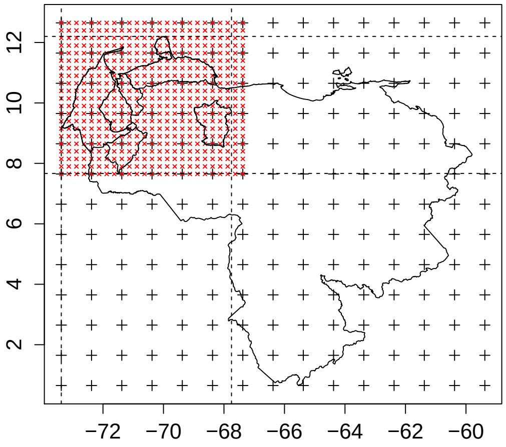
Arguments

shape	Object of class "SpatialPolygonsDataFrame"
origin.grid	External grid constructed by this same function. See Details
x.res	Longitudinal separation in degrees, defaults to 0.05 degrees
y.res	Latitudinal separation in degrees, defaults to 0.05 degrees
plot	Boolean. Shows the constructed grid over the shape

Details

About the `origin.grid` parametre. Say there is the need to work on a nation-wide scale. For this you construct a grid over the whole shapes constituting the country, naming is *extremal or external grid*. Now to study in detail a given state it is recomemded to construct a *smaller* grid covering only state of interest and not the whole nation. Doing this it is necesary to asure that the smaller grid *intersects* the external grid. Providing the parametre commandorigin.grid constructs a small grid over the given shape but that overlaps exactly with the external grid commandorigin.grid. An example that illustrates the above could be:

```
# External grid
VE <- get.shape.state(get.shape.state()$Abb)
External.Grid <- get.Grid.size(VE, plot=F, x.res=1, y.res=1)
# Small grid over a state
NE <- get.shape.state(c("MI","NE"))
Small.Grid <- get.Grid.size(NE, External.Grid, plot=T, x.res=0.25, y.res=0.25)
```



Value

`ncol`	Number of columns of the grid
`nrow`	Number of rows of the grid
`longs`	Longitudinal position for each column of the grid
`lats`	Latitudinal position for each row of the grid
`x.res`	Longitudinal resolution used
`y.res`	Latitudinal resolution used

Author(s)

A.M. Sajo-Castelli

See Also

`get.shape.venezuela`, `get.shape.state`, `get.shape.range`.

Examples

```
## Not run:
## Construct extremal grid for whole country
VE <- get.shape.state(get.shape.state()$Abb)
External.Grid <- get.Grid.size(VE)

## Build grid over Amazona state synchronized with External.Grid
AM <- get.shape.state("AM")
AM.Grid <- get.Grid.size(AM, origin.grid=External.Grid)

## Build grid over Amazona state
AM <- get.shape.state("AM")
```

```
AM.Grid <- get.Grid.size(AM)

## Another example:
VE = get.shape.state(get.shape.state()$Abb)
ZUFACO = get.shape.state(c('ZU','FA','CO'))
Main.grid=get.Grid.size(VE,x.res=1,y.res=1,plot=T)
sub.grid = get.Grid.size(ZUFACO,origin.grid=Main.grid, x.res=0.5,y.res=0.5,plot=TRUE)

## End(Not run)
```

get.shape.range *Get spatial range of an object*

Description

Extracts the longitudes and latitudes of an object of class SpatialPolygonsDataFrame.

Usage

```
get.shape.range(shape)
```

Arguments

shape — object of class SpatialPolygonsDataFrame. Object usually comes from get.shape.venezuela or get.shape.state.

Value

Matrix of one row and four columns, containing the Longitudinal and Latitudinal ranges. Of the form

```
     Long.start Long.end Lat.start Lat.end
[1,]  -73.37749 -59.7991 0.6492503 12.2012
```

Author(s)

A.M. Sajo-Castelli

See Also

get.shape.venezuela, get.shape.state, get.Grid.size.

Examples

```
VE <- get.shape.venezuela()
get.shape.range(VE)
```

get.shape.state *Retrive SHAPE files*

Description

These functions retreive the necesary SHAPE files to display the Venezuelan political border or any combination of states.

Usage

```
get.shape.state(abb, shape.file = "venezuelaestados")
get.shape.venezuela(shape.file = "venezuela")
```

Arguments

abb	a vector of characters containing the two letter abbreviations of the states to load.
shape.file	the base name of the SHAPE file to use.

Details

If the parameter abb is missing, then a data frame is shown and returned containing the states names, abbreviations and SHAPE file IDs.

Value

returns an object of class "SpatialPolygonsDataFrame" that can be plotted using the plot command.

Note

SHAPE files venezuela and "venezuela estados" have a slight size mismatch:

```
> VE <- get.shape.venezuela()
> VS <- get.shape.state(get.shape.state()$Abb)
> get.shape.range(VE)
      Long.start Long.end Lat.start  Lat.end
SHAPE   -73.3774 -59.7991 0.6498817 12.20108
> get.shape.range(VS)
       Long.start Long.end Lat.start  Lat.end
SHAPE   -73.37749 -59.7991 0.6492503 12.2012
```

Author(s)

R. Ramírez. Parque Tecnológico Sartenejas, Universidad Simón Bolívar. Venezuela.

Wrapped in R by A. M. Sajo-Castelli

References

Maps where constructed and exported from ArcGIS 2.x.

See Also

get.shape.range, get.Grid.size.

Examples

```
## Get national boudary SHAPE
VE <- get.shape.venezuela()
## Not run: plot(VE, asp=1, axes=T)

## Get list of all available shapes
get.shape.state()

## Get national and statal boudaries SHAPE
VS <- get.shape.state(get.shape.state()$Abb)
## Not run: plot(VS, col="gray80", asp=1, axes=F)

## Retrieve Zone III states
BOAMDA = get.shape.state(c("BO", "AM", "DA"))
## Not run: plot(BOAMDA, add=T, border="darkred", lwd=2, col="pink")
```

m12 *Smart modulo 12 for time aritmetics*

Description

Calculates which month corresponds to the number *x*, *smart* modulo 12-wise.

Usage

```
m12(x)
```

Arguments

x	integer

Value

Returns an integer 1 through 12 for the corresponding month of a positeve integer *x*, starting with *x*=1 being january. Note that *x*=13 is also january…

Author(s)

A.M. Sajo-Castelli

See Also

vetools, `diffmonths`, `tssum`, `diasdelmes`, `time2ym`, `ym2time`, `xts2ts`.

panorama *Overview of a* `collection` *of stations*

Description

These functions present an overview of the data quality for a `collection` of meteorological stations in a temporal or spatial perspective.

Usage

```
panorama(collection, main, cut, ylab.push.factor = 10, cut.col = "darkred",
    cut.lty = 1, cut.lwd = 2, col = "RoyalBlue", col.ramp = c("red",
        "pink", "blue"), col.line = "gray30", mar = c(5, 4 +
        ylab.push.factor, 3, 2), cex.axis = 0.8, cex.yaxis = 0.7,
    xlab = "Year", color.by.data = FALSE, ...)

panomapa(collection, main, axis = TRUE, xlab = "Long",
    ylab = "Lat", lab.col = "black", bg = NA, map.bg = NA,
    map.col = "black", col.ramp = c("Green3", "darkorange1",
    "red"), arrow.cex = 4.5, arrow.plot = TRUE,
    pt.col = rgb(0, 0, 0, 0.75), pt.cex = 4.5, pt.pch = 21,
    leg.pt.bg = pt.bg, leg.bg = NA, leg.title = "Lengevity\n(years)",
    leg.offset = c(0, 0), leg.y.intersp = 1.75)
```

Arguments

`arrow.cex`	Magnification passed to `arrow.plot`, defaults to 4.5
`arrow.plot`	Logical flag to indicate if to call `arrow.plot`, defaults to TRUE.
`axis`	Logical flag to indicate if to plot the axes, defaults to TRUE
`bg`	Backgrund color for the map, defaults to NA
`cex.axis`	Magnification for axis, defaults to 0.8
`cex.yaxis`	Magnification for y-axis, defaults to 0.8 = 0.7
`col`	`col` from `par`, defaults to "RoyalBlue"
`col.line`	Color for lines, defaults to "gray30"
`col.ramp`	Color for the color ramp, defaults to `c("red", "pink", "blue")` for `panorama` and to `c("Green3", "darkorange1", "red")` for `panomapa`
`color.by.data`	Logical flag to use `collection$data` to color the plotted boxes. This implies that all elements of `data` are between zero and one. Defaults to FALSE.
`collection`	An collection of stations. Object of class `Catalog`
`cut`	A concatenation of dates for which to trace a vertical line

`cut.col` Color to the `cut` line(s), defaults to "darkred". Can be a list

`cut.lty` Line type for the `cut` line(s), defaults to 1. Can be a list

`cut.lwd` Line width for the `cut` line(s), defaults to 2. Can be a list

`lab.col` Color for the labels, defaults to "black"

`leg.bg` Legend box Backgrund color, defaults to NA

`leg.offset` Legend offset, defaults to `c(0, 0)`

`leg.pt.bg` Legend points background color, defaults to `pt.bg`

`leg.title` Legend title, defaults to "Lengevity\n(years)"

`leg.y.intersp` Legend y interspace, is passed to `legend` and defaults to 1.75

`main` Main title

`map.bg` Map background color, defaults to NA

`map.col` map lines color, defaults to "black"

`mar` `par()$mar`, defaults to `c(5, 4 + ylab.push.factor, 3, 2)`

`pt.cex` Points magnification in map, defaults to 4.5

`pt.col` Points color in map, defaults to `rgb(0, 0, 0, 0.75)`

`pt.pch` Points pch in map, defaults to 21

`xlab` for `panorama` defaults to "Year" and for `panomapa` to "Long".

`ylab` y-axes label, defaults to "Lat"

`ylab.push.factor`
Factor in which to push the labels in `panorama`, defaults to 10

`...` Any valid parametres for `par()`

Value

These functions do not return anything.

Author(s)

A.M. Sajo-Castelli

See Also

vetools, Catalog Convention, summary.

Examples

```
## Not run:
panorama(collection)
collection
panomapa(collection)
plot(collection)
## End(Not run)
```

plotArrow *Plots a neat North arrow*

Description

Simple and configurable alternative to draw a "North Arrow" on maps.

Usage

```
plotArrow(shape="",
        pos = 1,
        offset.arrow = c(0, 0),
        north.lwd = par()$lwd+2,
        north.col = par()$col,
        ...)
```

Arguments

`shape`	The shape file used to estimate the x and y coordinates on where to plot the arrow's polygons
`pos`	Where to position the arrow: 1 SW, 2 SE, 3 NE, 4 NW.
`offset.arrow`	Offset pair (x.offset, y.offset).
`north.lwd`	Line width for the North lines.
`north.col`	Color to apply to the North lines
`...`	With `...`, it is possible to specify the color and thickness of the arrow via the `col` and `lwd` parameters. Overall magnification is controlled by `cex`.

Note

This implementation should support adding the scale bar.

Author(s)

A.M. Sajo-Castelli

See Also

`plotLayers`

`plotLayers` *Plot simultaneously one or more layers of information*

Description

Plots several layer of information, overlaying different kind of information. This funtion make it easy to plot several shapefiles/data pair information over one single plot.

Usage

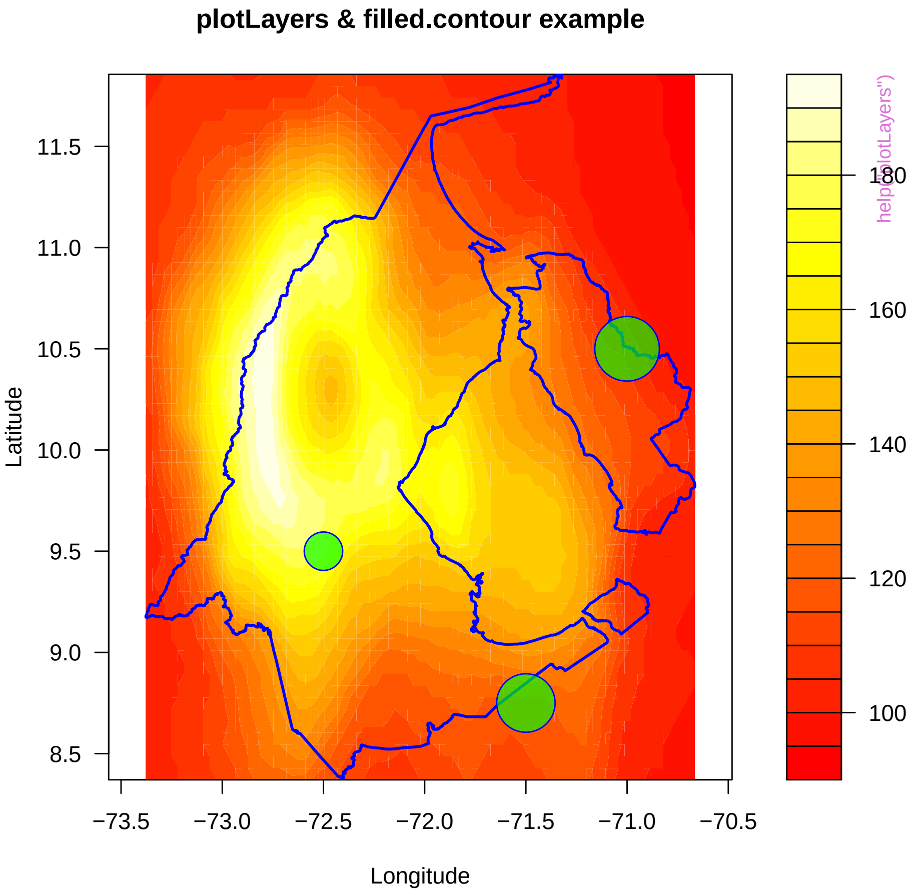
```
plotLayers(...)
```

Arguments

`...`	a list of lists, where each element of the list describes a layer of information. The list must contain a `FUN` member that indicates which function will be used to plot, generally `FUN` is `plot`, `text`, `points`, etc. The rest of the list describes the plotting attributes for each layer. See Examples.

Details

This function can also be embeded into `filled.contour` function. Example two produces the following graphic.



Value

Function does not return any value.

Author(s)

A.M. Sajo-Castelli

See Also

vetools

Examples

```
library(maptools)
library(vetools)

# Example 1 ####
ZU <- get.shape.state("ZU")
```

```
border <- list(FUN = plot, ZU, asp = 1, lwd = 2,
               border = "blue", col = NA, add = TRUE)
r <- get.shape.range(ZU)
x <- seq(r[1], r[2], length.out = nrow(volcano))
y <- seq(r[3], r[4], length.out = ncol(volcano))
image(x, y, volcano, col = heat.colors(100),
      axes = FALSE, xlab = NA, ylab = NA, asp = 1)
plotLayers(border)
plotArrow(ZU, cex = 0.666, offset.arrow = c(0.1, 0))
title(main = "Raster image combined with plotLayers")

# Example 2 ####
long=c(-72.5, -71.5, -71.0); lat=c(9.5, 8.75, 10.5);
mm = 1.5 * c(2.5, 3.8, 4.2)
data <- list(FUN = points, x = long, y = lat, pch = 21,
             bg = rgb(0, 1, 0, 0.666), col = "blue",
             cex = mm)
filled.contour(x, y, volcano, xlab = "Longitude",
               ylab = "Latitude", asp = 1,
               color.palette = heat.colors,
               plot.axis = { plotLayers(border, data) },
               main = "plotLayers & filled.contour example")

# Example 3 ####
pts <- cbind(r[1] + 2 * runif(10), r[3] + 3 * runif(10))
sts <- runif(10)
stations <- list(FUN = plot, x = pts[, 1], y = pts[, 2],
                 asp = 1, pch = 21, col = rgb(sts, 0, 0),
                 bg = 'white', cex = 2, lwd = 2,
                 xlim = r[1:2], ylim = r[3:4], axes = FALSE,
                 xlab = NA, ylab = NA)
labs <- list(FUN=text, x=pts[,1], y=pts[,2], labels=1:10,
             cex=0.7)
type = 1 + round(2 * sts)
LABELS = c('optimal', 'normal', 'critical')
status <- list(FUN = text, x = pts[, 1], y = pts[, 2],
               labels = LABELS[type], cex = 0.7,
               pos = 4, col = rgb(sts, 0, 0))
arrow <- list(FUN = plotArrow, shape = ZU, cex = 0.7)
plotLayers(stations, border, labs, status, arrow)
title(main = "plotLayers example", sub = "Zulia state")
```

read.HIDROX *Load environmental data from governmental sources*

Description

This sheet describes the functions to load environmental data from the formats used by **MARN** The Ministerio del Ambiente y Recursos Naturales.

MINAMB The Ministerio del Ambiente.
HIDROX The Argus data repository, see the references.

Each Ministry used a different data format to store the measured variables. For each available data, a read function is taylored.

Usage

```
read.HIDROX(file, state = NA, altitudes = NA, serial = NA, unit = NA)
read.MARN(file)
read.MINAMB(file, state = NA, YSPLIT = 20)
```

Arguments

file String containing the path to the file to load.

state A two letter character string identifying the state, see get.shape.state(NA) for a complete list.

altitudes A list containing information relative to the elevation of each station in the file.

serial A list containing information relative to the serial of each station in the file.

unit A character string identifying the unit of the measured data, e.g. mm/month.

YSPLIT This variable indicates from which decade (1900+YSPLIT) to consider between the 20th and 21st centuries.

Details

To explore the each data format, the package ships three test files.
See the folder system.file("tests",package="vetools").

Value

Returns a list of class Catalog with exactly two members, see Catalog Convention.

catalog The catalog, a list of each stations meta data.

data The data related to the catalog, a list of objects ts.

Author(s)

A.M. Sajo-Castelli

References

L. Bravo, S. Abad, I. Llatas, A. Salcedo, L. Delgado, S. Ramos, K. Cordova. Hidrox: Repositorio de Datos Hidroclimáticos para la Gestión de Riesgos Epidemiológicos y Ambientales. 2012. ISBN:987-9-8012596-2-6.

See Also

Catalog Convention.

Examples

```
## Not run:
collection.ZU = read.HIDROX('repo_est_ZU.csv', state="ZU", unit="Prec [mm/month]"))
summary(collection.ZU)
collection.ZU

## End(Not run)
```

---

summary.Catalog *Shows a summary, a panoramic overview in temporal or spatial fashion for a given* collection *of data/catalog pairs*

Description

Given a list in the Catalog Convention format, these functions print or plot a summary of the stations data and meta-data.

Usage

```
## S3 method for class 'Catalog'
summary(object, ...)
## S3 method for class 'Catalog'
plot(x, ...)
## S3 method for class 'Catalog'
print(x, ...)
```

Arguments

object	An object of class catalogo.
x	An object of class catalogo.
...	See panorama and panomapa.

Note

The method print calls the function panorama and the method plot calls panomapa.

Author(s)

A.M. Sajo-Castelli

See Also

CatalogConvention

Examples

```
## Not run:
collection = read.HIDROX('test-HIDROX.csv')
summary(collection)
print(collection)
plot(collection)
## End(Not run)
```

---

time2ym *Time related conversion functions*

Description

These functions convert between class "Date" and c(year, month) dates.

Usage

```
time2ym(d)
ym2time(e)
```

Arguments

d	object of class "Date" that can be converted to c(year, month)
e	object of class "ts"

Value

time2ym returns a vector of length 2 specifying year and month corresponding to a date given, compatible with commands start and end for objects of class "ts"

ym2time returns a class "Date" object determined by the specified year and month

Author(s)

A.M. Sajo-Castelli

See Also

vetools, diffmonths, tssum, diasdelmes, xts2ts.

`tssum` *The `sum` for time-series objects*

Description

This function is time related that helps manipulate time-series.

Usage

```
tssum(series, months = 1:12, max.na.fraction = 0.3, safe.check = FALSE)
```

Arguments

`series` a class `"ts"` object

`months` a vector of length 1 to 12 specifying the months to sum

`max.na.fraction` fraction of NAs to admit before discarding accumulated sum over `meses`

`safe.check` boolean specifying if some debbuging checks should be performed

Value

returns a time-series object of class `"ts"` of frequency `length(meses)` with the accumulated sum over the months defined in `meses`.

Author(s)

A.M. Sajo-Castelli

See Also

vetools, `diffmonths`, `diasdelmes`, `m12`, `time2ym`, `ym2time`, `xts2ts`.

`Vargas` *Rainfall in Vargas, Venezuela*

Description

Daily, monthly and quarterly precipitation values for meteorological stations located in the Vargas state, Venezuela.

There are two `collections` (data sets) of precipitation, *Vargas* and *Vargas2*, both data sets have the same source of meteorological stations located in the Vargas state, Venezuela. Data sets follows **vetools** Catalog Convention. See Catalog Convention.

Usage

```
data(Vargas)
data(Vargas2)
```

Format

The first collection *Vargas* contains four lists of 33 elements:

Vargas$catalog list of each station's meta data. Follows **vetools** Catalog Convention. To see meta data `summary(Vargas)`.

Vargas$daily list containing each station's daily time-series of class `"ts"` of frequency 365.25.

Vargas$data list containing each station's monthly time-series of class `"ts"` of frequency 12.

Vargas$quarterly list containing each station's quarterly time-series of class `"ts"` of frequency 4. The quarters are defined in base of Venezuela's rainy season: Feb-Apr, May-Jul, Aug-Oct, Nov-Jan.

The "data" and "quarterly" member series where built upon `daily` using the command `tssum` with arguments `meses=1:12` and `meses=c(2, 5, 8, 11) respectively.`

The second `collection` *Vargas2* contains three elements:

Vargas2$catalog list of 32 station's meta data. Follows **vetools** Catalog Convention. To see meta data `summary(Vargas2)`.

Vargas2$data list of length 32 containing each station's time-series of class `"ts"` on monthly basis. These series where completed using the Expectation-Maximization Algorithm.

Vargas2$statewise representative time-serie for the whole Vargas state. This series was build upon `Vargas2$data`.

Details

This data set provides day, monthly, quarterly and representative precipitation for 33 (32) meteorological stations located in the Vargas state of Venezuela. The region is delimited between -66.30917, -67.35 degrees and 10.46667 and 10.63 degrees (North), stations height vary between 0 and 1537 metres above sea level. Time-series for statios vary between 1948 and 2006.

The data set was imported into R using `read.MARN` function.

Source

Ministerio de Agricultura y Recursos Naturales. División De Hidrología, Meteorología y Oceanología (`http://www.minamb.gob.ve/`).

See Also

CuencaCaroni, **vetools** Catalog Convention, `read.MARN`, `disaggregate.MARN`, `tssum`.

Examples

```
## Not run:
data(Vargas, package='vetools')
summary(Vargas)
plot(Vargas$data[[2]])
start(Vargas$data[[1]])
end(Vargas$data[[1]])
frequency(Vargas$daily[[1]])
cat(Vargas$catalog[[1]]$Name)
## End(Not run)
```

`xts2ts` *Time-serie convertion routine*

Description

Converts from class `"xts"` to class `"ts"`

Usage

```
xts2ts(b.xts)
```

Arguments

`b.xts`	object of class `"xts"` to convert to class `"ts"`

Value

returns an object of class `"ts"`

Author(s)

A.M. Sajo-Castelli

See Also

vetools, `diffmonths`, `tssum`, `diasdelmes`, `time2ym`, `ym2time`, `m12`.

Index