# Package: aurelhy (via r-universe)

July 12, 2024

**Type** Package

**Version** 1.0.9

**Date** 2023-04-04

**Title** Hydrometeorological Interpolation

**Description** Hydrometeorological interpolation using the AURELHY method.

**Maintainer** Philippe Grosjean <phgrosjean@sciviews.org>

**Depends** R (>= 2.10.0)

**Imports** stats, graphics, sp, gstat

**Suggests** shapefiles, svUnit

**License** GPL-2

**URL** https://github.com/SciViews/aurelhy

**BugReports** https://github.com/SciViews/aurelhy/issues

**LazyLoad** yes

**NeedsCompilation** no

**#Roxygen** list(markdown = TRUE)

**#RoxygenNote** 7.1.0

**#VignetteBuilder** knitr

**Encoding** UTF-8

**Language** en-US

**Repository** https://sciviews.r-universe.dev

**RemoteUrl** https://github.com/SciViews/aurelhy

**RemoteRef** HEAD

**RemoteSha** 20dbbd38406faf16190244fc358d42b929a6c656

# Contents

---

| aurelhy-package | *Hydrometeorological Interpolation* |
|---|---|

---

### Description

Hydrometeorological interpolation using the AURELHY method.

### Details

| | |
|---|---|
| Package: | aurelhy |
| Title: | Hydrometeorological Interpolation |
| Type: | Package |
| Version: | 1.0-7 |
| Date: | 2015-07-16 |
| Authors@R: | c(person("Philippe", "Grosjean", role = c("aut", "cre"), email = "phgrosjean@sciviews. |
| URL: | https://github.com/phgrosjean/aurelhy |
| Depends: | sp, gstat |
| Suggests: | shapefiles |
| License: | GPL-2 |
| LazyLoad: | yes |
| Encoding: | UTF-8 |
| Author: Philippe Grosjean [aut, cre] | |
| Maintainer: | Philippe Grosjean <phgrosjean@sciviews.org> |

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

---

| aurelhy | *Create an 'aurelhy' object that contains required data to perform AU-RELHY interpolation* |
|---|---|

---

## Description

An 'aurelhy' object contains principal components calculated after the various variables describing the landscape, as well as other useful descriptors. Use the predict() method to interpolate some data with the AURELHY method.

## Usage

```
aurelhy(geotm, geomask, landmask = auremask(), x0 = 30, y0 = 30, step = 12,
  nbr.pc = 10, scale = FALSE, model = "data ~ .", vgmodel = gstat::vgm(1, "Sph", 10, 1),
   add.vars = NULL, var.name = NULL, resample.geomask = TRUE)

## S3 method for class 'aurelhy'
print(x, ...)
## S3 method for class 'aurelhy'
plot(x, y, main = "PCA on land descriptors", ...)
## S3 method for class 'aurelhy'
points(x, pch = ".", ...)
## S3 method for class 'aurelhy'
summary(object, ...)
## S3 method for class 'aurelhy'
update(object, nbr.pc, scale, model, vgmodel, ...)
## S3 method for class 'aurelhy'
predict(object, geopoints, variable, v.fit = NULL, ...)

## S3 method for class 'predict.aurelhy'
print(x, ...)
## S3 method for class 'predict.aurelhy'
summary(object, ...)
## S3 method for class 'predict.aurelhy'
plot(x, y, which = 1, ...)

## S3 method for class 'aurelhy'
as.geomat(x, what = "PC1", nodata = NA, ...)
## S3 method for class 'predict.aurelhy'
as.geomat(x,
    what = c("Interpolated", "Predicted", "KrigedResiduals", "KrigeVariance"),
 nodata = NA,...)
```

## Arguments

| | |
|---|---|
| geotm | a terrain model ('geotm' object) with enough resolution to be able to calculate all landscape descriptors (use `print()` or `plot()` methods of an 'auremask' object used to calculate landscape descriptors to check your terrain model is dense enough). |
| geomask | a 'geomask' object with same resolution and coverage of the 'geotm' object or the final interpolation grid (with `resample.geomask = FALSE`), and indicating which points should be considered for the interpolation (note that your terrain model must be larger than the targetted area by, at least, maximum distance of the mask in all directions in order to be able to calculate landscape descriptors for all considered points). |
| landmask | an 'auremask' object that defines the window of analysis used around each point ot calculate its landscape descriptors. |
| x0 | shift in X direction (longitude) where to consider the first point of the interpolation grid (note that interpolation grid must be less dense or equal to the terrain model grid, depending on the mask used). |
| y0 | shift in Y direction (latitude) for the first point of the interpolation grid. |
| step | resolution of the interpolation grid, i.e., we keep one point every `step` points from the original grid of the terrain model for constructing the interpolation grid. `step` must be a single integer larger or equal to one (may be equal to one only with rectangular 'auremask' objects). |
| nbr.pc | number of PCA's principal components to keep in the interpolation. This is the initial value; the example show you how you can change this after the 'aurelhy' object is calculated. |
| scale | should we scale the landscape descriptors (variance = 1) before performing the PCA? If `scale = FALSE` (by default), a PCA is run on the variance-covariance matrix (no scaling), otherwise, the PCA is run on the correlation matrix. |
| model | a formula describing the model used to predict the data. The left-hand side of the formula must always be 'data' and the right-hand considers all predictors spearated by a plus sign. To use all predictors, specify `data ~ .` (by default). |
| vgmodel | the variogram model to fit, as defined by the `gstat::vgm()` function of the gstat package |
| add.vars | additional variable(s) measured at the same points as the geotm object, or the final interpolation grid. They will be used as additional predictors. The example show you how you can add or remove such variables after the 'aurelhy' object is calculated. If `NULL` (by default), no additional variables will be used. |
| var.name | if `add.vars` is a 'geomat' object, you can give the name you want to use for this predictor here. |
| resample.geomask | |
| | do we resample the geomask using `x0`, `y0` and `step` to get the final mask of calculated points? If `TRUE` (by default), geomask should have the same grid as geotm. Otherwise, the geomask must exactly match the points where aurelhy should perform the interpolation. The default value allows for a backward-compatible behaviour of the function (aurelhy version =< 1.0-2). |

| | |
|---|---|
| x | an 'aurelhy' or 'predict.aurelhy' object, depending on the method invoked |
| y | a 'geopoints' object to create a plot best depicting the interpolation process, or nothing to just plot the interpolation grid. |
| main | the main title of the graph |
| pch | the symbol to use for plotting points. The default value, pch = "." prints a small (usually one pixel size) square |
| object | an 'aurelhy' object |
| geopoints | a 'geopoints' object with data to be interpolated. |
| variable | the name of the variable in the 'geopoints' object to interpolate |
| v.fit | the fitted variogram model used to krige residuals. If NULL (by default), a fitted model for the variogram is calculated, starting from the model provided in the 'aurelhy' object, 'vgm' slot. If FALSE, residuals are not kriged (useful, e.g., to save calculation time when one look for best predictors in the regression) |
| which | which graph to plot |
| what | what is extracted as a 'geomat' object |
| nodata | the code used to represent missing data in the 'geomat' object |
| ... | further arguments passed to the function |

## Details

aurelhy() creates a new 'aurelhy' object. The object has print() and plot() methods for further diagnostics. You should use the predict() method to perform the AURELHY interpolation on some data. The 'aurelhy' object is also easy to save for further reuse (it is designed so that the most time-consumming operations are done during its creation; so, it is supposed to be generated only once and reused for different interpolations on the same terrain model).

## Value

An 'aurelhy' object with all information required to perform an AURELHY interpolation with any 'geopoints' data.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## Source

Benichou P, Le Breton O (1987). Prise en compte de la topographie pour la cartographie des champs pluviometriques statistiques. La Meteorologie, 7:23-34.

## See Also

[geotm](), [auremask]()

**Examples**

```
# Create an aurelhy object for the Morocco terrain data
data(morocco)  # The terrain model with a grid of about 924x924m
data(mbord)    # A shape with the area around Morocco to analyze
data(mmask)    # A 924x924m grid with a mask covering territory to analyze
data(mseadist) # The distance to the sea for territory to analyze
data(mrain)    # Rain data measured at 43 stations to be interpolated

# Create a map with these data
image(morocco) # Plot the terrain model
grid()
lines(mbord, col = "red") # Add borders of territory to analyze in red

# Make sure we use all the stations from mrain in the geomask
mmask2 <- add.points(mmask, mrain)

# Now, create an aurelhy object with landscape description, using the
# first ten PCs, plus the distance to the sea (mseadist) for prediction
# Use a default radial window of analysis of 26km as maximum distance
# and an interpolation grid of 0.1x0.1degrees (roughly 11x11km)
# The variogram model is kept simple here, see ?gstat::vgm for other choices
# Be patient... this takes a little time to calculate!
maurelhy <- aurelhy(morocco, mmask2, auremask(), x0 = 30, y0 = 54, step = 12,
    scale = TRUE, nbr.pc = 10, vgmodel = gstat::vgm(100, "Sph", 1, 1),
    add.vars = mseadist, var.name = "seadist")
maurelhy
points(maurelhy) # Add the interpolated points on the map
points(mrain, col = "red") # Add location of weather stations in red

# Diagnostic of the PCA on land descriptors
summary(maurelhy)
plot(maurelhy)

# Interpolate 'rain' variable on the considered territory around Morocco
# Since we do not want negative values for this variable and it is log-normally
# distributed, we will interpolate log10(rain) instead
mrain$logRain <- log10(mrain$rain)
pmrain <- predict(maurelhy, mrain, "logRain")
pmrain

# Diagnostic of regression model
summary(pmrain) # Significant predictors at alpha = 0.01 are x, y, PC3, PC6 and PC7
# one could simplify the model as data ~ x + y + PC3 + PC6 + PC7
# but it is faster to keep the full model for final interpolation
# when we are only interested by the final interpolation or when processing
# is automated...
# Any of the predictors can be extracted from maurelhy as a geomat object
# for further inspection. For instance, let's look at PC3, PC6 and PC7 components
persp(as.geomat(maurelhy, "PC3"), expand = 50)
persp(as.geomat(maurelhy, "PC6"), expand = 50)
persp(as.geomat(maurelhy, "PC7"), expand = 50)
```

```
plot(pmrain, which = 1) # Residuals versus fitted (how residuals spread?)
plot(pmrain, which = 2) # Normal Q-Q plot of residuals (residuals distribution)
plot(pmrain, which = 3) # Best graph to look at residuals homoscedasticity
plot(pmrain, which = 4) # Cook's distance of residuals versus observation
plot(pmrain, which = 5) # Residuals leverage: are there influencial points?
# Map of predicted values
filled.contour(as.geomat(pmrain, "Predicted"), asp = 1,
    color.palette = terrain.colors, main = "Values predicted by the linear model")


# Residuals kriging diagnostic
plot(pmrain, which = 6) # Semi-variogram and adjusted model
filled.contour(as.geomat(pmrain, "KrigedResiduals"), asp = 1,
    color.palette = terrain.colors, main = "Kriged residuals")
filled.contour(as.geomat(pmrain, "KrigeVariance"), asp = 1,
    color.palette = terrain.colors, main = "Kriged residuals variance")
# As we can expect, kriging variance is larger in the south/south-west part
# where density of stations is low

# AURELHY interpolation diagnostic plots
# Graph showing the importance of predicted versus kriged residuals for
# all observations
plot(pmrain, which = 7)  # Model prediction and kriged residuals

# Extract interpolated log(rain) and transform back into rain (mm)
geomrain <- as.geomat(pmrain)
geomrain <- 10^geomrain

# How is interpolated rain distributed?
range(geomrain, na.rm = TRUE)
range(mrain$rain)

# Ooops! We have some very high values! How many?
sum(geomrain > 1000, na.rm = TRUE)
# This is probably due to a lack of data at high altitudes
# Let's truncate them to 1000 for a better graph
geomrain[geomrain > 1000] <- 1000
# ... and plot the result
image(geomrain, col = topo.colors(12))
contour(geomrain, add = TRUE)
lines(mbord, col = "red")
points(mrain, col = "red")

# A better plot for these interpolated rain data
filled.contour(coords(geomrain, "x"), coords(geomrain, "y"), geomrain,
    asp = 1, xlab = "Longitude", ylab = "Latitude",
    main = "AURELHY interpolated rain data", key.title = title(sub = "Rain (mm)\n\n"),
    color.palette = colorRampPalette(c("red", "gray", "blue"), bias = 2))


# One can experiment different interpolation parameters using update()
# Suppose we (1) don't want to scale PCs, (2) to keep only first 7 PCs,
# (3) we want an upgraded linear model like this:
# data <- a.x + b.y + c.z + d.PC3 + e.PC6 + f.PC7 + g.seadist + h.seadist^2 + i
```

```
# and (4) we want a Gaussian model for the semi-variogram
# (note that one can also regress against seadist2 <- seadist^2), just do:
# maurelhy2$seadist2 <- maurelhy$seadist^2
# Even with all these changes, you don't have to recompute maurelhy,
# just update() it and the costly steps of calculating landscape descriptors
# are reused (not the use of I() to protect calcs inside a formula)!
maurelhy2 <- update(maurelhy, scale = FALSE, nbr.pc = 7,
  model = data ~ x + y + z + PC3 + PC6 + PC7 + seadist + I(seadist^2),
  vgmodel = gstat::vgm(100, "Gau", 1, 1))
maurelhy2

# Diagnostic of the new PCA on land descriptors without scaling
summary(maurelhy2)
plot(maurelhy2)

# Interpolate with the new parameters
pmrain2 <- predict(maurelhy2, mrain, "logRain")
summary(pmrain2)

# A couple of graphs
plot(pmrain2, which = 1) # Residuals versus fitted (how residuals spread?)
plot(pmrain, which = 6) # Semi-variogram and adjusted model
plot(pmrain2, which = 7) # Model prediction and kriged residuals

#... Explore as much as you like until you find the set of parameters that suits you!
```

---

aurelhy-utilities          *Various utilities functions for AURELHY*

---

## Description

These functions manipulate geographical coordinates in various ways to optimize computation of the AURELHY method.

## Usage

```
deg.lat(latitude)
deg.lon(latitude)
polar.coords(geomat, x, y, maxdist)
match.coords(points, table, tol = 0.002)
coords(x, ...)
resample(x, ...)
add.points(x, ...)
## S3 method for class 'geomask'
add.points(x, geopoints, ...)
dist2sea(geotm)
```

## Arguments

| | |
|---|---|
| latitude | the latitude in decimal degrees |
| geomat | a 'geomat' object |
| x | X coordinate of the reference point for polar.coords(), or a correct object for the other functions |
| y | Y coordinate of the reference point |
| maxdist | maximum distance to consider in km. All points whose distance from the reference point is larger are not considered in the calculation |
| points | a list or data frame with X and Y coordinates of the points to match to the reference points (in decimal degrees, for instance) |
| table | a similar list or data frame with X(ref) and Y(ref) coordinates of the reference points to be matched (in the same units as for points) |
| tol | the maximum tolerance in X and Y units to consider points are matching, that is, X +/- tol = X(ref) and Y +/- tol = Y(ref) |
| ... | further arguments passed to the method |
| geopoints | a geopoints object from which we want to add corresponding points in a geomask |
| geotm | a geotm object |

## Details

deg.lat() and deg.lon() provide the length of one degree in, respectively, latitude and longitude in km, given the corresponding latitude in decimal degrees. The ellipsoid defined in WGS84 model is used for these calculations. polar.coords() calculates polar coordinates of points. match.coords() selects points with matching coordinates, given a tolerance distance between the reference points (i.e., from a geotm grid, using coords(my_geotm, "xy")) and the points to match (stations). coords() is a generic function that extracts geographical coordinates from one object in different fashions. resample is a generic function to resample a grid ('geomat' object). add.points add points from a geopoints object in a geomask. dist2sea() calculate the distance of points in a geotm object to the sea.

## Value

deg.lat() and deg.lon() return the length of one degree in km. polar.coords() returns a data frame with 'angle' in rad and 'dist'(ance) in km for the reference point to each point in the grid, within 'maxdist'. There is also a 'geomat' attribute containing the window of the initial 'geomat' object containing the considered points.

match.coords() returns a vector of logical of the same length as the number of colunms in the points data frame (that must contain 'x' and 'y' columns with coordinates of points to be matched).

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>, and Francois Delobel for dist2sea()

## See Also

[geomat](geomat), [auremask](auremask)

## Examples

```
# Size of one degree in latitude and longitude, given the latitude in decimal degrees
deg.lat(c(0, 15, 30, 45, 60, 75, 90))
# 110.574  110.649  110.852  111.132  111.412  111.618  111.694
deg.lon(c(0, 15, 30, 45, 60, 75, 90))
# 111.320  107.550  96.486  78.847  55.800  28.902  0.000
```

---

| auremask | *Create and manipulate a window of analysis for landscape descriptors used in AURELHY* |
|---|---|

---

## Description

An AURELHY window of analysis ('auremask' object) specifies the regions relative to the point that define the various variables describing the landscape.

## Usage

```
auremask(type = "radial", dist = c(1, 6, 11, 16, 21, 26),
    angles = 0:7 * pi/4 + 0.01, n = 11, keep.origin = FALSE)

## S3 method for class 'auremask'
print(x, geomat, ...)
## S3 method for class 'auremask'
plot(x, y, ...)
```

## Arguments

| | |
|---|---|
| type | the type of window, either "radial" (by default), or "rectangular" as in the initial version of the AURELHY method. |
| dist | A vector of distances (in km) to consider in the window for the "radial" window, or the distance to consider between two grid points for a "rectangular" window (in this case, if you provide several distances, only the smallest one will be considered) |
| angles | A vector of angles in radians to use to construct a "radial" window. This argument is ignored for "rectangular" window. Avoid to use angles parallels to the grid, like 0 or pi/4, because you will have points going into one or the other sector of your window of analysis, depending on rounding of the numbers in the floating-point calculations! A slight shift angle (0.01, by default) avoids this unstability |

| | |
|---|---|
| n | The number of grid points in latitude and longitude to use for a `"rectangular"` window. For instance, if n = 11, the window will be made of 11*11 = 121 points (minus one if `keep.origin` is `FALSE`). This argument is ignored for a `"radial"` window. |
| keep.origin | Is the origin where the window is centered considered as one point of the grid, or not (by default, not, as in the original implementation of the AURELHY method) |
| x | An 'auremask' object |
| geomat | A reference grid, as a 'geomat' object against which the window of analysis is tested (print or plot the number of points that are located in each sector of the window) |
| y | Same as `geomat` |
| ... | further arguments passed to the function |

## Details

auremask() creates a new window of analysis. The object has `print()` and `plot()` methods.

## Value

An 'auremask' object with all information required to mask a 'geotm' object (terrain model) for creating landscape variables required by the AURELHY method.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

[polar.coords](), [geomat]()

## Examples

```
# Default window of analysis
am <- auremask()
am
# Get an example terrain model and apply the window on it
data(morocco)
plot(am, morocco)
# Further statistics are displayed with print() if a grid is provided too
print(am, morocco)
```

---

geomat                          *A geomat, geotm or geomask object for AURELHY*

---

**Description**

Geomat are matrices of geographically referenced data. These are essentially georeferenced rect-
angular, regular grids of points. Data can be numeric (reals), integer, or logical (booleans). Objects
'geotm' are special 'geomat' matrices containing always integers and representing terrain models.
Objects 'geomask' are also special 'geomat' that only contain logical values. They are mainly used
to define a mask on top of a grid (which points to consider and which ones to eliminate from a
calculation).

**Usage**

```
geomat(x, size, xcenter, ycenter, coords = c(size = size, x = xcenter,
    y = ycenter), datatype = c("numeric", "integer", "logical"), nodata = NA)
geotm(x, size, xcenter, ycenter, coords = c(size = size, x = xcenter,
    y = ycenter))
geomask(x, size, xcenter, ycenter, coords = c(size = size, x = xcenter,
    y = ycenter))

read.geomat(file, type = "ascii", datatype = c("numeric", "integer", "logical"),
    ...)
read.geotm(file, type = "ascii", ...)
read.geomask(file, type = "ascii", threshold = 0, ...)

write.geomat(x, file, type = "ascii", integers = FALSE, nodata = -9999, ...)
write.geotm(x, file, type = "ascii", nodata = -9999, ...)
write.geomask(x, file, type = "ascii", nodata = -9999, ...)

as.geomat(x, ...)

## S3 method for class 'geomat'
print(x, ...)
## S3 method for class 'geomat'
coords(x, type = "par", ...)
## S3 method for class 'geomat'
resample(x, x0 = 1, y0 = 1, step = NULL, nx = 100, ny = nx,
    strict = FALSE, ...)
## S3 method for class 'geomat'
window(x, xlim, ylim, ...)
## S3 method for class 'geomat'
plot(x, y = NULL, max.xgrid = 100, nlevels = 50,
    color.palette = terrain.colors, xlab = "Longitude", ylab = "Latitude",
    asp = 1, ...)
## S3 method for class 'geomat'
image(x, max.xgrid = 500, col = terrain.colors(50),
```

```
        add = FALSE, xlab = if (add) "" else "Longitude",
        ylab = if (add) "" else "Latitude", asp = 1, ...)
## S3 method for class 'geomat'
contour(x, max.xgrid = 100, nlevels = 10, col = par("fg"),
        add = FALSE, xlab = if (add) "" else "Longitude",
        ylab = if (add) "" else "Latitude", asp = 1, ...)
## S3 method for class 'geomat'
persp(x, max.xgrid = 500, col = "green3",
        xlab = "Longitude", ylab = "Latitude", asp = 1, theta = 10, phi = 30,
        expand = 1, shade = 0.75, border = NA, box = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | An object (a matrix or data frame for geomat(), geotm(), or geomask(), a 'predict.aurelhy' object for as.geomat(), or a 'geomat' object for the other functions) |
| size | The size of a grid square (in decimal degrees) |
| xcenter | The position of the center of the top-left square of the grid, that is, its longitude in decimal degrees |
| ycenter | Idem, but latitude in decimal degrees |
| coords | A named vector of three numbers: 'size', 'x' and 'y' as above |
| datatype | The type of data to store in the grid, ort to read/write on the file. Can be 'numeric' (reals), 'integer', or 'logical' (booleans) |
| nodata | The number to use to represent missing data in the grid (by default it is NA). For file operations, it is the numerical code used to represent missing or not applicable cell in the file. By default, it is -9999 in ASCII grid format |
| file | The path to the file used for reading or writing data |
| type | The type of data to read/write. Currently, only \"ascii\", which means ARC/INFO ASCII GRID format (.asc file). For coords(), it is the type of coordinates to ba calculated: "par" is the vector defining the coordinates as 'size' of the cell, 'x' and 'y' coordinates of the center of the top-left square in the grid and the 'x1', 'y1' coordinates of the top-left point and 'x2', 'y2' coordinates of the bottom-right points covered by the grid. If "x", or "y", coords() returns a vector of the coordinates of centers of the grid points. Finally if "xy", then, coords() returns a data frame with 'x' and 'y' coordinates of all points in the grid (center of rectangles) |
| threshold | Value (single integer) above which all data are converted to TRUE. The rest is converted to FALSE, except missing data that are encoded as NA during the conversion into logical values |
| integers | Do we read/write integers (saves memory and disk space used to represent the grid) |
| x0 | The X origin of the new grid |
| y0 | The Y origin of the new grid |
| step | The step to use for resampling (step = 2 means we take one point every two original points in the grid). |

| | |
|---|---|
| nx | The desired number of points in the X direction (longitude). `resample()` is a quick method that takes a point every n points in the grid without doing more calculation. The final number of points is an integer value of points that can be resampled without interpolation |
| ny | idem than nx, but in the Y direction (latitude) |
| strict | do we interpolated the grid so that we obtain exactly nx and ny point (when `strict = TRUE`)? By default, not (`strict = FALSE`) and we span as far as possible to the right and to the bottom for the interpolated grid |
| xlim | A vector of two numbers defining the limits to use in X direction (longitude) for the window |
| ylim | A vector of two numbers defining the limits to use in Y direction (latitude) for the window |
| y | Unused argument to match `plot()` method definition |
| max.xgrid | The maximum number of points in x direction to use. If the grid that is plotted is denser, it is furst resampled to avoid drawing a graph with too much points |
| nlevels | the number of contour levles to calculate |
| color.palette | a color palette generation function |
| col | A vector of colors to use for the plot |
| xlab | The label of the X axis (″Longitude″ by default) |
| ylab | The label of the Y axis (″Latitude″ by default) |
| asp | The aspect ratio between 'x' and 'y'. The default value of `asp = 1` should usually not be changed. |
| add | Do we add the graph to an existing graph device, or do we plot a fresh new graph? |
| theta | angles defining the viewing direction. `theta` gives the azimuthal direction |
| phi | `phi` is the colatitude angle of the viewing direction |
| expand | the expansion level to use for the z-axis in the perspective |
| shade | the shade at a surface facet is computed as `((1+d)/2)^shade`, where d is the dot product of a unit vector normal to the facet and a unit vector in the direction of a light source. Values of shade close to one yield shading similar to a point light source model and values close to zero produce no shading. Values in the range 0.5 to 0.75 provide an approximation to daylight illumination. |
| border | the color of the borders of facets. If `NA`, no border is drawn. This is usually a good value when shading is used |
| box | If `TRUE`, a box, aznd axes are drawn around the perspective plot |
| ... | Further arguments passed to the functions (only used for the plotting method) |

## Value

An object of class, respectively 'geomat', 'geotm' or 'geomask' inheriting from 'matrix' is created. Methods either return an object of same class, or are used for their side effect of plotting a graph. Objects 'geotm' and 'geomask' also inherit from 'geomat'.

A 'geomat' object. For the `print()` method, size of the grid is presented in km.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

aurelhy, auremask

## Examples

```
# Create a simple geomat object containing random numbers
(gm <- geomat(matrix(rnorm(120), nrow = 10), 0.1, 10, 20))
# Get coordinates for this grid
coords(gm)
# Longitudes (x) and latitudes (y) for the center of all squares
coords(gm, type = "x")
coords(gm, type = "y")
# Coordinates of the center of all squares
coords(gm, type = "xy")

# Resample the grid to take one point every second points in the original grid
resample(gm, step = 2)

# Extract a window from the grid (keep only squares with centers in the window)
window(gm, xlim = c(9.5, 10.2), ylim = c(19.5, 20.6))

# Plot this grid in different ways
plot(gm)
image(gm)
contour(gm)
persp(gm, expand = 100)

# Now load real data (Morocco terrain model)
data(morocco)
morocco
image(morocco)
contour(morocco, add = TRUE)
grid()

# The mask of points inside Morocco territory was obtained like that:
#library(splancs)
#data(mbord)
#inm <- inout(coords(morocco, "xy"), mbord[[1]])
#mmask <- morocco
#mmask[inm] <- 1
#mmask[!inm] <- 0
#mmask[is.na(morocco)] <- NA
#mmask <- geomask(mmask, coords = coords(mmask))

data(mmask)
image(mmask)

# Get Morocco frontiers from a shapefile
```

```
# To read it from an ESRI shape
#mbord <- read.geoshapes("morocco_border.shp")

data(mbord)
lines(mbord, col = "red")
```

---

geopoints                     *A 'geopoints' object containing one or more georeferenced data*

---

### Description

Geospoints objects contain data for one or more points defined by their longitude and latitude in
decimal degrees. These objects can be read or write to ERSI shape files, or DBF database.

### Usage

```
geopoints(x)
read.geopoints(File, format)
write.geopoints(x, file, arcgis = FALSE,...)

## S3 method for class 'geopoints'
print(x, ...)
## S3 method for class 'geopoints'
points(x, ...)
```

### Arguments

| | |
|---|---|
| x | A 'geoshapes' object or a data frame with columns 'x' and 'y' for longitudes and latitudes of the points in decimal degrees. For print() and points() methods, it is a 'geopoints' object |
| File | The path to a .shp (ESRI shape file) or .dbf (DBase) file to import |
| format | Either "shp" or "dbf". If you do not provide this argument, the format is guess from the File extension |
| file | The path to an ESRI file where to write data, without extension. Three files are created, with respective extensions .shp, .shx, and .dbf |
| arcgis | If TRUE, the header of the DBF table is made compatible with ArcGIS, that is, dot (.) is replaced by underline (_) |
| ... | Further arguments passed to the functions (not used yet) |

### Details

geopoints() converts a 'geoshapes' object or a data frame into a 'geopoints' object. read.geoshapes()
and write.geoshapes() read and write shapes from or to ESRI shape files or DBase files on disk.
The 'geoshapes' objects have methods to print them, and to add them to graphs (points at corre-
sponding coordinates).

## Value

A 'geopoints' object is returned from geopoints() and read.geopoints(). The other functions are used for their side-effect rather than for returning something useful.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

[geomat](#), [geoshapes](#)

## Examples

```
data(mpet)
mpet

# Plot of Morocco terrain and add the stations location in red
data(morocco)
image(morocco)
points(mpet, col = 2)
```

---

| geoshapes | *A 'geoshapes' object containing one or more georeferenced shapes* |

---

## Description

Geoshapes objects contain one or more shapes (that is, polygons, points, or polylines) defined by their longitude and latitude in decimal degrees. These objects can be read or write to ERSI shape files.

## Usage

```
geoshapes(x, name = "1", dbf = NULL)
read.geoshapes(shpFile, dbf = TRUE)
write.geoshapes(x, file, type = c("polygon", "point", "polyLine"),
    dbf = TRUE, arcgis = FALSE,...)

## S3 method for class 'geoshapes'
print(x, ...)
## S3 method for class 'geoshapes'
lines(x, which = 1, ...)
## S3 method for class 'geoshapes'
points(x, which = "all", ...)
```

## Arguments

| | |
|---|---|
| x | A data frame with columns 'x' and 'y' for longitudes and latitudes of the points in decimal degrees, or a list of such data frames for geoshapes(); a 'geoshapes' object for the other functions |
| name | The name to use for the shape in case a data frame is passed to geoshapes(). Ignored if a list is passed to the function |
| dbf | A data frame to record as 'dbf' attribute for geoshapes, or a flag indicating to read or write DBF data too, if the file exists |
| shpFile | The path to a .shp file (ESRI shape file) to import |
| file | The path to an ESRI file where to write data, without extension. Three files are created, with respective extensions .shp, .shx, and .dbf |
| type | The type of shape to write in the ESRI shape file |
| arcgis | If TRUE, the header of the DBF table is made compatible with ArcGIS, that is, dot (.) is replaced by underline (_) |
| which | The index of the shape to use, or its name |
| ... | Further arguments passed to the functions (not used yet) |

## Details

geoshapes() converts a data frame or a list into a 'geoshapes' object. read.geoshapes() and write.geoshapes() read and write shapes from or to ESRI shape files on disk. The 'geoshapes' objects have methods to print them (very concisely), and to add them to graphs, as polygons lines(), or as separate points points().

## Value

A 'geoshapes' object is returned from geoshapes() and read.geoshapes(). The other functions are used for their side-effect rather than for returning something useful.

## Author(s)

Philippe Grosjean <phgrosjean@sciviews.org>

## See Also

[geomat](), [geopoints]()

## Examples

```
data(mbord) # Morocco borders
mbord

# Plot of Morocco terrain and add the borders in red
data(morocco)
image(morocco)
lines(mbord, col = 2)
```

```
# Simulate the creation of a geoshapes object with two shapes
geoshapes(list(a = mbord[[1]], b = mbord[[1]]))
```

---

| mbord | *A geoshapes object with a polygon of the area to analyze (around Morocco)* |
|---|---|

---

### Description

The mbord dataset is a 'geoshapes' object (georeferenced shapes).

### Usage

```
data(mbord)
```

### Format

This 'geoshapes' object contains a polygon defining the area to analyze around Morocco in decimal degrees (longitudes - latitudes).

### See Also

geoshapes, morocco , mmask

---

| mmask | *A geomask object masking the Morocco terrain model* |
|---|---|

---

### Description

The mmask dataset is a 'geomask' object (georeferenced mask). It indicates which ones of the points in the morocco terrain model do belong to the territory to analyze around Morocco (TRUE), or are land outside (FALSE). Points in the sea are flagged as NA.

### Usage

```
data(mmask)
```

### Format

This 'geomask' object contains a grid of 1986 x 1866 booleans masking the morocco terrain model.

### See Also

geomask, morocco , mbord

---

morocco                              *A digital elevation model with a grid of roughly 1km x 1km of Morocco*

---

### Description

The morocco dataset is a 'geotm' object (georeferenced terrain model).

### Usage

```
data(morocco)
```

### Format

This 'geotm' object contains a grid of 1986x1866 elevation points (in m).

### See Also

[geotm](), [mbord](), [mmask](), [mseadist]()

---

mpet                                 *A geopoints object with PET values measured at different weather stations*

---

### Description

The mpet dataset is a 'geopoints' object (georeferenced points data).

### Usage

```
data(mpet)
```

### Format

This 'geopoints' object contains a data frame with PET measurements (D1 - D36) made at 18 weather stations (names in STATION and coordinates in x and y).

### See Also

[geopoints](), [morocco](), [mbord](), [mrain]()

---

mrain                               *A geopoints object with normalized rain values (mm) measured at*
                                    *different weather stations*

---

### Description

The `mrain` dataset is a 'geopoints' object (georeferenced points data).

### Usage

```
data(mrain)
```

### Format

This 'geopoints' object contains a data frame with rain measurements (`rain` variable) made at 43
weather stations (coordinates in x and y variables).

### See Also

[geopoints](), [morocco](), [mbord](), [mpet]()

---

mseadist               *A geomat object with distance from the sea for the morocco data*

---

### Description

The `mseadist` dataset is a 'geomat' object (georeferenced data).

### Usage

```
data(mseadist)
```

### Format

This 'geomat' object contains a matrix with distance from the sea for all points in the `morocco`
dataset. It can be used as supplemntary variable for AURELHY interpolation.

### See Also

[geomat](), [morocco](), [mbord](), [mpet]()

---

unitTests.aurelhy            *Unit tests for the package aurelhy*

---

### Description

Performs unit tests defined in this package by running `example(unitTests.aurelhy)`. Tests are in `runit*.R` files located in the '/unitTests' subdirectory or one of its subdirectories ('/inst/unitTests' and subdirectories in package sources).

### Author(s)

Philippe Grosjean (`<phgrosjean@sciviews.org>`)

### Examples

```
if (require(svUnit)) {
    clearLog()


    ## This test is now moved to the tests directory
 runTest(svSuite("package:aurelhy"), "aurelhy")



    ## Check errors at the end (needed to interrupt R CMD check)
    errorLog()
}
```

# Index