Package 'RSAGA'

July 21, 2025

Type Package

Title SAGA Geoprocessing and Terrain Analysis

Version 1.4.2

Date 2025-02-01

Description Provides access to geocomputing and terrain analysis functions of the geographical information system (GIS) 'SAGA' (System for Automated Geoscientific Analyses) from within R by running the command line version of SAGA. This package furthermore provides several R functions for handling ASCII grids, including a flexible framework for applying local functions (including predict methods of fitted models) and focal functions to multiple grids. SAGA GIS is available under GPL-2 / LGPL-2 licences from <https://sourceforge.net/projects/saga-gis/>.

License GPL-2 | file LICENSE

URL https://github.com/r-spatial/RSAGA

Depends gstat, plyr, R (>= 3.5), shapefiles

Imports magrittr, stats, stringr, utils

Suggests gam, knitr, rmarkdown, sf, testthat, tibble

VignetteBuilder knitr

Encoding UTF-8

LazyData true

LazyLoad yes

RoxygenNote 7.3.2

SystemRequirements SAGA GIS (2.3 LTS - 9.7.2)

Collate 'gridtools.R' 'RSAGA-core.R' 'RSAGA-modules.R' 'RSAGA-utils.R' 'RSAGA-package.R' 'landslides.R' 'dem.R' 'study_area.R'

NeedsCompilation no

Author Alexander Brenning [aut, cre] (ORCID: <https://orcid.org/0000-0001-6640-679X>), Donovan Bangs [aut],

Marc Becker [aut],
Patrick Schratz [ctb] (ORCID: <https://orcid.org/0000-0003-0748-6624>),
Fabian Polakowski [ctb]

Maintainer Alexander Brenning <alexander.brenning@uni-jena.de>

Repository CRAN

Date/Publication 2025-02-04 15:30:02 UTC

Contents

RSAGA-package *RSAGA: SAGA Geoprocessing and Terrain Analysis in R*

Description

RSAGA provides direct access to SAGA GIS functions including, for example, a comprehensive set of terrain analysis algorithms for calculating local morphometric properties (slope, aspect, curvature), hydrographic characteristics (size, height, and aspect of catchment areas), and other process-related terrain attributes (potential incoming solar radiation, topographic wetness index, and more). In addition, (R)SAGA provides functions for importing and exporting different grid file formats, and tools for preprocessing grids, e.g. closing gaps or filling sinks.

Details

RSAGA adds a framework for creating custom-defined focal functions, e.g. specialized filter and terrain attributes such as the topographic wind shelter index, within R. This framework can be used to apply predict methods of fitted statistical models to stacks of grids representing predictor variables. Furthermore, functions are provided for conveniently picking values at point locations from a grid using kriging or nearest neighbour interpolation.

RSAGA requires SAGA GIS (versions 2.3.1 LTS - 9.7.2) are currently supported) and its user-contributed modules to be available on your computer. These can be downloaded under GPL from `https://sourceforge.net/projects/saga-gis/`. Please check the help page for `rsaga.env()` to make sure that RSAGA can find your local installation of SAGA. You may need to 'tell' RSAGA where to find SAGA GIS.

Thanks to Olaf Conrad, Andre Ringeler and all the other SAGA GIS developers and contributors of this excellent geocomputing tool! Thanks to Rainer Hurling, Johan van de Wauw, Massimo Di Stefano and others for helping to adapt SAGA to and test it on unix and Max OSX.

Author(s)

Alexander Brenning, Donovan Bangs and Marc Becker

References

Brenning, A., 2008. Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In J. Boehner, T. Blaschke and L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, vol. 19), p. 23-32.

Conrad, O., Bechtel, M., Bock, M., Dietrich, H., Fischer, E., Gerlitz, L., Wichmann, V., & Boehner, J. (2015). System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geoscientific Model Development*, 8, 1991-2007

See Also

Useful links:

- https://github.com/r-spatial/RSAGA

---

centervalue *Pick Center Value from Matrix*

Description

Pick the value in the center of a square matrix. Auxiliary function to be used by functions called by `focal.function()`.

Usage

```
centervalue(x)
```

Arguments

x	a square matrix

Details

See for example the code of `resid.median()`. Intended for use with square moving window matrices with an odd number of columns and rows.

Value

value of the matrix entry in the middle of the matrix

See Also

`focal.function()`, `resid.median()`

Examples

```
( m <- matrix( round(runif(9,1,10)), ncol=3 ) )
centervalue(m)
```

---

create.variable.name *Convert file name to variable name*

Description

Convert a file name into a variable name

Usage

```
create.variable.name(filename, prefix = NULL, fsep = .Platform$file.sep)
```

Arguments

filename	character string
prefix	character string: optional prefix to be added
fsep	character used to separate path components

Value

character string: a variable name, see example below

Examples

```
## Not run:
create.variable.name("C:/my-path/my-file-name.Rd",prefix="res")
# Result: "res.my.file.name"

## End(Not run)
```

---

dem *DEM for the* landslides *dataset*

Description

Digital elevation model (DEM) for the landslides dataset's study area in southern Ecuador

Usage

```
dem
```

Format

Digital elevation model given as a list consisting of the elements `header` (nine properties) and `data` (grid elevation values in m a.s.l.). The 10 m x 10 m digital elevation model was triangulated from aerial imagery as described by Jordan *et al.* (2005) and provided as a courtesy of Lars Ungerechts (2010).

Details

Landslide data provided here are a subset of that used by Muenchow *et al.* (2012) to predict spatially landslide susceptibility using generalized additive models (GAMs). Specifically, the here provided landslides belong to the "natural" part of the *RBSF* area. Please refer also to the accompanying vignette for an introductory tutorial on the use of the RSAGA package for terrain analysis, geoprocessing, and model-building using these data.

Note

Loading `landslides` replaces any existing objects in the current work environment named `dem`, `landslides`, and `study_area`.

Source

Ungerechts, L. (2010): DEM 10m (triangulated from aerial photo - b/w). Retrieved from `http://vhrz669.hrz.uni-marbur`

Jordan, E., Ungerechts, L., Caceres, B. Penafiel, A. and Francou, B. (2005): Estimation by photogrammetry of the glacier recession on the Cotopaxi Volcano (Ecuador) between 1956 and 1997. *Hydrological Sciences* 50, 949-961.

See Also

landslides

Examples

```
## Not run:
library("RSAGA")
data(dem)

# Print the DEM header:
dem$header

# Write the DEM to a SAGA grid:
write.sgrd(data = dem, file = "dem", header = dem$header, env = env)

# Calculate slope of DEM:
rsaga.slope(in.dem = "dem", out.slope = "slope", method = "poly2zevenbergen")

## End(Not run)
```

`focal.function` *Local and Focal Grid Functions*

Description

`focal.function` cuts out square or circular moving windows from a grid (matrix) and applies a user-defined matrix function to calculate e.g. a terrain attribute or filter the grid. The function is suitable for large grid files as it can process them row by row. `local.function` represents the special case of a moving window of radius 1. Users can define their own functions operating on moving windows, or use simple functions such as `median` to define filters.

Usage

```
focal.function(
  in.grid,
  in.factor.grid,
  out.grid.prefix,
  path = NULL,
  in.path = path,
  out.path = path,
  fun,
  varnames,
  radius = 0,
  is.pixel.radius = TRUE,
  na.strings = "NA",
  valid.range = c(-Inf, Inf),
  nodata.values = c(),
  out.nodata.value,
  search.mode = c("circle", "square"),
  digits = 4,
  hdr.digits = 10,
  dec = ".",
  quiet = TRUE,
  nlines = Inf,
  mw.to.vector = FALSE,
  mw.na.rm = FALSE,
  ...
)

gapply(in.grid, fun, varnames, mw.to.vector = TRUE, mw.na.rm = TRUE, ...)

local.function(...)
```

Arguments

`in.grid`	file name of input ASCII grid, relative to `in.path`

in.factor.grid	optional file name giving a gridded categorical variables defining zones; zone boundaries are used as breaklines for the moving window (see Details)
out.grid.prefix	character string (optional), defining a file name prefix to be used for the output file names; a dash (-) will separate the prefix and the varnames
path	path in which to look for in.grid and write output grid files; see also in.path and out.path, which overwrite path if they are specified
in.path	path in which to look for in.grid (defaults to path)
out.path	path in which to write output grid files; defaults to path
fun	a function, or name of a function, to be applied on the moving window; see Details
varnames	character vector specifying the names of the variable(s) returned by fun; if missing, focal.function will try to determine the varnames from fun itself, or from a call to fun if this is a function (see Details)
radius	numeric value specifying the (circular or square) radius of the moving window; see is.pixel.radius and search.mode; note that all data within distance <=radius will be included in the moving window, not <radius.
is.pixel.radius	logical: if TRUE (default), the radius will be interpreted as a (possibly non-integer) number of pixels; if FALSE, it is interpreted as a radius measured in the grid (map) units.
na.strings	passed on to scan()
valid.range	numeric vector of length 2, specifying minimum and maximum valid values read from input file; all values <valid.range[1] or >valid.range[1] will be converted to NA.
nodata.values	numeric vector: any values from the input grid file that should be converted to NA, in addition to the nodata value specified in the grid header
out.nodata.value	numeric: value used for storing NAs in the output file(s); if missing, use the same nodata value as specified in the header of the input grid file
search.mode	character, either "circle" (default) for a circular search window, or "square" for a squared one.
digits	numeric, specifying the number of digits to be used for output grid file.
hdr.digits	numeric, specifying the number of digits to be used for the header of the output grid file (default: 10; see write.ascii.grid.header()).
dec	character, specifying the decimal mark to be used for input and output.
quiet	If TRUE, gives some output ("*") after every 10th line of the grid file and when the job is done.
nlines	Number of lines to be processed; useful for testing purposes.
mw.to.vector	logical: Should the content of the moving window be coerced (from a matrix) to a vector?
mw.na.rm	logical: Should NAs be removed from moving window prior to passing the data to fun? Only applicable when mw.to.vector=TRUE.
...	Arguments to be passed to fun; local.function: arguments to be passed to focal.function.

Details

`focal.function` passes a square matrix of size `2*radius+1` to the function `fun` if `mw.to.vector=FALSE` (default), or a vector of length `<=(2*radius+1)^2` if `mw.to.vector=TRUE`. This matrix or vector will contain the content of the moving window, which may possibly contain `NA`s even if the `in.grid` has no nodata values, e.g. due to edge effects. If `search.mode="circle"`, values more than `radius` units (pixels or grid units, depending on `is.pixel.radius`) away from the center pixel / matrix entry will be set to `NA`. In addition, `valid.range`, `nodata.values`, and the nodata values specified in the `in.grid` are checked to assign further `NA`s to pixels in the moving window. Finally, if `in.factor.grid` specifies zones, all pixels in the moving window that belong to a different zone than the center pixel are set to `NA`, or, in other words, zone boundaries are used as breaklines.

The function `fun` should return a single numeric value or a numeric vector. As an example, the function `resid.minmedmax()` returns the minimum, median and maximum of the difference between the values in the moving window and the value in the center grid cell. In addition to the (first) argument receiving the moving window data, `fun` may have additional arguments; the `...` argument of `focal.function` is passed on to `fun`. `resid.quantile()` is a function that uses this feature.

Optionally, `fun` should support the following feature: If no argument is passed to it, then it should return a character vector giving variable names to be used for naming the output grids. The call `resid.minmedmax()`, for example, returns `c("rmin","rmed","rmax")`; this vector must have the same length as the numeric vector returned when moving window data is passed to the function. This feature is only used if no `varnames` argument is provided. Note that the result is currently being `abbreviate()`d to a length of 6 characters.

Input and output file names are built according to the following schemes:

Input: `[<in.path>/]<in.grid>`

Zones: `[<in.path>/]<in.factor.grid>` (if specified)

Output: `[<out.path>/][<out.grid.prefix>-]<varnames>.asc`

For the input files, `.asc` is used as the default file extension, if it is not specified by the user.

Value

`focal.function` and `local.function` return the character vector of output file names.

Note

These functions are not very efficient ways of calculating e.g. (focal) terrain attributes compared to for example the SAGA modules, but the idea is that you can easily specify your own functions without starting to mess around with C code. For example try implementing a median filter as a SAGA module... or just use the code shown in the example!

Author(s)

Alexander Brenning

References

Brenning, A. (2008): Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In: J. Boehner, T. Blaschke, L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, 19), 23-32.

See Also

multi.focal.function(), multi.local.function(), resid.median(), resid.minmedmax(), relative.position(), resid.quantile(), resid.quartiles(), relative.rank(), wind.shelter(), create.variable.name()

Examples

```
## Not run:
# A simple median filter applied to dem.asc:
gapply("dem","median",radius=3)
# Same:
#focal.function("dem",fun="median",radius=3,mw.to.vector=TRUE,mw.na.rm=TRUE)
# See how the filter has changed the elevation data:
d1 = as.vector(read.ascii.grid("dem")$data)
d2 = as.vector(read.ascii.grid("median")$data)
hist(d1-d2,br=50)

## End(Not run)
# Wind shelter index used by Plattner et al. (2004):
## Not run:
ctrl = wind.shelter.prep(6,-pi/4,pi/12,10)
focal.function("dem",fun=wind.shelter,control=ctrl,
    radius=6,search.mode="circle")

## End(Not run)
# Or how about this, if "aspect" is local terrain exposure:
## Not run:
gapply("aspect","cos") # how "northerly-exposed" is a pixel?
gapply("aspect","sin") # how "easterly-exposed" is a pixel?
# Same result, but faster:
focal.function("aspect",fun=function(x) c(cos(x),sin(x)), varnames=c("cos","sin"))

## End(Not run)
```

---

grid.predict *Helper function for applying predict methods to stacks of grids.*

Description

This function can be used to apply the predict method of hopefully any fitted predictive model pixel by pixel to a stack of grids representing the explanatory variables. It is intended to be called primarily by multi.local.function() or multi.focal.function().

Usage

```
grid.predict(
  fit,
  predfun,
  trafo,
  control.predict,
  predict.column,
  trace = 0,
  location,
  ...
)
```

Arguments

fit	a model object for which prediction is desired
predfun	optional prediction function; if missing, the fit's predict() method is called. In some cases it may be convenient to define a wrapper function for the predict method that may be passed as predfun argument.
trafo	an optional function(x) that takes a data.frame x and returns a data.frame with the same number of rows; this is intended to perform transformations on the input variables, e.g. derive a log-transformed variable from the raw input read from the grids, or more complex variables such as the NDVI etc.; the data.frame resulting from a call to trafo (if provided) is passed to predfun
control.predict	an optional list of arguments to be passed on to predfun; this may be e.g. type="response" to obtain probability prediction maps from a logistic regression model
predict.column	optional character string: Some predict methods (e.g. predict.lda) return a data.frame with several columns, e.g. one column per class in a classification problem. predict.column is used to pick the one that is of interest
trace	integer >=0: positive values give more (=2) or less (=1) information on predictor variables and predictions
location	optional location data received from multi.focal.function; is added to the newdata object that is passed on to predfun.
...	these arguments are provided by the calling function, usually multi.local.function() or multi.focal.function(). They contain the explanatory (predictor) variables required by the fit model.

Details

grid.predict is a simple wrapper function. First it binds the arguments in \dots together in a data.frame with the raw predictor variables that have been read from their grids by the caller, multi.local.function() (or multi.focal.function()). Then it calls the optional trafo function to transform or combine predictor variables (e.g. perform log transformations, ratioing, arithmetic operations such as calculating the NDVI). Finally the predfun (or, typically, the default predict() method of fit) is called, handing over the fit, the predictor data.frame, and the optional control.predict arguments.

Value

`grid.predict` returns the result of the call to `predfun` or the default `predict()` method.

Note

Though `grid.predict` can in principle deal with `predict` methods returning factor variables, its usual caller `multi.local.function()` / `multi.focal.function()` cannot; classification models should be dealt with by setting a `type="prob"` (for `rpart`) or `type="response"` (for logistic regression and logistic additive model) argument, for example (see second Example below).

Author(s)

Alexander Brenning

References

Brenning, A. (2008): Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In: J. Boehner, T. Blaschke, L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, 19), 23-32.

See Also

`focal.function()`, `multi.local.function()`, `multi.focal.function()`

Examples

```
## Not run:
# Assume that d is a data.frame with point observations
# of a numerical response variable y and predictor variables
# a, b, and c.
# Fit a generalized additive model to y,a,b,c.
# We want to model b and c as nonlinear terms:
require(gam)
fit <- gam(y ~ a + s(b) + s(c), data = d)
multi.local.function(in.grids = c("a", "b", "c"),
    out.varnames = "pred",
    fun = grid.predict, fit = fit )
    # Note that the 'grid.predict' uses by default the
    # predict method of 'fit'.
# Model predictions are written to a file named pred.asc

## End(Not run)

## Not run:
# A fake example of a logistic additive model:
require(gam)
fit <- gam(cl ~ a + s(b) + s(c), data = d, family = binomial)
multi.local.function(in.grids = c("a", "b", "c"),
    out.varnames = "pred",
    fun = grid.predict, fit = fit,
```

```
    control.predict = list(type = "response") )
    # 'control.predict' is passed on to 'grid.predict', which
    # dumps its contents into the arguments for 'fit''s
    # 'predict' method.
# Model predictions are written to a file named pred.asc

## End(Not run)
```

`grid.to.xyz` *Convert Grid Matrix to (x,y,z) data.frame*

Description

Convert a grid matrix to a (x,y,z) data.frame.

Usage

```
grid.to.xyz(data, header, varname = "z", colnames = c("x", "y", varname))
```

Arguments

`data`	grid data: either a grid data matrix, or a list with components `data` (a matrix with the grid data) and `header` (the grid header information); see `read.ascii.grid()` for details
`header`	optional list giving grid header information; see `read.ascii.grid()` for details
`varname`	character: name to be assigned to the column with the z values in the output data.frame
`colnames`	names to be given to the columns corresponding to the x and y coordinates and the grid variable in the output data.frame

Value

a data.frame with three columns (names are specified in the `colnames` argument) giving the x and y coordinates and the attribute values at the locations given by the grid `data`.

See Also

`read.ascii.grid()`, `pick.from.ascii.grid()`

Examples

```
## Not run:
d = read.ascii.grid("dem")
xyz = grid.to.xyz(d,varname="elevation")
str(xyz)

## End(Not run)
```

`landslides` *Landslide inventory data*

Description

Landslide data from southern Ecuador

Landslide data from southern Ecuador

Usage

```
landslides
```

Format

The landslide dataset consists of three objects: `landslides`, `dem`, and `study_area`.

- `landslides`: A `data.frame` with 1535 rows and 3 variables representing landslide initiation points in the *Reserva Biologica San Francisco* (RBSF) area of the tropical Andes in Southern Ecuador. The variables are:
  - `lslpts`: Landslide initiation point (boolean)
  - `x` and `y`: Coordinates in UTM zone 17S (EPSG: 32717)

  The landslide inventory was mapped by Stoyan (2000) in the field and by the presence of landslide scars in aerial imagery.

Details

Landslide data provided here are a subset of that used by Muenchow *et al.* (2012) to predict spatially landslide susceptibility using generalized additive models (GAMs). Specifically, the here provided landslides belong to the "natural" part of the *RBSF* area. Please refer also to the accompanying vignette for an introductory tutorial on the use of the RSAGA package for terrain analysis, geoprocessing, and model-building using these data.

Source

Muenchow, J., Brenning, A., Richter, R. (2012): Geomorphic process rates of landslides along a humidity gradient in the tropical Andes, Geomorphology 139-140, 271-284. DOI: 10.1016/j.geomorph.2011.10.029.

Stoyan, R. (2000): Aktivitaet, Ursachen und Klassifikation der Rutschungen in San Francisco/Suedecuador. Unpublished diploma thesis, University of Erlangen-Nuremberg, Germany.

See Also

dem, study_area

Examples

```
## Not run:
library("RSAGA")
data(landslides)
data(dem)

# Print the DEM header:
dem$header

# Write the DEM to a SAGA grid:
write.sgrd(data = dem, file = "dem", header = dem$header, env = env)

# Calculate slope of DEM:
rsaga.slope(in.dem = "dem", out.slope = "slope", method = "poly2zevenbergen")

# Pick slope values at landslide points,
# added to landslides data.frame as variable "slope":
landslides <- pick.from.saga.grid(data = landslides,
                                  filename = "slope",
                                  varname = "slope")

## End(Not run)
```

`match.arg.ext` *Extended Argument Matching*

Description

`match.arg.ext` matches `arg` against a set of candidate values as specified by `choices`; it extends `match.arg()` by allowing `arg` to be a numeric identifier of the `choices`.

Usage

```
match.arg.ext(
  arg,
  choices,
  base = 1,
  several.ok = FALSE,
  numeric = FALSE,
  ignore.case = FALSE
)
```

Arguments

`arg`	a character string or numeric value
`choices`	a character vector of candidate values
`base`	numeric value, specifying the numeric index assigned to the first element of `choices`

several.ok	logical specifying if arg should be allowed to have more than one element
numeric	logical specifying if the function should return the numerical index (counting from base) of the matched argument, or, by default, its name
ignore.case	logical specifying if the matching should be case sensitive

Details

When choices are missing, they are obtained from a default setting for the formal argument arg of the function from which match.arg.ext was called.

Matching is done using pmatch() (indirectly through a call to match.arg(), so arg may be abbreviated.

If arg is numeric, it may take values between base and length(choices)+base-1. base=1 will give standard 1-based R indices, base=0 will give indices counted from zero as used to identify SAGA modules in library RSAGA.

Value

If numeric is false and arg is a character string, the function returns the unabbreviated version of the unique partial match of arg if there is one; otherwise, an error is produced if several.ok is false, as per default. When several.ok is true and there is more than one match, all unabbreviated versions of matches are returned.

If numeric is false but arg is numeric, match.arg.ext returns name of the match corresponding to this index, counting from base; i.e. arg=base corresponds to choices[1].

If numeric is true, the function returns the numeric index(es) of the partial match of arg, counted from base to length(choices)+base-1. If arg is already numeric, the function only checks whether it falls into the valid range from arg to length(choices)+base-1 and returns arg.

Author(s)

Alexander Brenning

See Also

match.arg(), pmatch()

Examples

```
# Based on example from 'match.arg':
require(stats)
center <- function(x, type = c("mean", "median", "trimmed")) {
  type <- match.arg.ext(type,base=0)
  switch(type,
         mean = mean(x),
         median = median(x),
         trimmed = mean(x, trim = .1))
}
x <- rcauchy(10)
center(x, "t")       # Works
center(x, 2)         # Same, for base=0
```

```
center(x, "med")     # Works
center(x, 1)         # Same, for base=0
try(center(x, "m"))  # Error
```

`multi.focal.function` *Local and Focal Grid Function with Multiple Grids as Inputs*

Description

`multi.focal.function` cuts out square or circular moving windows from a stack of grids (matrices) and applies a user-defined matrix function that takes multiple arguments to this data. `multi.local.function` is a more efficiently coded special case of moving windows of size 0, i.e. functions applied to individual grid cells of a stack of grids. This is especially useful for applying `predict` methods of statistical models to a stack of grids containing the explanatory variables (see Examples and `grid.predict()`). The function is suitable for large grid files as it can process them row by row; but it may be slow because one call to the focal function is generated for each grid cell.

Usage

```
multi.focal.function(
  in.grids,
  in.grid.prefix,
  in.factor.grid,
  out.grid.prefix,
  path = NULL,
  in.path = path,
  out.path = path,
  fun,
  in.varnames,
  out.varnames,
  radius = 0,
  is.pixel.radius = TRUE,
  na.strings = "NA",
  valid.ranges,
  nodata.values = c(),
  out.nodata.value,
  search.mode = c("circle", "square"),
  digits = 4,
  hdr.digits = 10,
  dec = ".",
  quiet = TRUE,
  nlines = Inf,
  mw.to.vector = FALSE,
  mw.na.rm = FALSE,
  pass.location = FALSE,
  ...
)
```

```
multi.local.function(
  in.grids,
  in.grid.prefix,
  out.grid.prefix,
  path = NULL,
  in.path = path,
  out.path = path,
  fun,
  in.varnames,
  out.varnames,
  na.strings = "NA",
  valid.ranges,
  nodata.values = c(),
  out.nodata.value,
  digits = 4,
  hdr.digits = 10,
  dec = ".",
  quiet = TRUE,
  nlines = Inf,
  na.action = stats::na.exclude,
  pass.location = FALSE,
  ...
)
```

Arguments

`in.grids`	character vector: file names of input ASCII grids, relative to `in.path`; `in.grid.prefix` will be used as a prefix to the file name if specified; default file extension: `.asc`
`in.grid.prefix`	character string (optional), defining a file name prefix to be used for the input file names; a dash (`-`) will separate the prefix and the `in.varnames`
`in.factor.grid`	optional file name giving a gridded categorical variables defining zones; zone boundaries are used as breaklines for the moving window (see Details)
`out.grid.prefix`	character string (optional), defining a file name prefix to be used for the output file names; a dash (`-`) will separate the prefix and the `out.varnames`
`path`	path in which to look for `in.grids` and write output grid files; see also `in.path` and `out.path`, which overwrite `path` if they are specified
`in.path`	path in which to look for `in.grids` (defaults to `path`)
`out.path`	path in which to write output grid files; defaults to `path`
`fun`	a function, or name of a function, to be applied on the moving window; see Details; `fun` is expected to accept named arguments with the names given by `in.varnames`; `grid.predict()` is a wrapper function that can be used for applying a model's `predict` method to a stack of grids; see Details. In `multi.local.function`, `fun` must be able to process arguments that are vectors of equal length (e.g., a vector of 50 slope angles, another vector of 50 elevation values, etc.).

in.varnames	character vector: names of the variables corresponding to the in.grids; if missing, same as in.grids; if specified, must have the same length and order as in.grids
out.varnames	character vector specifying the name(s) of the variable(s) returned by fun; if missing, multi.focal.function will try to determine the varnames from fun itself, or or from a call to fun if this is a function (see Details)
radius	numeric value specifying the (circular or square) radius of the moving window; see is.pixel.radius and search.mode; note that all data within distance <=radius will be included in the moving window, not <radius.
is.pixel.radius	logical: if TRUE (default), the radius will be interpreted as a (possibly non-integer) number of pixels; if FALSE, it is interpreted as a radius measured in the grid (map) units.
na.strings	passed on to scan()
valid.ranges	optional list of length length(in.grids) with numeric vector of length 2, specifying minimum and maximum valid values read from input file; all values <valid.ranges[[i]][1] or >valid.ranges[[i]][1] will be converted to NA.
nodata.values	numeric vector: any values from the input grid file that should be converted to NA, in addition to the nodata value specified in the grid header
out.nodata.value	numeric: value used for storing NAs in the output file(s); if missing, use the same nodata value as specified in the header of the input grid file
search.mode	character, either "circle" (default) for a circular search window, or "square" for a squared one.
digits	numeric, specifying the number of digits to be used for output grid file.
hdr.digits	numeric, specifying the number of digits to be used for the header of the output grid file (default: 10; see write.ascii.grid.header()).
dec	character, specifying the decimal mark to be used for input and output.
quiet	If FALSE, gives some output ("*") after every 10th line of the grid file and when the job is done.
nlines	Number of lines to be processed; useful for testing purposes.
mw.to.vector	logical: Should the content of the moving window be coerced (from a matrix) to a vector?
mw.na.rm	logical: Should NAs be removed from moving window prior to passing the data to fun? Only applicable when mw.to.vector=TRUE.
pass.location	logical: Should the x,y coordinates of grid points (center of grid cells) be passed to fun? If TRUE, two additional arguments named arguments x and y are passed to fun; NOTE: This currently only works for radius=0, otherwise a warning is produced and pass.location is reset to FALSE.
...	Arguments to be passed to fun; local.function: arguments to be passed to focal.function.
na.action	function: determines if/how NA values are omitted from the stack of input variables; use na.exclude() (default) or na.pass() if fun can handle NA values correctly

Details

multi.local.function is probably most useful for applying the predict method of a fitted model to a grids representing the predictor variables. An example is given below and in more detail in Brenning (2008) (who used multi.focal.function for the same purpose); see also grid.predict().

multi.local.function is essentially the same as multi.focal.function for radius=0, but coded MUCH more efficiently. (The relevant code will eventually migrate into multi.focal.function as well, but requires further testing.) Applying a GAM to the data set of Brenning (2008) takes about 1/100th the time with multi.local.function compared to multi.focal.function.

multi.focal.function extends focal.function() by allowing multiple input grids to be passed to the focal function fun operating on moving windows. It passes square matrices of size 2*radius+1 to the function fun if mw.to.vector=FALSE (default), or a vector of length <=(2*radius+1)^2 if mw.to.vector=TRUE; one such matrix or vector per input grid will be passed to fun as an argument whose name is specified by in.varnames.

These matrices or vectors will contain the content of the moving window, which may possibly contain NAs even if the in.grid has no nodata values, e.g. due to edge effects. If search.mode="circle", values more than radius units (pixels or grid units, depending on is.pixel.radius) away from the center pixel / matrix entry will be set to NA. In addition, valid.range, nodata.values, and the nodata values specified in the in.grid are checked to assign further NAs to pixels in the moving window. Finally, if in.factor.grid specifies zones, all pixels in the moving window that belong to a different zone than the center pixel are set to NA, or, in other words, zone boundaries are used as breaklines.

The function fun should return a single numeric value or a numeric vector, such as a regression result or a vector of class probabilities returned by a soft classifier. In addition to the named arguments receiving the moving window data, fun may have additional arguments; the ... argument of focal.function is passed on to fun. grid.predict() uses this feature.

Optionally, fun should support the following feature: If no argument is passed to it, then it should return a character vector giving variable names to be used for naming the output grids.

For the input files, .asc is used as the default file extension, if it is not specified by the user.

See focal.function() for details.

Value

multi.focal.function returns the character vector of output file names.

Note

multi.focal.function can do all the things focal.function() can do.

Author(s)

Alexander Brenning

References

Brenning, A. (2008): Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In: J. Boehner, T. Blaschke, L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, 19), 23-32.

See Also

focal.function(), grid.predict()

Examples

```
## Not run:
# Assume that d is a data.frame with point observations
# of a numerical response variable y and predictor variables
# a, b, and c.
# Fit a generalized additive model to y,a,b,c.
# We want to model b and c as nonlinear terms:
require(gam)
fit <- gam(y ~ a + s(b) + s(c), data = d)
multi.local.function(in.grids = c("a", "b", "c"),
    out.varnames = "pred",
    fun = grid.predict, fit = fit )
    # Note that the 'grid.predict' uses by default the
    # predict method of 'fit'.
# Model predictions are written to a file named pred.asc

## End(Not run)

## Not run:
# A fake example of a logistic additive model:
require(gam)
fit <- gam(cl ~ a + s(b) + s(c), data = d, family = binomial)
multi.local.function(in.grids = c("a", "b", "c"),
    out.varnames = "pred",
    fun = grid.predict, fit = fit,
    control.predict = list(type = "response") )
    # 'control.predict' is passed on to 'grid.predict', which
    # dumps its contents into the arguments for 'fit''s
    # 'predict' method.
# Model predictions are written to a file named pred.asc

## End(Not run)
```

---

pick.from.points *Pick Variable from Spatial Dataset*

Description

These functions pick (i.e. interpolate without worrying too much about theory) values of a spatial variables from a data stored in a data.frame, a point shapefile, or an ASCII or SAGA grid, using nearest neighbor or kriging interpolation. `pick.from.points` and `[internal.]pick.from.ascii.grid` are the core functions that are called by the different wrappers.

Usage

```
pick.from.points(
  data,
  src,
  pick,
  method = c("nearest.neighbour", "krige"),
  set.na = FALSE,
  radius = 200,
  nmin = 0,
  nmax = 100,
  sill = 1,
  range = radius,
  nugget = 0,
  model = vgm(sill - nugget, "Sph", range = range, nugget = nugget),
  log = rep(FALSE, length(pick)),
  X.name = "x",
  Y.name = "y",
  cbind = TRUE
)

pick.from.shapefile(data, shapefile, X.name = "x", Y.name = "y", ...)

pick.from.ascii.grid(
  data,
  file,
  path = NULL,
  varname = NULL,
  prefix = NULL,
  method = c("nearest.neighbour", "krige"),
  cbind = TRUE,
  parallel = FALSE,
  nsplit,
  quiet = TRUE,
  ...
)

pick.from.ascii.grids(
  data,
  file,
  path = NULL,
  varname = NULL,
```

```
  prefix = NULL,
  cbind = TRUE,
  quiet = TRUE,
  ...
)

internal.pick.from.ascii.grid(
  data,
  file,
  path = NULL,
  varname = NULL,
  prefix = NULL,
  method = c("nearest.neighbour", "krige"),
  nodata.values = c(-9999, -99999),
  at.once,
  quiet = TRUE,
  X.name = "x",
  Y.name = "y",
  nlines = Inf,
  cbind = TRUE,
  range,
  radius,
  na.strings = "NA",
  ...
)

pick.from.saga.grid(
  data,
  filename,
  path,
  varname,
  prec = 7,
  show.output.on.console = FALSE,
  env = rsaga.env(),
  ...
)
```

Arguments

data	data.frame giving the coordinates (in columns specified by X.name, Y.name) of point locations at which to interpolate the specified variables or grid values
src	data.frame
pick	variables to be picked (interpolated) from src; if missing, use all available variables, except those specified by X.name and Y.name
method	interpolation method to be used; uses a partial match to the alternatives "nearest.neighbor" (currently the default) and "krige"
set.na	logical: if a column with a name specified in pick already exists in data, how should it be dealt with? set.na=FALSE (default) only overwrites existing data if

	the interpolator yields a non-NA result; set.na=TRUE passes NA values returned by the interpolator on to the results data.frame
radius	numeric value specifying the radius of the local neighborhood to be used for interpolation; defaults to 200 map units (presumably meters), or, in the functions for grid files, 2.5*cellsize.
nmin	numeric, for method="krige" only: see gstat::krige() function in package **gstat**
nmax	numeric, for method="krige" only: see gstat::krige() function in package **gstat**
sill	numeric, for method="krige" only: the overall sill parameter to be used for the variogram
range	numeric, for method="krige" only: the variogram range
nugget	numeric, for method="krige" only: the nugget effect
model	for method="krige" only: the variogram model to be used for interpolation; defaults to a spherical variogram with parameters specified by the range, sill, and nugget arguments; see gstat::vgm() in package **gstat** for details
log	logical vector, specifying for each variable in pick if interpolation should take place on the logarithmic scale (default: FALSE)
X.name	name of the variable containing the x coordinates
Y.name	name of the variable containing the y coordinates
cbind	logical: should the new variables be added to the input data.frame (cbind=TRUE, the default), or should they be returned as a separate vector or data.frame? cbind=FALSE
shapefile	point shapefile
...	arguments to be passed to pick.from.points, and to internal.pick.from.ascii.grid in the case of pick.from.ascii.grid
file	file name (relative to path, default file extension .asc) of an ASCII grid from which to pick a variable, or an open connection to such a file
path	optional path to file
varname	character string: a variable name for the variable interpolated from grid file file in pick.from.*.grid; if missing, variable name will be determined from filename by a call to create.variable.name()
prefix	an optional prefix to be added to the varname
parallel	logical (default: FALSE): enable parallel processing; requires additional packages such as (formerly) doSNOW or **doMC**. See example below and plyr::ddply()
nsplit	split the data.frame data in nsplit disjoint subsets in order to increase efficiency by using plyr::ddply() in package **plyr**. The default seems to perform well in many situations.
quiet	logical: provide information on the progress of grid processing on screen? (only relevant if at.once=FALSE and method="nearest.neighbour")
nodata.values	numeric vector specifying grid values that should be converted to NA; in addition to the values specified here, the nodata value given in the input grid's header will be used

at.once	logical: should the grid be read as a whole or line by line? at.once=FALSE is useful for processing large grids that do not fit into memory; the argument is currently by default FALSE for method="nearest.neighbour", and it currently MUST be TRUE for all other methods (in these cases, TRUE is the default value); piecewise processing with at.once=FALSE is always faster than processing the whole grid at.once
nlines	numeric: stop after processing nlines lines of the input grid; useful for testing purposes
na.strings	passed on to scan()
filename	character: name of a SAGA grid file, default extension .sgrd
prec	numeric, specifying the number of digits to be used in converting a SAGA grid to an ASCII grid in pick.from.saga.grid
show.output.on.console	a logical (default: FALSE), indicates whether to capture the output of the command and show it on the R console (see system(), rsaga.geoprocessor()).
env	list: RSAGA geoprocessing environment created by rsaga.env()

Details

pick.from.points interpolates the variables defined by pick in the src data.frame to the locations provided by the data data.frame. Only nearest neighbour and ordinary kriging interpolation are currently available. This function is intended for 'data-rich' situations in which not much thought needs to be put into a geostatistical analysis of the spatial structure of a variable. In particular, this function is supposed to provide a simple, 'quick-and-dirty' interface for situations where the src data points are very densely distributed compared to the data locations.

pick.from.shapefile is a front-end of pick.from.points for point shapefiles.

pick.from.ascii.grid retrieves data values from an ASCII raster file using either nearest neighbour or ordinary kriging interpolation. The latter may not be possible for large raster data sets because the entire grid needs to be read into an R matrix. Split-apply-combine strategies are used to improve efficiency and allow for parallelization.

The optional parallelization of pick.from.ascii.grid computation requires the use of a *parallel backend* package such as (formerly) doSNOW or **doMC**, and the parallel backend needs to be registered before calling this function with parallel=TRUE. The example section provides an example using **doSNOW** on Windows. I have seen 25-40% reduction in processing time by parallelization in some examples that I ran on a dual core Windows computer.

pick.from.ascii.grids performs multiple pick.from.ascii.grid calls. File path and prefix arguments may be specific to each file (i.e. each may be a character vector), but all interpolation settings will be the same for each file, limiting the flexibility a bit compared to individual pick.from.ascii.grid calls by the user. pick.from.ascii.grids currently processes the files sequentially (i.e. parallelization is limited to the pick.from.ascii.grid calls within this function).

pick.from.saga.grid is the equivalent to pick.from.ascii.grid for SAGA grid files. It simply converts the SAGA grid file to a (temporary) ASCII raster file and applies pick.from.ascii.grid.

internal.pick.from.ascii.grid is an internal 'workhorse' function that by itself would be very inefficient for large data sets data. This function is called by pick.from.ascii.grid, which uses a split-apply-combine strategy implemented in the **plyr** package.

Value

If `cbind=TRUE`, columns with the new, interpolated variables are added to the input data.frame `data`.

If `cbind=FALSE`, a data.frame only containing the new variables is returned (possibly coerced to a vector if only one variable is processed).

Note

`method="krige"` requires the **gstat** package.

`pick.from.shapefile` requires the **shapefiles** package.

The nearest neighbour interpolation currently randomly breaks ties if `pick.from.points` is used, and in a deterministic fashion (rounding towards greater grid indices, i.e. toward south and east) in the grid functions.

Author(s)

Alexander Brenning

References

Brenning, A. (2008): Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In: J. Boehner, T. Blaschke, L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, 19), 23-32.

See Also

`grid.to.xyz()`, `gstat::vgm()`, `gstat::krige()`, `read.ascii.grid()`, `write.ascii.grid()`

Examples

```
## Not run:
# assume that 'dem' is an ASCII grid and d a data.frame with variables x and y
pick.from.ascii.grid(d, "dem")
# parallel processing on Windows using the doSNOW package:
# ---outdated - doSNOW has been superseded---
## require(doSNOW)
## registerDoSNOW(cl <- makeCluster(2, type = "SOCK")) # DualCore processor
## pick.from.ascii.grid(d, "dem", parallel = TRUE)
# produces two (ignorable) warning messages when using doSNOW
# typically 25-40% faster than the above on my DualCore notebook
## stopCluster(cl)

## End(Not run)

## Not run:
# use the meuse data for some tests:
require(gstat)
data(meuse)
data(meuse.grid)
meuse.nn = pick.from.points(data=meuse.grid, src=meuse,
```

```
    pick=c("cadmium","copper","elev"), method="nearest.neighbour")
meuse.kr = pick.from.points(data=meuse.grid, src=meuse,
    pick=c("cadmium","copper","elev"), method="krige", radius=100)
# it does make a difference:
plot(meuse.kr$cadmium,meuse.nn$cadmium)
plot(meuse.kr$copper,meuse.nn$copper)
plot(meuse.kr$elev,meuse.nn$elev)

## End(Not run)
```

`read.ascii.grid` *Read/write ASCII, SAGA and Rd Grid Files*

Description

These functions provide simple interfaces for reading and writing grids from/to ASCII grids and Rd files. Grids are stored as matrices, their headers in lists.

Usage

```
read.ascii.grid(
  file,
  return.header = TRUE,
  print = 0,
  nodata.values = c(),
  at.once = TRUE,
  na.strings = "NA"
)

read.ascii.grid.header(file, ...)

read.sgrd(
  fname,
  return.header = TRUE,
  print = 0,
  nodata.values = c(),
  at.once = TRUE,
  prec = 7,
  ...
)

read.Rd.grid(fname, return.header = TRUE)

write.ascii.grid(
  data,
  file,
  header = NULL,
  write.header = TRUE,
```

```
  digits,
  hdr.digits = 10,
  dec = ".",
  georef = "corner"
)

write.ascii.grid.header(file, header, georef, dec = ".", hdr.digits = 10)

write.sgrd(
  data,
  file,
  header = NULL,
  prec = 7,
  hdr.prec = 10,
  georef = "corner",
  ...
)

write.Rd.grid(data, file, header = NULL, write.header = TRUE, compress = TRUE)
```

Arguments

file	file name of an ASCII grid (extension defaults to .asc if not specified), or a connection open for reading or writing, as required
return.header	logical: should the grid header be returned (default), or just the grid data matrix? In the former case, read.ascii.grid returns a list with two components named data and header.
print	numeric, specifying how detailed the output reporting the progress should be (currently 0 to 2, 0 being minimum output).
nodata.values	optional numeric vector specifying nodata values to be used in addition to the nodata value specified in the grid header; nodata values are converted to NA.
at.once	logical: if TRUE, read the whole grid with one scan command; if FALSE, read it row by row using scan with option nlines=1.
na.strings	passed on to scan().
...	read.sgrd, write.sgrd: additional arguments to be passed to rsaga.geoprocessor
fname	file name of a grid stored as an R (.Rd) file; extension defaults to .Rd
prec	integer: number of digits of temporary ASCII grid used for importing or exporting a SAGA grid
data	grid data: a data matrix, or a list with components data (the grid data matrix) and header (the grid header information).
header	optional list argument specifying the grid header information as returned by the read.ascii.grid or read.ascii.grid.header function; see Details
write.header	logical: should the header be written with the grid data? (default: TRUE)
digits	numeric: if not missing, write data rounded to this many decimal places
hdr.digits	numeric: see hdr.prec

dec	character (default: "."): decimal mark used in input or output file
georef	character: specifies whether the output grid should be georeferenced by the "center" or "corner" of its lower left grid cell; defaults to "corner".
hdr.prec	numeric: write (non-integer) header data with this many decimal places; a value of 9 or higher is recommended for compatibility with SAGA GIS (default: 10)
compress	logical: should the .Rd file written by write.Rd.file be compressed? (default: TRUE)

Value

The read.* functions return either a list with components data (the grid data matrix) and header (the grid header information, see below), if return.header=TRUE, or otherwise just the grid data matrix return.header=FALSE.

The grid data matrix is a numeric matrix whose first column corresponds to the first (i.e. northernmost) row of the grid. Columns run from left = West to right = East.

The header information returned by the read.ascii.grid[.header] functions (if return.header=TRUE) is a list with the following components:

ncols	Number of grid columns.
nrows	Number of grid rows.
xllcorner	x coordinate of the corner of the lower left grid cell.
yllcorner	y coordinate of the corner of the lower left grid cell.
cellsize	Single numeric value specifying the size of a grid cell or pixel in both x and y direction.
nodata_value	Single numeric value being interpreted as NA (typically -9999.
xllcenter	x coordinate of the center of the lower left grid cell
yllcenter	y coordinate of the center of the lower left grid cell

Note: The order of the components, especially of ?llcorner and ?llcenter, may change, depending on the order in which they appear in the grid header and on the georeferencing method (center or corner) used for the grid. The ?llcorner and ?llcenter attributes differ only by cellsize/2.

Note

read.sgrd and write.sgrd import/export grids indirectly by creating temporary ASCII grid files (this explains why write.sgrd has prec and hdr.prec arguments). Consider using sf::read_sf() in package sf instead, which is likely more efficient but may require coercion of your gridded data to/from an object supported by sf.

The read.Rd.grid and write.Rd.grid functions use the load and save commands to store a grid. The variable name used is data, which is either a numeric matrix or a list with components data (the grid data matrix) and header (the grid header information).

Author(s)

Alexander Brenning

See Also

sf::read_sf() and sf::write_sf() in package sf, and readAsciiGrid and writeAsciiGrid in package maptools

---

relative.position *Relative Topographic Position*

Description

relative.position and relative.rank are used with focal.function() to determine the relative value of a grid cell compared to its surroundings, either on a metric scale or based on ranks.

Usage

```
relative.position(x)

relative.rank(x, ties.method = "average")
```

Arguments

x	a square matrix with the grid data from the moving window, possibly containing NA values
ties.method	see rank()

Value

If x is provided, a numeric value in the interval [0,1] is returned.

If x is missing, a character vector of same length giving suggested variable (or file) names, here "relpos" and "relrank", respectively. See focal.function() for details.

See Also

focal.function(), rank(), centervalue()

Examples

```
m = matrix( round(runif(9,1,10)), ncol=3 )
print(m)
relative.position(m)
relative.rank(m)
## Not run:
focal.function("dem",fun=relative.rank,radius=5)
focal.function("dem",fun=relative.position,radius=5)
relrank = as.vector(read.ascii.grid("relrank")$data)
relpos  = as.vector(read.ascii.grid("relpos")$data)
plot(relpos,relrank,pch=".")
cor(relpos,relrank,use="complete.obs",method="pearson")

## End(Not run)
```

`resid.median` *Residual Median and Quantile Filters for Grids*

Description

These functions use the median and other quantiles to describe the difference between a grid value and its neighborhood. They are designed for use with `focal.function()`.

Usage

```
## S3 method for class 'median'
resid(x)

## S3 method for class 'minmedmax'
resid(x)

## S3 method for class 'quantile'
resid(x, probs)

## S3 method for class 'quartiles'
resid(x)
```

Arguments

`x`	a square matrix with the grid data from the moving window, possibly containing `NA` values
`probs`	numeric vector of probabilities in [0,1] to be passed to `quantile()`

Details

These functions are designed for being called by `focal.function()`, which repeatedly passes the contents of a square or circular moving window to these functions.

The `resid.median` function rests the value of the central grid cell from the median of the whole moving window. Thus, in terms of topography, a positive residual median indicates that this grid cell stands out compared to its surroundings. `resid.quantile` gives more flexibility in designing such residual attributes.

Value

If `x` is provided, a numeric vector of length 1 (`resid.median`), 3 (`resid.minmedmax` and `resid.quartiles`), or `length(probs)` (`resid.quantile`).

If `x` is missing, a character vector of same length giving suggested variable (or file) names, such as `"rmed"`. See `focal.function()` for details.

See Also

`focal.function()`, `quantile()`, `median()`, `centervalue()`

`rsaga.add.grid.values.to.points` *Add Grid Values to Point Shapefile*

Description

Pick values from SAGA grids and attach them as a new variables to a point shapefile.

Usage

```
rsaga.add.grid.values.to.points(
  in.shapefile,
  in.grids,
  out.shapefile,
 method = c("nearest.neighbour", "bilinear", "idw", "bicubic.spline", "b.spline"),
  ...
)
```

Arguments

`in.shapefile`	Input point shapefile (default extension: `.shp`).
`in.grids`	Input: character vector with names of (one or more) SAGA GIS grid files to be converted into a point shapefile.
`out.shapefile`	Output point shapefile (default extension: `.shp`).
`method`	interpolation method to be used; choices: nearest neighbour interpolation (default), bilinear interpolation, inverse distance weighting, bicubic spline interpolation, B-splines.
`...`	Optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment.

Details

Retrieves information from the selected grids at the positions of the points of the selected points layer and adds it to the resulting layer.

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses module `Add Grid Values to Points` in SAGA GIS library `shapes_grid`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA modules)

See Also

pick.from.points(), pick.from.ascii.grid(), pick.from.saga.grid(), rsaga.grid.to.points()

---

rsaga.close.gaps *SAGA Modules Close Gaps and Close One Cell Gaps*

Description

Close (Interpolate) Gaps

Usage

```
rsaga.close.gaps(in.dem, out.dem, threshold = 0.1, ...)

rsaga.close.one.cell.gaps(in.dem, out.dem, ...)
```

Arguments

in.dem	input: digital elevation model (DEM) as SAGA grid file (default file extension: .sgrd)
out.dem	output: DEM grid file without no-data values (gaps). Existing files will be overwritten!
threshold	tension threshold for adjusting the interpolator (default: 0.1)
...	optional arguments to be passed to rsaga.geoprocessor(), including the env RSAGA geoprocessing environment

Details

rsaga.close.one.cell.gaps only fill gaps whose neighbor grid cells have non-missing data.

In rsaga.close.gaps, larger tension thresholds can be used to reduce overshoots and undershoots in the surfaces used to fill (interpolate) the gaps.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses modules 7 (`rsaga.close.gaps` and 6 `rsaga.close.one.cell.gaps` from the SAGA library `grid_tools`.

SAGA GIS 2.0.5+ has a new additional module `Close Gaps with Spline`, which can be accessed using `rsaga.geoprocessor()` (currently no R wrapper available). See `rsaga.get.usage("grid_tools","Close Gaps with Spline")` or in version 2.1.0+ call `rsaga.html.help("grid_tools","Close Gaps with Spline")`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.geoprocessor()`, `rsaga.env()`

Examples

```
## Not run:
# using SAGA grids:
rsaga.close.gaps("rawdem.sgrd","dem.sgrd")
# using ASCII grids:
rsaga.esri.wrapper(rsaga.close.gaps,in.dem="rawdem",out.dem="dem")

## End(Not run)
```

---

rsaga.contour *Contour Lines from a Grid*

Description

Creates a contour lines shapefile from a grid file in SAGA grid format.

Usage

```
rsaga.contour(
  in.grid,
  out.shapefile,
  zstep,
  zmin,
  zmax,
  vertex = "xy",
  env = rsaga.env(),
  ...
)
```

Arguments

in.grid	input: digital elevation model (DEM) as SAGA grid file (default file extension: .sgrd)
out.shapefile	output: contour line shapefile. Existing files will be overwritten!
zstep, zmin, zmax	lower limit, upper limit, and equidistance of contour lines
vertex	optional parameter: vertex type for resulting contours. Default "xy" (or 0). Only available with SAGA GIS 2.1.3+. • 0 "xy" • 1 "xyz"
env	A SAGA geoprocessing environment, see rsaga.env()
...	arguments to be passed to rsaga.geoprocessor()

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

rsaga.geoprocessor()

---

rsaga.copy.sgrd *Create a copy of a SAGA grid file*

Description

Creates a copy of a SAGA grid file, optionally overwriting the target file if it already exists. Intended mainly for internal use by RSAGA functions, currently in particular rsaga.inverse.distance().

Usage

```
rsaga.copy.sgrd(in.grid, out.grid, overwrite = TRUE, env = rsaga.env())
```

Arguments

in.grid	name of a SAGA GIS grid file; file extension can be omitted
out.grid	name of a SAGA GIS grid file; file extension can be omitted
overwrite	logical; if TRUE (the default), overwrite out.grid if it already exists; if FALSE and the out.grid already exists, copying will be skipped without causing an error.
env	a SAGA geoprocessing environment as created by rsaga.env()

Value

a vector containing the results of the multiple `file.copy()` calls

Note

SAGA grid files consist of three (or more) individual files with file extensions `.mgrd`, `.sgrd` and `.sdat`. The files with these three file extensions are copied, any additional files (e.g. a history file) are ignored.

`rsaga.env` *Function to set up RSAGA geoprocessing environment: Set up the RSAGA Geoprocessing Environment*

Description

`rsaga.env` creates a list with system-dependent information on SAGA path, module path and data (working) directory. This kind of a list is required by most RSAGA geoprocessing functions and is referred to as the 'RSAGA geoprocessing environment.'

Usage

```
rsaga.env(
  path = NULL,
  modules = NULL,
  workspace = ".",
  cmd = ifelse(Sys.info()["sysname"] == "Windows", "saga_cmd.exe", "saga_cmd"),
  version = NULL,
  cores,
  parallel = FALSE,
  root = NULL,
  lib.prefix
)
```

Arguments

`path`	path in which to find `cmd`; `rsaga.env` is usually able to find SAGA on your system if it is installed; see Details.
`modules`	path in which to find SAGA libraries; see Details
`workspace`	path of the working directory for SAGA; defaults to the current directory (`"."`).
`cmd`	name of the SAGA command line program; defaults to `saga_cmd.exe`, its name under Windows
`version`	optional character string: SAGA GIS (API) version, e.g. `"2.0.8"`; if missing, a call to `rsaga.get.version()` is used to determine version number of SAGA API

cores	optional numeric argument, or NA: number of cores used by SAGA GIS; supported only by SAGA GIS 2.1.0 (and higher), ignored otherwise (with a warning). Multicore-enabled SAGA GIS modules such as the one used by rsaga.pisr() seem to run in multicore mode by default when this argument is not specified, therefore cores should only be specified to use a smaller number of cores than available on a machine.
parallel	optional logical argument (default: FALSE): if TRUE, run RSAGA functions that are capable of parallel processing in parallel mode; note that this is completely independent of the behavior of SAGA GIS (which can be controlled using the cores argument); currently only some RSAGA functions support parallel processing (e.g., pick.from.ascii.grid() or rsaga.get.modules()). parallel=TRUE requires that a parallel backend such as **doSNOW** or **doMC** is available and has been started prior to calling any parallelized RSAGA function, otherwise warnings may be generated
root	optional root path to SAGA GIS installation. It is used if RSAGA performs a search for the SAGA command line program (s. search). If left empty, on Windows C:/ is used, on Linux /usr and on Mac OS /usr/local/Cellar.
lib.prefix	character string: a possible (platform-dependent) prefix for SAGA GIS library names; if missing (recommended), a call to rsaga.lib.prefix() tries to determine the correct prefix, e.g. "" on Windows systems and "lib" on non-Windows systems with SAGA GIS pre-2.1.0. Try specifying "" or "lib" manually if this causes problems, and contact the package maintainer if the detection mechanism fails on your system (indicate your Sys.info()["sysname"] and your SAGA GIS version)

Details

IMPORTANT: Unlike R functions such as options(), which changes and saves settings somewhere in a global variable, rsaga.env() does not actually 'save' any settings, it simply creates a list that can (and has to) be passed to other rsaga.* functions. See example below.

We strongly recommend to install SAGA GIS on Windows in C:/Program Files/SAGA-GIS, C:/Program Files (x86)/SAGA-GIS,C:/SAGA-GIS, C:/OSGeo4W64/apps/saga-lts or C:/OSGeo4W64/apps/saga. If you use a standalone version of SAGA GIS in a different path, please refer to section 2 bellow.

There are three ways to create a RSAGA environment with rsaga.env:

1. No paths to the SAGA command line program and to the SAGA modules are specified by the user through the arguments path and modules. On Windows rsaga.env tries to find the SAGA command line program in the following folders C:/Progra~1/SAGA, C:/Progra~2/SAGA, C:/Progra~1/SAGA-GIS, C:/Progra~2/SAGA-GIS, C:/SAGA-GIS, C:/OSGeo4W64/apps/saga-lts and C:/OSGeo4W64/apps/saga. If this fails and attempt is being made to find the SAGA command line program with a search on C:/ (The drive letter can be changed with the root argument). The subfolder tools (SAGA Version < 3.0.0 subfolder modules) is checked for the SAGA module libraries. On Unix systems rsaga.env tries to find the SAGA command line program in various default paths. Additionally, on Unix systems the PATH environment variable is checked for the path to the SAGA command line program and the SAGA_MLB environment variable is checked for the SAGA module libraries. If this fails, a search for the SAGA command line program and the module libraries is performed on /usr. If no SAGA command line program can be found, please specify the paths as described in section 2.

2. The user specifies both the path to the SAGA command line program and to the SAGA module libraries. Both paths are checked if they are valid. Use this if SAGA GIS is located in a non-standard path or if you use more than one SAGA GIS version.
3. The user specifies only the path to the SAGA command line program. A search for the SAGA modules is performed as described in section 1.

Value

A list with components `workspace`, `cmd`, `path`, `modules`, `version`, `numeric_version`, `cores` and `parallel` with values as passed to `rsaga.env` or default values as described in the Details section.

Note

Note that the default `workspace` is `"."`, not `getwd()`; i.e. the default SAGA workspace folder is not fixed, it changes each time you change the R working directory using `setwd`.

Author(s)

Alexander Brenning and Marc Becker

See Also

`rsaga.get.version()`

Examples

```
## Not run:
# Check the default RSAGA environment on your computer:
myenv <- rsaga.env()
myenv
# SAGA data in C:/sagadata, binaries in C:/SAGA-GIS, modules in C:/SAGA-GIS/modules:
myenv <- rsaga.env(workspace="C:/sagadata", path="C:/SAGA-GIS")
# Unix: SAGA in /usr/bin (instead of the default /usr/local/bin),
# and modules in /use/lib/saga:
# myenv <- rsaga.env(path="/usr/bin")
# Use the 'myenv' environment for SAGA geoprocessing:
rsaga.hillshade("dem","hillshade",env=myenv)
# ...creates (or overwrites) grid "C:/sagadata/hillshade.sgrd"
# derived from digital elevation model "C:/sagadata/dem.sgrd"

# Same calculation with different SAGA version:
# (I keep several versions in SAGA-GIS_x.x.x folders:)
myenv05 = rsaga.env(path = "C:/Progra~1/SAGA-GIS_2.0.5")
rsaga.hillshade("dem","hillshade205",env=myenv05)

## End(Not run)
```

`rsaga.esri.to.sgrd` *Convert ESRI ASCII/binary grids to SAGA grids*

Description

`rsaga.esri.to.sgrd` converts grid files from ESRI's ASCII (.asc) and binary (.flt) format to SAGA's (version 2) grid format (.sgrd).

Usage

```
rsaga.esri.to.sgrd(
  in.grids,
  out.sgrds = set.file.extension(in.grids, ".sgrd"),
  in.path,
  ...
)
```

Arguments

`in.grids`	character vector of ESRI ASCII/binary grid files (default file extension: `.asc`); files should be located in folder `in.path`
`out.sgrds`	character vector of output SAGA grid files; defaults to `in.grids` with file extension being replaced by `.sgrd`, which is also the default extension if file names without extension are specified; files will be placed in the current SAGA workspace (default: `rsaga.env()$workspace`, or `env$workspace` if an `env` argument is provided
`in.path`	folder with `in.grids`
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

If multiple `in.grids` are converted, the result will be a vector of numerical error codes of the same length, or the combination of the console outputs with `c()`.

Note

This function uses module 1 from the SAGA library `io_grid`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

rsaga.esri.wrapper() for an efficient way of applying RSAGA to ESRI ASCII/binary grids; rsaga.env()

---

rsaga.esri.wrapper *Use RSAGA functions for ESRI grids*

Description

This wrapper converts input grid files provided in ESRI binary (.flt) or ASCII (.asc) formats to SAGA's (version 2) grid format, calls the RSAGA geoprocessing function, and converts the output grids back to the ESRI grid format. Conversion can also be limited to either input or output grids.

Usage

```
rsaga.esri.wrapper(
  fun,
  in.esri = TRUE,
  out.esri = TRUE,
  env = rsaga.env(),
  esri.workspace = env$workspace,
  format = "ascii",
  georef = "corner",
  prec = 5,
  esri.extension,
  condensed.res = TRUE,
  clean.up = TRUE,
  intern = TRUE,
  ...
)
```

Arguments

fun	function: one of the RSAGA geoprocessing functions, such as rsaga.close.gaps() or rsaga.hillshade() etc.
in.esri	logical: are input grids provided as ESRI grids (in.esri=TRUE) or as SAGA grids?
out.esri	logical: should output grids be converted to ESRI grids?
env	RSAGA environment as returned by rsaga.env()
esri.workspace	directory for the input and output ESRI ASCII/binary grids
format	output file format, either "ascii" (default; equivalent: format=1) for ASCII grids or "binary" (equivalent: 0) for binary ESRI grids (.flt).
georef	character: "corner" (equivalent numeric code: 0) or "center" (default; equivalent: 1). Determines whether the georeference will be related to the center or corner of its extreme lower left grid cell.

prec	number of digits when writing floating point values to ASCII grid files (only relevant if out.esri=TRUE).
esri.extension	extension for input/output ESRI grids: defaults to .asc for format="ascii", and to .flt for format="binary"
condensed.res	logical: return only results of the RSAGA geoprocessing function fun (condensed.res=TRUE), or include the results of the import and export operations, i.e. the calls to rsaga.esri.to.sgrd() and rsaga.sgrd.to.esri()? (see Value)
clean.up	logical: delete intermediate SAGA grid files?
intern	intern argument to be passed to rsaga.geoprocessor(); see Value
...	additional arguments for fun; NOTE: ESRI ASCII/float raster file names should NOT include the file extension (.asc, .flt); the file extension is defined by the esri.extension and format arguments!

Details

ESRI ASCII/float raster file names should NOT include the file extension (.asc, .flt); the file extension is defined by the esri.extension and format arguments!

Value

The object returned depends on the condensed.res arguments and the intern argument passed to the rsaga.geoprocessor().

If condensed.res=TRUE and intern=FALSE, a single numerical error code (0: success) is returned. If condensed.res=TRUE and intern=TRUE (default), a character vector with the module's console output is returned (invisibly).

If condensed.res=FALSE the result is a list with components in.res, geoproc.res and out.res. Each of these components is either an error code (for intern=FALSE) or (for intern=TRUE) a character vector with the console output of the input (rsaga.esri.to.sgrd()), the geoprocessing (fun), and the output conversion (rsaga.sgrd.to.esri()) step, respectively. For in.esri=FALSE or out.esri=FALSE, the corresponding component is NULL.

Note

Note that the intermediate grids as well as the output grids may overwrite existing files with the same file names without prompting the user. See example below.

See Also

rsaga.esri.to.sgrd(), rsaga.sgrd.to.esri(), rsaga.geoprocessor(), rsaga.env()

Examples

```
## Not run:
rsaga.esri.wrapper(rsaga.hillshade,in.dem="dem",out.grid="hshd",condensed.res=FALSE,intern=FALSE)
# if successful, returns list(in.res=0,geoproc.res=0,out.res=0),
# and writes hshd.asc; intermediate files dem.sgrd, dem.hgrd, dem.sdat,
# hshd.sgrd, hshd.hgrd, and hshd.sdat are deleted.
# hshd.asc is overwritten if it already existed.
```

```
## End(Not run)
```

`rsaga.fill.sinks` *Fill Sinks*

Description

Several methods for filling closed depressions in digital elevation models that would affect hydrological modeling.

Usage

```
rsaga.fill.sinks(
  in.dem,
  out.dem,
  method = "planchon.darboux.2001",
  out.flowdir,
  out.wshed,
  minslope,
  ...
)
```

Arguments

`in.dem`	Input: digital elevation model (DEM) as SAGA grid file (default extension: `.sgrd`).
`out.dem`	Output: filled, depression-free DEM (SAGA grid file). Existing files will be overwritten!
`method`	The depression filling algorithm to be used (character). One of `"planchon.darboux.2001"` (default), `"wang.liu.2006"`, or `"xxl.wang.liu.2006"`.
`out.flowdir`	(only for `"wang.liu.2001"`): Optional output grid file for computed flow directions (see Notes).
`out.wshed`	(only for `"wang.liu.2001"`): Optional output grid file for watershed basins.
`minslope`	Minimum slope angle (in degree) preserved between adjacent grid cells (default value of `0.01` only for `method="planchon.darboux.2001"`, otherwise no default).
`...`	Optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment.

Details

This function bundles three SAGA modules for filling sinks using three different algorithms (`method` argument).

`"planchon.darboux.2001"`: The algorithm of Planchon and Darboux (2001) consists of increasing the elevation of pixels in closed depressions until the sink disappears and a minimum slope angle of `minslope` (default: `0.01` degree) is established.

`"wang.liu.2006"`: This module uses an algorithm proposed by Wang and Liu (2006) to identify and fill surface depressions in DEMs. The method was enhanced to allow the creation of hydrologically sound elevation models, i.e. not only to fill the depressions but also to preserve a downward slope along the flow path. If desired, this is accomplished by preserving a minimum slope gradient (and thus elevation difference) between cells. This is the fully featured version of the module creating a depression-free DEM, a flow path grid and a grid with watershed basins. If you encounter problems processing large data sets (e.g. LIDAR data) with this module try the basic version (`xxl.wang.lui.2006`).

`"xxl.wang.liu.2006"`: This modified algorithm after Wang and Liu (2006) is designed to work on large data sets.

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

The function writes SAGA grid files containing of the depression-free preprocessed DEM, and optionally the flow directions and watershed basins.

Note

The flow directions are coded as 0 = north, 1 = northeast, 2 = east, ..., 7 = northwest.

If `minslope=0`, depressions will only be filled until a horizontal surface is established, which may not be helpful for hydrological modeling.

Author(s)

Alexander Brenning (R interface), Volker Wichmann (SAGA module)

References

Planchon, O., and F. Darboux (2001): A fast, simple and versatile algorithm to fill the depressions of digital elevation models. Catena 46: 159-176.

Wang, L. & H. Liu (2006): An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. International Journal of Geographical Information Science, Vol. 20, No. 2: 193-213.

See Also

`rsaga.sink.removal()`, `rsaga.sink.route()`.

`rsaga.filter.gauss` *Gauss Filter*

Description

Smooth a grid using a Gauss filter.

Usage

```
rsaga.filter.gauss(
  in.grid,
  out.grid,
  sigma,
  radius = ceiling(2 * sigma),
  env = rsaga.env(),
  ...
)
```

Arguments

`in.grid`	input: SAGA GIS grid file (default file extension: `.sgrd`)
`out.grid`	output: SAGA GIS grid file
`sigma`	numeric, >0.0001: standard deviation parameter of Gauss filter
`radius`	positive integer: radius of moving window
`env`	list, setting up a SAGA geoprocessing environment as created by `rsaga.env()`
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.filter.simple()`

`rsaga.filter.simple` *Simple Filters*

Description

Apply a smoothing, sharpening or edge filter to a SAGA grid.

Usage

```
rsaga.filter.simple(
  in.grid,
  out.grid,
  mode = "circle",
  method = c("smooth", "sharpen", "edge"),
  radius,
  env = rsaga.env(),
  ...
)
```

Arguments

`in.grid`	input: SAGA grid file (default file extension: `.sgrd`)
`out.grid`	output: SAGA grid file
`mode`	character or numeric: shape of moving window, either `"square"` (=0) or `"circle"` (=1, default)
`method`	character or numeric: `"smooth"` (=0), `"sharpen"` (=1), or `"edge"` (=2)
`radius`	positive integer: radius of moving window
`env`	list, setting up a SAGA geoprocessing environment as created by `rsaga.env()`
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.filter.gauss()`

Examples

```
## Not run: rsaga.filter.simple("dem","dem-smooth",radius=4)
```

`rsaga.geoprocessor` *Generic R interface for SAGA modules*

Description

This function is the workhorse of the R–SAGA interface: It calls the SAGA command line tool to run SAGA modules and pass arguments.

Usage

```
rsaga.geoprocessor(
  lib,
  module = NULL,
  param = list(),
  show.output.on.console = TRUE,
  invisible = TRUE,
  intern = TRUE,
  prefix = NULL,
  flags = ifelse(show.output.on.console, "q", "s"),
  cores,
  env = rsaga.env(),
  display.command = FALSE,
  reduce.intern = TRUE,
  check.module.exists = TRUE,
  warn = options("warn")$warn,
  argsep = " ",
  check.parameters = TRUE,
  ...
)
```

Arguments

`lib`	Name of the SAGA library to be called (see Details).
`module`	Number (`>=0`) or name of the module to called within the library `lib` (see Details).
`param`	A list of named arguments to be passed to the SAGA module (see Examples).
`show.output.on.console`	a logical (default: `TRUE`), indicates whether to capture the output of the command and show it on the R console (see `system()`).
`invisible`	a logical, indicates whether the command window should be visible on the screen.
`intern`	a logical, indicates whether to make the output of the command an R object

prefix — optional character string: prefix such as "-h" used in the saga_cmd call; mostly for internal purposes; call saga_cmd -h from the command line for details; see also flags

flags — optional character string indicating any command line flags; supported only by SAGA GIS 2.1.0 (and higher), quietly ignored otherwise: "q": no progress report (the default for show.output.on.console=TRUE); "r": no messages report; "s": silent mode, i.e. no progress and no messages report (the default for show.output.on.console=FALSE); other flag options probably not relevant within RSAGA

cores — optional numeric argument, or NA: number of cores used by SAGA GIS; supported only by SAGA GIS 2.1.0 (and higher), ignored otherwise (with a warning); overwrites the cores setting specified in the env argument (see rsaga.env()). Multicore-enabled SAGA GIS modules such as the one used by rsaga.pisr() seem to run in multicore mode by default when this argument is not specified, therefore cores should only be specified to use a smaller number of cores than available on a machine.

env — A SAGA geoprocessing environment, i.e. a list with information on the SAGA and SAGA modules paths and the name of the working directory in which to look for input and output files. (Defaults: see rsaga.env().)

display.command — Display the DOS command line for executing the SAGA module (including all the arguments to be passed). Default: FALSE.

reduce.intern — If intern=TRUE, reduce the text output of SAGA returned to R by eliminating redundant lines showing the progress of module execution etc. (default: TRUE).

check.module.exists — logical (default: TRUE): call rsaga.module.exists() to determine if the specified module can be called in the current SAGA installation

warn — logical (default: TRUE): for internal purposes - can be used to suppress warning messages generated by failed SAGA_CMD calls; currently used by rsaga.get.lib.modules() and related functions; see options() argument warn for details

argsep — character (default: " "; currently for internal use): defines the character symbol used as a separator between each argument name and argument value passed to saga_cmd. SAGA GIS 2.1.0 (RC1) seems to move toward "=" as a separator, but " " still works and some modules (e.g. the used by rsaga.pisr) don't seem to work with argsep="=". Future releases of RSAGA may change the default argsep value and/or delete or ignore this argument and/or move it to rsaga.env().

check.parameters — logical(default: TRUE): Check if correct parameters are used.

... — Additional arguments to be passed to base::system().

Details

This workhorse function establishes the interface between the SAGA command line program and R by submitting a system call. This is a low-level function that may be used for directly accessing

SAGA; specific functions such as rsaga.hillshade are intended to be more user-friendly interfaces to the most frequently used SAGA modules. These higher-level interfaces support default values for the arguments and perform some error checking; they should therefore be preferred if available.

A warning is issued if the RSAGA version is not one of 2.0.4-2.0.8 or 2.1.0-2.1.4

Value

The type of object returned depends on the intern argument passed to system().

If intern=FALSE, a numerical error/success code is returned, where a value of 0 corresponds to success and a non-zero value indicates an error. Note however that the function always returns a success value of 0 if wait=FALSE, i.e. if it does not wait for SAGA to finish.

If intern=TRUE (default), the console output of SAGA is returned as a character vector. This character vector lists the input file names and modules arguments, and gives a more or less detailed report of the function's progress. Redundant information can be removed by setting reduce.intern=TRUE.

Note

Existing output files will be overwritten by SAGA without prompting!

If a terrain analysis function is not directly interfaced by one of the RSAGA functions, you might still find it in the growing set of SAGA libraries and modules. The names of all libraries available in your SAGA installation can be obtained using rsaga.get.libraries() (or by checking the directory listing of the modules folder in the SAGA directory). The names and numeric codes of all available modules (globally or within a specific library) are retrieved by rsaga.get.modules(). Full-text search in library and module names is performed by rsaga.search.modules(). For information on the usage of SAGA command line modules, see rsaga.get.usage(), or the RSAGA interface function if available.

display.command=TRUE is mainly intended for debugging purposes to check if all arguments are passed correctly to SAGA CMD.

Author(s)

Alexander Brenning (R interface); Olaf Conrad and the SAGA development team (SAGA development)

References

Brenning, A., 2008. Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. In J. Boehner, T. Blaschke and L. Montanarella (eds.), SAGA - Seconds Out (= Hamburger Beitraege zur Physischen Geographie und Landschaftsoekologie, vol. 19), p. 23-32.

See Also

rsaga.env(), rsaga.get.libraries(), rsaga.get.modules(), rsaga.search.modules(), rsaga.get.usage(); rsaga.esri.wrapper() for a wrapper for ESRI ASCII/binary grids; rsaga.hillshade() and other higher-level functions.

Examples

```
## Not run:
rsaga.hillshade("dem","hillshade",exaggeration=2)
# using the RSAGA geoprocessor:
rsaga.geoprocessor("ta_lighting",0,list(ELEVATION="dem.sgrd",SHADE="hillshade",EXAGGERATION=2))
# equivalent DOS command line call:
# saga_cmd.exe ta_lighting 0 -ELEVATION dem.sgrd -SHADE hillshade -EXAGGERATION 2

## End(Not run)
```

---

rsaga.get.modules *Find SAGA libraries and modules*

Description

These functions list the SAGA libraries (`rsaga.get.libraries`) and modules (`rsaga.get.lib.modules`, `rsaga.get.modules`) available in a SAGA installation, and allow performing a full-text search among these functions.

Usage

```
rsaga.get.modules(
  libs,
  env = rsaga.env(),
  interactive = FALSE,
  parallel = env$parallel
)

rsaga.get.libraries(path = rsaga.env()$modules, dll)

rsaga.get.lib.modules(lib, env = rsaga.env(), interactive = FALSE)

rsaga.module.exists(libs, module, env = rsaga.env(), ...)

rsaga.search.modules(
  text,
  modules,
  search.libs = TRUE,
  search.modules = TRUE,
  env = rsaga.env(),
  ignore.case = TRUE,
  ...
)
```

Arguments

`libs`	character vector with the names of libraries in which to look for modules; if missing, all libraries will be processed

env	a SAGA geoprocessing environment as created by rsaga.env()
interactive	logical (default FALSE): should modules be returned that can only be executed in interactive mode (i.e. using SAGA GUI)?
parallel	logical (defaults to env$parallel): if TRUE, run in parallel mode; requires a parallel backend such as **doSNOW** or **doMC**
path	path of SAGA library files (modules subfolder in the SAGA installation folder); defaults to the path determined by rsaga.env().
dll	file extension of dynamic link libraries
lib	character string with the name of the library in which to look for modules
module	module name or numeric code
...	currently only interactive to be passed on to rsaga.get.lib.modules
text	character string to be searched for in the names of available libraries and/or modules
modules	optional list: result of rsaga.get.modules; if missing, a list of available modules will be retrieved using that function
search.libs	logical (default TRUE); see search.modules
search.modules	logical (default TRUE): should text be searched for in library and/or module names?
ignore.case	logical (default FALSE): should the text search in library/module names be case-sensitive?

Value

rsaga.get.libraries returns a character vector with the names of all SAGA libraries available in the folder env$modules.

rsaga.get.lib.modules returns a data.frame with:

- name: the names of all modules in library lib
- code: their numeric identifiers
- interactive: a logical variable indicating whether a module can only be executed in interactive (SAGA GUI) mode

rsaga.get.modules returns a list with, for each SAGA library in libs, a data.frame with module information as given by rsaga.get.lib.modules. If libs is missing, all modules in all libraries will be retrieved.

Note

For information on the usage of SAGA command line modules, see rsaga.get.usage(), or rsaga.html.help() (in SAGA GIS 2.1.0+), or the RSAGA interface function, if available.

See Also

rsaga.get.usage(), rsaga.html.help(), rsaga.geoprocessor(), rsaga.env()

Examples

```
## Not run:
# make sure that 'rsaga.env' can find 'saga_cmd.exe'
# before running this:
rsaga.get.libraries()
# list all modules in my favorite libraries:
rsaga.get.modules(c("io_grid", "grid_tools", "ta_preprocessor",
    "ta_morphometry", "ta_lighting", "ta_hydrology"))
# list *all* modules (quite a few!):
# rsaga.get.modules(interactive=TRUE)

# find modules that remove sink from DEMs:
rsaga.search.modules("sink")
# find modules that close gaps (no-data areas) in grids:
rsaga.search.modules("gap")

## End(Not run)
```

---

rsaga.get.modules.path

Internal functions that determine OS-specific path in which modules might be located.

Description

Internal functions that determine OS-specific path in which modules might be located.

Usage

```
rsaga.get.modules.path(sysname = Sys.info()["sysname"], saga.path, root, cmd)
```

Arguments

sysname	character: name of the operating system, determined by default by base::Sys.info(): e.g., "Windows", "Linux", "Darwin" (for Mac OSX), or "FreeBSD"
saga.path	character: path with SAGA GIS binaries, as determined (e.g.) by rsaga.default.path
root	root path to SAGA GIS installation
cmd	name of the SAGA command line program

Value

character string: identified path with SAGA modules; stops with an error if not found

rsaga.get.usage *Usage of SAGA command line modules*

Description

rsaga.get.usage provides information on the usage of and arguments required by SAGA command line modules.

Usage

```
rsaga.get.usage(lib, module, env = rsaga.env(), show = TRUE)
```

Arguments

lib	name of the SAGA library
module	name or numeric identifier of SAGA module in library lib
env	a SAGA geoprocessing environment as created by rsaga.env()
show	logical (default: TRUE); display usage in the R console?

Details

This function is intended to provide information required to use the rsaga.geoprocessor() and for writing your own high-level interface function for SAGA modules. R–SAGA interfaces already exist for some SAGA modules, e.g. rsaga.hillshade(), rsaga.local.morphometry(), but there are many more.

Value

The character vector with usage information is invisibly returned.

See Also

rsaga.html.help(), rsaga.geoprocessor(), rsaga.env(), rsaga.get.modules()

Examples

```
## Not run:
rsaga.get.usage("io_grid",1)
rsaga.get.usage("ta_preprocessor",2)
rsaga.get.usage("ta_morphometry",0)
# in SAGA GIS 2.1.0+, compare:
rsaga.html.help("io_grid",1)
# etc.

## End(Not run)
```

`rsaga.get.version` *Determine SAGA GIS version*

Description

Determine SAGA GIS version.

Usage

```
rsaga.get.version(env = rsaga.env(version = NA), ...)
```

Arguments

`env`	list, setting up a SAGA geoprocessing environment as created by `rsaga.env()`. Note that `version=NA` ensures that `rsaga.env()` won't call `rsaga.get.version` itself.
`...`	additional arguments to `rsaga.geoprocessor()`

Details

The function first attempts to determine the SAGA version directly through a system call `saga_cmd --version`, which is supported by SAGA GIS 2.0.8+. If this fails, `saga_cmd -h` is called, and it is attempted to extract the version number of the SAGA API from the output generated, which works for 2.0.4 - 2.0.7.

Value

A character string defining the SAGA GIS (API) version. E.g., `"2.0.8"`.

See Also

`rsaga.env()`

Examples

```
## Not run:
myenv <- rsaga.env()
myenv$version
# rsaga.env actually calls rsaga.get.version:
rsaga.get.version()

# I keep several versions of SAGA GIS in SAGA-GIS_2.0.x folders:
myenv05 = rsaga.env(path = "C:/Progra~1/SAGA-GIS_2.0.5", version = NA)
# Check if it's really version 2.0.5 as suggested by the folder name:
rsaga.get.version(env = myenv05)

## End(Not run)
```

`rsaga.grid.calculus` *SAGA Module Grid Calculus*

Description

Perform Arithmetic Operations on Grids

Usage

```
rsaga.grid.calculus(in.grids, out.grid, formula, env = rsaga.env(), ...)

rsaga.linear.combination(
  in.grids,
  out.grid,
  coef,
  cf.digits = 16,
  remove.zeros = FALSE,
  remove.ones = TRUE,
  env = rsaga.env(),
  ...
)
```

Arguments

`in.grids`	input character vector: SAGA grid files (default file extension: `.sgrd`)
`out.grid`	output: grid file resulting from the cell-by-cell application of 'formula' to the grids. Existing files will be overwritten!
`formula`	character string of formula specifying the arithmetic operation to be performed on the `in.grids` (see Details); if this is a formula, only the right hand side will be used.
`env`	RSAGA geoprocessing environment, generated by a call to `rsaga.env()`
`...`	optional arguments to be passed to `rsaga.geoprocessor()`
`coef`	numeric: coefficient vector to be used for the linear combination of the `in.grids`. If `coef` as one more element than `in.grids`, the first one will be interpreted as an intercept.
`cf.digits`	integer: number of digits used when converting the coefficients `coef` to character strings (trailing zeros will be removed)
`remove.zeros`	logical: if `TRUE`, terms (grids) with coefficient (numerically) equal to zero (after rounding to `cf.digits` digits) will be removed from the formula
`remove.ones`	logical: if `TRUE` (the default), factors equal to 1 (after rounding to `cf.digits` digits) will be removed from the formula

Details

The `in.grids` are represented in the `formula` by the letters `a` (for `in.grids[1]`), `b` etc. Thus, if `in.grids[1]` is Landsat TM channel 3 and `in.grids[2]` is channel 4, the NDVI formula (TM3-TM4)/(TM3+TM4) can be represented by the character string `"(a-b)/(a+b)"` (any spaces are removed) or the formula `~(a-b)/(a+b)` in the `formula` argument.

In addition to +, -, *, and /, the following operators and functions are available for the `formula` definition: `+ ^` power + `sin(a)` sine + `cos(a)` cosine + `tan(a)` tangent + `asin(a)` arc sine + `acos(a)` arc cosine + `atan(a)` arc tangent + `atan2(a,b)` arc tangent of b/a + `abs(a)` absolute value + `int(a)` convert to integer + `sqr(a)` square + `sqrt(a)` square root + `ln(a)` natural logarithm + `log(a)` base 10 logarithm + `mod(a,b)` modulo + `gt(a, b)` returns 1 if a greater b + `lt(a, b)` returns 1 if a lower b + `eq(a, b)` returns 1 if a equal b + `ifelse(switch, x, y)` returns x if switch equals 1 else y

Using `remove.zeros=FALSE` might have the side effect that no data areas in the grid with coefficient 0 are passed on to the results grid. (To be confirmed.)

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`local.function()`, `focal.function()`, and `multi.focal.function()` for a more flexible framework for combining grids or applying local and focal functions; `rsaga.geoprocessor()`, `rsaga.env()`

Examples

```
## Not run:
# using SAGA grids:
# calculate the NDVI from Landsat TM bands 3 and 4:
rsaga.grid.calculus(c("tm3.sgrd","tm4.sgrd"), "ndvi.sgrd", ~(a-b)/(a+b))
# apply a linear regression equation to grids:
coefs = c(20,-0.6)
# maybe from a linear regression of mean annual air temperature (MAAT)
# against elevation - something like:
# coefs = coef( lm( maat ~ elevation ) )
rsaga.linear.combination("elevation.sgrd", "maat.sgrd", coefs)
# equivalent:
rsaga.grid.calculus("elevation.sgrd", "maat.sgrd", "20 - 0.6*a")

## End(Not run)
```

`rsaga.grid.to.points` *Convert SAGA grid file to point shapefile*

Description

Convert SAGA grid file to point (or polygon) shapefile - either completely or only a random sample of grid cells.

Usage

```
rsaga.grid.to.points(
  in.grids,
  out.shapefile,
  in.clip.polygons,
  exclude.nodata = TRUE,
  type = "nodes",
  env = rsaga.env(),
  ...
)

rsaga.grid.to.points.randomly(in.grid, out.shapefile, freq, ...)
```

Arguments

`in.grids`	Input: names of (possibly several) SAGA GIS grid files to be converted into a point shapefile.
`out.shapefile`	Output: point shapefile (default extension: `.shp`). Existing files will be overwritten!
`in.clip.polygons`	optional polygon shapefile to be used for clipping/masking an area
`exclude.nodata`	logical (default: `TRUE`), in newer SAGA versions numeric (see Note below): skip 'nodata' grid cells?
`type`	character string: `"nodes"`: create point shapefile of grid center points; `"cells"` (only supported by SAGA GIS 2.0.6+): create polygon shapefile with grid cell boundaries
`env`	RSAGA geoprocessing environment created by `rsaga.env()`; required by `rsaga.grid.to.points` to determine version-dependent SAGA module name and arguments
`...`	Optional arguments to be passed to `rsaga.geoprocessor()`
`in.grid`	Input: SAGA grid file from which to sample.
`freq`	integer >=1: sampling frequency: on average 1 out of 'freq' grid cells are selected

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

These functions use modules `Grid Cells to Points/Polygons` (previously called `Grid Values to Points` and in some earlier versions `Grid Values to Shapes`) and `Grid Values to Points (randomly)` in SAGA library `shapes_grid`.

The SAGA 2.0.6+ version of this module is more flexible as it allows to create grid cell polygons instead of center points (see argument `type`).

Since somewhere between SAGA 9.0.0 and 9.3.3, the `NODATA` argument is numeric with values of 0: include all cells, 1: include cell if at least one grid provides data, and 2: exclude cell if at least one grid does not provide data. From 9.3.0 up, `exclude.nodata=TRUE` is interpreted as (and converted to) 1.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA modules)

See Also

`rsaga.add.grid.values.to.points()`

Examples

```
## Not run:
# one point per grid cell, exclude nodata areas:
rsaga.grid.to.points("dem", "dempoints")
# take only every 20th point, but to not exclude nodata areas:
rsaga.grid.to.points.randomly("dem", "dempoints20", freq = 20)

## End(Not run)
```

---

rsaga.hillshade *Analytical hillshading Analytical hillshading calculation.*

Description

Analytical hillshading Analytical hillshading calculation.

Usage

```
rsaga.hillshade(
  in.dem,
  out.grid,
  method = "standard",
  azimuth = 315,
  declination = 45,
  exaggeration = 4,
  ...
)
```

Arguments

in.dem	Input digital elevation model (DEM) as SAGA grid file (default extension: .sgrd).
out.grid	Output hillshading grid (SAGA grid file). Existing files will be overwritten!
method	Available choices (character or numeric): "standard" (or 0 - default), "max90deg.standard" (1), "combined.shading" (2), "ray.tracing" (3). See Details.
azimuth	Direction of the light source, measured in degree clockwise from the north direction; default 315, i.e. northwest.
declination	Declination of the light source, measured in degree above the horizon (default 45).
exaggeration	Vertical exaggeration of elevation (default: 4). The terrain exaggeration factor allows to increase the shading contrasts in flat areas.
...	Optional arguments to be passed to rsaga.geoprocessor(), including the env RSAGA geoprocessing environment.

Details

The Analytical Hillshading algorithm is based on the angle between the surface and the incoming light beams, measured in radians.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

While the default azimuth of 315 degree (northwest) is not physically meaningful on the northern hemisphere, a northwesterly light source is required to properly depict relief in hillshading images. Physically correct southerly light sources results a hillshade that would be considered by most people as inverted: hills look like depressions, mountain chains like troughs.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

rsaga.solar.radiation(), rsaga.insolation()

Examples

```
## Not run: rsaga.hillshade("dem.sgrd","hillshade")
```

`rsaga.html.help` *HTML help on a SAGA module or library*

Description

This function opens SAGA's HTML documentation for the specified library or module. Works with SAGA GIS 2.1.0(+), for earlier versions a web page with the SAGA GIS wiki is displayed.

Usage

```
rsaga.html.help(
  lib,
  module = NULL,
  use.program.folder = TRUE,
  env = rsaga.env(),
  ...
)
```

Arguments

`lib`	name of the SAGA library, or one of the `rsaga.` module functions such as `rsaga.hillshade()`
`module`	name or numeric identifier of SAGA module in library `lib`; `module=NULL` takes you to the main help page of the SAGA library `lib`
`use.program.folder`	logical; if `TRUE` (the default), attempt to write SAGA GIS documentation to a `"help"` subfolder of `env$path`; the `"help"` folder is created if it doesn't exist. If `FALSE`, create SAGA GIS documentation files in this R session's temporary folder as obtained using `tempdir()`
`env`	a SAGA geoprocessing environment as created by `rsaga.env()`
`...`	additional arguments to `browseURL()`

Details

Requires SAGA GIS 2.1.0(+), with earlier versions use `rsaga.get.usage()`.

Value

Returns nothing

See Also

`rsaga.get.usage()`, `rsaga.geoprocessor()`, `rsaga.env()`

Examples

```
## Not run:
# Requires SAGA GIS 2.1.0+:
rsaga.html.help("io_grid")
rsaga.html.help("io_grid",0)
rsaga.html.help("io_grid","Import ESRI Arc/Info Grid")

## End(Not run)
```

`rsaga.import.gdal` *Import Grid Files to SAGA grid format using GDAL*

Description

These functions provide simple interfaces for reading and writing grids from/to ASCII grids and Rd files. Grids are stored in matrices, their headers in lists.

Usage

```
rsaga.import.gdal(in.grid, out.grid, env = rsaga.env(), ...)
```

Arguments

`in.grid`	file name of a grid in a format supported by GDAL
`out.grid`	output SAGA grid file name; defaults to `in.grid` with the file extension being removed; file extension should not be specified, it defaults to `.sgrd`
`env`	RSAGA geoprocessing environment created by `rsaga.env()`
`...`	additional arguments to be passed to `rsaga.geoprocessor`

Details

The GDAL Raster Import module of SAGA imports grid data from various file formats using the Geospatial Data Abstraction Library (GDAL) by Frank Warmerdam. GDAL Versions are specific to SAGA versions:

- SAGA 2.1.2 - 2.2.0: GDAL v.1.11.0
- SAGA 2.2.1 - 2.2.3: GDAL v.2.1.0 dev
- ...
- SAGA 8.4.1: GDAL v3.3.0 More information is available at https://gdal.org/.

If `in.grid` has more than one band (e.g. RGB GEOTIFF), then output grids with file names of the form $in.grid\_01.sgrd$, $in.grid\_02.sgrd$ etc. are written, one for each band.

Numerous raster formats are currently supported. For SAGA 8.4.1 see e.g. https://saga-gis.sourceforge.io/saga_tool_doc/8.4.1/io_gdal_0.html

Value

the result of the `rsaga.geoprocessor()` call

Author(s)

Alexander Brenning (R interface), Olaf Conrad / Andre Ringeler (SAGA module), Frank Warmerdam (GDAL)

References

GDAL website: https://gdal.org/

See Also

`read.ascii.grid`, `rsaga.esri.to.sgrd`, `read.sgrd`, `read.Rd.grid`

---

rsaga.insolation *Incoming Solar Radiation (Insolation)*

Description

This function calculates the amount of incoming solar radiation (insolation) depending on slope, aspect, and atmospheric properties. Module not available in SAGA GIS 2.0.6 and 2.0.7.

Usage

```
rsaga.insolation(
  in.dem,
  in.vapour,
  in.latitude,
  in.longitude,
  out.direct,
  out.diffuse,
  out.total,
  horizontal = FALSE,
  solconst = 8.164,
  atmosphere = 12000,
  water.vapour.pressure = 10,
  type = c("moment", "day", "range.of.days", "same.moment.range.of.days"),
  time.step = 1,
  day.step = 5,
  days,
  moment,
  latitude,
  bending = FALSE,
  radius = 6366737.96,
  lat.offset = "user",
```

```
  lat.ref.user = 0,
  lon.offset = "center",
  lon.ref.user = 0,
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	Name of input digital elevation model (DEM) grid in SAGA grid format (default extension: .sgrd)
in.vapour	Optional input: SAGA grid file giving the water vapour pressure in mbar
in.latitude	Optional input: SAGA grid file giving for each pixel the latitude in degree
in.longitude	Optional input: SAGA grid file giving for each pixel the longitude in degree
out.direct	Optional output grid file for direct insolation
out.diffuse	Optional output grid file for diffuse insolation
out.total	Optional output grid file for total insolation, i.e. the sum of direct and diffuse insolation
horizontal	logical; project radiation onto a horizontal surface? (default: FALSE, i.e. use the actual inclined surface as a reference area)
solconst	solar constant in Joule; default: 8.164 J/cm2/min (=1360.7 kWh/m2; the more commonly used solar constant of 1367 kWh/m2 corresponds to 8.202 J/cm2/min)
atmosphere	height of atmosphere in m; default: 12000m
water.vapour.pressure	if no water vapour grid is given, this argument specifies a constant water vapour pressure that is uniform in space; in mbar, default 10 mbar
type	type of time period: "moment" (equivalent: 0) for a single instant, "day" (or 1) for a single day, "range.of.days" (or 2), or "same.moment.range.of.days" (or 3) for the same moment in a range of days; default: "moment"
time.step	time resolution in hours for discretization within a day
day.step	time resolution in days for a range of days
days	numeric vector of length 2, specifying the first and last day of a range of days (for types 2 and 3)
moment	if type="moment" or "same.moment.range.of.days", moment specifies the time of the day (hour between 0 and 24) for which the insolation is to be calculated
latitude	if no in.latitude grid is given, this will specify a fixed geographical latitude for the entire grid
bending	should planetary bending be modeled? (default: FALSE)
radius	planetary radius
lat.offset	latitude relates to grids "bottom"(equivalent code: 0), "center" (1), "top" (2), or "user"-defined reference (default: "user"); in the latter case, lat.ref.user defines the reference

lat.ref.user	if in.latitude is missing and lat.offset="user", then this numeric value defines the latitudinal reference (details??)
lon.offset	local time refers to grid's "left" edge (code 0), "center" (1), "right" edge (2), or a "user"-defined reference.
lon.ref.user	if in.longitude is missing and lon.offset="user", then this numeric value defines the reference of the local time (details??)
env	RSAGA geoprocessing environment obtained with rsaga.env(); this argument is required for version control (see Note)
...	optional arguments to be passed to rsaga.geoprocessor(), including the env RSAGA geoprocessing environment

Details

Calculation of incoming solar radiation (insolation). Based on the SADO (System for the Analysis of Discrete Surfaces) routines developed by Boehner & Trachinow.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses module Insolation (code: 3) from SAGA library ta_lighting. It is available in SAGA GIS 2.0.4 and 2.0.5 but not 2.0.6 and 2.0.7; see rsaga.pisr().

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

rsaga.solar.radiation(), rsaga.pisr(), rsaga.hillshade()

---

rsaga.intersect.polygons

Spatial intersection of two polygon layers

Description

The function rsaga.intersect.polygons calculates the geometric intersection of two overlayed polygon layers using SAGA module "Intersect".

Usage

```
rsaga.intersect.polygons(
  layer_a = NULL,
  layer_b = NULL,
  result = NULL,
  split = FALSE,
  load = NULL,
  env = rsaga.env()
)
```

Arguments

layer_a	A character string representing the path to a polygon shapefile.
layer_b	A character string representing the path to a polygon shapefile with which to intersect layer_a.
result	A character string indicating where the resulting shapefile should be stored.
split	If TRUE, multipart polygons become separated polygons (default: FALSE).
load	Deprecated, will be removed in a future release. Ignored if FALSE, and causes an error if TRUE (default: NULL).
env	RSAGA geoprocessing environment created by rsaga.env().

Details

Function gIntersection in rgeos package can also be used to define the intersection between two polygon layers. However, rsaga.intersect.polygons() will be usually much faster, especially when intersecting thousands of polygons.

Value

The function saves the output shapefile to the path indicated in function argument result.

Author(s)

Jannes Muenchow and Alexander Brenning (R interface), Olaf Conrad and Angus Johnson (SAGA modules)

---

rsaga.inverse.distance

Spatial Interpolation Methods

Description

Spatial interpolation of point data using inverse distance to a power (inverse distance weighting, IDW), nearest neighbors, or modified quadratic shephard.

Usage

```
rsaga.inverse.distance(
  in.shapefile,
  out.grid,
  field,
  power = 1,
  maxdist,
  nmax = 100,
  target,
  env = rsaga.env(),
  ...
)

rsaga.nearest.neighbour(
  in.shapefile,
  out.grid,
  field,
  target,
  env = rsaga.env(),
  ...
)

rsaga.modified.quadratic.shephard(
  in.shapefile,
  out.grid,
  field,
  quadratic.neighbors = 13,
  weighting.neighbors = 19,
  target,
  env = rsaga.env(),
  ...
)

rsaga.triangulation(
  in.shapefile,
  out.grid,
  field,
  target,
  env = rsaga.env(),
  ...
)
```

Arguments

in.shapefile	Input: point shapefile (default extension: .shp).
out.grid	Output: filename for interpolated grid (SAGA grid file). Existing files will be overwritten!

field	numeric or character: number or name of attribute in the shapefile's attribute table to be interpolated; the first attribute is represented by a zero.
power	numeric (>0): exponent used in inverse distance weighting (usually 1 or 2)
maxdist	numeric: maximum distance of points to be used for inverse distance interpolation (search radius); no search radius is applied when this argument is missing or equals Inf
nmax	Maximum number of nearest points to be used for interpolation; nmax=Inf is a valid value (no upper limit)
target	required argument of type list: parameters identifying the target area, e.g. the x/y extent and cellsize, or name of a reference grid; see rsaga.target().
env	RSAGA geoprocessing environment created by rsaga.env(), required because module(s) depend(s) on SAGA version
...	Optional arguments to be passed to rsaga.geoprocessor(), including the env RSAGA geoprocessing environment.
quadratic.neighbors	integer >=5; default 13.
weighting.neighbors	integer >=3; default 19.

Details

These functions use modules from the grid_gridding SAGA GIS library. They do not support SAGA GIS 2.0.4, which differs in some argument names and parameterizations. Target grid parameterization by grid file name currently doesn't work with SAGA GIS 2.1.0 Release Candidate 1 (see also rsaga.target()); stay tuned for future updates and fixes.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

The 'Inverse Distance Weighted' module of SAGA GIS not only support inverse-distance weighted interpolation, but also exponential and other weighting schemes (command line argument WEIGHTING); these are however not accessible through this function, but only through the rsaga.geoprocessor, if needed. See rsaga.get.usage("grid_gridding","Inverse Distance Weighted") for details.

See the example section in the help file for shapefiles::write.shapefile() in package shapefiles to learn how to apply these interpolation functions to a shapefile exported from a data.frame.

Modified Quadratic Shephard method: based on module 660 in TOMS (see references).

Author(s)

Alexander Brenning (R interface), Andre Ringeler and Olaf Conrad (SAGA modules)

References

QSHEP2D: Fortran routines implementing the Quadratic Shepard method for bivariate interpolation of scattered data (see R. J. Renka, ACM TOMS 14 (1988) pp.149-150). Classes: E2b. Interpolation of scattered, non-gridded multivariate data.

See Also

rsaga.target(); gstat::idw() in package gstat.

---

rsaga.lib.prefix *Determine prefix for SAGA GIS library names*

Description

Internal function that determines the possible prefix for SAGA GIS library names - relevant for non-Windows SAGA GIS pre-2.1.0.

Usage

```
rsaga.lib.prefix(env)
```

Arguments

env	list, setting up a SAGA geoprocessing environment as created by rsaga.env().

Details

Some non-Windows versions of saga_cmd require library names with a "lib" prefix, e.g. libio_grid instead of io_grid. This function, which is called by rsaga.env() tries to guess this behavior based on the operating system and SAGA GIS version.

Value

A character string, either "" or "lib".

See Also

rsaga.env()

Examples

```
## Not run:
env = rsaga.env()
# obtained by a call to rsaga.lib.prefix:
env$lib.prefix

# more explicitly:
rsaga.lib.prefix(env=env)

## End(Not run)
```

`rsaga.local.morphometry` *Local Morphometry*

Description

Calculates local morphometric terrain attributes (i.e. slope, aspect and curvatures). Intended for use with SAGA versions 2.1.0 and older. Use `rsaga.slope.asp.curv()` for SAGA 2.1.1+

Usage

```
rsaga.local.morphometry(
  in.dem,
  out.slope,
  out.aspect,
  out.curv,
  out.hcurv,
  out.vcurv,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)

rsaga.slope(
  in.dem,
  out.slope,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)

rsaga.aspect(
  in.dem,
  out.aspect,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)

rsaga.curvature(
  in.dem,
  out.curv,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)
```

```
rsaga.plan.curvature(
  in.dem,
  out.hcurv,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)

rsaga.profile.curvature(
  in.dem,
  out.vcurv,
  method = "poly2zevenbergen",
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	input: digital elevation model (DEM) as SAGA grid file (default file extension: .sgrd)
out.slope	optional output: slope (in radians)
out.aspect	optional output: aspect (in radians; north=0, clockwise angles)
out.curv	optional output: curvature
out.hcurv	optional output: horizontal curvature (plan curvature)
out.vcurv	optional output: vertical curvature (profile curvature)
method	character (or numeric): algorithm (see References): • 0 Maximum Slope - Travis et al. (1975) ("maxslope", or 0) • 1 Max. Triangle Slope - Tarboton (1997) ("maxtriangleslope", or 1) • 2 Least Squares Fit Plane - Costa-Cabral and Burgess (1996) ("lsqfitplane", or 2) • 3 Fit 2nd Degree Polynomial - Bauer et al. (1985) ("poly2bauer", or 3) • 4 Fit 2nd Degree Polynomial - Heerdegen and Beran (1982) ("poly2heerdegen", or 4) • 5 default: Fit 2nd Degree Polynomial - Zevenbergen and Thorne (1987) ("poly2zevenbergen", or 5) • 6 Fit 3rd Degree Polynomial - Haralick (1983) ("poly3haralick", or 6).
env	list, setting up a SAGA geoprocessing environment as created by rsaga.env()
...	further arguments to rsaga.geoprocessor()

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Author(s)

Alexander Brenning and Donovan Bangs (R interface), Olaf Conrad (SAGA module)

References

For references and algorithm changes in SAGA GIS 2.1.1+ see `rsaga.slope.asp.curv()`.

See Also

`rsaga.slope.asp.curv()`, `rsaga.parallel.processing()`, `rsaga.geoprocessor()`, `rsaga.env()`

Examples

```
## Not run:
# a simple slope algorithm:
rsaga.slope("lican.sgrd","slope","maxslope")
# same for ASCII grids (default extension .asc):
rsaga.esri.wrapper(rsaga.slope,in.dem="lican",out.slope="slope",method="maxslope")

## End(Not run)
```

---

rsaga.parallel.processing

Parallel Processing

Description

Calculate the size of the local catchment area (contributing area), the catchment height, catchment slope and aspect, and flow path length, using parallel processing algorithms including the recommended multiple flow direction algorithm. This set of algorithms processes a digital elevation model (DEM) downwards from the highest to the lowest cell.
No longer supported with SAGA GIS 2.1.3+. See `rsaga.topdown.processing()`.

Usage

```
rsaga.parallel.processing(
  in.dem,
  in.sinkroute,
  in.weight,
  out.carea,
  out.cheight,
  out.cslope,
  out.caspect,
  out.flowpath,
  step,
  method = "mfd",
  linear.threshold = Inf,
```

```
  convergence = 1.1,
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	input: digital elevation model (DEM) as SAGA grid file (default file extension: .sgrd)
in.sinkroute	optional input: SAGA grid with sink routes
in.weight	optional intput: SAGA grid with weights
out.carea	output: catchment area grid
out.cheight	optional output: catchment height grid
out.cslope	optional output: catchment slope grid
out.caspect	optional output: catchment aspect grid
out.flowpath	optional output: flow path length grid
step	integer >=1: step parameter
method	character or numeric: choice of processing algorithm: Deterministic 8 ("d8" or 0), Rho 8 ("rho8" or 1), Braunschweiger Reliefmodell ("braunschweig" or 2), Deterministic Infinity ("dinf" or 3), Multiple Flow Direction ("mfd" or 4, the default), Multiple Triangular Flow Direction ("mtfd", or 5).
linear.threshold	numeric (number of grid cells): threshold above which linear flow (i.e. the Deterministic 8 algorithm) will be used; linear flow is disabled for linear.threshold=Inf (the default)
convergence	numeric >=0: a parameter for tuning convergent/ divergent flow; default value of 1.1 gives realistic results and should not be changed
env	list, setting up a SAGA geoprocessing environment as created by rsaga.env()
...	further arguments to rsaga.geoprocessor()

Details

Refer to the references for details on the available algorithms.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Note

This function uses module Parallel Processing (version 2.0.7+: Catchment Area (Parallel) from SAGA library ta_hydrology.

The SAGA GIS 2.0.6+ version of the module adds more (optional) input and output grids that are currently not supported by this wrapper function. Use `rsaga.geoprocessor()` for access to these options, and see `rsaga.get.usage("ta_hydrology","Catchment Area (Parallel)")` for information on new arguments.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module), Thomas Grabs (MTFD algorithm)

References

Deterministic 8:

O'Callaghan, J.F., Mark, D.M. (1984): The extraction of drainage networks from digital elevation data. Computer Vision, Graphics and Image Processing, 28: 323-344.

Rho 8:

Fairfield, J., Leymarie, P. (1991): Drainage networks from grid digital elevation models. Water Resources Research, 27: 709-717.

Braunschweiger Reliefmodell:

Bauer, J., Rohdenburg, H., Bork, H.-R. (1985): Ein Digitales Reliefmodell als Vorraussetzung fuer ein deterministisches Modell der Wasser- und Stoff-Fluesse. Landschaftsgenese und Landschaftsoekologie, H. 10, Parameteraufbereitung fuer deterministische Gebiets-Wassermodelle, Grundlagenarbeiten zu Analyse von Agrar-Oekosystemen, eds.: Bork, H.-R., Rohdenburg, H., p. 1-15.

Deterministic Infinity:

Tarboton, D.G. (1997): A new method for the determination of flow directions and upslope areas in grid digital elevation models. Water Resources Research, 33(2): 309-319.

Multiple Flow Direction:

Freeman, G.T. (1991): Calculating catchment area with divergent flow based on a regular grid. Computers and Geosciences, 17: 413-22.

Quinn, P.F., Beven, K.J., Chevallier, P., Planchon, O. (1991): The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. Hydrological Processes, 5: 59-79.

Multiple Triangular Flow Direction:

Seibert, J., McGlynn, B. (2007): A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models. Water Resources Research, 43, W04501.

See Also

`rsaga.topdown.processing()`, `rsaga.wetness.index()`, `rsaga.geoprocessor()`, `rsaga.env()`

Examples

```
## Not run:
# SAGA GIS 2.0.6+:
rsaga.get.usage("ta_hydrology","Catchment Area (Parallel)")
# earlier versions of SAGA GIS:
#rsaga.get.usage("ta_hydrology","Parallel Processing")
```

```
# execute model with typical settings:
rsaga.parallel.processing(in.dem = "dem", out.carea = "carea", out.cslope = "cslope")
# cslope is in radians - convert to degree:
fac = round(180/pi, 4)
formula = paste(fac, "*a", sep = "")
rsaga.grid.calculus("cslope", "cslopedeg", formula)

## End(Not run)
```

`rsaga.pisr` *Potential incoming solar radiation*

Description

This function calculates the potential incoming solar radiation in an area using different atmospheric models; module available in SAGA GIS 2.0.6+.

Usage

```
rsaga.pisr(
  in.dem,
  in.svf.grid = NULL,
  in.vapour.grid = NULL,
  in.latitude.grid = NULL,
  in.longitude.grid = NULL,
  out.direct.grid,
  out.diffuse.grid,
  out.total.grid = NULL,
  out.ratio.grid = NULL,
  out.duration,
  out.sunrise,
  out.sunset,
  local.svf = TRUE,
  latitude,
  unit = c("kWh/m2", "kJ/m2", "J/cm2"),
  solconst = 1367,
  enable.bending = FALSE,
  bending.radius = 6366737.96,
  bending.lat.offset = "user",
  bending.lat.ref.user = 0,
  bending.lon.offset = "center",
  bending.lon.ref.user = 0,
  method = c("height", "components", "lumped"),
  hgt.atmosphere = 12000,
  hgt.water.vapour.pressure = 10,
  cmp.pressure = 1013,
  cmp.water.content = 1.68,
  cmp.dust = 100,
```

```
  lmp.transmittance = 70,
  time.range = c(0, 24),
  time.step = 0.5,
  start.date = list(day = 21, month = 3),
  end.date = NULL,
  day.step = 5,
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	name of input digital elevation model (DEM) grid in SAGA grid format (default extension: .sgrd)
in.svf.grid	Optional input grid in SAGA format: Sky View Factor; see also local.svf
in.vapour.grid	Optional input grid in SAGA format: Water vapour pressure (mbar); see also argument hgt.water.vapour.pressure
in.latitude.grid	Optional input grid in SAGA format: Latitude (degree) of each grid cell
in.longitude.grid	see in.latitude.grid
out.direct.grid	Output grid: Direct insolation (unit selected by unit argument)
out.diffuse.grid	Output grid: Diffuse insolation
out.total.grid	Optional output grid: Total insolation, i.e. sum of direct and diffuse incoming solar radiation
out.ratio.grid	Optional output grid: Direct to diffuse ratio
out.duration	Optional output grid: Duration of insolation
out.sunrise	Optional output grid: time of sunrise; only calculated if time span is set to single day
out.sunset	Time of sunset; see out.sunrise
local.svf	logical (default: TRUE; if TRUE, use sky view factor based on local slope (after Oke, 1988), if no sky view factor grid is provided in in.svf.grid
latitude	Geographical latitude in degree North (negative values indicate southern hemisphere)
unit	unit of insolation output grids: "kWh/m2" (default) "kJ/m2", or "J/cm2"
solconst	solar constant, defaults to 1367 W/m2
enable.bending	logical (default: FALSE): incorporate effects of planetary bending?
bending.radius	Planetary radius, default 6366737.96
bending.lat.offset	if bending is enabled: latitudinal reference is "user"-defined (default), or relative to "top", "center" or "bottom" of grid?

bending.lat.ref.user
: user-defined lat. reference for bending, see bending.lat.offset

bending.lon.offset
: longitudinal reference, i.e. local time, is "user"-defined, or relative to "top", "center" (default) or "bottom" of grid?

bending.lon.ref.user
: user-defined reference for local time (Details??)

method
: specifies how the atmospheric components should be accounted for: either based on the height of atmosphere and vapour pressure ("height", or numeric code 0), or air pressure, water and dust content ("components", code 1), or lumped atmospheric transmittance ("lumped", code 0)

hgt.atmosphere
: Height of atmosphere (in m); default 12000 m

hgt.water.vapour.pressure
: Water vapour pressure in mbar (default 10 mbar); This value is used if no vapour pressure grid is given in argument in.vapour.grid

cmp.pressure
: atmospheric pressure in mbar, defaults to 1013 mbar

cmp.water.content
: water content of a vertical slice of the atmosphere in cm: between 1.5 and 1.7cm, average 1.68cm (default)

cmp.dust
: dust factor in ppm; defaults to 100 ppm

lmp.transmittance
: transmittance of the atmosphere in percent; usually between 60 (humid areas) and 80 percent (deserts)

time.range
: numeric vector of length 2: time span (hours of the day) for numerical integration

time.step
: time step in hours for numerical integration

start.date
: list of length two, giving the start date in day and month components as numbers; these numbers are one-based (SAGA_CMD uses zero-based numbers internally), i.e. Jan. 1st is list(day=1,month=1)

end.date
: see start.date

day.step
: if days indicates a range of days, this specifies the time step (number of days) for calculating the incoming solar radiation

env
: RSAGA geoprocessing environment obtained with rsaga.env(); this argument is required for version control (see Note)

...
: optional arguments to be passed to rsaga.geoprocessor()

Details

According to SAGA GIS 2.0.7 documentation, "Most options should do well, but TAPES-G based diffuse irradiance calculation ("Atmospheric Effects" methods 2 and 3) needs further revision!" I.e. be careful with method = "components" and method = "lumped".

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This module is computationally very intensive (depending on the size of the grid and the time resolution, of course). The performance seems to have much improved in SAGA GIS 2.1.0, which by default runs this module in multicore mode (at the release candidate 1 for Windows does).

SAGA_CMD uses zero-based days and months, but this R function uses the standard one-based days and months (e.g. day 1 is the first day of the month, month 1 is January) and translates to the SAGA system.

This function uses module Potential Incoming Solar Radiation from SAGA library `ta_lighting` in SAGA version 2.0.6+.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

References

Boehner, J., Antonic, O. (2009): Land surface parameters specific to topo-climatology. In: Hengl, T. and Reuter, H. I. (eds.): Geomorphometry - Concepts, Software, Applications. Elsevier.

Oke, T.R. (1988): Boundary layer climates. London, Taylor and Francis.

Wilson, J.P., Gallant, J.C. (eds.), 2000: Terrain analysis - principles and applications. New York, John Wiley and Sons.

See Also

`rsaga.hillshade()`; for similar modules in older SAGA versions (pre-2.0.6) see `rsaga.solar.radiation()` and `rsaga.insolation()`

`rsaga.pisr2` *Potential incoming solar radiation SAGA 2.2.2+*

Description

This function calculates the potential incoming solar radiation in an area using different atmospheric models; This function reflects changes to the module with SAGA 2.2.2+. For SAGA versions 2.0.6 to 2.2.1 please see `rsaga.pisr()`.

Usage

```
rsaga.pisr2(
  in.dem,
  in.svf.grid = NULL,
  in.vapour.grid = NULL,
  in.linke.grid = NULL,
  out.direct.grid,
  out.diffuse.grid,
  out.total.grid = NULL,
```

```
  out.ratio.grid = NULL,
  out.duration,
  out.sunrise,
  out.sunset,
  local.svf = TRUE,
  location = c("latitude", "grid"),
  latitude = 53,
  unit = c("kWh/m2", "kJ/m2", "J/cm2"),
  shadow = c("slim", "fat", "none"),
  solconst = 1367,
  method = c("height", "components", "lumped", "hofierka"),
  linke.default = 3,
  vapour.default = 10,
  hgt.atmosphere = 12000,
  cmp.pressure = 1013,
  cmp.water.content = 1.68,
  cmp.dust = 100,
  lmp.transmittance = 70,
  time.range = c(0, 24),
  time.step = 0.5,
  start.date = list(day = 31, month = 10, year = 2015),
  end.date = NULL,
  day.step = 5,
  env = rsaga.env(),
  ...
)
```

Arguments

`in.dem`	name of input digital elevation model (DEM) grid in SAGA grid format (default extension: `.sgrd`)
`in.svf.grid`	Optional input grid in SAGA format: Sky View Factor; see also `local.svf`
`in.vapour.grid`	Optional input grid in SAGA format: Water vapour pressure (mbar), for use with `method = "height"`; default 10 mbar
`in.linke.grid`	Optional input grid in SAGA format: Linke turbidity coefficient, for use with `method = "hofierka"`; default 3.0
`out.direct.grid`	Output grid: Direct insolation (unit selected by `unit` argument)
`out.diffuse.grid`	Output grid: Diffuse insolation
`out.total.grid`	Optional output grid: Total insolation, i.e. sum of direct and diffuse incoming solar radiation
`out.ratio.grid`	Optional output grid: Direct to diffuse ratio
`out.duration`	Optional output grid: Duration of insolation.
`out.sunrise`	Optional output grid: time of sunrise; only calculated if time span is set to single day

out.sunset	Time of sunset; see out.sunrise
local.svf	logical (default: TRUE; if TRUE, use sky view factor based on local slope (after Oke, 1988), if no sky view factor grid is provided in in.svf.grid
location	specified whether to use constant latitude supplied by latitude below ("latitude" or code 0; default) or as calculated from the grid system ("grid" or code 1)
latitude	Geographical latitude in degree North (negative values indicate southern hemisphere)
unit	unit of insolation output grids: "kWh/m2" (default) "kJ/m2", or "J/cm2"
shadow	specifies how topographic shading is modeled: "slim" (or numeric code 0), "fat" (or code 1; the default), or "none" (code 2). Quoting SAGA 7.8.2 help: Choose 'slim' to trace grid node's shadow, 'fat' to trace the whole cell's shadow, or ignore shadowing effects. The first is slightly faster but might show some artifacts. (End quote.)
solconst	solar constant, defaults to 1367 W/m2
method	specifies how the atmospheric components should be accounted for: either based on the height of atmosphere and vapour pressure ("height", or numeric code 0), or air pressure, water and dust content ("components", code 1), or lumped atmospheric transmittance ("lumped", code 2), or by the method of Hofierka and Suri, 2009 ("hofierka", code 3). Default: "lumped".
linke.default	SAGA argument GRD_LINKE_DEFAULT, defaults to 3.0
vapour.default	SAGA argument GRD_VAPOUR_DEFAULT, defaults to 10.0
hgt.atmosphere	Height of atmosphere (in m); default 12000 m. For use with method = "height"
cmp.pressure	atmospheric pressure in mbar, defaults to 1013 mbar. For use with method = "components"
cmp.water.content	water content of a vertical slice of the atmosphere in cm: between 1.5 and 1.7cm, average 1.68cm (default). For use with method = "components"
cmp.dust	dust factor in ppm; defaults to 100 ppm. For use with method = "components"
lmp.transmittance	transmittance of the atmosphere in percent; usually between 60 (humid areas) and 80 percent (deserts)
time.range	numeric vector of length 2: time span (hours of the day) for numerical integration
time.step	time step in hours for numerical integration
start.date	list of length three, giving the start date in day, month, and year components as numbers, i.e. Jan. 1st 2015 is list(day=1,month=1,year=2015)
end.date	see start.date
day.step	if days indicates a range of days, this specifies the time step (number of days) for calculating the incoming solar radiation
env	RSAGA geoprocessing environment obtained with rsaga.env(); this argument is required for version control (see Note)
...	optional arguments to be passed to rsaga.geoprocessor()

Details

According to SAGA GIS 2.0.7 documentation, "Most options should do well, but TAPES-G based diffuse irradiance calculation ("Atmospheric Effects" methods 2 and 3) needs further revision!" I.e. be careful with `method = "components"` and `method = "lumped"`.

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses module Potential Incoming Solar Radiation from SAGA library `ta_lighting` in SAGA versions >2.2.3. Duration of insolation (`"out.duration"`) is only calculated when the time period is set to a single day.

The SAGA module in version 8.5.x does not correctly use the date arguments; it is therefore not supported. Several SAGA versions >=8.6.0 and <8.5.0 did, however, produce plausible results and correctly interpreted the date arguments according to output shown when `show.output.on.console=TRUE` was used.

Author(s)

Alexander Brenning & Donovan Bangs (R interface), Olaf Conrad (SAGA module)

References

Boehner, J., Antonic, O. (2009): Land surface parameters specific to topo-climatology. In: Hengl, T. and Reuter, H. I. (eds.): Geomorphometry - Concepts, Software, Applications. Elsevier.

Oke, T.R. (1988): Boundary layer climates. London, Taylor and Francis.

Wilson, J.P., Gallant, J.C. (eds.), 2000: Terrain analysis - principles and applications. New York, John Wiley and Sons.

Hofierka, J., Suri, M. (2002): The solar radiation model for Open source GIS: implementation and applications. International GRASS users conference in Trento, Italy, September 2002

See Also

`rsaga.pisr()`; for similar modules in older SAGA versions (pre-2.0.6) see `rsaga.solar.radiation()` and `rsaga.insolation()`; `rsaga.hillshade()`

`rsaga.set.env` *Internal function that sets the RSAGA Geoprocessing Evironment manually*

Description

Internal function that sets the RSAGA Geoprocessing Evironment manually

Usage

```
rsaga.set.env(
  workspace = NULL,
  cmd = NULL,
  path = NULL,
  modules = NULL,
  version = NA,
  cores = NULL,
  parallel = NULL
)
```

Arguments

`workspace`	path of the working directory for SAGA; defaults to the current directory (`"."`).
`cmd`	name of the SAGA command line program; defaults to `saga_cmd.exe`, its name under Windows
`path`	path in which to find `cmd`; `rsaga.env` is usually able to find SAGA on your system if it is installed; see Details.
`modules`	path in which to find SAGA libraries; see Details
`version`	optional character string: SAGA GIS (API) version, e.g. `"2.0.8"`; if missing, a call to `rsaga.get.version()` is used to determine version number of SAGA API
`cores`	optional numeric argument, or `NA`: number of cores used by SAGA GIS; supported only by SAGA GIS 2.1.0 (and higher), ignored otherwise (with a warning). Multicore-enabled SAGA GIS modules such as the one used by `rsaga.pisr()` seem to run in multicore mode by default when this argument is not specified, therefore `cores` should only be specified to use a smaller number of cores than available on a machine.
`parallel`	optional logical argument (default: `FALSE`): if `TRUE`, run RSAGA functions that are capable of parallel processing in parallel mode; note that this is completely independent of the behavior of SAGA GIS (which can be controlled using the `cores` argument); currently only some RSAGA functions support parallel processing (e.g., `pick.from.ascii.grid()` or `rsaga.get.modules()`). `parallel=TRUE` requires that a parallel backend such as **doSNOW** or **doMC** is available and has been started prior to calling any parallelized RSAGA function, otherwise warnings may be generated

Value

a list containing the above information

`rsaga.sgrd.to.esri` *Convert SAGA grids to ESRI ASCII/binary grids*

Description

`rsaga.sgrd.to.esri` converts grid files from SAGA's (version 2) grid format (.sgrd) to ESRI's ASCII (.asc) and binary (.flt) format.

Usage

```
rsaga.sgrd.to.esri(
  in.sgrds,
  out.grids,
  out.path,
  format = "ascii",
  georef = "corner",
  prec = 5,
  ...
)
```

Arguments

`in.sgrds`	character vector of SAGA grid files (`.sgrd`) to be converted; files are expected to be found in folder `rsaga.env()$workspace`, or, if an optional `env` argument is provided, in `env$workspace`
`out.grids`	character vector of ESRI ASCII/float output file names; defaults to `in.sgrds` with the file extension being replaced by `.asc` or `.flt`, depending on `format`. Files will be placed in folder `out.path`, existing files will be overwritten
`out.path`	folder for `out.grids`
`format`	output file format, either `"ascii"` (default; equivalent: `format=1`) for ASCII grids or `"binary"` (equivalent: `0`) for binary ESRI grids (`.flt`).
`georef`	character: `"corner"` (equivalent numeric code: `0`) or `"center"` (default; equivalent: `1`). Determines whether the georeference will be related to the center or corner of its extreme lower left grid cell.
`prec`	number of digits when writing floating point values to ASCII grid files; either a single number (to be replicated if necessary), or a numeric vector of length `length(in.grids)`
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses module 0 from the SAGA library `io_grid`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.esri.wrapper()` for an efficient way of applying RSAGA to ESRI ASCII/binary grids; `rsaga.env()`

`rsaga.sink.removal` *Sink Removal Remove sinks from a digital elevation model by deepening drainage routes or filling sinks.*

Description

Sink Removal Remove sinks from a digital elevation model by deepening drainage routes or filling sinks.

Usage

```
rsaga.sink.removal(in.dem, in.sinkroute, out.dem, method = "fill", ...)
```

Arguments

`in.dem`	input: digital elevation model (DEM) as SAGA grid file (default file extension: `.sgrd`)
`in.sinkroute`	optional input: sink route grid file
`out.dem`	output: modified DEM
`method`	character string or numeric value specifying the algorithm (partial string matching will be applied): `"deepen drainage route"` (or 0): reduce the elevation of pixels in order to achieve drainage out of the former sinks `"fill sinks"` (or 1): fill sinks until none are left
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This function uses module 1 from SAGA library `ta_preprocessor`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.sink.route()`, `rsaga.fill.sinks()`

Examples

```
## Not run: rsaga.sink.route("dem","sinkroute")
rsaga.sink.removal("dem","sinkroute","dem-preproc",method="deepen")
## End(Not run)
```

`rsaga.sink.route` *Sink Drainage Route Detection*

Description

Sink drainage route detection.

Usage

```
rsaga.sink.route(in.dem, out.sinkroute, threshold, thrsheight = 100, ...)
```

Arguments

`in.dem`	input: digital elevation model (DEM) as SAGA grid file (default file extension: `.sgrd`)
`out.sinkroute`	output: sink route grid file: non-sinks obtain a value of 0, sinks are assigned an integer between 0 and 8 indicating the direction to which flow from this sink should be routed
`threshold`	logical: use a threshold value?
`thrsheight`	numeric: threshold value (default: `100`)
`...`	optional arguments to be passed to `rsaga.geoprocessor()`, including the `env` RSAGA geoprocessing environment

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

I assume that flow directions are coded as 0 = north, 1 = northeast, 2 = east, ..., 7 = northwest, as in `rsaga.fill.sinks()`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

See Also

`rsaga.sink.removal()`

Examples

```
## Not run: rsaga.sink.route("dem","sinkroute")
rsaga.sink.removal("dem","sinkroute","dem-preproc",method="deepen")
## End(Not run)
```

---

rsaga.slope.asp.curv *Slope, Aspect, Curvature*

Description

Calculates local morphometric terrain attributes (i.e. slope, aspect, and curvatures). Intended for use with SAGA v 2.1.1+. For older versions use `rsaga.local.morphometry()`.

Usage

```
rsaga.slope.asp.curv(
  in.dem,
  out.slope,
  out.aspect,
  out.cgene,
  out.cprof,
  out.cplan,
  out.ctang,
  out.clong,
  out.ccros,
  out.cmini,
  out.cmaxi,
  out.ctota,
```

```
  out.croto,
  method = "poly2zevenbergen",
  unit.slope = "radians",
  unit.aspect = "radians",
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	input: digital elevation model as SAGA grid file (.sgrd)
out.slope	optional output: slope
out.aspect	optional output: aspect
out.cgene	optional output: general curvature (1 / map units)
out.cprof	optional output: profile curvature (vertical curvature; 1 / map units)
out.cplan	optional output: plan curvature (horizontal curvature; 1 / map units)
out.ctang	optional output: tangential curvature (1 / map units)
out.clong	optional output: longitudinal curvature (1 / map units) Zevenbergen & Thorne (1987) refer to this as profile curvature
out.ccros	optional output: cross-sectional curvature (1 / map units) Zevenbergen & Thorne (1987) refer to this as the plan curvature
out.cmini	optional output: minimal curvature (1 / map units)
out.cmaxi	optional output: maximal curvature (1 / map units)
out.ctota	optional output: total curvature (1 / map units)
out.croto	optional output: flow line curvature (1 / map units)
method	character algorithm (see References): • 0 Maximum Slope - Travis et al. (1975) ("maxslope") • 1 Max. Triangle Slope - Tarboton (1997) ("maxtriangleslope") • 2 Least Squares Fit Plane - Costa-Cabral & Burgess (1996) ("lsqfitplane") • 3 Fit 2nd Degree Polynomial - Evans (1979) ("poly2evans") • 4 Fit 2nd Degree Polynomial - Heerdegen and Beran (1982) ("poly2heerdegen") • 5 Fit 2nd Degree Polynomial - Bauer et al. (1985) ("poly2bauer") • 6 default: Fit 2nd Degree Polynomial - Zevenbergen & Thorne (1987) ("poly2zevenbergen") • 7 Fit 3rd Degree Polynomial - Haralick (1983) ("poly3haralick")
unit.slope	character or numeric (default "radians"): • 0 "radians" • 1 "degrees" • 2 "percent"
unit.aspect	character or numeric (default is 0, or "radians"): • 0 "radians" • 1 "degrees"
env	list, setting up a SAGA geoprocessing environment as created by rsaga.env()
...	further arguments to rsaga.geoprocessor()

Details

Profile and plan curvature calculation (`out.cprof`, `out.cplan`) changed in SAGA GIS 2.1.1+ compared to earlier versions. See the following thread on sourceforge.net for an ongoing discussion: https://sourceforge.net/p/saga-gis/discussion/354013/thread/e9d07075/#5727

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Author(s)

Alexander Brenning and Donovan Bangs (R interface), Olaf Conrad (SAGA module)

References

General references:

Jones KH (1998) A comparison of algorithms used to compute hill slope as a property of the DEM. Computers and Geosciences. 24 (4): 315-323.

References on specific methods:

Maximum Slope:

Travis, M.R., Elsner, G.H., Iverson, W.D., Johnson, C.G. (1975): VIEWIT: computation of seen areas, slope, and aspect for land-use planning. USDA F.S. Gen. Tech. Rep. PSW-11/1975, 70 p. Berkeley, California, U.S.A.

Maximum Triangle Slope:

Tarboton, D.G. (1997): A new method for the determination of flow directions and upslope areas in grid digital elevation models. Water Resources Research, 33(2): 309-319.

Least Squares or Best Fit Plane:

Beasley, D.B., Huggins, L.F. (1982): ANSWERS: User's manual. U.S. EPA-905/9-82-001, Chicago, IL, 54 pp.

Costa-Cabral, M., Burges, S.J. (1994): Digital Elevation Model Networks (DEMON): a model of flow over hillslopes for computation of contributing and dispersal areas. Water Resources Research, 30(6): 1681-1692.

Fit 2nd Degree Polynomial:

Evans, I.S. (1979): An integrated system of terrain analysis and slope mapping. Final Report on grant DA-ERO-591-73-G0040. University of Durham, England.

Bauer, J., Rohdenburg, H., Bork, H.-R. (1985): Ein Digitales Reliefmodell als Vorraussetzung fuer ein deterministisches Modell der Wasser- und Stoff-Fluesse. Landschaftsgenese und Landschaftsoekologie, H. 10, Parameteraufbereitung fuer deterministische Gebiets-Wassermodelle, Grundlagenarbeiten zur Analyse von Agrar-Oekosystemen, eds.: Bork, H.-R., Rohdenburg, H., p. 1-15.

Heerdegen, R.G., Beran, M.A. (1982): Quantifying source areas through land surface curvature. Journal of Hydrology, 57.

Zevenbergen, L.W., Thorne, C.R. (1987): Quantitative analysis of land surface topography. Earth Surface Processes and Landforms, 12: 47-56.

Fit 3.Degree Polynomial:

Haralick, R.M. (1983): Ridge and valley detection on digital images. Computer Vision, Graphics and Image Processing, 22(1): 28-38.

For a discussion on the calculation of slope by ArcGIS check these links:

https://community.esri.com/?c=93&f=1734&t=239914

https://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?topicname=how_slope_works

See Also

rsaga.local.morphometry(), rsaga.parallel.processing(), rsaga.geoprocessor(), rsaga.env()

Examples

```
## Not run:
# Simple slope, aspect, and general curvature in degrees:
rsaga.slope.asp.curv("lican.sgrd", "slope", "aspect", "curvature",
                     method = "maxslope", unit.slope = "degrees", unit.aspect = "degrees")
# same for ASCII grids (default extension .asc):
rsaga.esri.wrapper(rsaga.slope.asp.curv,
                   in.dem="lican", out.slope="slope",
                   out.aspect = "aspect", out.cgene = "curvature",
                   method="maxslope", unit.slope = "degrees", unit.aspect = "degrees")

## End(Not run)
```

---

rsaga.solar.radiation *Potential incoming solar radiation*

Description

This function calculates the potential incoming solar radiation in an area either using a lumped atmospheric transmittance model or estimating it based on water and dust content. Use rsaga.pisr() instead with SAGA GIS 2.0.6+.

Usage

```
rsaga.solar.radiation(
  in.dem,
  out.grid,
  out.duration,
  latitude,
  unit = c("kWh/m2", "J/m2"),
  solconst = 1367,
  method = c("lumped", "components"),
  transmittance = 70,
```

```
  pressure = 1013,
  water.content = 1.68,
  dust = 100,
  time.range = c(0, 24),
  time.step = 1,
  days = list(day = 21, month = 3),
  day.step = 5,
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	name of input digital elevation model (DEM) grid in SAGA grid format (default extension: .sgrd)
out.grid	output grid file for potential incoming solar radiation sums
out.duration	Optional output grid file for duration of insolation
latitude	Geographical latitude in degree North (negative values indicate southern hemisphere)
unit	unit of the out.grid output: "kWh/m2" (default) or "J/m2"
solconst	solar constant, defaults to 1367 W/m2
method	specifies how the atmospheric components should be accounted for: either based on a lumped atmospheric transmittance as specified by argument transmittance ("lumped", or numeric code 0; default); or by calculating the components corresponding to water and dust ("components", code 1)
transmittance	transmittance of the atmosphere in percent; usually between 60 (humid areas) and 80 percent (deserts)
pressure	atmospheric pressure in mbar
water.content	water content of a vertical slice of the atmosphere in cm: between 1.5 and 1.7cm, average 1.68cm (default)
dust	dust factor in ppm; defaults to 100ppm
time.range	numeric vector of length 2: time span (hours of the day) for numerical integration
time.step	time step in hours for numerical integration
days	either a list with components day and month specifying a single day of the year for radiation modeling; OR a numeric vector of length 2 specifying the start and end date (see Note below)
day.step	if days indicates a range of days, this specifies the time step (number of days) for calculating the incoming solar radiation
env	RSAGA geoprocessing environment obtained with rsaga.env(); this argument is required for version control (see Note)
...	optional arguments to be passed to rsaga.geoprocessor()

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (default) a character vector with the module's console output.

Note

This module ceased to exist under SAGA GIS 2.0.6+, which has a similar (but more flexible) module Potential Solar Radiation that is interfaced by `rsaga.pisr()`.

SAGA_CMD uses zero-based days and months, but this R function uses the standard one-based days and months (e.g. day 1 is the first day of the month, month 1 is January) and translates to the SAGA system.

In SAGA 2.0.2, solar radiation sums calculated for a range of days, say `days=c(a,b)` actually calculate radiation only for days `a,...,b-1` (in steps of `day.step` - I used `day.step=1` in this example). The setting `a=b` however gives the same result as `b=a+1`, and indeed `b=a+2` gives twice the radiation sums and potential sunshine duration that `a=b` and `b=a+1` both give.

The solar radiation module of SAGA 2.0.1 had a bug that made it impossible to pass a range of `days` of the year or a range of hours of the day (`time.range`) to SAGA. These options work in SAGA 2.0.1.

This function uses module Incoming Solar Radiation from SAGA GIS library `ta_lighting`.

Author(s)

Alexander Brenning (R interface), Olaf Conrad (SAGA module)

References

Wilson, J.P., Gallant, J.C. (eds.), 2000: Terrain analysis - principles and applications. New York, John Wiley & Sons.

See Also

`rsaga.hillshade()`, `rsaga.insolation()`

Examples

```
## Not run:
# potential solar radiation on Nov 7 in Southern Ontario...
rsaga.solar.radiation("dem","solrad","soldur",latitude=43,
    days=list(day=7,month=11),time.step=0.5)

## End(Not run)
```

`rsaga.target` *Define target grid for interpolation*

Description

Define the resolution and extent of a target grid for interpolation by SAGA modules based on (1) user-provided x/y coordinates, (2) an existing SAGA grid file, or (3) the header data of an ASCII grid. Intended to be used with RSAGA's interpolation functions.

Usage

```
rsaga.target(
  target = c("user.defined", "target.grid", "header"),
  user.cellsize = 100,
  user.x.extent,
  user.y.extent,
  target.grid,
  header,
  env = rsaga.env()
)
```

Arguments

`target`	character: method used for defining the target grid
`user.cellsize`	Only for `target="user.defined"`: raster resolution (in the grid's map units)
`user.x.extent`	See `user.y.extent`
`user.y.extent`	Only for `target="user.defined"`: numeric vectors of length 2: minimum and maximum coordinates of grid cell center points
`target.grid`	Only for `target="target.grid"`: character string giving the name of a SAGA grid file that specifies the extent and resolution of the target grid; this target grid file may be overwritten, depending on the specifics of the SAGA GIS module used.
`header`	Only for `target="header"`: list: ASCII grid header (as returned e.g. by `read.ascii.grid.header()`) or defined manually; must at least have components `ncols`, `nrows`, `cellsize`, and either `x/yllcorner` or `x/yllcenter`.
`env`	A SAGA geoprocessing environment, see `rsaga.env()`.)

Value

a list containing information such as the bounding box of a target grid

Note

This function is to be used with RSAGA functions `rsaga.inverse.distance()`, `rsaga.nearest.neighbour()` and `rsaga.modified.quadratic.shephard()`. Note that these are currently only compatible with SAGA GIS 2.0.5 and higher.

See Also

read.ascii.grid.header()

Examples

```
## Not run:
# IDW interpolation of attribute "z" from the point shapefile
# 'points.shp' to a grid with the same extent and resolution
# as the (pre-existing) geology grid:
rsaga.inverse.distance("points", "dem", field = "z", maxdist = 1000,
    target = rsaga.target(target="target.grid",
    target.grid = "geology"))

## End(Not run)
```

---

rsaga.topdown.processing

Top-Down Processing

Description

Calculate the size of the local catchment area (contributing area), accumulated material, and flow path length, using top-down processing algorithms from the highest to the lowest cell.
Top-Down Processing is new with SAGA GIS 2.1.3. See rsaga.parallel.processing() with older versions.

Usage

```
rsaga.topdown.processing(
  in.dem,
  in.sinkroute,
  in.weight,
  in.mean,
  in.material,
  in.target,
  in.lin.val,
  in.lin.dir,
  out.carea,
  out.mean,
  out.tot.mat,
  out.acc.left,
  out.acc.right,
  out.flowpath,
  step,
  method = "mfd",
  linear.threshold = Inf,
  convergence = 1.1,
```

```
  env = rsaga.env(),
  ...
)
```

Arguments

in.dem	input: digital elevation model (DEM) as SAGA grid file (default file extension: .sgrd)
in.sinkroute	optional input: SAGA grid with sink routes
in.weight	optional input: SAGA grid with weights
in.mean	optional input: SAGA grid for mean over catchment calculation
in.material	optional input: SAGA grid with material
in.target	optional input: SAGA grid of accumulation target
in.lin.val	optional input: SAGA grid providing values to be compared with linear flow threshold instead of catchment area
in.lin.dir	optional input: SAGA grid to be used for linear flow routing, if the value is a valid direction (0-7 = N, NE, E, SE, S, SW, W, NW)
out.carea	output: catchment area grid
out.mean	optional output: mean over catchment grid
out.tot.mat	optional output: total accumulated material grid
out.acc.left	optional output: accumulated material from left side grid
out.acc.right	optional output: accumulated material from right side grid
out.flowpath	optional output: flow path length grid
step	integer >=1: step parameter
method	character or numeric: choice of processing algorithm (default "mfd", or 4): • 0 Deterministic 8 ("d8" or 0) • 1 Rho 8 ("rho8", or 1) • 2 Braunschweiger Reliefmodell ("braunschweig" or 2) • 3 Deterministic Infinity ("dinf" or 3) • 4 Multiple Flow Direction ("mfd" or 4) • 5 Multiple Triangular Flow Direction ("mtfd", or 5) • 6 Multiple Maximum Gradient Based Flow Direction ("mdg", or 6)
linear.threshold	numeric (number of grid cells): threshold above which linear flow (i.e. the Deterministic 8 algorithm) will be used; linear flow is disabled for linear.threshold=Inf (the default)
convergence	numeric >=0: a parameter for tuning convergent/ divergent flow; default value of 1.1 gives realistic results and should not be changed
env	list, setting up a SAGA geoprocessing environment as created by rsaga.env()
...	further arguments to rsaga.geoprocessor()

Details

Refer to the references for details on the available algorithms.

Value

The type of object returned depends on the `intern` argument passed to the `rsaga.geoprocessor()`. For `intern=FALSE` it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning and Donovan Bangs (R interface), Olaf Conrad (SAGA module), Thomas Grabs (MTFD algorithm)

References

Deterministic 8:

O'Callaghan, J.F., Mark, D.M. (1984): The extraction of drainage networks from digital elevation data. Computer Vision, Graphics and Image Processing, 28: 323-344.

Rho 8:

Fairfield, J., Leymarie, P. (1991): Drainage networks from grid digital elevation models. Water Resources Research, 27: 709-717.

Braunschweiger Reliefmodell:

Bauer, J., Rohdenburg, H., Bork, H.-R. (1985): Ein Digitales Reliefmodell als Vorraussetzung fuer ein deterministisches Modell der Wasser- und Stoff-Fluesse. Landschaftsgenese und Landschaftsoekologie, H. 10, Parameteraufbereitung fuer deterministische Gebiets-Wassermodelle, Grundlagenarbeiten zu Analyse von Agrar-Oekosystemen, eds.: Bork, H.-R., Rohdenburg, H., p. 1-15.

Deterministic Infinity:

Tarboton, D.G. (1997): A new method for the determination of flow directions and upslope areas in grid digital elevation models. Water Resources Research, 33(2): 309-319.

Multiple Flow Direction:

Freeman, G.T. (1991): Calculating catchment area with divergent flow based on a regular grid. Computers and Geosciences, 17: 413-22.

Quinn, P.F., Beven, K.J., Chevallier, P., Planchon, O. (1991): The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. Hydrological Processes, 5: 59-79.

Multiple Triangular Flow Direction:

Seibert, J., McGlynn, B. (2007): A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models. Water Resources Research, 43, W04501.

Multiple Flow Direction Based on Maximum Downslope Gradient:

Qin, C.Z., Zhu, A-X., Pei, T., Li, B.L., Scholten, T., Zhou, C.H. (2011): An approach to computing topographic wetness index based on maximum downslope gradient. Precision Agriculture, 12(1): 32-43.

See Also

rsaga.parallel.processing(), rsaga.wetness.index(), rsaga.geoprocessor(), rsaga.env()

Examples

```
## Not run:
# Calculation of contributing area with default settings:
rsaga.topdown.processing(in.dem = "dem", out.carea = "carea")
# Calculation of contributing area by maximunm downslope gradient:
rsaga.topdown.processing(in.dem = "dem", out.carea = "carea",
                         method = "mdg")

## End(Not run)
```

---

rsaga.union.polygons *Spatial union of two polygon layers*

Description

The function rsaga.union.polygons uses SAGA function "Union" to calculate the geometric union of two polygon layers. This corresponds to the intersection and the symmetrical difference of the two layers.

Usage

```
rsaga.union.polygons(
  layer_a = NULL,
  layer_b = NULL,
  result = NULL,
  split = FALSE,
  load = NULL,
  env = rsaga.env()
)
```

Arguments

layer_a	A character string representing the path to a polygon shapefile.
layer_b	A character string representing the path to a polygon shapefile with which to union layer_a.
result	character, path indicating where to store the output shapefile.
split	If TRUE, multipart polygons become separated polygons (default: FALSE).
load	Deprecated, will be removed in a future release. Ignored if FALSE, and causes an error if TRUE (default: NULL)
env	RSAGA geoprocessing environment created by rsaga.env(), required because module(s) depend(s) on SAGA version.

Details

Function `gUnion()` in `rgeos` package can also be used for joining intersecting polygon geometries. However, `rsaga.union.polygons()` will be usually much faster, especially when joining thousands of polygons.

Value

The function saves the output shapefile to the path indicated in function argument `result`.

Author(s)

Jannes Muenchow and Alexander Brenning (R interface), Olaf Conrad and Angus Johnson (SAGA modules)

`rsaga.wetness.index` *SAGA Modules SAGA Wetness Index*

Description

Calculate the SAGA Wetness Index (SWI), a modified topographic wetness index (TWI)

Usage

```
rsaga.wetness.index(
  in.dem,
  out.wetness.index,
  out.carea,
  out.cslope,
  out.mod.carea,
  suction,
  area.type,
  slope.type,
  slope.min,
  slope.offset,
  slope.weight,
  t.param,
  env = rsaga.env(),
  ...
)
```

Arguments

`in.dem`	input: digital elevation model (DEM) as SAGA grid file (default file extension: `.sgrd`)
`out.wetness.index`	output file (optional): wetness index grid file name. Existing files of the same name will be overwritten!

out.carea	output file (optional): catchment area grid file name
out.cslope	output file (optional): catchment slope grid file name
out.mod.carea	output file (optional): file name of modified catchment area grid
suction	SAGA GIS 2.1.0+: positive numeric value (optional): the lower this value is the stronger is the suction effect; defaults to a value of 10 (more detailed information is currently not available in the SAGA GIS documentation
area.type	character or numeric (optional): type of area: "absolute" (or numeric code 0): absolute catchment area; "square root" (code 1; the default e.g. in SAGA 2.3.1): square root of catchment area; "specific" (code 2; the default e.g. in SAGA 8.4.1): specific catchment area
slope.type	character or numeric (optional): type of slope: "local" (or numeric code 0): local slope; "catchment" (or code 1; the default): catchment slope.
slope.min	numeric (optional): minimum slope; default: 0
slope.offset	numeric (optional): offset slope; default: 0.1
slope.weight	numeric (optional): weighting factor for slope in index calculation; default: 1
t.param	SAGA GIS up to version 2.0.8: positive numeric value (optional): undocumented
env	A SAGA geoprocessing environment, see rsaga.env().)
...	optional arguments to be passed to rsaga.geoprocessor()

Details

The SAGA Wetness Index is similar to the Topographic Wetness Index (TWI), but it is based on a modified catchment area calculation (out.mod.carea), which does not treat the flow as a thin film as done in the calculation of catchment areas in conventional algorithms. As a result, the SWI tends to assign a more realistic, higher potential soil wetness than the TWI to grid cells situated in valley floors with a small vertical distance to a channel.

This module and its arguments changed substantially from SAGA GIS 2.0.8 to version 2.1.0. It appears to me that the new algorithm is similar (but not identical) to the old one when using area.type="absolute" and slope.type="local" but I haven't tried out all possible options. This help file will be updated as soon as additional documentation becomes available.

Value

The type of object returned depends on the intern argument passed to the rsaga.geoprocessor(). For intern=FALSE it is a numerical error code (0: success), or otherwise (the default) a character vector with the module's console output.

Author(s)

Alexander Brenning (R interface), Juergen Boehner and Olaf Conrad (SAGA module)

References

Boehner, J., Koethe, R. Conrad, O., Gross, J., Ringeler, A., Selige, T. (2002): Soil Regionalisation by Means of Terrain Analysis and Process Parameterisation. In: Micheli, E., Nachtergaele, F., Montanarella, L. (ed.): Soil Classification 2001. European Soil Bureau, Research Report No. 7, EUR 20398 EN, Luxembourg. pp.213-222.

Boehner, J. and Selige, T. (2006): Spatial prediction of soil attributes using terrain analysis and climate regionalisation. In: Boehner, J., McCloy, K.R., Strobl, J. [Ed.: SAGA - Analysis and Modelling Applications, Goettinger Geographische Abhandlungen, Goettingen: 13-28.

See Also

rsaga.parallel.processing(), rsaga.geoprocessor(), rsaga.env()

Examples

```
## Not run:
# using SAGA grids:
rsaga.wetness.index("dem.sgrd","swi.sgrd")

## End(Not run)
```

---

set.file.extension *Determine or modify file name extensions*

Description

Function get.file.extension determines the file extension, set.file.extension changes it, and default.file.extension changes it only if it is not already specified.

Usage

```
set.file.extension(filename, extension, fsep = .Platform$file.sep)

get.file.extension(filename, fsep = .Platform$file.sep)

default.file.extension(filename, extension, force = FALSE)
```

Arguments

filename	character vector: file name(s), possibly including paths and extensions; a file name ending with a "." is interpreted as having extension "", while a file name that doesn't contain a "." is interpreted has having no extension.
extension	character string: file extension, without the dot
fsep	character: separator between paths
force	logical argument to default.file.extension: force the file extension to be extension (same result as set.file.extension), or only set it to extension if it has not been specified?

Value

character vector of same length as `filename`

Examples

```
fnm = c("C:/TEMP.DIR/temp","C:/TEMP.DIR/tmp.txt","tempfile.")
get.file.extension(fnm)
set.file.extension(fnm,extension=".TMP")
default.file.extension(fnm,extension=".TMP")
```

---

study_area *Study area mask for the* `landslides` *dataset*

Description

Study area polygon for the `landslides` dataset

Usage

```
study_area
```

Format

An `sf` polygon object representing the outlines of the natural part of the RBSF study area.

Source

Muenchow, J., Brenning, A., Richter, R. (2012): Geomorphic process rates of landslides along a humidity gradient in the tropical Andes, Geomorphology 139-140, 271-284. DOI: 10.1016/j.geomorph.2011.10.029.

Stoyan, R. (2000): Aktivitaet, Ursachen und Klassifikation der Rutschungen in San Francisco/Suedecuador. Unpublished diploma thesis, University of Erlangen-Nuremberg, Germany.

See Also

landslides

wind.shelter *Wind Shelter Index*

Description

wind.shelter is a function to be used with focal.function() to calculate a topographic wind shelter index from a digital elevation model, which is a proxy for snow accumulation on the lee side of topographic obstacles. wind.shelter.prep performs some preparatory calculations to speed up repeated calls to wind.shelter.

Usage

```
wind.shelter(x, prob = NULL, control)

wind.shelter.prep(radius, direction, tolerance, cellsize = 90)
```

Arguments

x	square matrix of elevation data
prob	numeric: quantile of slope values to be used in computing the wind shelter index; if NULL, use max (equivalent to prob=1)
control	required argument: the result of a call to wind.shelter.prep
radius	radius (>1) of circle segment to be used (number of grid cells, not necessarily an integer)
direction	wind direction: direction from which the wind originates; North = 0 = 2*pi, clockwise angles.
tolerance	directional tolerance
cellsize	grid cellsize

Details

wind.shelter implements a wind shelter index used by Plattner et al. (2004) for modeling snow accumulation patterns on a glacier in the Austrian Alps. It is a modified version of the algorithm of Winstral et al. (2002). The wind shelter index of Plattner et al. (2004) is defined as:
Shelter index(S) = arctan(max((z(x0)-z(x)) / |x0-x| : x in S)),

where S = S(x0,a,da,d) is the set of grid nodes within a distance <=d from x0, only considering grid nodes in directions between a-da and a+da from x0.

The present implementation generalizes this index by replacing max by the quantile function; the max function is used if prob=NULL, and the same result is obtained for prob=1 using the quantile function.

Value

The function `wind.shelter` returns the wind shelter index as described above if a numeric matrix `x` is provided. If it is missing, it returns the character string `"windshelter"`.

`wind.shelter.prep` returns a list with components `mask` and `dist`. Both are square matrices with `2*(ceiling(radius)+1)` columns and rows:

`mask`	indicates which grid cell in the moving window is within the specified circle segment (value `FALSE`) or not (`TRUE`)
`dist`	the precomputed distances of a grid cell to the center of the moving window, in map units

Note

The wind shelter index only makes sense if elevation is measured in the same units as the horizontal map units used for the `cellsize` argument (i.e. usually meters).

`wind.shelter` and `wind.shelter.prep` do not restrict the calculation to a circular area; this is done by `focal.function()` when used in combination with that function (assuming `search.mode="circle"`).

Note that the present definition of the wind shelter index returns negative values for surfaces that are completely exposed toward the specified direction. This may make sense if interpreted as a "wind exposure index", or it might be appropriate to set negative wind shelter values to 0.

Author(s)

Alexander Brenning

References

Plattner, C., Braun, L.N., Brenning, A. (2004): Spatial variability of snow accumulation on Vernagtferner, Austrian Alps, in winter 2003/2004. Zeitschrift fuer Gletscherkunde und Glazialgeologie, 39: 43-57.

Winstral, A., Elder, K., Davis, R.E. (2002): Spatial snow modeling of wind-redistributed snow using terrain-based parameters. Journal of Hydrometeorology, 3: 524-538.

See Also

`focal.function()`, `quantile()`

Examples

```
# Settings used by Plattner et al. (2004):
ctrl = wind.shelter.prep(6,-pi/4,pi/12,10)
## Not run: focal.function("dem.asc",fun=wind.shelter,control=ctrl,
    radius=6,search.mode="circle")
## End(Not run)
```

Index