

Package ‘RSAGA’

April 17, 2009

Type Package

Title SAGA Geoprocessing and Terrain Analysis in R

Version 0.9-5

Date 2009-03-01

Author Alexander Brenning

Maintainer Alexander Brenning <brenning@uwaterloo.ca>

Description RSAGA provides access to geocomputing and terrain analysis functions of SAGA from within R by running the command line version of SAGA. In addition, several R functions for handling and manipulating ASCII grids are provided, including a flexible framework for applying local functions (including predict methods of fitted models) or focal functions to multiple grids. SAGA is available under GPL via <http://sourceforge.net/projects/saga-gis/>.

License GPL-2

SystemRequirements Windows, SAGA (>=2.0.2)

Suggests gstat, shapefiles

OS_type windows

Repository CRAN

Date/Publication 2009-03-03 20:19:35

R topics documented:

RSAGA-package	2
centervalue	3
create.variable.name	4
focal.function	5
grid.predict	8
grid.to.xyz	10
match.arg.ext	11

multi.focal.function	13
pick.from.points	16
read.ascii.grid	19
relative.position	21
resid.median	22
rsaga.add.grid.values.to.points	23
rsaga.close.gaps	24
rsaga.contour	26
rsaga.env	26
rsaga.esri.to.sgrd	28
rsaga.esri.wrapper	29
rsaga.fill.sinks	31
rsaga.filter.gauss	33
rsaga.filter.simple	34
rsaga.geoprocessor	35
rsaga.get.modules	37
rsaga.get.usage	39
rsaga.grid.calculus	40
rsaga.grid.to.points	42
rsaga.hillshade	43
rsaga.html.help	44
rsaga.import.gdal	45
rsaga.insolation	47
rsaga.inverse.distance	49
rsaga.local.morphometry	51
rsaga.ordinary.kriging	53
rsaga.parallel.processing	54
rsaga.sgrd.to.esri	56
rsaga.sink.removal	58
rsaga.sink.route	59
rsaga.solar.radiation	60
rsaga.target	62
rsaga.wetness.index	63
set.file.extension	65
wind.shelter	66
Index	68

Description

RSAGA provides access to geocomputing and terrain analysis functions of SAGA from within R by running the command line version of SAGA. In addition, several R functions for handling and manipulating ASCII grids are provided, including a flexible framework for applying local functions (including predict methods of fitted models) or focal functions to multiple grids.

Details

Package: RSAGA
Type: Package
Version: 0.9-5
Date: 2009-03-01
License: GPL-2

RSAGA provides direct access to SAGA functions including 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.

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 the free SAGA GIS ($\geq 2.0.2$) and its user-contributed modules to be available on your computer. These can be downloaded under GPL via <http://www.saga-gis.org/> or directly from <http://sourceforge.net/projects/saga-gis/>. Please check `rsaga.env` to make sure that RSAGA can find your local installation of SAGA.

Thanks to Olaf Conrad, Andre Ringeler and all the other SAGA developers and contributors providing this excellent geocomputing tool!

Author(s)

Alexander Brenning <brenning@uwaterloo.ca>

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`.

Author(s)

Alexander Brenning

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

<code>filename</code>	character string
<code>prefix</code>	character string: optional prefix to be added
<code>fsep</code>	character used to separate path components

Author(s)

Alexander Brenning

Examples

```
create.variable.name("C:/my-path/my-file-name.Rd", prefix="res")
```

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, dec = ".", quiet = TRUE, nlines = Inf,
  mw.to.vector = FALSE, mw.na.rm = FALSE, ...)
local.function(...)
gapply(in.grid, fun, varnames, mw.to.vector=TRUE, mw.na.rm=TRUE, ...)
```

Arguments

<code>in.grid</code>	file name of input ASCII grid, relative to <code>in.path</code>
<code>in.factor.grid</code>	optional file name giving a gridded categorical variables defining zones; zone boundaries are used as breaklines for the moving window (see Details)
<code>out.grid.prefix</code>	character string (optional), defining a file name prefix to be used for the output file names; a dash (-) will separate the prefix and the <code>varnames</code>
<code>path</code>	path in which to look for <code>in.grid</code> and write output grid files; see also <code>in.path</code> and <code>out.path</code> , which overwrite <code>path</code> if they are specified
<code>in.path</code>	path in which to look for <code>in.grid</code> (defaults to <code>path</code>)
<code>out.path</code>	path in which to write output grid files; defaults to <code>path</code>
<code>fun</code>	a function, or name of a function, to be applied on the moving window; see Details
<code>varnames</code>	character vector specifying the names of the variable(s) returned by <code>fun</code> ; if missing, <code>focal.function</code> will try to determine the <code>varnames</code> from <code>fun</code> itself, or from a call to <code>fun</code> if this is a function (see Details)
<code>radius</code>	numeric value specifying the (circular or square) radius of the moving window; see <code>is.pixel.radius</code> and <code>search.mode</code> ; note that all data within distance \leq <code>radius</code> will be included in the moving window, not $<$ <code>radius</code> .

<code>is.pixel.radius</code>	logical: if TRUE (default), the <code>radius</code> 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.
<code>valid.range</code>	numeric vector of length 2, specifying minimum and maximum valid values read from input file; all values <code><valid.range[1]</code> or <code>>valid.range[1]</code> will be converted to NA.
<code>nodata.values</code>	numeric vector: any values from the input grid file that should be converted to NA, in addition to the <code>nodata</code> value specified in the grid header
<code>out.nodata.value</code>	numeric: value used for storing NAs in the output file(s); if missing, use the same <code>nodata</code> value as specified in the header of the input grid file
<code>na.strings</code>	passed on to <code>scan</code>
<code>search.mode</code>	character, either <code>"circle"</code> (default) for a circular search window, or <code>"square"</code> for a squared one.
<code>digits</code>	numeric, specifying the number of digits to be used for output grid file.
<code>dec</code>	character, specifying the decimal mark to be used for input and output.
<code>quiet</code>	If TRUE, gives some output (<code>"*"</code>) after every 10th line of the grid file and when the job is done.
<code>nlines</code>	Number of lines to be processed; useful for testing purposes.
<code>mw.to.vector</code>	logical: Should the content of the moving window be coerced (from a matrix) to a vector?
<code>mw.na.rm</code>	logical: Should NAs be removed from moving window prior to passing the data to <code>fun</code> ? Only applicable when <code>mw.to.vector=TRUE</code> .
<code>...</code>	Arguments to be passed to <code>fun</code> ; <code>local.function</code> : arguments to be passed to <code>focal.function</code> .

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 $\leq (2 * radius + 1)^2$ if `mw.to.vector=TRUE`. This matrix or vector will contain the content of the moving window, which may possibly contain 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. 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("\dQuote{rmin}, \dQuote{rmed}, \dQuote{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 *abbreviated* 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 Beitrage zur Physischen Geographie und Landschaftsoekologie, 19), 23-32. <http://www.environment.uwaterloo.ca/u/brenning/Brenning-2008-RSAGA.pdf>

See Also

`multi.focal.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.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

<code>predict.column</code>	optional character string: Some predict methods (e.g. <code>predict.lda</code>) return a <code>data.frame</code> with several columns, e.g. one column per class in a classification problem. <code>predict.column</code> is used to pick the one that is of interest
<code>trace</code>	integer ≥ 0 : positive values give more (=2) or less (=1) information on predictor variables and predictions
<code>location</code>	optional location data received from <code>multi.focal.function</code> ; is added to the <code>newdata</code> object that is passed on to <code>predfun</code> .
<code>...</code>	these arguments are provided by the calling function, usually <code>multi.focal.function</code> . They contain the explanatory (predictor) variables required by the <code>fit</code> model.

Details

`grid.predict` is a simple wrapper function. First it binds the arguments in `...` together in a `data.frame` with the raw predictor variables that have been read from their grids by the caller, `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.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 Beitrage zur Physischen Geographie und Landschaftsoekologie, 19), 23-32. <http://www.environment.uwaterloo.ca/u/brenning/Brenning-2008-RSAGA.pdf>

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.focal.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.focal.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
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

Author(s)

Alexander Brenning

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)
```

match.arg.ext	<i>Extended Argument Matching</i>
---------------	-----------------------------------

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

<code>arg</code>	a character string or numeric value
<code>choices</code>	a character vector of candidate values
<code>base</code>	numeric value, specifying the numeric index assigned to the first element of <code>choices</code>
<code>several.ok</code>	logical specifying if <code>arg</code> should be allowed to have more than one element
<code>numeric</code>	logical specifying if the function should return the numerical index (counting from <code>base</code>) of the matched argument, or, by default, its name
<code>ignore.case</code>	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 signalled 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

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. 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,
  dec = ".", quiet = TRUE, nlines = Inf, mw.to.vector = FALSE,
  mw.na.rm = FALSE, 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.factor.grid	optional file name giving a gridded categorical variables defining zones; zone boundaries are used as breaklines for the moving window (see Details)
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
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

<code>in.varnames</code>	character vector: names of the variables corresponding to the <code>in.grids</code> ; if missing, same as <code>in.grids</code> ; if specified, must have the same length and order as <code>in.grids</code>
<code>out.varnames</code>	character vector specifying the name(s) of the variable(s) returned by <code>fun</code> ; if missing, <code>multi.focal.function</code> will try to determine the varnames from <code>fun</code> itself, or from a call to <code>fun</code> if this is a function (see Details)
<code>radius</code>	numeric value specifying the (circular or square) radius of the moving window; see <code>is.pixel.radius</code> and <code>search.mode</code> ; note that all data within distance \leq radius will be included in the moving window, not $<$ radius.
<code>is.pixel.radius</code>	logical: if TRUE (default), the <code>radius</code> 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.
<code>valid.ranges</code>	optional list of length <code>length(in.grids)</code> 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.
<code>nodata.values</code>	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
<code>out.nodata.value</code>	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
<code>search.mode</code>	character, either "circle" (default) for a circular search window, or "square" for a squared one.
<code>digits</code>	numeric, specifying the number of digits to be used for output grid file.
<code>dec</code>	character, specifying the decimal mark to be used for input and output.
<code>quiet</code>	If TRUE, gives some output ("*") after every 10th line of the grid file and when the job is done.
<code>nlines</code>	Number of lines to be processed; useful for testing purposes.
<code>mw.to.vector</code>	logical: Should the content of the moving window be coerced (from a matrix) to a vector?
<code>mw.na.rm</code>	logical: Should NAs be removed from moving window prior to passing the data to <code>fun</code> ? Only applicable when <code>mw.to.vector=TRUE</code> .
<code>pass.location</code>	logical: Should the x,y coordinates of grid points (center of grid cells) be passed to <code>fun</code> ? If TRUE, two additional arguments named <code>arguments x</code> and <code>y</code> are passed to <code>fun</code> ; NOTE: This currently only works for <code>radius=0</code> , otherwise a warning is produced and <code>pass.location</code> is reset to FALSE.
<code>na.strings</code>	passed on to <code>scan</code>
<code>...</code>	Arguments to be passed to <code>fun</code> ; <code>local.function</code> : arguments to be passed to <code>focal.function</code> .

Details

`multi.focal.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); see also [grid.predict](#).

`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 $\leq (2 * radius + 1) ^ 2$ 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 Beitrage zur Physischen Geographie und Landschaftsoekologie, 19), 23-32. <http://www.environment.uwaterloo.ca/u/brenning/Brenning-2008-RSAGA.pdf>

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.focal.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.focal.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` is the core function that is 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, varname, prefix,
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, shapefile	data.frame or point shapefile
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, nmax	numeric, for method="krige" only: see 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 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, Y.name	names of the variables containing the x and 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

<code>file</code>	file name (relative to <code>path</code> , default file extension <code>.asc</code>) of an ASCII grid from which to pick a variable, or an open connection to such a file
<code>path</code>	optional path to <code>file</code>
<code>varname</code>	character string: a variable name for the variable interpolated from grid file <code>file</code> in <code>pick.from.*.grid</code> ; if missing, variable name will be determined from <code>filename</code> by a call to <code>create.variable.name</code>
<code>prefix</code>	an optional prefix to be added to the <code>varname</code>
<code>nodata.values</code>	numeric vector specifying grid values that should be converted to NA; in addition to the values specified here, the <code>nodata</code> value given in the input grid's header will be used
<code>at.once</code>	logical: should the grid be read as a whole or line by line? <code>at.once=FALSE</code> is useful for processing large grids that do not fit into memory; the argument is currently by default <code>FALSE</code> for <code>method="nearest.neighbour"</code> , and it currently MUST be <code>TRUE</code> for all other methods (in these cases, <code>TRUE</code> is the default value); piecewise processing with <code>at.once=FALSE</code> is always faster than processing the whole grid <code>at.once</code>
<code>quiet</code>	logical: provide information on the progress of grid processing on screen? (only relevant if <code>at.once=FALSE</code> and <code>method="nearest.neighbour"</code>)
<code>nlines</code>	numeric: stop after processing <code>nlines</code> lines of the input grid; useful for testing purposes
<code>filename</code>	character: name of a SAGA grid file, default extension <code>.sgrd</code>
<code>prec</code>	numeric, specifying the number of digits to be used in converting a SAGA grid to an ASCII grid in <code>pick.from.saga.grid</code>
<code>na.strings</code>	passed on to <code>scan</code>
<code>env</code>	list: RSAGA geoprocessing environment created by <code>rsaga.env</code>
<code>show.output.on.console</code>	a logical (default: <code>FALSE</code>), indicates whether to capture the output of the command and show it on the R console (see <code>system</code> , <code>rsaga.geoprocessor</code>).
<code>...</code>	arguments to be passed to <code>pick.from.points</code>

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

~put references to the literature/web site here ~

See Also[grid.to.xyz](#), [read.ascii.grid](#), [write.ascii.grid](#)**Examples**

```
# 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)
```

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 in 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, dec = ".", georef = "corner")
write.ascii.grid.header(file, header, georef, dec = ".")
write.sgrd(data, file, header = NULL, prec = 7,
  georef = "corner", ...)
```

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

Arguments

<code>file</code>	file name of an ASCII grid (extension defaults to <code>.asc</code> if not specified), or a connection open for reading or writing, as required
<code>fname</code>	file name of a grid stored as an R (<code>.Rd</code>) file; extension defaults to <code>.Rd</code>
<code>return.header</code>	logical: should the grid header be returned (default), or just the grid data matrix? In the former case, <code>read.ascii.grid</code> returns a list with two components named <code>data</code> and <code>header</code> .
<code>print</code>	numeric, specifying how detailed the output reporting the progress should be (currently 0 to 2, 0 being minimum output).
<code>nodata.values</code>	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 <code>NA</code> .
<code>at.once</code>	logical: if <code>TRUE</code> , read the whole grid with one <code>scan</code> command; if <code>FALSE</code> , read it row by row using <code>scan</code> with option <code>nlines=1</code> .
<code>data</code>	grid data: a data matrix, or a list with components <code>data</code> (the grid data matrix) and <code>header</code> (the grid header information).
<code>header</code>	optional list argument specifying the grid header information as returned by the <code>read.ascii.grid</code> or <code>read.ascii.grid.header</code> function; see Details
<code>write.header</code>	logical: should the header be written with the grid data? (default: <code>TRUE</code>)
<code>digits</code>	numeric: if not missing, write grid data rounded to this many digits
<code>dec</code>	character (default: <code>"."</code>): decimal mark used in input or output file
<code>georef</code>	character: specifies whether the output grid should be georeferenced by the <code>"center"</code> or <code>"corner"</code> of its lower left grid cell; defaults to <code>"corner"</code> .
<code>compress</code>	logical: should the <code>.Rd</code> file written by <code>write.Rd.file</code> be compressed? (default: <code>TRUE</code>)
<code>prec</code>	integer: number of digits of temporary ASCII grid used for importing or exporting a SAGA grid
<code>na.strings</code>	passed on to <code>scan</code> .
<code>...</code>	<code>read.sgrid, write.sgrid</code> : additional arguments to be passed to <code>rsaga.geoprocessor</code>

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:

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

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

`write.ascii.grid`, `write.ascii.grid.header`, `read.Rd.grid`

`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.

Author(s)

Alexander Brenning

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

```
resid.median(x)
resid.minmedmax(x)
resid.quantiles(x)
resid.quantile(x, probs)
```

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.quantiles`), 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.

Author(s)

Alexander Brenning

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. THIS SAGA MODULE CURRENTLY SEEMS TO CRASH SAGA (but not R).

Usage

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

Arguments

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

Note

This function uses module 0 in SAGA 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

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

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, ...)
```

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
`...` 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.

Author(s)

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

See Also

[rsaga.geoprocessor](#)

rsaga.env *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. Such a list is required by all RSAGA geoprocessing functions.

Usage

```
rsaga.env( workspace=".", cmd="saga_cmd.exe", path, modules,
           check.libpath=TRUE, check.SAGA=TRUE, check.PATH=TRUE,
           check.windowsdefault=.Platform$OS.type=="windows" )
```

Arguments

<code>workspace</code>	path of the working directory for SAGA; defaults to the current directory (" . ").
<code>cmd</code>	name of the SAGA command line program; defaults to <code>saga_cmd.exe</code> , its name under Windows
<code>path</code>	path in which to find <code>cmd</code> ; <code>rsaga.env</code> is usually able to find SAGA on your system if it is installed; see Details.
<code>modules</code>	path in which to find SAGA libraries; see Details
<code>check.libpath</code>	if TRUE (default), first look for SAGA in the folder where the RSAGA package is installed
<code>check.SAGA</code>	if TRUE (default), next check the path given by the environment variable SAGA, if it exists
<code>check.PATH</code>	if TRUE (default), next look for SAGA in all the paths in the PATH environment variable
<code>check.windowsdefault</code>	if TRUE, look for SAGA in the folder <code>C:/Progra~1/saga_vc</code> . While SAGA does not have an installer program, unpacking the downloaded SAGA zip file would currently create a folder <code>saga_vc</code> . Therefore this should be a good guess.

Details

`rsaga.env` tries to compile information on the SAGA environment; this is not easy because there is no standard installation folder and procedure. If `path` is missing, `rsaga.env` first looks for an environment variable SAGA; if this is undefined, it checks the current working directory, then the paths given in the PATH environment variable, and finally the function's guess is "C:/Progra~1/saga_vc".

The default `modules` folder is the value of the `SAGA_MLB` environment variable. If this is undefined, the "modules" subfolder of the `path` folder is `rsaga.env`'s final guess.

Value

A list with components `workspace`, `cmd`, `path`, and `modules`, 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`.

There is no guarantee that this package will work with a version of SAGA other than 2.0.2. It is therefore recommended to keep a copy of this SAGA version in the RSAGA library folder, where

RSAGA will by default look first SAGA. This version will then be used even if a newer version of SAGA is available somewhere else on your computer.

Author(s)

Alexander Brenning

Examples

```
## Not run:
# Check the default RSAGA environment on your computer:
rsaga.env()
# SAGA data in C:/sagadata, binaries in C:/saga_vc:
myenv <- rsaga.env(workspace="C:/sagadata", path="C:/saga_vc")
# 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"
## 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

<code>fun</code>	function: one of the RSAGA geoprocessing functions, such as <code>rsaga.close.gaps</code> or <code>rsaga.hillshade</code> etc.
<code>in.esri</code>	logical: are input grids provided as ESRI grids (<code>in.esri=TRUE</code>) or as SAGA grids?
<code>out.esri</code>	logical: should output grids be converted to ESRI grids?
<code>env</code>	RSAGA environment as returned by <code>rsaga.env</code>
<code>esri.workspace</code>	directory for the input and output ESRI ASCII/binary grids

<code>format</code>	output file format, either "ascii" (default; equivalent: <code>format=1</code>) for ASCII grids or "binary" (equivalent: 0) for binary ESRI grids (<code>.flt</code>).
<code>georef</code>	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.
<code>prec</code>	number of digits when writing floating point values to ASCII grid files (only relevant if <code>out.esri=TRUE</code>)
<code>esri.extension</code>	extension for input/output ESRI grids: defaults to <code>.asc</code> for <code>format="ascii"</code> , and to <code>.flt</code> for <code>format="binary"</code>
<code>condensed.res</code>	logical: return only results of the RSAGA geoprocessing function <code>fun(condensed.res=TRUE)</code> , 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)
<code>clean.up</code>	logical: delete intermediate SAGA grid files?
<code>intern</code>	intern argument to be passed to rsaga.geoprocessor ; see Value
<code>...</code>	additional arguments for <code>fun</code> ; NOTE: ESRI ASCII/float raster file names should NOT include the file extension (<code>.asc</code> , <code>.flt</code>); the file extension is defined by the <code>esri.extension</code> and <code>format</code> 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.

Author(s)

Alexander Brenning

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=F
# 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

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

Arguments

<code>in.grid</code>	input: SAGA grid file (default file extension: <code>.sgrd</code>)
<code>out.grid</code>	output: SAGA grid file
<code>sigma</code>	numeric, >0.0001: standard deviation parameter of Gauss filter
<code>radius</code>	positive integer: radius of moving window
<code>...</code>	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> 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 in the SAGA library `grid_filter`.

This SAGA module had a bug under 2.0.1 which has been corrected in version 2.0.2. (SAGA used to crash when this module was called.)

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, ...)
```

Arguments

<code>in.grid</code>	input: SAGA grid file (default file extension: <code>.sgrd</code>)
<code>out.grid</code>	output: SAGA grid file
<code>mode</code>	character or numeric: shape of moving window, either "square" (=0) or "circle" (=1, default)
<code>method</code>	character or numeric: "smooth" (=0), "sharpen" (=1), or "edge" (=2)
<code>radius</code>	positive integer: radius of moving window
<code>...</code>	optional arguments to be passed to rsaga.geoprocessor , including the <code>env</code> 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.

The `mode` argument is passed to SAGA as a `-MODE` command line option. This option used to be called `-SEARCH_MODE` under SAGA 2.0.1, so this function will cause an error under SAGA 2.0.1.

Note

This function uses module 0 in the SAGA library `grid_filter`.

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(), silent = TRUE,
  beep.off, show.output.on.console = TRUE, invisible = TRUE,
  intern = TRUE, env = rsaga.env(), display.command = FALSE,
  reduce.intern=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).
silent	SAGA command line option <code>-silent</code> : avoid asking for interactive user input; does NOT turn off SAGA's beep.
beep.off	is currently ignored (did never really work and produced some unwanted side effects); a warning is produced if <code>beep.off</code> is specified.
show.output.on.console	a logical (default: <code>TRUE</code>), 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
env	A SAGA geoprocessing environment, i.e. currently 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: <code>FALSE</code> .
reduce.intern	If <code>intern=TRUE</code> , reduce the text output of SAGA returned to R by eliminating redundant lines showing the progress of module execution etc. (default: <code>TRUE</code>).
...	Additional arguments to be passed to 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.

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 cancelled out 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)

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 -silent -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 to perform a full-text search among these functions.

Usage

```
rsaga.get.libraries(path = rsaga.env()$modules,
  dll = .Platform$dynlib.ext)
rsaga.get.lib.modules(lib, env = rsaga.env(), interactive = FALSE)
rsaga.get.modules(libs, env = rsaga.env(), ...)
rsaga.search.modules(text, modules, search.libs = TRUE,
  search.modules = TRUE, env = rsaga.env(),
  ignore.case = TRUE, ...)
```

Arguments

<code>text</code>	character string to be searched for in the names of available libraries and/or modules
<code>search.libs</code> , <code>search.modules</code>	logical (default TRUE: should <code>text</code> be searched for in library and/or module names?
<code>ignore.case</code>	logical (default FALSE): should the text search in library/module names be case sensitive?
<code>lib</code> , <code>libs</code>	character vector (<code>libs</code>) or character string (<code>lib</code>) with the name(s) of library/ies in which to look for modules; if <code>libs</code> is missing, all libraries will be processed
<code>modules</code>	optional list: result of <code>rsaga.get.modules</code> ; if missing, a list of available modules will be retrieved using that function
<code>env</code>	list, setting up a SAGA geoprocessing environment as created by <code>rsaga.env</code>
<code>path</code>	path of SAGA library files (<code>modules</code> subfolder in the SAGA installation folder); defaults to the path determined by <code>rsaga.env</code> .
<code>dll</code>	file extension of dynamic link libraries

`interactive` logical (default `FALSE`): should modules be returned that can only be executed in interactive mode (i.e. using SAGA GUI)?

... currently only `interactive` to be passed on to `rsaga.get.lib.modules`

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:

<code>name</code>	the names of all modules in library <code>lib</code> ,
<code>code</code>	their numeric identifiers,
<code>interactive</code>	and 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](#)), or the RSAGA interface function, if available.

Author(s)

Alexander Brenning

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.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

<code>lib</code>	name of the SAGA library
<code>module</code>	name or numeric identifier of SAGA module in library <code>lib</code>
<code>env</code>	list, setting up a SAGA geoprocessing environment as created by <code>rsaga.env</code>
<code>show</code>	logical (default: TRUE; display usage on 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`. For information on the usage and arguments

Value

The character vector with usage information is invisibly returned.

Author(s)

Alexander Brenning

See Also

`rsaga.geoprocessor`, `rsaga.env`, `rsaga.html.help`

Examples

```
## Not run:
rsaga.get.usage("io_grid", 1)
rsaga.get.usage("ta_preprocessor", 2)
rsaga.get.usage("ta_morphometry", 0)
## 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, ...)
rsaga.linear.combination(in.grids, out.grid,
    coef, cf.digits = 16, remove.zeros = FALSE,
    remove.ones = TRUE, ...)
```

Arguments

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

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
`sqrt(a)` square root
`ln(a)` natural 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 (default) a character vector with the module's console output.

Note

This function uses module 1 in the SAGA library `grid_calculus`.

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

Arguments

<code>in.grids</code>	Input: names of (possibly several) SAGA grid files to be converted into a point shapefile.
<code>in.grid</code>	Input: SAGA grid file from which to sample.
<code>out.shapefile</code>	Output: point shapefile (default extension: <code>.shp</code>). Existing files will be overwritten!
<code>in.clip.polygons</code>	optional polygon shapefile to be used for clipping/masking an area
<code>exclude.nodata</code>	logical (default: <code>TRUE</code>): skip 'nodata' grid cells?
<code>freq</code>	integer ≥ 1 : sampling frequency in percent
<code>...</code>	Optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> RSAGA geoprocessing environment.

Note

These functions use modules 3 and 4 in SAGA library `shapes_grid`.

Author(s)

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

See Also

[rsaga.add.grid.values.to.points](#)

Examples

```
## Not run:
rsaga.grid.to.points.randomly("dem", "dempoints", freq = 20)
## End(Not run)
```

rsaga.hillshade	<i>Analytical hillshading</i>
-----------------	-------------------------------

Description

Analytical hillshading calculation.

Usage

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

Arguments

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

This function uses module 0 from SAGA library `ta_lighting`.

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 tries to obtain SAGA's HTML help for the specified library or module. NOTE: HTML help files are not provided with all SAGA distributions.

Usage

```
rsaga.html.help(lib, module, env = rsaga.env(), ...)
```

Arguments

<code>lib</code>	name of the SAGA library, or one of the <code>rsaga.</code> module functions such as rsaga.hillshade
<code>module</code>	name or numeric identifier of SAGA module in library <code>lib</code> ; <code>module=NULL</code> links to the main help page of the SAGA library <code>lib</code>
<code>env</code>	list, setting up a SAGA geoprocessing environment as created by rsaga.env
<code>...</code>	additional arguments to be passed to browseURL

Details

Doesn't seem to work with SAGA GIS 2.0.2+, needs to be updated, sorry. Please use [rsaga.get.usage](#) or [rsaga.search.modules](#) or [rsaga.get.modules](#) instead.

Deprecated details on this function: This help is not always available, there are some mismatches between libraries and their HTML files, and the HTML files are designed for use with SAGA GUI, not with the command line version. Some HTML help pages are also linked to the wrong module; in this case the SAGA library's help page may provide a link to the module's help page.

In many cases [rsaga.get.usage](#) will provide more reliable information.

Author(s)

Alexander Brenning

See Also

[rsaga.get.usage](#), [rsaga.geoprocessor](#), [rsaga.env](#), [browseURL](#)

Examples

```
## Not run: rsaga.html.help(rsaga.parallel.processing)
```

`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, ...)
```

Arguments

<code>in.grid</code>	file name of a grid in a format supported by GDAL
<code>out.grid</code>	output SAGA grid file name; defaults to <code>in.grid</code> with the file extension being removed; file extension should not be specified, it defaults to <code>.sgrd</code>
<code>...</code>	additional arguments to be passed to <code>rsaga.geoprocessor</code>

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. More information is available at <http://www.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.

The following raster formats are currently supported:

VRT	Virtual Raster
GTiff	GeoTIFF
NITF	National Imagery Transmission Format
HFA	Erdas Imagine Images (.img)
SAR-CEOS	CEOS SAR Image
CEOS	CEOS Image
ELAS	ELAS
AIG	Arc/Info Binary Grid
AAIGrid	Arc/Info ASCII Grid
SDTS	SDTS Raster
DTED	DTED Elevation Raster
PNG	Portable Network Graphics
JPEG	JPEG JFIF
MEM	In Memory Raster
JDEM	Japanese DEM (.mem)
GIF	Graphics Interchange Format (.gif)
ESAT	Envisat Image Format
BSB	Maptech BSB Nautical Charts
XPM	X11 PixMap Format
BMP	MS Windows Device Independent Bitmap
PCIDSK	PCIDSK Database File
PNM	Portable Pixmap Format (netpbm)
DOQ1	USGS DOQ (Old Style)
DOQ2	USGS DOQ (New Style)
ENVI	ENVI .hdr Labelled

EHdr ESRI .hdr Labelled
PAux PCI .aux Labelled
MFF Atlantis MFF Raster
MFF2 Atlantis MFF2 (HKV) Raster
FujiBAS Fuji BAS Scanner Image
GSC GSC Geogrid
FAST EOSAT FAST Format
BT VTP .bt (Binary Terrain) 1.3 Format
LIB NOAA Polar Orbiter Level 1b Data Set
FIT FIT Image
USGSDEM USGS Optional ASCII DEM
GXF GeoSoft Grid Exchange Format

Author(s)

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

References

GDAL website: <http://www.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.

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, ...)
```

Arguments

<code>in.dem</code>	Name of input digital elevation model (DEM) grid in SAGA grid format (default extension: <code>.sgrd</code>)
<code>in.vapour</code>	Optional input: SAGA grid file giving the water vapour pressure in mbar
<code>in.latitude</code>	Optional input: SAGA grid file giving for each pixel the latitude in degree
<code>in.longitude</code>	Optional input: SAGA grid file giving for each pixel the longitude in degree
<code>out.direct</code>	Optional output grid file for direct insolation
<code>out.diffuse</code>	Optional output grid file for diffuse insolation
<code>out.total</code>	Optional output grid file for total insolation, i.e. the sum of direct and diffuse insolation
<code>horizontal</code>	logical; project radiation onto a horizontal surface? (default: <code>FALSE</code> , i.e. use the actual inclined surface as a reference area)
<code>solconst</code>	solar constant in Joule; default: 8.164 J
<code>atmosphere</code>	height of atmosphere in m; default: 12000m
<code>water.vapour.pressure</code>	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
<code>type</code>	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"
<code>time.step</code>	time resolution in hours for discretization within a day
<code>day.step</code>	time resolution in days for a range of days
<code>days</code>	numeric vector of length 2, specifying the first and last day of a range of days (for types 2 and 3)
<code>moment</code>	if <code>type="moment"</code> or <code>"same.moment.range.of.days"</code> , <code>moment</code> specifies the time of the day (hour between 0 and 24) for which the insolation is to be calculated
<code>latitude</code>	if no <code>in.latitude</code> grid is given, this will specify a fixed geographical latitude for the entire grid
<code>bending</code>	should planetary bending be modeled? (default: <code>FALSE</code>)
<code>radius</code>	planetary radius
<code>lat.offset</code>	latitude relates to grids "bottom" (equivalent code: 0), "center" (1), "top" (2), or "user"-defined reference (default: "user"); in the latter case, <code>lat.ref.user</code> defines the reference
<code>lat.ref.user</code>	if <code>in.latitude</code> is missing and <code>lat.offset="user"</code> , then this numeric value defines the latitudinal reference (details??)
<code>lon.offset</code>	local time refers to grid's "left" edge (code 0), "center" (1), "right" edge (2), or a "user"-defined reference.
<code>lon.ref.user</code>	if <code>in.longitude</code> is missing and <code>lon.offset="user"</code> , then this numeric value defines the reference of the local time (details??)
<code>...</code>	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> 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 3 from SAGA library `ta_lighting`.

Author(s)

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

See Also

[rsaga.solar.radiation](#), [rsaga.hillshade](#)

`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 = 100, nmax = 10,
  target = rsaga.target(), ...)
rsaga.nearest.neighbour(in.shapefile, out.grid, field,
  target = rsaga.target(), ...)
rsaga.modified.quadratic.shephard(in.shapefile, out.grid, field,
  quadratic.neighbors = 13, weighting.neighbors = 19,
  target = rsaga.target(), ...)
```

Arguments

<code>in.shapefile</code>	Input: point shapefile (default extension: <code>.shp</code>).
<code>out.grid</code>	Output: filename for interpolated grid (SAGA grid file). Existing files will be overwritten!
<code>field</code>	numeric(!): number (not name!) of attribute in the shapefile's attribute table to be interpolated; the first attribute is represented by a zero.
<code>power</code>	numeric (>0): exponent used in inverse distance weighting (usually 1 or 2)
<code>maxdist</code>	numeric: maximum distance of points to be used for inverse distance interpolation (search radius)
<code>nmax</code>	Maximum number of nearest points to be used for interpolation
<code>quadratic.neighbors</code>	integer >=5; ??
<code>weighting.neighbors</code>	integer >=3; ??
<code>target</code>	list: parameters identifying the target area, e.g. the lower left corner and size of grid, or name of a reference grid; see <code>rsaga.target</code> .
<code>...</code>	Optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env RSAGA</code> geoprocessing environment.

Details

Inverse distance weighting (IDW) uses module 0 in the SAGA library `grid_gridding`. Nearest neighbour interpolation uses module 1, and triangulation is performed by module 4.

Note

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

Modified Quadratic Shepard 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.ordinary.kriging`, and `idw` in package `gstat`.

```
rsaga.local.morphometry
    Local Morphometry
```

Description

Calculates local morphometric terrain attributes (i.e. slope, aspect and curvatures).

Usage

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

Arguments

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

Note

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

Author(s)

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

References

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: Bauer, J., Rohdenburg, H., Bork, H.-R. (1985): Ein Digitales Reliefmodell als Voraussetzung fuer ein deterministisches Modell der Wasser- und Stoff-Fluesse. Landschaftsgenese und Landschaftsoekologie, H. 10, Parametereaufbereitung 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 Polynom Haralick, R.M. (1983): Ridge and valley detection on digital images. Computer Vision, Graphics and Image Processing, 22(1): 28-38.

See Also

[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.ordinary.kriging
      Local Ordinary Kriging
```

Description

Perform ordinary kriging using a local search neighborhood (local ordinary kriging). Also supports block kriging.

Usage

```
rsaga.ordinary.kriging(in.shapefile, out.grid,
                      out.variance.grid, field,
                      model = c("spherical", "exponential", "gaussian"),
                      nugget = 0, sill = 10, range = 100,
                      log.transform = FALSE, maxdist = 1000, blocksize,
                      nmin = 4, nmax = 20, target = rsaga.target(), ...)
```

Arguments

<code>in.shapefile</code>	Input: point shapefile (default extension: .shp).
<code>out.grid</code>	Output: filename for interpolated grid (SAGA grid file). Existing files will be overwritten!
<code>out.variance.grid</code>	Output (optional): SAGA grid for kriging variances
<code>field</code>	numeric(!): number (not name!) of attribute in the shapefile's attribute table to be interpolated; the first attribute is represented by a zero.
<code>model</code>	character: variogram model to be used; defaults to "spherical".
<code>nugget</code>	numeric (≥ 0): Nugget effect
<code>sill</code>	numeric (≥ 0): Sill of the variogram
<code>range</code>	numeric (≥ 0): Variogram range
<code>log.transform</code>	logical: apply a log transformation to the observations? (default: FALSE).
<code>maxdist</code>	numeric: maximum distance of nearest points to be used for kriging (search radius)
<code>nmin</code>	numeric: Minimum number of points (within the local search neighborhood) required for interpolation.
<code>nmax</code>	numeric: Maximum number of nearest points to be used for interpolation
<code>blocksize</code>	numeric: block size for block kriging; block kriging is applied if this parameter is specified. If <code>blocksize</code> is missing (default), ordinary (point) kriging is used.
<code>target</code>	list: parameters identifying the target area, e.g. the lower left corner and size of grid, or name of a reference grid; see rsaga.target .
<code>...</code>	Optional arguments to be passed to rsaga.geoprocessor , including the <code>env</code> RSAGA geoprocessing environment.

Note

This function uses module 4 ("Ordinary Kriging") in SAGA library `grid_gridding` (users of the GUI of SAGA GIS should not be confused by the fact that the "Ordinary Kriging" module appears first in the GUI's module listing - it is in fact module 4).

The SAGA module support some other variogram models(?), but I am not quite sure what they are doing, so they (and the associated additional parameters) are currently not supported by this wrapper function. The module's usage page also mentions a `FORMULA` argument, but this seems to be a mistake.

Author(s)

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

See Also

`rsaga.inverse.distance`, `rsaga.target`; see also `krige` in package `gstat`.

Examples

```
## Not run:
# Krige attribute 0 from the points shapefile to
# a grid with the same extent and resolutionn as the
# (pre-existing) geology grid:
rsaga.ordinary.krigening("points", "dem", field = 0, maxdist = 1000,
  target = rsaga.target(target="target.grid",
    target.grid = "geology"))
# Specify a target grid manually (see rsaga.target):
rsaga.ordinary.krigening("points", "dem", field = 0, radius = 1000,
  target = rsaga.target("grid.system",
    system.nx = 200, system.ny = 300,
    system.xy = c(604853,7465013), system.d = 50))
## 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.

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, ...)
```

Arguments

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

Note

This function uses module 0 from SAGA library `ta_hydrology`.

Author(s)

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

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 Voraussetzung 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 Ressources 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.

See Also

[rsaga.wetness.index](#), [rsaga.geoprocessor](#), [rsaga.env](#)

`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

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

Description

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

<code>in.dem</code>	input: digital elevation model (DEM) as SAGA grid file (default file extension: <code>.sgrd</code>)
<code>in.sinkroute</code>	optional input: sink route grid file
<code>out.dem</code>	output: modified DEM
<code>method</code>	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
<code>...</code>	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> 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, thrheight = 100, ...)
```

Arguments

<code>in.dem</code>	input: digital elevation model (DEM) as SAGA grid file (default file extension: <code>.sgrd</code>)
<code>out.sinkroute</code>	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
<code>threshold</code>	logical: use a threshold value?
<code>thrheight</code>	numeric: threshold value (default: 100)
<code>...</code>	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> 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`.

This function uses module 0 from SAGA library `ta_preprocessor`.

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.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.

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, ...)
```

Arguments

<code>in.dem</code>	name of input digital elevation model (DEM) grid in SAGA grid format (default extension: <code>.sgrd</code>)
<code>out.grid</code>	output grid file for potential incoming solar radiation sums
<code>out.duration</code>	Optional output grid file for duration of insolation
<code>latitude</code>	Geographical latitude in degree North (negative values indicate southern hemisphere)
<code>unit</code>	unit of the <code>out.grid</code> output: "kWh/m2" (default) or "J/m2"
<code>solconst</code>	solar constant, defaults to 1367 W/m2
<code>method</code>	specifies how the atmospheric components should be accounted for: either based on a lumped atmospheric transmittance as specified by argument <code>transmittance</code> ("lumped", or numeric code 0; default); or by calculating the components corresponding to water and dust ("components", code 1)
<code>transmittance</code>	transmittance of the atmosphere in percent; usually between 60 (humid areas) and 80 percent (deserts)
<code>pressure</code>	atmospheric pressure in mbar
<code>water.content</code>	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	a list with components <code>day</code> and <code>month</code> specifying a single day of the year for radiation modeling
day.step	if <code>days</code> indicates a range of days, this specifies the time step (number of days) for calculating the incoming solar radiation
...	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env</code> RSAGA geoprocessing environment

Note

SAGA 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 2 from SAGA 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)
# ...in fact a cold, cloudy, windy day...
## 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) fitting the extent to the input data, (2) an existing SAGA grid file, (3) user-defined parameters, or (4) the header data of an ASCII grid. Intended to be used with RSAGA's interpolation functions.

Usage

```
rsaga.target(target = c("user.defined", "grid.system",
  "target.grid", "header"),
  user.cellsize = 100, user.fit.extent = TRUE,
  user.x.extent, user.y.extent, user.bbox,
  system.nx, system.ny, system.xy, system.d,
  target.grid, header)
```

Arguments

target	character: method used for defining the target grid
user.fit.extent	Only for target="user.defined": logical; if TRUE, use the dimensions of an input grid supplied to the SAGA module, e.g. to <code>rsaga.ordinary.kriging</code> . The other <code>user.*</code> variables should not be provided if <code>user.fit.extent=TRUE</code> .
user.cellsize	Only for target="user.defined": raster resolution
user.x.extent, user.y.extent	Only for target="user.defined": numeric vectors of length 2: minimum and maximum coordinates of grid cell center points
user.bbox	Only for target="user.defined": alternative way of specifying extent (either use <code>bbox</code> OR <code>user.*.extent</code>): 2x2 matrix of the form <code>rbind(user.x.extent, user.y</code>
system.nx, system.ny	Only for target="grid.system": number of columns and rows of the grid
system.xy	Only for target="grid.system": numeric vector of length 2 giving the x and y coordinates at the center of the grid's lower left cell
system.d	Only for target="grid.system": cellsize
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
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 <code>ncols</code> , <code>nrows</code> , <code>cellsize</code> , and either <code>x/yllcorner</code> or <code>x/yllcenter</code> .

Note

This function is to be used with RSAGA functions `rsaga.ordinary.kriging`, `rsaga.inverse.distance`, `rsaga.nearest.neighbour` and `rsaga.modified.quadratic.shephard`.

Author(s)

Alexander Brenning

See Also

`read.ascii.grid.header`

Examples

```
## Not run:
# Krige attribute 0 from the points shapefile to
# a grid with the same extent and resolutionn as the
# (pre-existing) geology grid:
rsaga.ordinary.kriging("points", "dem", field = 0, maxdist = 1000,
  target = rsaga.target(target="target.grid",
    target.grid = "geology"))
# Specify a target grid manually (see above):
rsaga.ordinary.kriging("points", "dem", field = 0, radius = 1000,
  target = rsaga.target("grid.system",
    system.nx = 200, system.ny = 300,
    system.xy = c(604853,7465013), system.d = 50))
## End(Not run)
```

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, t.param, ...)
```

Arguments

`in.dem` input: digital elevation model (DEM) as SAGA grid file (default file extension: `.sgrd`)

`out.wetness.index` output (optional): wetness index grid. Existing files of the same name will be overwritten!

<code>out.carea</code>	output (optional): catchment area
<code>out.cslope</code>	output (optional): catchment slope
<code>out.mod.carea</code>	output (optional): modified catchment area
<code>t.param</code>	positive numeric value (optional): undocumented
<code>...</code>	optional arguments to be passed to <code>rsaga.geoprocessor</code> , including the <code>env RSAGA</code> geoprocessing environment

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.

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 15 from the SAGA library `ta_hydrology`.

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.

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 as having no extension
extension	character string: file extension, without the dot
fsep	character: separator between paths
force	logical argument to <code>default.file.extension</code> : force the file extension to be extension (same result as <code>set.file.extension</code>), or only set it to extension if it has not been specified?

Value

character vector of same length as filename

Author(s)

Alexander Brenning

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")
```

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

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

$$\text{Shelter index}(S) = \arctan(\max((z(x_0) - z(x)) / |x_0 - x| : x \text{ in } S)),$$

where $S = S(x_0, a, da, d)$ is the set of grid nodes within a distance $\leq d$ from x_0 , only considering grid nodes in directions between $a - da$ and $a + da$ from x_0 .

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 $2 * (\text{ceiling}(\text{radius}) + 1)$ columns and rows:

<code>mask</code>	indicates which grid cell in the moving window is within the specified circle segment (value <code>FALSE</code>) or not (<code>TRUE</code>)
<code>dist</code>	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 onto 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

*Topic **file**

- `read.ascii.grid`, 19
- `rsaga.esri.to.sgrd`, 27
- `rsaga.import.gdal`, 44
- `rsaga.sgrd.to.esri`, 55
- `set.file.extension`, 64

*Topic **interface**

- `read.ascii.grid`, 19
- RSAGA-package, 2
- `rsaga.add.grid.values.to.points`, 23
- `rsaga.close.gaps`, 24
- `rsaga.contour`, 25
- `rsaga.env`, 26
- `rsaga.esri.to.sgrd`, 27
- `rsaga.esri.wrapper`, 28
- `rsaga.fill.sinks`, 30
- `rsaga.filter.gauss`, 32
- `rsaga.filter.simple`, 33
- `rsaga.geoprocessor`, 34
- `rsaga.get.modules`, 36
- `rsaga.get.usage`, 38
- `rsaga.grid.calculus`, 39
- `rsaga.grid.to.points`, 41
- `rsaga.hillshade`, 42
- `rsaga.html.help`, 43
- `rsaga.import.gdal`, 44
- `rsaga.insolation`, 46
- `rsaga.inverse.distance`, 48
- `rsaga.local.morphometry`, 50
- `rsaga.ordinary.kriging`, 52
- `rsaga.parallel.processing`, 53
- `rsaga.sgrd.to.esri`, 55
- `rsaga.sink.removal`, 57
- `rsaga.sink.route`, 58
- `rsaga.solar.radiation`, 59
- `rsaga.target`, 61
- `rsaga.wetness.index`, 62

*Topic **package**

- RSAGA-package, 2

*Topic **spatial**

- `focal.function`, 4
- `grid.predict`, 7
- `grid.to.xyz`, 9
- `multi.focal.function`, 12
- `pick.from.points`, 16
- `read.ascii.grid`, 19
- `relative.position`, 21
- `resid.median`, 22
- RSAGA-package, 2
- `rsaga.add.grid.values.to.points`, 23
- `rsaga.close.gaps`, 24
- `rsaga.contour`, 25
- `rsaga.env`, 26
- `rsaga.esri.to.sgrd`, 27
- `rsaga.esri.wrapper`, 28
- `rsaga.fill.sinks`, 30
- `rsaga.filter.gauss`, 32
- `rsaga.filter.simple`, 33
- `rsaga.geoprocessor`, 34
- `rsaga.get.modules`, 36
- `rsaga.get.usage`, 38
- `rsaga.grid.calculus`, 39
- `rsaga.grid.to.points`, 41
- `rsaga.hillshade`, 42
- `rsaga.import.gdal`, 44
- `rsaga.insolation`, 46
- `rsaga.inverse.distance`, 48
- `rsaga.local.morphometry`, 50
- `rsaga.ordinary.kriging`, 52
- `rsaga.parallel.processing`, 53
- `rsaga.sgrd.to.esri`, 55
- `rsaga.sink.removal`, 57
- `rsaga.sink.route`, 58
- `rsaga.solar.radiation`, 59
- `rsaga.target`, 61
- `rsaga.wetness.index`, 62

- wind.shelter, 65
- *Topic **utilities**
 - centervalue, 3
 - create.variable.name, 3
 - match.arg.ext, 10
 - rsaga.html.help, 43
 - set.file.extension, 64
- abbreviate, 6
- browseURL, 44
- centervalue, 3, 21, 22
- create.variable.name, 3, 7, 17
- default.file.extension
 - (set.file.extension), 64
- focal.function, 3, 4, 9, 14, 15, 21, 22, 40, 65, 66
- gapply (focal.function), 4
- get.file.extension
 - (set.file.extension), 64
- grid.predict, 7, 9, 12–15
- grid.to.xyz, 9, 18
- local.function, 40
- local.function (focal.function), 4
- match.arg, 10, 11
- match.arg.ext, 10
- median, 22
- multi.focal.function, 7, 8, 12, 40
- pick.from.ascii.grid, 10, 23
- pick.from.ascii.grid
 - (pick.from.points), 16
- pick.from.points, 16, 23
- pick.from.saga.grid, 23
- pick.from.saga.grid
 - (pick.from.points), 16
- pick.from.shapefile
 - (pick.from.points), 16
- pmatch, 11
- predict, 8
- quantile, 22, 66
- rank, 21
- read.ascii.grid, 10, 18, 19
- read.ascii.grid.header, 61, 62
- read.Rd.grid (read.ascii.grid), 19
- read.sgrd (read.ascii.grid), 19
- relative.position, 7, 21
- relative.rank, 7
- relative.rank
 - (relative.position), 21
- resid.median, 3, 7, 22
- resid.minmedmax, 6, 7
- resid.minmedmax (resid.median), 22
- resid.quantile, 6, 7
- resid.quantile (resid.median), 22
- resid.quartiles, 7
- resid.quartiles (resid.median), 22
- RSAGA (RSAGA-package), 2
- RSAGA-package, 2
- rsaga.add.grid.values.to.points, 23, 42
- rsaga.aspect
 - (rsaga.local.morphometry), 50
- rsaga.close.gaps, 24, 29
- rsaga.close.one.cell.gaps
 - (rsaga.close.gaps), 24
- rsaga.contour, 25
- rsaga.curvature
 - (rsaga.local.morphometry), 50
- rsaga.env, 2, 17, 24, 26, 28–30, 34–38, 40, 44, 51, 55, 56, 63
- rsaga.esri.to.sgrd, 27, 29, 30
- rsaga.esri.wrapper, 28, 28, 35, 56
- rsaga.fill.sinks, 30, 57, 58
- rsaga.filter.gauss, 32, 33
- rsaga.filter.simple, 32, 33
- rsaga.geoprocessor, 17, 23–25, 28–33, 34, 37–44, 47–52, 54–58, 60, 63
- rsaga.get.lib.modules
 - (rsaga.get.modules), 36
- rsaga.get.libraries, 35
- rsaga.get.libraries
 - (rsaga.get.modules), 36
- rsaga.get.modules, 35, 36, 44
- rsaga.get.usage, 35, 37, 38, 44
- rsaga.grid.calculus, 39
- rsaga.grid.to.points, 23, 41
- rsaga.hillshade, 29, 35, 38, 42, 44, 48, 60

`rsaga.html.help`, 37, 38, 43
`rsaga.import.gdal`, 44
`rsaga.insolation`, 43, 46, 60
`rsaga.inverse.distance`, 48, 53, 62
`rsaga.linear.combination`
 (`rsaga.grid.calculus`), 39
`rsaga.local.morphometry`, 38, 50
`rsaga.modified.quadratic.shephard`,
 62
`rsaga.modified.quadratic.shephard`
 (`rsaga.inverse.distance`),
 48
`rsaga.nearest.neighbour`, 62
`rsaga.nearest.neighbour`
 (`rsaga.inverse.distance`),
 48
`rsaga.ordinary.kriging`, 49, 52, 62
`rsaga.parallel.processing`, 51, 53,
 63
`rsaga.plan.curvature`
 (`rsaga.local.morphometry`),
 50
`rsaga.profile.curvature`
 (`rsaga.local.morphometry`),
 50
`rsaga.search.modules`, 35, 44
`rsaga.search.modules`
 (`rsaga.get.modules`), 36
`rsaga.sgrd.to.esri`, 29, 30, 55
`rsaga.sink.removal`, 32, 57, 58
`rsaga.sink.route`, 32, 57, 58
`rsaga.slope`
 (`rsaga.local.morphometry`),
 50
`rsaga.solar.radiation`, 43, 48, 59
`rsaga.target`, 49, 52, 53, 61
`rsaga.triangulation`
 (`rsaga.inverse.distance`),
 48
`rsaga.wetness.index`, 55, 62

`scan`, 5, 14, 17, 20
`set.file.extension`, 64
`system`, 17, 34, 35

`wind.shelter`, 7, 65
`write.ascii.grid`, 18
`write.ascii.grid`
 (`read.ascii.grid`), 19
`write.Rd.grid`(`read.ascii.grid`),
 19
`write.sgrd`(`read.ascii.grid`), 19