Package: GeoDist (via r-universe)

August 27, 2024

Type Package

Title Constrained distance calculation and associated geotools

Version 0.1.0.8000

Author Sébastien Rochette

Maintainer Sébastien Rochette <sebastienrochettefr@gmail.com>

Description This package allows the calculation of distances that are constrained by frontiers, islands, mountains, ... These distances are then implemented in classical geotools like kriging with a modified version of geoR functions.

License GPL-3

Depends R (>= 3.0), geoR

Imports dplyr, graphics, gstat, igraph, magrittr, methods, parallel, raster, readr, rlang, snow, sp, stats, tcltk, utils

Suggests knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Repository https://statnmap.r-universe.dev

RemoteUrl https://github.com/statnmap/GeoDist

RemoteRef HEAD

RemoteSha c536c04e90041499214a6e2c95d9edc3c4f56037

Contents

autofitVariogram.dist *Automatically fitting a variogram*

Description

Automatically fitting a variogram to the data on which it is applied. The automatic fitting is done through fit.variogram. In fit.variogram the user had to supply an initial estimate for the sill, range etc. autofitVariogram provides this estimate based on the data and then calls fit.variogram. For details, see autofitVariogram

Usage

```
autofitVariogram.dist(
  formula,
  input_data,
  model = c("Sph", "Exp", "Gau", "Ste"),
  kappa = c(0.05, seq(0.2, 2, 0.1), 5, 10),
  fix.values = c(NA, NA, NA),
  verbose = FALSE,
  GLS.model = NA,
  start_vals = c(NA, NA, NA),
  cutoff,
  miscFitOptions = list(),
  dist.mat = NULL,
  ...
)
```

Arguments

`formula`	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name 'z', for ordinary and simple kriging use the formula 'z~1'; for simple kriging also define 'beta' (see below); for universal kriging, suppose 'z' is linearly dependent on 'x' and 'y', use the formula 'z~x+y'.

`input_data`	An object of SpatialPointsDataFrame-class.
`model`	The list of variogrammodels that will be tested.
`kappa`	Smoothing parameter of the Matern model. Provide a list if you want to check more than one value.
`fix.values`	Can be used to fix a variogram parameter to a certain value. It consists of a list with a length of three. The items describe the fixed value for the nugget, range and sill respectively. They need to be given in that order. Setting the value to NA means that the value is not fixed.
`verbose`	logical, if TRUE the function will give extra feedback on the fitting process
`GLS.model`	If a variogram model is passed on through this parameter a Generalized Least Squares sample variogram is calculated.
`start_vals`	Can be used to give the starting values for the variogram fitting. The items describe the fixed value for the nugget, range and sill respectively. They need to be given in that order. Setting the value to NA means that the value will be automatically chosen.
`cutoff`	Parameter included in variogram (maximum distance of variogram). Default to diagonal * 0.35.
`miscFitOptions`	A list with named arguments that provide additional control over the fitting process. For example: `list(merge.small.bins = TRUE)`. If the list is empty, autofitVariogram uses default values. The following parameters can be set: **`merge.small.bins`:** logical, when TRUE, the function checks if there are bins with less than 5 points. If so, the first two bins are merged and the check is repeated. This is done until all bins have more than `min.np.bin` points. **`min.np.bin`:** integer, the minimum number of points allowed in a bin before we start merging bins. See also `merge.small.bins`.
`dist.mat`	Square matrix of distances between points of the dataset ## Not implemented yet
`...`	parameters that are passed on to variogram when calculating the sample variogram.

`dist.obstacle` *Calculate 2D distances from points to points avoiding obstacles*

Description

Calculate 2D distances from points to points avoiding obstacles

Usage

```
dist.obstacle(
  from,
  to,
  r.ref,
  dref.r,
  n.cores,
  closest = FALSE,
  step.ang = 5,
  r.ref3D = FALSE,
  filename = paste0(tempdir(), "/dref.r.RData"),
  small.dist = TRUE,
  keep.path = FALSE,
  longlat = raster::isLonLat(r.ref),
  tol = 1e-05,
  igraph = FALSE
)
```

Arguments

`from`	Point data set from which to calculate distances. May be a 2-col matrix of (x,y) coordinates or a SpatialPointDataFrame. Should be in same projection than r.ref. This can also be an object of class distref.data.
`to`	Point data set from which to calculate distances. May be a 2-col matrix of (x,y) coordinates or a SpatialPointDataFrame. Should be in same projection than r.ref. If not set to = from. This can also be an object of class distref.data. If "to" is provided, it should be the bigger dataset.
`r.ref`	reference raster used for distance without obstacles calculation
`dref.r`	object of class distref.raster. Reference raster already processed using distref.raster (e.g. usefull if previously saved using filename.)
`n.cores`	number of cores to run parallel calculation. Default to all cores using parallel::detectCores()
`closest`	if a point is over an obstacle, set to TRUE to find the closest non-obstacle r.ref cell
`step.ang`	Step for angles classes breaks (in degrees). This means that a distance error is allowed in future calculations. Distance error (%) = 100 * (1 - cos((step.ang/2)*pi/180)). With step.ang = 5 degrees, error < 0.1%. Default set to 5.
`r.ref3D`	Logical. Wether the reference raster should be used as a 3D surface to calculate distance accounting for elevation.
`filename`	Path where to save the dref.r issued from reference raster. This may be usefull for re-using the raster for different datasets. Default to paste0(tempdir(),"/dref.r.RData")
`small.dist`	if TRUE, distances <= 1 paths are directly calculated.
`keep.path`	Logical. Whether to keep SpatialLines of paths. This allows real distances when 2 points are in the same raster cell or close.
`longlat`	Logical. if FALSE, Euclidean distance, if TRUE Great Circle distance. Defined from r.ref.
`tol`	numeric Tolerance to define distances between points to be null.
`igraph`	Logical. Wether to calculate all distances through igraph (TRUE) or use `Pt2Pts.wo.obstacle` to use igraph only when necessary. Using igraph maybe less precise but a little quicker. Default to FALSE.

Value

If keep.path is false, returns a matrix of distances with rows = "from" and cols = "to". If keep.path is true, returns a list where dist.mat is the matrix of distance and listLines is a list of SpatialLines. Each list is a SpatialLinesDataFrame corresponding to a starting point in "from".

Examples

```
## Not run:
# For "from" and "to" being SpatialPoints
allpath <- dist.obstacle(from, to, r.ref, keep.path = TRUE)
plot(from)
points(to)
# All path from "from[1,]" to all "to"
lines(allpath$allLines[[1]])

## End(Not run)
```

`distref.data` *Get characteristics of data regarding the reference raster*

Description

Get characteristics of data regarding the reference raster

Usage

```
distref.data(
  data.pt,
  r.ref,
  closest = FALSE,
  longlat = raster::isLonLat(r.ref)
)
```

Arguments

`data.pt`	Point data set. May be a 2-col matrix of (x,y) coordinates or a SpatialPoint-DataFrame. Should be in same projection than r.ref
`r.ref`	Reference raster
`closest`	if a point is over an obstacle, set to TRUE to find the closest non-obstacle r.ref cell
`longlat`	Logical. if FALSE, Euclidean distance, if TRUE Great Circle distance. Defined from r.ref.

`distref.data-class` *Class for distref.data*

Description

Class for distref.data

Slots

`data.pt.N` numeric.

`NA.pos` numeric

`NA.dist` numeric

`distref.raster` *Transfrom reference raster as a network of paths for distance calculation*

Description

Transfrom reference raster as a network of paths for distance calculation

Usage

```
distref.raster(
  r.ref,
  step.ang = 5,
  filename = paste0(tempdir(), "/dref.r.RData"),
  r.ref3D = FALSE,
  n.cores,
  save.rdists
)
```

Arguments

`r.ref`	Raster (1-layer) that will be used to calculate distances. Its resolution impacts precision on distances.
`step.ang`	Step for angles classes breaks (in degrees). This means that a distance error is allowed in future calculations. Distance error (%) = 100 * (1 - cos((step.ang/2)*pi/180)). With step.ang = 5 degrees, error < 0.1%. Default set to 5.
`filename`	Path where to save the dref.r issued from reference raster. This may be usefull for re-using the raster for different datasets. Default to paste0(tempdir(), "/dref.r.RData")
`r.ref3D`	Logical. Wether the reference raster should be used as a 3D surface to calculate distance accounting for elevation.

`n.cores`	number of cores to run parallel calculation. Default to all cores using parallel::detectCores()
`save.rdists`	optional. Path to file where to save the distances matrix between all non-NA raster cells. Saved as csv. If not provided, distances matrix will not be calculated as it may be to big to be stored in memory.

Details

Not all direct possible paths are calculated. Direct paths are retained in a focal window around cells. The size of the focal window is determined by the step.ang value. All paths retained are stored in a undirected graph (as from library igraph). All paths real distances are also stored to be weights for future points to points distances calculations.

Value

coords.r coordinates of all cells of the raster r.ref.NA.nb cell that are NA in the reference raster (obstacles) cell.Relpos.ngb row and col of neighbors retained after the angle calculation, relative to the cell of origin (moving window) cell.Relpos.cross row and col of cells crossed by lines between origin and neighbors, relative to the cell of origin (moving window)

`distref.raster-class` *Class for distref.raster*

Description

Class for distref.raster

Slots

`r.ref.path` path of the raster

`r.ref.dim` dimensions of the raster

`r.ref.N` cells numerotation

`r.ref.NA.nb` numerotation of NA cells

`adj.ref.graph` adjacency graph

`path.w` weight (distance) of paths

`eyefit.large` *Eyefit function of geoR with higher range for phi and sigma*

Description

Eyefit function of geoR with higher range for phi and sigma

Usage

```
eyefit.large(vario, max.phi = 2, max.sigma = 2, silent = FALSE)
```

Arguments

`vario`	An empirical variogram object as returned by the function `variog`.
`max.phi`	Numeric as multiplied by max(vario$u). Default to 2.
`max.sigma`	Numeric as multiplied by max(vario$v). Default to 2.
`silent`	logical indicating wheather or not the fitted variogram must be returned.

`Find.ngb.wo.obstacle` *Find neighbours without obstacles in different directions*

Description

Find neighbours without obstacles in different directions

Usage

```
Find.ngb.wo.obstacle(
  i,
  r.ref,
  coords.r,
  r.ref.NA.nb,
  cell.Relpos.ngb,
  cell.Relpos.cross
)
```

Arguments

`i`	cell number
`r.ref`	reference raster
`coords.r`	coordinates of the reference raster
`r.ref.NA.nb`	Numerotation of NA ref.raster cells
`cell.Relpos.ngb`	Relative position of neighbours in the moving window
`cell.Relpos.cross`	Numerotation of positions of neighbours of the moving window

Value

matrix with all neighbours of each cell that are not NA

`idw.dist` *Inverse distance calculation using custom distances*

Description

Inverse distance calculation using custom distances

Usage

```
idw.dist(
  data,
  coords,
  locations,
  loc.dist,
  idp = 2,
  max.dist,
  nmin = 1,
  nmax,
  longlat = TRUE
)
```

Arguments

`data`	vector of values to be interpolated at data coordinates
`coords`	coordinates of observation points. Can also be an object of class SpatialPoints*. Not necessary with loc.dist.
`locations`	coordinates of points where to provide predictions. Can also be an object of class SpatialPoints*. Not necessary with loc.dist.
`loc.dist`	Matrix of distances between data (rows) and locations (cols)
`idp`	specify the inverse distance weighting power. Default to 2.
`max.dist`	Maximum distance radius for neighbours search in the idw interpolation. maxdist is in km when longlat is TRUE (non-projected crs), in meters otherwise. If loc.dist is specified, maxdist should be in same distance unit.
`nmin`	if the number of nearest observations within distance maxdist is less than nmin, a missing value will be generated; see maxdist
`nmax`	the number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations. By default, all observations are used
`longlat`	Logical Wether coordinates are not-projected (TRUE) or are projected (FALSE).

idw.dist.special *Special case idw for reduced time calculation*

Description

Special case idw for reduced time calculation

Usage

```
idw.dist.special(data, loc.dist.order, loc.dist.nmax, idp)
```

Arguments

`data`	vector of data to interpolate on points. Refer to output of Prep_loc_dist
`loc.dist.order`	matrix of nmax lines and nb points columns with position of data to keep
`loc.dist.nmax`	matrix of nmax lines and nb points columns with distances between kept data and points position
`idp`	specify the inverse distance weighting power. Default to 2.

krige.conv.dist *Kriging. Modified from function krige.conv in geoR library.*

Description

Kriging. Modified from function krige.conv in geoR library.

Usage

```
krige.conv.dist(
  geodata,
  coords = geodata$coords,
  data = geodata$data,
  locations,
  borders,
  krige,
  output,
  dist.mat,
  loc.dist,
  loc.loc.dist
)
```

Arguments

`geodata`	a list containing elements `coords` and `data` as described next. Typically an object of the `class` `"geodata"` - a **geoR** data-set. If not provided the arguments `coords` and `data` must be provided instead.
`coords`	an $n \times 2$ matrix or data-frame with the 2-D coordinates of the n data locations. By default it takes the component `coords` of the argument `geodata`, if provided.
`data`	a vector with *n* data values. By default it takes the component `data` of the argument `geodata`, if provided.
`locations`	an $N \times 2$ matrix or data-frame with the 2-D coordinates of the N prediction locations, or a list for which the first two components are used. Input is internally checked by the function `check.locations`.
`borders`	optional. By default reads the element `borders` from the `geodata` object, if present. Setting to `NULL` prevents this behavior. If a two column matrix defining a polygon is provided the prediction is performed only at locations inside this polygon.
`krige`	a list defining the model components and the type of kriging. It can take an output to a call to `krige.control` or a list with elements as for the arguments in `krige.control`. Default values are assumed for arguments or list elements not provided. See the description of arguments in 'krige.control' below.
`output`	a list specifying output options. It can take an output to a call to `output.control` or a list with elements as for the arguments in `output.control`. Default values are assumed for arguments not provided. See documentation for `output.control` for further details.
`dist.mat`	Square matrix of distances between points of the dataset
`loc.dist`	is a n (data) x N (locations) matrix with distances between data points and prediction locations.
`loc.loc.dist`	is a N (locations) x N (locations) matrix with distances between prediction locations.

`likfit.dist` *Fit of gaussian field. Modified from function likfit in geoR.*

Description

Fit of gaussian field. Modified from function likfit in geoR.

Usage

```
likfit.dist(
  geodata,
  coords = geodata$coords,
  data = geodata$data,
  trend = "cte",
  ini.cov.pars,
  fix.nugget = FALSE,
  nugget = 0,
  fix.kappa = TRUE,
  kappa = 0.5,
  fix.lambda = TRUE,
  lambda = 1,
  fix.psiA = TRUE,
  psiA = 0,
  fix.psiR = TRUE,
  psiR = 1,
  cov.model,
  dist.mat,
  realisations,
  lik.method = "ML",
  components = TRUE,
  nospatial = TRUE,
  limits = pars.limits(),
  print.pars = FALSE,
  messages,
  ...
)
```

Arguments

`geodata`	a list containing elements `coords` and `data` as described next. Typically an object of the class `"geodata"`. If not provided the arguments `coords` and `data` must be provided instead.
`coords`	an $n \times 2$ matrix where each row has the 2-D coordinates of the n data locations. By default it takes the component `coords` of the argument `geodata`, if provided.
`data`	a vector with *n* data values. By default it takes the component `data` of the argument `geodata`, if provided.
`trend`	specifies the mean part of the model. See documentation of `trend.spatial` for further details. Defaults to `"cte"`.
`ini.cov.pars`	initial values for the covariance parameters: σ^2 (partial sill) and ϕ (range parameter). Typically a vector with two components. However a matrix can be used to provide several initial values. See `DETAILS` below.
`fix.nugget`	logical, indicating whether the parameter τ^2 (nugget variance) should be regarded as fixed (`fix.nugget = TRUE`) or should be estimated (`fix.nugget = FALSE`). Defaults to `FALSE`.
`nugget`	value of the nugget parameter. Regarded as a fixed value if `fix.nugget = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to zero.
`fix.kappa`	logical, indicating whether the extra parameter κ should be regarded as fixed (`fix.kappa = TRUE`) or should be estimated (`fix.kappa = FALSE`). Defaults to `TRUE`.
`kappa`	value of the extra parameter κ. Regarded as a fixed value if `fix.kappa = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 0.5.

	This parameter is valid only if the covariance function is one of: `"matern"`, `"powered.exponential"`, `"cauchy"` or `"gneiting.matern"`. For more details on covariance functions see documentation for `cov.spatial`.
`fix.lambda`	logical, indicating whether the Box-Cox transformation parameter λ should be regarded as fixed (`fix.lambda = TRUE`) or should be be estimated (`fix.lambda = FALSE`). Defaults to `TRUE`.
`lambda`	value of the Box-Cox transformation parameter λ. Regarded as a fixed value if `fix.lambda = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 1. Two particular cases are $\lambda = 1$ indicating no transformation and $\lambda = 0$ indicating log-transformation.
`fix.psiA`	logical, indicating whether the anisotropy angle parameter ψ_R should be regarded as fixed (`fix.psiA = TRUE`) or should be estimated (`fix.psiA = FALSE`). Defaults to `TRUE`.
`psiA`	value (in radians) for the anisotropy angle parameter ψ_A. Regarded as a fixed value if `fix.psiA = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 0. See `coords.aniso` for further details on anisotropy correction.
`fix.psiR`	logical, indicating whether the anisotropy ratio parameter ψ_R should be regarded as fixed (`fix.psiR = TRUE`) or should be estimated (`fix.psiR = FALSE`). Defaults to `TRUE`.
`psiR`	value, always greater than 1, for the anisotropy ratio parameter ψ_R. Regarded as a fixed value if `fix.psiR = TRUE` otherwise as the initial value for the minimisation algorithm. Defaults to 1. See `coords.aniso` for further details on anisotropy correction.
`cov.model`	a string specifying the model for the correlation function. For further details see documentation for `cov.spatial`. Reads values from an `variomodel` object passed to `ini.cov.pars` if any, otherwise defaults to the *exponential* model.
`dist.mat`	Square matrix of distances between data points
`realisations`	optional. Logical or a vector indicating the number of replication for each datum. For further information see `DETAILS` below and documentation for `as.geodata`.
`lik.method`	(formely method.lik) options are `"ML"` for maximum likelihood and `"REML"` for restricted maximum likelihood. Defaults to `"ML"`.
`components`	an $n \times 3$ data-frame with fitted values for the three model components: trend, spatial and residuals. See the section `DETAILS` below for the model specification.
`nospatial`	logical. If `TRUE` parameter estimates for the model without spatial component are included in the output.
`limits`	values defining lower and upper limits for the model parameters used in the numerical minimisation. The auxiliary function `pars.limits` is called to set the limits. See also **Limits** in DETAILS below.
`print.pars`	logical. If `TRUE` the parameters and the value of the negative log-likelihood (up to a constant) are printed each time the function to be minimised is called.
`messages`	logical. Indicates whether status messages should be printed on the screen (or output device) while the function is running.

`...`	additional parameters to be passed to the minimisation function. Typically arguments of the type `control()` which controls the behavior of the minimisation algorithm. For further details see documentation for the minimisation function `optim`.

`Prep_loc_dist` *Prepare loc.dist for idw special case*

Description

Prepare loc.dist for idw special case

Usage

```
Prep_loc_dist(loc.dist, nmin = 1, nmax, max.dist)
```

Arguments

`loc.dist`	Matrix of distances between data (rows) and locations (cols)
`nmin`	if the number of nearest observations within distance maxdist is less than nmin, a missing value will be generated; see maxdist
`nmax`	the number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations. By default, all observations are used
`max.dist`	Maximum distance radius for neighbours search in the idw interpolation.

`Pt2Pts.wo.obstacle` *Calculate distance from one point to a set of points using adjacency matrix*

Description

Calculate distance from one point to a set of points using adjacency matrix

Usage

```
Pt2Pts.wo.obstacle(
  dref.from,
  dref.to,
  dref.r,
  r.ref,
  longlat,
  small.dist = TRUE,
  keep.path = FALSE
)
```

Arguments

`dref.from`	object of class distref.data for a single point
`dref.to`	object of class distref.data
`dref.r`	object of class distref.raster corresponding to r.ref
`r.ref`	reference raster (only used for its coordinates here)
`longlat`	Logical. if FALSE, Euclidean distance, if TRUE Great Circle distance. Defined from r.ref.
`small.dist`	if TRUE, distances <= 1 paths are directly calculated.
`keep.path`	Logical. Whether to keep SpatialLines of paths. This allows real distances when 2 points are in the same raster cell or close.

Value

Vector of distances of length(B)

Examples

```
## Not run:
# Need gdistance ??

## End(Not run)
```

---

sph2car *Computes cartesian coordinates from long,lat geographical coordinates Use of geoid with radius and flattening to calculate correct x,y,z coordinates*

Description

Computes cartesian coordinates from long,lat geographical coordinates Use of geoid with radius and flattening to calculate correct x,y,z coordinates

Usage

```
sph2car(long, lat, radius = 6378137, f = 1/298.257223563, deg = TRUE)
```

Arguments

`long`	longitude values, can also contain a matrix of (long, lat), or (long, lat and and radius) in that order.
`lat`	latitude values.
`radius`	major (equatorial) radius of the ellipsoid. The default value is for WGS84
`f`	numeric. Ellipsoid flattening. The default value is for WGS84
`deg`	Specifies if input is in degrees (default) or radians.

Details

Use geosphere::refEllipsoids() for other radius and f possibilites.

`variog.dist` *Variogram calculation. Modified from geoR.*

Description

Variogram calculation. Modified from geoR.

Usage

```
variog.dist(
  geodata,
  coords = geodata$coords,
  data = geodata$data,
  uvec = "default",
  breaks = "default",
  trend = "cte",
  lambda = 1,
  option = c("bin", "cloud", "smooth"),
  estimator.type = c("classical", "modulus"),
  nugget.tolerance,
  max.dist,
  dist.mat,
  pairs.min = 2,
  bin.cloud = FALSE,
  direction = "omnidirectional",
  tolerance = pi/8,
  unit.angle = c("radians", "degrees"),
  angles = FALSE,
  messages,
  ...
)
```

Arguments

`geodata`	a list containing element `coords` as described next. Typically an object of the class `"geodata"` - a **geoR** data-set. If not provided the arguments `coords` must be provided instead.
`coords`	an $n \times 2$ matrix containing coordinates of the n data locations in each row. Defaults to `geodata$coords`, if provided.
`data`	a vector or matrix with data values. If a matrix is provided, each column is regarded as one variable or realization. Defaults to `geodata$data`, if provided.

`uvec`	a vector with values used to define the variogram binning. Only used when `option = "bin"`. See `DETAILS` below for more details on how to specify the bins.
`breaks`	a vector with values to define the variogram binning. Only used when `option = "bin"`. See `DETAILS` below for more details on how to specify the bins.
`trend`	specifies the mean part of the model. See documentation of `trend.spatial` for further details. Defaults to `"cte"`.
`lambda`	values of the Box-Cox transformation parameter. Defaults to 1 (no transformation). If another value is provided the variogram is computed after transforming the data. A case of particular interest is $\lambda = 0$ which corresponds to log-transformation.
`option`	defines the output type: the options `"bin"` returns values of binned variogram, `"cloud"` returns the variogram cloud and `"smooth"` returns the kernel smoothed variogram. Defaults to `"bin"`.
`estimator.type`	`"classical"` computes the classical method of moments estimator. `"modulus"` returns the variogram estimator suggested by Hawkins and Cressie (see Cressie, 1993, pg 75). Defaults to `"classical"`.
`nugget.tolerance`	a numeric value. Points which are separated by a distance less than this value are considered co-located. Defaults to zero.
`max.dist`	a numerical value defining the maximum distance for the variogram. Pairs of locations separated for distance larger than this value are ignored for the variogram calculation. If not provided defaults takes the maximum distance among all pairs of data locations.
`dist.mat`	Square matrix of distances between points of the dataset
`pairs.min`	a integer number defining the minimum numbers of pairs for the bins. For `option = "bin"`, bins with number of pairs smaller than this value are ignored. Defaults to `NULL`.
`bin.cloud`	logical. If `TRUE` and `option = "bin"` the cloud values for each class are included in the output. Defaults to `FALSE`.
`direction`	a numerical value for the directional (azimuth) angle. This used to specify directional variograms. Default defines the omnidirectional variogram. The value must be in the interval $[0, \pi]$ radians ($[0, 180]$ degrees).
`tolerance`	numerical value for the tolerance angle, when computing directional variograms. The value must be in the interval $[0, \pi/2]$ radians ($[0, 90]$ degrees). Defaults to $\pi/8$.
`unit.angle`	defines the unit for the specification of angles in the two previous arguments. Options are `"radians"` and `"degrees"`, with default to `"radians"`.
`angles`	Logical with default to `FALSE`. If `TRUE` the function also returns the angles between the pairs of points (unimplemented).
`messages`	logical. Indicates whether status messages should be printed on the screen (or output device) while the function is running.
`...`	arguments to be passed to the function `ksmooth`, if `option = "smooth"`.

Index