

Package ‘gstat’

February 1, 2005

Version 0.9-18

Date 2004/11/18

Title uni- and multivariable geostatistical modelling, prediction and simulation

Author Edzer J. Pebesma <e.pebesma@geog.uu.nl> and others

Maintainer Edzer J. Pebesma <e.pebesma@geog.uu.nl>

Description variogram modelling; simple, ordinary and universal point or block (co)kriging, sequential Gaussian or indicator (co)simulation;

Depends R (>= 2.0.0)

Imports lattice

License GPL

URL <http://www.gstat.org/>

R topics documented:

bpy.colors	2
bubble	3
fit.lmc	4
fit.variogram	5
fit.variogram.reml	7
fulmar	8
get.contr	9
gstat-internal	10
gstat	10
image	13
krige.cv	15
krige	17
makegrid	19
mapasp	20
map.to.lev	20
meuse.all	21
meuse.alt	22
meuse.grid	23
meuse	24

ncp.grid	25
ossfim	26
oxford	27
pcb	28
plot.gstatVariogram	29
plot.pointPairs	31
plot.variogramCloud	33
point.in.polygon	34
predict.gstat	35
select.spatial	38
show.vgms	39
sic2004	40
variogram.line	42
variogram	43
vgm	45
zerodist	47

Index **49**

bpy.colors	<i>blue-pink-yellow color scheme that prints well as grey tone</i>
------------	--

Description

Create a vector of ‘n’ “contiguous” colors.

Usage

```
bpy.colors(n = 100, cutoff.tails = 0.1)
```

Arguments

`n` number of colors (≥ 1) to be in the palette

`cutoff.tails` tail fraction to be cut off. This palette runs from black to white if `cutoff.tails` is 0; by cutting off the tails, it runs from blue to yellow, which looks nicer.

Value

A character vector, ‘cv’, of color names. This can be used either to create a user-defined color palette for subsequent graphics by ‘palette(cv)’, a ‘col=’ specification in graphics functions or in ‘par’.

Note

This color map prints well on black-and-white printers.

Author(s)

unknown

References

<http://www.ihe.uni-karlsruhe.de/mitarbeiter/vonhagen/palette.en.html>;
gnuplot has this color map

See Also

[rainbow](#), [cm.colors](#)

Examples

```
bpy.colors(10)
p <- expand.grid(x=1:30,y=1:30)
p$z <- p$x + p$y
image(p, col = bpy.colors(100))
# require(lattice)
# trellis.par.set("regions", list(col=bpy.colors())) # make default
```

bubble	<i>Create a bubble plot of spatial data</i>
--------	---

Description

Create a bubble plot of spatial data, with options for bicolour residual plots (xyplot wrapper)

Usage

```
bubble(data, xcol = 1, ycol = 2, zcol = 3, fill = TRUE, maxsize = 3,
       do.sqrt = TRUE, pch, col = c(2,3), key.entries = quantile(data[,zcol]),
       main, identify = FALSE, labels = row.names(data), ...)
```

Arguments

data	data frame from which x- and y-coordinate and z-variable are taken
xcol	x-coordinate column number or (quoted) name
ycol	y-coordinate column number or (quoted) name
zcol	z-variable column number or (quoted) name
fill	logical; if TRUE, filled circles are plotted (pch = 16), else open circles (pch = 1); the pch argument overrides this
maxsize	cex value for largest circle
do.sqrt	logical; if TRUE the plotting symbol area (sqrt(diameter)) is proportional to the value of the z-variable; if FALSE, the symbol size (diameter) is proportional to the z-variable
pch	plotting character
col	colours to be used; numeric vector of size two: first value is for negative values, second for positive values.
key.entries	the values that will be plotted in the key; by default the five quantiles min, q.25, median q.75, max
main	character; plot title

identify	logical; if true, regular plot is called instead of <code>xyplot</code> , and followed by a call to <code>identify()</code> .
labels	labels argument, passed to <code>plot</code> when <code>identify</code> is TRUE
...	arguments, passed to <code>xyplot</code> , or <code>plot</code> if identification is required.

Value

returns (or plots) the bubble plot; if `identify` is TRUE, returns the indexes (row numbers, not the row.names shown) of identified points.

Author(s)

Edzer J. Pebesma

References**See Also**

`xyplot`, `mapasp`, `identify`

Examples

```
data(meuse)
bubble(meuse, max = 2.5, main = "cadmium concentrations (ppm)",
       key.entries = c(.5,1,2,4,8,16))
bubble(meuse, "x", "y", "zinc", main = "zinc concentrations (ppm)",
       key.entries = 100 * 2^(0:4))
```

fit.lmc

Fit a Linear Model of Coregionalization to a Multivariable Sample Variogram

Description

Fit a Linear Model of Coregionalization to a Multivariable Sample Variogram; in case of a single variogram model (i.e., no nugget) this is equivalent to Intrinsic Correlation

Usage

```
fit.lmc(v, g, model, fit.ranges = FALSE, fit.lmc = !fit.ranges,
       correct.diagonal = 1.0, ...)
```

Arguments

v	multivariable sample variogram, output of <code>variogram</code>
g	gstat object, output of <code>gstat</code>
model	variogram model, output of <code>vgm</code> ; if supplied this value is used as initial value for each fit

<code>fit.ranges</code>	logical; determines whether the range coefficients (excluding that of the nugget component) should be fitted; or logical vector: determines for each range parameter of the variogram model whether it should be fitted or fixed.
<code>fit.lmc</code>	logical; if TRUE, each coefficient matrices of partial sills is guaranteed to be positive definite
<code>correct.diagonal</code>	multiplicative correction factor to be applied to partial sills of direct variograms only; the default value, 1.0, does not correct. If you encounter problems with singular covariance matrices during cokriging or cosimulation, you may want to try to increase this to e.g. 1.01
<code>...</code>	parameters that get passed to fit.variogram

Value

returns an object of class `gstat`, with fitted variograms;

Note

This function does not use the iterative procedure proposed by M. Goulard and M. Voltz (Math. Geol., 24(3): 269-286; reproduced in Goovaerts' 1997 book) but uses simply two steps: first, each variogram model is fitted to a direct or cross variogram; next each of the partial sill coefficient matrices is approached by its in least squares sense closest positive definite matrices (by setting any negative eigenvalues to zero).

The argument `correct.diagonal` was introduced by experience: by zeroing the negative eigenvalues for fitting positive definite partial sill matrices, apparently still perfect correlation may result, leading to singular cokriging/cosimulation matrices. If someone knows of a more elegant way to get around this, please let me know.

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org/>

See Also

[variogram](#), [vgm](#), [fit.variogram](#), `demo(cokriging)`

Examples

<code>fit.variogram</code>	<i>Fit a Variogram Model to a Sample Variogram</i>
----------------------------	--

Description

Fit ranges and/or sills from a simple or nested variogram model to a sample variogram

Usage

```
fit.variogram(object, model, fit.sills = TRUE, fit.ranges = TRUE,
             fit.method = 7, debug.level = 1, warn.if.neg = FALSE )
```

Arguments

<code>object</code>	sample variogram, output of variogram
<code>model</code>	variogram model, output of vgm
<code>fit.sills</code>	logical; determines whether the partial sill coefficients (including nugget variance) should be fitted; or logical vector: determines for each partial sill parameter whether it should be fitted or fixed.
<code>fit.ranges</code>	logical; determines whether the range coefficients (excluding that of the nugget component) should be fitted; or logical vector: determines for each range parameter whether it should be fitted or fixed.
<code>fit.method</code>	fitting method, used by <code>gstat</code> . The default method uses weights N_h/h^2 with N_h the number of point pairs and h the distance. This criterion is not supported by theory, but by practice. For other values of <code>fit.method</code> , see table 4.2 in the <code>gstat</code> manual.
<code>debug.level</code>	integer; set <code>gstat</code> internal debug level
<code>warn.if.neg</code>	logical; if TRUE a warning is issued whenever a sill value of a direct variogram becomes negative

Value

returns a fitted variogram model (of class `variogram.model`).

This is a `data.frame` has two attributes: (i) `singular` a logical attribute that indicates whether the non-linear fit converged, or ended in a singularity, and (ii) `SSErr` a numerical attribute with the (weighted) sum of squared errors of the fitted model.

Note

If fitting the range(s) is part of the job of this function, the results may well depend on the starting values, given in argument `model`. This is nothing new, but generally true for non-linear regression problems. This function uses the internal `gstat` (C) code, which iterates over (a) a direct (least squares) fit of the partial sills and (b) an iterated search, using gradients, for the optimal range value(s), until convergence of after a combined step ((a) and (b)) is reached.

If for a direct (i.e. not a cross) variogram a sill parameter (partial sill or nugget) becomes negative, `fit.variogram` is called again with this parameter set to zero, and with a FALSE flag to further fit this sill. This implies that once at the search space boundary, a sill value does not never away from it.

Author(s)

Edzer J. Pebesma

References<http://www.gstat.org/>

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences, 30: 683-691.

See Also[variogram](#), [vgm](#)**Examples**

```
data(meuse)
vgml <- variogram(log(zinc)~1, ~x+y, meuse)
fit.variogram(vgml, vgm(1,"Sph",300,1))
```

fit.variogram.reml REML Fit Direct Variogram Partial Sills to Data

Description

Fit Variogram Sills to Data, using REML (only for direct variograms; not for cross variograms)

Usage

```
fit.variogram.reml(formula, locations, data, model, debug.level = 1, set, degree)
```

Arguments

formula	formula defining the response vector and (possible) regressors; in case of absence of regressors, use e.g. z~1
locations	spatial data locations; a formula with the coordinate variables in the right hand (dependent variable) side.
data	data frame where the names in formula and locations are to be found
model	variogram model to be fitted, output of vgm
debug.level	debug level; set to 65 to see the iteration trace and log likelihood
set	additional options that can be set; use <code>set=list(iter=100)</code> to set the max. number of iterations to 100.
degree	order of trend surface in the location, between 0 and 3

Valuean object of class "variogram.model"; see [fit.variogram](#)

Note

This implementation only uses REML fitting of sill parameters. For each iteration, an $n \times n$ matrix is inverted, with n the number of observations, so for large data sets this method becomes rather, ehm, demanding. I guess there is much more to likelihood variogram fitting in package `geOR`, and probably also in `nlme`.

Author(s)

Edzer J. Pebesma

References

Christensen, R. Linear models for multivariate, Time Series, and Spatial Data, Springer, NY, 1991.
Kitanidis, P., Minimum-Variance Quadratic Estimation of Covariances of Regionalized Variables, *Mathematical Geology* 17 (2), 195–208, 1985

See Also

[fit.variogram](#),

Examples

```
data(meuse)
fit.variogram.reml(log(zinc)~1, ~x+y, meuse, model = vgm(1, "Sph", 900,1))
```

fulmar

Fulmaris glacialis data

Description**Usage**

```
data(fulmar)
```

Format

This data frame contains the following columns:

year year of measurement: 1998 or 1999

x x-coordinate in UTM31

y y-coordinate in UTM31

depth sea water depth, in m

coast distance to coast, in m

fulmar observed density (number of birds per square km)

Note

References

See Also

[ncp.grid](#)

Examples

```
data(fulmar)
summary(fulmar)
```

<code>get.contr</code>	<i>Calculate contrasts from multivariable predictions</i>
------------------------	---

Description

Given multivariable predictions and prediction (co)variances, calculate contrasts and their (co)variance

Usage

```
get.contr(data, gstat.object, X, ids = names(gstat.object$data))
```

Arguments

<code>data</code>	data frame, output of predict.gstat
<code>gstat.object</code>	object of class <code>gstat</code> , used to extract ids; may be missing if <code>ids</code> is used
<code>X</code>	contrast vector or matrix; the number of variables in <code>gstat.object</code> should equal the number of elements in <code>X</code> if <code>X</code> is a vector, or the number of rows in <code>X</code> if <code>X</code> is a matrix.
<code>ids</code>	character vector with (selection of) id names, present in <code>data</code>

Details

From `data`, we can extract the $(n \times 1)$ vector with multivariable predictions, say y , and its $(n \times n)$ covariance matrix V . Given a contrast matrix in X , this function computes the contrast vector is $C = X'y$ and $Var(C) = X'VX$.

Value

a data frame containing for each row in `data` the generalized least squares estimates (named `beta.1`, `beta.2`, ...), their variances (named `var.beta.1`, `var.beta.2`, ...) and covariances (named `cov.beta.1.2`, `cov.beta.1.3`, ...)

Note

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org/>

See Also

[predict.gstat](#)

Examples

gstat-internal	<i>Gstat Internal Functions</i>
----------------	---------------------------------

Description

gstat internal functions

Note

these functions should not be called by users directly

Author(s)

Edzer J. Pebesma

gstat	<i>Create gstat objects, or subset it</i>
-------	---

Description

Function that creates gstat objects; objects that hold all the information necessary for univariate or multivariate geostatistical prediction (simple, ordinary or universal (co)kriging), or its conditional or unconditional Gaussian or indicator simulation equivalents. Multivariate gstat object can be subsetted.

Usage

```
gstat(g, id, formula, locations, data, model = NULL, beta, nmax = Inf,
      nmin = 0, maxdist = Inf, dummy = FALSE, set, fill.all = FALSE,
      fill.cross = TRUE, variance = "identity", weights = NULL, merge,
      degree = 0)
print.gstat(x, ...)
```

Arguments

<code>g</code>	gstat object to append to; if missing, a new gstat object is created
<code>id</code>	identifier of new variable; if missing, <code>varn</code> is used with <code>n</code> the number for this variable. If a cross variogram is entered, <code>id</code> should be a vector with the two <code>id</code> values, e.g. <code>c("zn", "cd")</code> , further only supplying arguments <code>g</code> and <code>model</code> . It is advisable not to use expressions, such as <code>log(zinc)</code> , as identifiers, as this may lead to complications later on.
<code>formula</code>	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name <code>z</code> , for ordinary and simple kriging use the formula <code>z~1</code> ; for simple kriging also define <code>beta</code> (see below); for universal kriging, suppose <code>z</code> is linearly dependent on <code>x</code> and <code>y</code> , use the formula <code>z~x+y</code>
<code>locations</code>	formula with only independent variables that define the spatial data locations (coordinates), e.g. <code>~x+y</code> ; if <code>data</code> is of class <code>spatial.data.frame</code> , this argument may be ignored, as it can be derived from the data
<code>data</code>	data frame; contains the dependent variable, independent variables, and locations.
<code>model</code>	variogram model for this <code>id</code> ; defined by a call to <code>vgm</code> ; see argument <code>id</code> to see how cross variograms are entered
<code>beta</code>	only for simple kriging (and simulation based on simple kriging); vector with the trend coefficients (including intercept); if no independent variables are defined the model only contains an intercept and this should be the simple kriging mean
<code>nmax</code>	for local kriging: 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
<code>nmin</code>	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
<code>maxdist</code>	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
<code>dummy</code>	logical; if TRUE, consider this data as a dummy variable (only necessary for unconditional simulation)
<code>set</code>	named list with optional parameters to be passed to <code>gstat</code> (only <code>set</code> commands of <code>gstat</code> are allowed, and not all of them may be relevant; see the manual for <code>gstat</code> stand-alone, URL below)
<code>x</code>	gstat object to print
<code>fill.all</code>	logical; if TRUE, fill all of the direct variogram and, depending on the value of <code>fill.cross</code> also all cross variogram model slots in <code>g</code> with the given variogram model
<code>fill.cross</code>	logical; if TRUE, fill all of the cross variograms, if FALSE fill only all direct variogram model slots in <code>g</code> with the given variogram model (only if <code>fill.all</code> is used)
<code>variance</code>	character; variance function to transform to non-stationary covariances; "identity" does not transform, other options are "mu" (Poisson) and "mu(1-mu)" (binomial)
<code>weights</code>	numeric vector; if present, covariates are present, and variograms are missing weights are passed to OLS prediction routines; if variograms are given, weights

	should be $1/\text{variance}$, where variance specifies location-specific measurement error as in Delhomme, J.P. Kriging in the hydrosiences. <i>Advances in Water Resources</i> , 1(5):251-266, 1978; see also the section Kriging with known measurement errors in the gstat user's manual, URL see below.
merge	either character vector of length 2, indicating two ids that share a common mean; the more general gstat merging of any two coefficients across variables is obtained when a list is passed, with each element a character vector of length 4, in the form <code>c("id1", 1, "id2", 2)</code> . This merges the first parameter for variable <code>id1</code> to the second of variable <code>id2</code> .
degree	order of trend surface in the location, between 0 and 3
...	arguments that are passed to the printing of variogram models only

Details

to print the full contents of the object `g` returned, use `as.list(g)` or `print.default(g)`

Value

an object of class `gstat`, which inherits from `list`. Its components are:

data	list; each element is a list with the formula, locations, data, nvars, beta, etc., for a variable
model	list; each element contains a variogram model; names are those of the elements of data; cross variograms have names of the pairs of data elements, separated by a <code>.</code> (e.g.: <code>var1.var2</code>)
set	list; named list, corresponding to <code>set name=value</code> ; gstat commands (look up the set command in the gstat manual for a full list)

Note

The function currently copies the data objects into the `gstat` object, so this may become a large object. I would like to copy only the name of the data frame, but could not get this to work. Any help is appreciated.

Subsetting (see examples) is done using the `id`'s of the variables, or using numeric subsets. Subsetted `gstat` objects only contain cross variograms if (i) the original `gstat` object contained them and (ii) the order of the subset indexes increases, numerically, or given the order they have in the `gstat` object.

The `merge` item may seem obscure. Still, for collocated cokriging, it is needed. See texts by Goovaerts, Wackernagel, Chiles and Delfiner, or look for standardised ordinary kriging in the 1992 *Deutsch and Journel* or *Isaaks and Srivastava*. In these cases, two variables share a common mean parameter. Gstat generalises this case: any two variables may share any of the regression coefficients; allowing for instance analysis of covariance models, when variograms left out (see e.g. R. Christensen's "Plane answers" book on linear models. The tests directory of the package contains examples in file `merge.R`).

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org/>

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30: 683-691.

See Also

[predict.gstat](#), [krige](#)

Examples

```
data(meuse)
# let's do some manual fitting of two direct variograms and a cross variogram
g <- gstat(id = "ln.zinc", formula = log(zinc)~1, locations = ~x+y,
          data = meuse)
g <- gstat(g, id = "ln.lead", formula = log(lead)~1, locations = ~x+y,
          data = meuse)
# examine variograms and cross variogram:
plot(variogram(g))
# enter direct variograms:
g <- gstat(g, id = "ln.zinc", model = vgm(.55, "Sph", 900, .05))
g <- gstat(g, id = "ln.lead", model = vgm(.55, "Sph", 900, .05))
# enter cross variogram:
g <- gstat(g, id = c("ln.zinc", "ln.lead"), model = vgm(.47, "Sph", 900, .03))
# examine fit:
plot(variogram(g), model = g$model, main = "models fitted by eye")
# see also demo(cokriging) for a more efficient approach
g["ln.zinc"]
g["ln.lead"]
g[c("ln.zinc", "ln.lead")]
g[1]
g[2]

# Inverse distance interpolation with inverse distance power set to .5:
# (kriging variants need a variogram model to be specified)
data(meuse)
data(meuse.grid)
meuse.gstat <- gstat(id = "zinc", formula = zinc ~ 1, locations = ~ x + y,
                   data = meuse, nmax = 7, set = list(idp = .5))
meuse.gstat
z <- predict(meuse.gstat, meuse.grid)
library(lattice) # for levelplot
levelplot(zinc.pred~x+y, z, aspect = mapasp(z))
# see demo(cokriging) and demo(examples) for further examples,
# and the manuals for predict.gstat and image
```

image

Image Gridded Coordinates in Data Frame

Description

Image gridded data, held in a data frame, keeping the right aspect ratio for axes, and the right cell shape

Usage

```
image.data.frame(x, zcol = 3, xcol = 1, ycol = 2, ...)
xyz2img(xyz, zcol = 3, xcol = 1, ycol = 2, tolerance = 10 * .Machine$double.eps)
```

Arguments

<code>x</code>	data frame (or matrix) with x-coordinate, y-coordinate, and z-coordinate in its columns
<code>zcol</code>	column number of z-variable
<code>xcol</code>	column number of x-coordinate
<code>ycol</code>	column number of y-coordinate
<code>...</code>	arguments, passed to <code>image.default</code>
<code>xyz</code>	data frame (same as <code>x</code>)
<code>tolerance</code>	maximum deviation for coordinates from being exactly on a regularly spaced grid

Value

`image.data.frame` plots an image from gridded data, organized in arbitrary order, in a data frame. It uses `xyz2img` and `image.default` for this. In the S-Plus version, `xyz2img` tries to make an image object with a size such that it will plot with an equal aspect ratio; for the R version, `image.data.frame` uses the `asp=1` argument to guarantee this.

`xyz2img` returns a list with components: `z`, a matrix containing the z-values; `x`, the increasing coordinates of the rows of `z`; `y`, the increasing coordinates of the columns of `z`. This list is suitable input to `image.default`.

Note

I wrote this function before I found out about `levelplot`, a Lattice/Trellis function that lets you control the aspect ratio by the `aspect` argument, and that automatically draws a legend, and therefore I now prefer `levelplot` over `image`. Plotting points on a `levelplots` is probably done with providing a panel function and using `lpoints`.

(for S-Plus only –) it is hard (if not impossible) to get exactly right cell shapes (e.g., square for a square grid) without altering the size of the plotting region, but this function tries hard to do so by extending the image to plot in either x- or y-direction. The larger the grid, the better the approximation. Geographically correct images can be obtained by modifying `par("pin")`. Read the examples, image a 2 x 2 grid, and play with `par("pin")` if you want to learn more about this.

Author(s)

Edzer J. Pebesma

References**See Also**

Examples

```

data(meuse)
data(meuse.grid)
g <- gstat(formula=log(zinc)~1,locations=~x+y,data=meuse,model=vgm(1,"Exp",300))
x <- predict(g, meuse.grid)
image(x, 4, main="kriging variance and data points")
points(meuse$x, meuse$y, pch = "+")
# non-square cell test:
image(x[((x$y - 20) %% 80) == 0,], main = "40 x 80 cells")
image(x[((x$x - 20) %% 80) == 0,], main = "80 x 40 cells")
# the following works for square cells only:
oldpin <- par("pin")
ratio <- length(unique(x$x))/length(unique(x$y))
par(pin = c(oldpin[2]*ratio,oldpin[2]))
image(x, main="Exactly square cells, using par(pin)")
par(pin = oldpin)
library(lattice)
levelplot(var1.var~x+y, x, aspect = mapasp(x), main = "kriging variance")

```

krige.cv

*(co)kriging cross validation, n-fold or leave-one-out***Description**

Cross validation functions for simple, ordinary or universal point (co)kriging, kriging in a local neighbourhood.

Usage

```

gstat.cv(object, nfold, remove.all = FALSE, verbose = FALSE,
         all.residuals = FALSE, ...)
krige.cv(formula, locations, data, model = NULL, beta = NULL, nmax = Inf,
         nmin = 0, maxdist = Inf, nfold = nrow(data), verbose = FALSE, ...)

```

Arguments

<code>object</code>	object of class <code>gstat</code> ; see function gstat
<code>nfold</code>	apply n-fold cross validation; if <code>nfold</code> is set to <code>nrow(data)</code> (the default), leave-one-out cross validation is done; if set to e.g. 5, five-fold cross validation is done
<code>remove.all</code>	logical; if TRUE, remove observations at cross validation locations not only for the first, but for all subsequent variables as well
<code>verbose</code>	logical; if TRUE, progress is printed
<code>all.residuals</code>	logical; if TRUE, residuals for all variables are returned instead of for the first variable only
<code>...</code>	other arguments that will be passed to predict.gstat in case of <code>gstat.cv</code> , or to gstat in case of <code>krige.cv</code>

formula	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name <code>z</code> , for ordinary and simple kriging use the formula <code>z~1</code> ; for simple kriging also define <code>beta</code> (see below); for universal kriging, suppose <code>z</code> is linearly dependent on <code>x</code> and <code>y</code> , use the formula <code>z~x+y</code>
locations	formula with only independent variables that define the spatial data locations (coordinates), e.g. <code>~x+y</code> ; if <code>data</code> is of class <code>spatial.data.frame</code> , this argument may be ignored, as it can be derived from the data
data	data frame; should contain the dependent variable, independent variables, and coordinates.
model	variogram model of dependent variable (or its residuals), defined by a call to <code>vgm</code> or <code>fit.variogram</code>
beta	only for simple kriging (and simulation based on simple kriging); vector with the trend coefficients (including intercept); if no independent variables are defined the model only contains an intercept and this should be the simple kriging mean
nmax	for local kriging: 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
nmin	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
maxdist	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply

Details

Leave-one-out cross validation (LOOCV) visits a data point, and predicts the value at that location by leaving out the observed value, and proceeds with the next data point. (The observed value is left out because kriging would otherwise predict the value itself.) N-fold cross validation makes a partitions the data set in N parts. For all observation in a part, predictions are made based on the remaining N-1 parts; this is repeated for each of the N parts. N-fold cross validation may be faster than LOOCV.

Value

data frame containing the coordinates of `data` or those of the first variable in `object`, and columns of prediction and prediction variance of cross validated data points, observed values, residuals, `zscore` (residual divided by kriging standard error), and `fold`.

If `all.residuals` is true, a data frame with residuals for all variables is returned, without coordinates.

Note

Leave-one-out cross validation seems to be much faster in plain (stand-alone) `gstat`, apparently quite a bit of the effort is spent moving data around from R to `gstat`.

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org/>

See Also

[krige](#), [gstat](#), [predict.gstat](#)

Examples

```
data(meuse)
m <- vgm(.59, "Sph", 874, .04)
# five-fold cross validation:
x <- krige.cv(log(zinc)~1, ~x+y, model = m, data = meuse, nmax = 40, nfold=5)
bubble(x, z = "residual", main = "log(zinc): 5-fold CV residuals")
```

krige

Simple, Ordinary or Universal, global or local, Point or Block Kriging, or simulation.

Description

Function for simple, ordinary or universal kriging (sometimes called external drift kriging), kriging in a local neighbourhood, point kriging or kriging of block mean values (rectangular or irregular blocks), and conditional (Gaussian or indicator) simulation equivalents for all kriging varieties. For multivariable prediction, see [gstat](#) and [predict.gstat](#)

Usage

```
krige(formula, locations, data, newdata, model, beta, nmax = Inf,
      nmin = 0, maxdist = Inf, block, nsim = 0, indicators = FALSE,
      na.action = na.pass, ...)
```

Arguments

formula	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name <i>z</i> , for ordinary and simple kriging use the formula <i>z~1</i> ; for simple kriging also define <i>beta</i> (see below); for universal kriging, suppose <i>z</i> is linearly dependent on <i>x</i> and <i>y</i> , use the formula <i>z~x+y</i>
locations	formula with only independent variables that define the spatial data locations (coordinates), e.g. <i>~x+y</i> ; if <i>data</i> is of class <code>spatial.data.frame</code> , this argument may be ignored, as it can be derived from the data
data	data frame; should contain the dependent variable, independent variables, and coordinates.
newdata	data frame with prediction/simulation locations; should contain columns with the independent variables (if present) and the coordinates with names as defined in <code>locations</code>
model	variogram model of dependent variable (or its residuals), defined by a call to vgm or fit.variogram

<code>beta</code>	only for simple kriging (and simulation based on simple kriging); vector with the trend coefficients (including intercept); if no independent variables are defined the model only contains an intercept and this should be the simple kriging mean
<code>nmax</code>	for local kriging: 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
<code>nmin</code>	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
<code>maxdist</code>	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
<code>block</code>	block size; a vector with 1, 2 or 3 values containing the size of a rectangular in x-, y- and z-dimension respectively (0 if not set), or a data frame with 1, 2 or 3 columns, containing the points that discretize the block in the x-, y- and z-dimension to define irregular blocks relative to (0,0) or (0,0,0)—see also the details section of predict.gstat . By default, predictions or simulations refer to the support of the data values.
<code>nsim</code>	integer; if set to a non-zero value, conditional simulation is used instead of kriging interpolation. For this, sequential Gaussian or indicator simulation is used (depending on the value of <code>indicators</code>), following a single random path through the data.
<code>indicators</code>	logical, only relevant if <code>nsim</code> is non-zero; if TRUE, use indicator simulation; else use Gaussian simulation
<code>na.action</code>	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'. Missing values in coordinates and predictors are both dealt with.
<code>...</code>	other arguments that will be passed to gstat

Details

This function is a simple wrapper function around [gstat](#) and [predict.gstat](#) for univariate kriging prediction and conditional simulation methods available in `gstat`. For multivariate prediction or simulation, or for other interpolation methods provided by `gstat` (such as inverse distance weighted interpolation or trend surface interpolation) use the functions [gstat](#) and [predict.gstat](#) directly.

For further details, see [predict.gstat](#).

Value

a data frame containing the coordinates of `newdata`, and columns of prediction and prediction variance (in case of kriging) or the `abs(nsim)` columns of the conditional Gaussian or indicator simulations

Note

Daniel G. Krige is a South African scientist who was a mining engineer when he first used generalised least squares prediction with spatial covariances in the 50's. George Matheron coined the term *kriging* in the 60's for the action of doing this, although very similar approaches had been taken in the field of meteorology. Beside being Krige's name, I consider "krige" to be to "kriging" what "predict" is to "prediction".

Author(s)

Edzer J. Pebesma

References

N.A.C. Cressie, 1993, Statistics for Spatial Data, Wiley.

<http://www.gstat.org/>

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences, 30: 683-691.

See Also[gstat](#), [predict.gstat](#)**Examples**

```

data(meuse)
data(meuse.grid)
m <- vgm(.59, "Sph", 874, .04)
# ordinary kriging:
x <- krige(log(zinc)~1, ~x+y, model = m, data = meuse, newd = meuse.grid)
library(lattice)
levelplot(varl.pred~x+y, x, aspect = mapasp(x),
           main = "ordinary kriging predictions")
levelplot(varl.var~x+y, x, aspect = mapasp(x),
           main = "ordinary kriging variance")
# simple kriging:
x <- krige(log(zinc)~1, ~x+y, model = m, data = meuse, newdata = meuse.grid,
           beta=5.9)
# residual variogram:
m <- vgm(.4, "Sph", 954, .06)
# universal block kriging:
x <- krige(log(zinc)~x+y, ~x+y, model = m, data = meuse, newdata =
           meuse.grid, block = c(40,40))
levelplot(varl.pred~x+y, x, aspect = mapasp(x),
           main = "universal kriging predictions")
levelplot(varl.var~x+y, x, aspect = mapasp(x),
           main = "universal kriging variance")

```

makegrid

make regular grid with square cells and round numbers

Description

make regular grid with square cells and round numbers

Usage

```
makegrid(x, y, n=10000, nsig=2, margin=1.05, cell.size)
```

Arguments

<code>x</code>	x-coordinate
<code>y</code>	y-coordinate
<code>n</code>	approximate number of cells in grid
<code>nsig</code>	number of significant digits to which cell size and origin are rounded
<code>margin</code>	margin around the x and y coordinate limits
<code>cell.size</code>	cell size; if missing, a reasonable, and rounded, estimate is made

Value

data frame with the following elements:

<code>x</code>	x-coordinates
<code>y</code>	y-coordinate

Examples

```
data(meuse)
grd <- makegrid(meuse$x, meuse$y, 1000)
diff(grd$x)
diff(grd$y)
summary(grd)
grd <- makegrid(meuse$x, meuse$y, cell.size = 40)
diff(grd$x)
diff(grd$y)
summary(grd)
```

mapasp

Calculate plot aspect ratio for geographic maps

Description

Calculate plot aspect ratio for geographic maps

Usage

```
mapasp(data, x = data$x, y = data$y, get.number = FALSE)
```

Arguments

<code>data</code>	data frame
<code>x</code>	x-coordinates
<code>y</code>	y-coordinates
<code>get.number</code>	logical; if true the aspect value is returned else, and in R >= 2.0.0, "iso" is returned

Value

on R version 2.0.0 or higher, returns "iso", else returns $\text{diff}(\text{range}(y))/\text{diff}(\text{range}(x))$

See Also

[image.data.frame](#), [krige](#)

map.to.lev

rearrange data frame for plotting with levelplot

Description

rearrange data frame for plotting with levelplot

Usage

```
map.to.lev(data, xcol = 1, ycol = 2, zcol = c(3, 4), ns = names(data)[zcol])
```

Arguments

data	data frame, e.g. output from krige or predict.gstat
xcol	x-coordinate column number
ycol	y-coordinate column number
zcol	z-coordinate column number range
ns	names of the set of z-columns to be viewed

Value

data frame with the following elements:

x	x-coordinate for each row
y	y-coordinate for each row
z	column vector with each of the elements in columns zcol of data stacked
name	factor; name of each of the stacked z columns

See Also

[image.data.frame](#), [krige](#); for examples see [predict.gstat](#); `levelplot` in package `lattice`.

meuse.all

Meuse river data set – original, full data set

Description

This data set gives locations and top soil heavy metal concentrations (ppm), along with a number of soil and landscape variables, collected in a flood plain of the river Meuse, near the village Stein. Heavy metal concentrations are bulk sampled from an area of approximately 15 m x 15 m.

Usage

```
data(meuse.all)
```

Format

This data frame contains the following columns:

sample sample number

x a numeric vector; x-coordinate (m) in RDM (Dutch topographical map coordinates)

y a numeric vector; y-coordinate (m) in RDM (Dutch topographical map coordinates)

cadmium topsoil cadmium concentration, ppm.; note that zero cadmium values in the original data set have been shifted to 0.2 (half the lowest non-zero value)

copper topsoil copper concentration, ppm.

lead topsoil lead concentration, ppm.

zinc topsoil zinc concentration, ppm.

elev relative elevation

om organic matter, as percentage

ffreq flooding frequency class

soil soil type

lime lime class

landuse landuse class

dist.m distance to river Meuse (metres), as obtained during the field survey

in.pit logical; indicates whether this is a sample taken in a pit

in.meuse155 logical; indicates whether the sample is part of the meuse (i.e., filtered) data set; in addition to the samples in a pit, an sample (139) with outlying zinc content was removed

in.BMcD logical; indicates whether the sample is used as part of the subset of 98 points in the various interpolation examples of Burrough & McDonnell

Note

sample refers to original sample number. Eight samples were left out because they were not indicative for the metal content of the soil. They were taken in an old pit. One sample contains an outlying zinc value, which was also discarded for the meuse (155) data set.

References

P.A. Burrough, R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

<http://www.gstat.org/>

See Also

[meuse](#)

Examples

```
data(meuse.all)
summary(meuse.all)
```

meuse.alt

Meuse river altitude data set

Description

This data set gives a point set with altitudes, digitized from the 1:10,000 topographical map of the Netherlands.

Usage

```
data(meuse.alt)
```

Format

This data frame contains the following columns:

x a numeric vector; x-coordinate (m) in RDM (Dutch topographical map coordinates)

y a numeric vector; y-coordinate (m) in RDM (Dutch topographical map coordinates)

alt altitude in m. above NAP (Dutch zero for sea level)

References

<http://www.gstat.org/>

See Also

[meuse](#)

Examples

```
data(meuse.alt)
library(lattice)
xyplot(y~x, meuse.alt, aspect = "iso")
```

`meuse.grid`*Prediction Grid for Meuse Data Set*

Description

The `meuse.grid` data frame has 3103 rows and 2 columns; a grid with 40 m x 40 m spacing that covers the Meuse Study area

Usage

```
data(meuse.grid)
```

Format

This data frame contains the following columns:

x a numeric vector; x-coordinate (see [meuse](#))

y a numeric vector; y-coordinate (see [meuse](#))

dist distance to the Meuse river; obtained by a spread (spatial distance) GIS operation, from border of river; normalized to [0, 1]

ffreq flood frequency; the lower the value, the larger the flood frequency; the origin of this item is questionable

part.a arbitrary division of the area in two areas, a and b

part.b see `part.a`

soil soil type; it is questionable whether these data come from a real soil map

Details

`x` and `y` are in RDM, the Dutch topographical map coordinate system. Roger Bivand projected this to UTM in the R-Grass interface package.

Source

<http://www.gstat.org/>

References

See the [meuse](#) documentation

Examples

```
data(meuse.grid)
library(lattice)
xyplot(y~x, meuse.grid, asp=mapasp(meuse.grid), pch="+")
```

meuse

Meuse river data set

Description

This data set gives locations and top soil heavy metal concentrations (ppm), along with a number of soil and landscape variables, collected in a flood plain of the river Meuse, near the village Stein. Heavy metal concentrations are bulk sampled from an area of approximately 15 m x 15 m.

Usage

```
data(meuse)
```

Format

This data frame contains the following columns:

x a numeric vector; x-coordinate (m) in RDM (Dutch topographical map coordinates)

y a numeric vector; y-coordinate (m) in RDM (Dutch topographical map coordinates)

cadmium topsoil cadmium concentration, ppm.; note that zero cadmium values in the original data set have been shifted to 0.2 (half the lowest non-zero value)

copper topsoil copper concentration, ppm.

lead topsoil lead concentration, ppm.

zinc topsoil zinc concentration, ppm.

elev relative elevation

dist distance to river Meuse; obtained from the nearest cell in [meuse.grid](#), which in turn was derived by a spread (spatial distance) GIS operation, therefore it is accurate up to 20 metres; normalized [0, 1]

om organic matter, as percentage

ffreq flooding frequency class

soil soil type

lime lime class

landuse landuse class

dist.m distance to river Meuse (metres), as obtained during the field survey

Note

Sample refers to original sample number (9 of the original 164 samples were discarded)

References

P.A. Burrough, R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

<http://www.gstat.org/>

See Also

[meuse.all](#)

Examples

```
data(meuse)
summary(meuse)
```

`ncp.grid`*Grid for the NCP, the Dutch part of the North Sea*

Description**Usage**

```
data(ncp.grid)
```

Format

This data frame contains the following columns:

x x-coordinate, UTM31

y y-coordinate, UTM31

depth sea water depth, m

coast distance to coast, m

area identifier for administrative sub-areas

Note**References****See Also**

[fulmar](#)

Examples

```
data(ncp.grid)
summary(ncp.grid)
```

`ossfim`*Kriging standard errors as function of grid spacing and block size*

Description

Calculate, for a given variogram model, ordinary block kriging standard errors as a function of sampling spaces and block sizes

Usage

```
ossfim(spacings = 1:5, block.sizes = 1:5, model, nmax = 25, debug = 0)
```

Arguments

<code>spacings</code>	range of grid (data) spacings to be used
<code>block.sizes</code>	range of block sizes to be used
<code>model</code>	variogram model, output of <code>vgm</code>
<code>nmax</code>	set the kriging neighbourhood size
<code>debug</code>	debug level; set to 32 to see a lot of output

Value

data frame with columns `spacing` (the grid spacing), `block.size` (the block size), and `kriging.se` (block kriging standard error)

Note

The idea is old, simple, but still of value. If you want to map a variable with a given accuracy, you will have to sample it. Suppose the variogram of the variable is known. Given a regular sampling scheme, the kriging standard error decreases when either (i) the data spacing is smaller, or (ii) predictions are made for larger blocks. This function helps quantifying this relationship. `Ossfim` probably refers to “optimal sampling scheme for isarithmic mapping”.

Author(s)

Edzer J. Pebesma

References

Burrough, P.A., R.A. McDonnell (1999) Principles of Geographical Information Systems. Oxford University Press (e.g., figure 10.11 on page 261)

Burgess, T.M., R. Webster, A.B. McBratney (1981) Optimal interpolation and isarithmic mapping of soil properties. IV Sampling strategy. The journal of soil science 32(4), 643-660.

McBratney, A.B., R. Webster (1981) The design of optimal sampling schemes for local estimation and mapping of regionalized variables: 2 program and examples. Computers and Geosciences 7: 335-365.

read more on a simplified, web-based version on <http://www.gstat.org/ossfim.html>

See Also[krige](#)**Examples**

```
x <- ossfim(1:15,1:15, model = vgm(1,"Exp",15))
library(lattice)
levelplot(kriging.se~spacing+block.size, x,
  main = "Ossfim results, variogram 1 Exp(15)")
# if you wonder about the decrease in the upper left corner of the graph,
# try the above with nmax set to 100, or perhaps 200.
```

oxford

*Oxford soil samples***Description**

Data: 126 soil augerings on a 100 x 100m square grid, with 6 columns and 21 rows. Grid is oriented with long axis North-north-west to South-south-east Origin of grid is South-south-east point, 100m outside grid.

Original data are part of a soil survey carried out by P A. Burrough in 1967. The survey area is located on the chalk downlands on the Berkshire Downs in Oxfordshire, UK. Three soil profile units were recognised on the shallow Rendzina soils; these are Ia - very shallow, grey calcareous soils less than 40cm deep over chalk; Ct - shallow to moderately deep, grey-brown calcareous soils on calcareous colluvium, and Cr: deep, moderately acid, red-brown clayey soils. These soil profile classes were registered at every augering.

In addition, an independent landscape soil map was made by interpolating soil boundaries between these soil types, using information from the changes in landform. Because the soil varies over short distances, this field mapping caused some soil borings to receive a different classification from the classification based on the point data.

Also registered at each auger point were the site elevation (m), the depth to solid chalk rock (in cm) and the depth to lime in cm. Also, the percent clay content, the Munsell colour components of VALUE and CHROMA , and the lime content of the soil (as tested using HCl) were recorded for the top two soil layers (0-20cm and 20-40cm).

Samples of topsoil taken as a bulk sample within a circle of radius 2.5m around each sample point were used for the laboratory determination of Mg (ppm), OM1 %, CEC as mequ/100g air dry soil, pH, P as ppm and K (ppm).

Usage

```
data(oxford)
```

Format

This data frame contains the following columns:

PROFILE profile number

XCOORD x-coordinate, field, non-projected

YCOORD y-coordinate, field, non-projected

ELEV elevation, m.
PROFCLASS soil class, obtained by classifying the soil profile at the sample site
MAPCLASS soil class, obtained by looking up the site location in the soil map
VAL1 Munsell colour component VALUE, 0-20 cm
CHR1 Munsell colour component CHROMA, 20-40 cm
LIME1 Lime content (tested using HCl), 0-20 cm
VAL2 Munsell colour component VALUE, 0-20 cm
CHR2 Munsell colour component CHROMA, 20-40 cm
LIME2 Lime content (tested using HCl), 20-40 cm
DEPTHCM soil depth, cm
DEP2LIME depth to lime, cm
PCLAY1 percentage clay, 0-20 cm
PCLAY2 percentage clay, 20-40 cm
MG1 Magnesium content (ppm), 0-20 cm
OM1 organic matter (%), 0-20 cm
CEC1 CES as mequ/100g air dry soil, 0-20 cm
PH1 pH, 0-20 cm
PHOS1 Phosphorous, 0-20 cm, ppm
POT1 K (potassium), 0-20 cm, ppm

Note

oxford.jpg, in the gstat package data directory, shows an image of the soil map for the region

References

P.A. Burrough, R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

Examples

```
data(oxford)
summary(oxford)
```

pcb

PCB138 measurements in sediment at the NCP, the Dutch part of the North Sea

Description

This data set gives a point set with altitudes, digitized from the 1:10,000 topographical map of the Netherlands.

Usage

```
data(pcb)
```

Format

This data frame contains the following columns:

year measurement year

x x-coordinate; UTM31

y y-coordinate; UTM31

coast distance to coast, m.

depth sea water depth, m.

PCB138 PCB-138, measured on the sediment fraction smaller than 63 μm , in $\mu\text{g}/\text{kg}$ dry matter;
BUT SEE NOTE BELOW

yf year; as factor

Note

A note of caution: The PCB-138 data are provided only to be able to re-run the analysis done in Pebesma and Duin (2004; see references below). If you want to use these data for comparison with PCB measurements elsewhere, or if you want to compare them to regulation standards, or want to use these data for any other purpose, you should first contact <mailto:basisinfodesk@rikz.rws.minvenw.nl>. The reason for this is that several normalisations were carried out that are not reported here, nor in the paper below.

References

<http://www.gstat.org/>, <http://www.rikz.nl/>

Edzer J. Pebesma, Richard N.M. Duin, 2004. Spatio-temporal mapping of sea floor sediment pollution in the North Sea. Paper presented at GeoENV2004, Oct 12-14, 2004, Neuchatel; proceedings to be published by Springer. A copy of the paper can be requested from <mailto:e.pebesma@geog.uu.nl>

See Also

[ncp.grid](#)

Examples

```
data(pcb)
library(lattice)
xyplot(y~x|as.factor(yf), pcb, aspect = "iso")
# demo(pcb)
```

```
plot.gstatVariogram
```

Plot a Sample Variogram

Description

Creates a variogram plot

Usage

```
plot.gstatVariogram(x, model = NULL, ylim, xlim, xlab = "distance",
  ylab = "semivariance", multipanel = TRUE, plot.numbers = FALSE,
  scales, ids = x$id, group.id = TRUE, skip, layout, ...)
plot.variogramMap(x, np = FALSE, skip, threshold, ...)
```

Arguments

<code>x</code>	object of class "gstatVariogram", obtained from the function <code>variogram</code> , possibly containing directional or cross variograms
<code>model</code>	in case of a single variogram: a variogram model, as obtained from <code>vgm</code> or <code>fit.variogram</code> , to be drawn as a line in the variogram plot; in case of a set of variograms and cross variograms: a list with variogram models
<code>ylim</code>	numeric vector of length 2, limits of the y-axis
<code>xlim</code>	numeric vector of length 2, limits of the x-axis
<code>xlab</code>	x-axis label
<code>ylab</code>	y-axis label
<code>multipanel</code>	logical; if TRUE, directional variograms are plotted in different panels, if FALSE, directional variograms are plotted in the same graph, using color, colored lines and symbols to distinguish them
<code>plot.numbers</code>	logical; if TRUE, plot number of point pairs next to each plotted semivariance symbol
<code>scales</code>	optional argument that will be passed to <code>xyplot</code> in case of the plotting of variograms and cross variograms; use the value <code>list(relation = "same")</code> if y-axes need to share scales
<code>ids</code>	ids of the data variables and variable pairs
<code>group.id</code>	logical; control for directional multivariate variograms: if TRUE, panels divide direction and colors indicate variables (ids), if FALSE panels divide variables/variable pairs and colors indicate direction
<code>skip</code>	logical; can be used to arrange panels, see <code>xyplot</code>
<code>layout</code>	integer vector; can be used to set panel layout: <code>c(ncol, nrow)</code>
<code>np</code>	logical; if TRUE, plot number of point pairs, if FALSE plot semivariances
<code>threshold</code>	semivariogram map values based on fewer point pairs than threshold will not be plotted
<code>...</code>	any arguments that will be passed to the panel plotting functions (such as <code>auto.key</code> in examples below)

Value

returns (or plots) the variogram plot

Note

currently, plotting models and/or point pair numbers is not supported when a variogram is both directional and multivariable; also, three-dimensional directional variograms will probably not be displayed correctly.

Author(s)

Edzer J. Pebesma

References<http://www.gstat.org>**See Also**[variogram](#), [fit.variogram](#), [vgm variogram.line](#),**Examples**

```

data(meuse)
vgm1 <- variogram(log(zinc)~1, ~x+y, meuse)
plot(vgm1)
model.1 <- fit.variogram(vgm1,vgm(1,"Sph",300,1))
plot(vgm1, model=model.1)
plot(vgm1, plot.numbers = TRUE, pch = "+")
vgm2 <- variogram(log(zinc)~1, ~x+y, meuse, alpha=c(0,45,90,135))
plot(vgm2)
# the following demonstrates plotting of directional models:
model.2 <- vgm(.59,"Sph",926,.06,anis=c(0,0.3))
plot(vgm2, model=model.2)

g = gstat(id="zinc < 200", form=I(zinc<200)~1,loc=~x+y,data=meuse)
g = gstat(g, id="zinc < 400", form=I(zinc<400)~1,loc=~x+y,data=meuse)
g = gstat(g, id="zinc < 800", form=I(zinc<800)~1,loc=~x+y,data=meuse)
# calculate multivariable, directional variogram:
v = variogram(g, alpha=c(0,45,90,135))
plot(v, group.id = FALSE, auto.key = TRUE) # id and id pairs panels
plot(v, group.id = TRUE, auto.key = TRUE) # direction panels

if (require(sp)) {
  plot(variogram(g, cutoff=1000, width=100, map=TRUE),
       main = "(cross) semivariance maps")
  plot(variogram(g, cutoff=1000, width=100, map=TRUE), np=TRUE,
       main = "number of point pairs")
}

```

plot.pointPairs *Plot a point pairs, identified from a variogram cloud*

Description

Plot a point pairs, identified from a variogram cloud

Usage

```

plot.pointPairs(x, data, xcol = data$x, ycol = data$y, xlab = "x coordinate",
               ylab = "y coordinate", col.line = 2, line.pch = 0, ...)

```

Arguments

x	object of class "pointPairs", obtained from the function plot.variogramCloud , containing point pair indices
data	data frame to which the indices refer (from which the variogram cloud was calculated)
xcol	numeric vector with x-coordinates of data
ycol	numeric vector with y-coordinates of data
xlab	x-axis label
ylab	y-axis label
col.line	color for lines connecting points
line.pch	if non-zero, symbols are also plotted at the middle of line segments, to mark lines too short to be visible on the plot; the color used is <code>col.line</code> ; the value passed to this argument will be used as plotting symbol (pch)
...	arguments, further passed to <code>xypplot</code>

Value

plots the data locations, with lines connecting the point pairs identified (and referred to by indices in) x

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org>

See Also

[plot.variogramCloud](#)

Examples

```
### The following requires interaction, and is therefore outcommented
#data(meuse)
#vgml <- variogram(log(zinc)~1, ~x+y, meuse, cloud = TRUE)
#pp <- plot(vgml, id = TRUE)
### Identify the point pairs
#plot(pp, data = meuse) # meuse has x and y as coordinates
```

```
plot.variogramCloud
```

Plot and Identify Data Pairs on Sample Variogram Cloud

Description

Plot a sample variogram cloud, possibly with identification of individual point pairs

Usage

```
plot.variogramCloud(x, identify = FALSE, digitize = FALSE, xlim, ylim, xlab, ylab,
                    keep = FALSE, ...)
```

Arguments

<code>x</code>	object of class <code>variogramCloud</code>
<code>identify</code>	logical; if TRUE, the plot allows identification of a series of individual point pairs that correspond to individual variogram cloud points (use left mouse button to select; right mouse button ends)
<code>digitize</code>	logical; if TRUE, select point pairs by digitizing a region with the mouse (left mouse button adds a point, right mouse button ends)
<code>xlim</code>	limits of x-axis
<code>ylim</code>	limits of y-axis
<code>xlab</code>	x axis label
<code>ylab</code>	y axis label
<code>keep</code>	logical; if TRUE and <code>identify</code> is TRUE, the labels identified and their position are kept and glued to object <code>x</code> , which is returned. Subsequent calls to <code>plot</code> this object will now have the labels shown, e.g. to plot to hardcopy
<code>...</code>	parameters that are passed through to plot.gstatVariogram (in case of <code>identify = FALSE</code>) or to <code>plot</code> (in case of <code>identify = TRUE</code>)

Value

If `identify` or `digitize` is TRUE, a data frame of class `pointPairs` with in its rows the point pairs identified (pairs of row numbers in the original data set); if `identify` is F, a plot of the variogram cloud, which uses [plot.gstatVariogram](#)

If in addition to `identify`, `keep` is also TRUE, an object of class `variogramCloud` is returned, having attached to it attributes "sel" and "text", which will be used in subsequent calls to `plot.variogramCloud` with `identify` set to FALSE, to plot the text previously identified.

If in addition to `digitize`, `keep` is also TRUE, an object of class `variogramCloud` is returned, having attached to it attribute "poly", which will be used in subsequent calls to `plot.variogramCloud` with `digitize` set to FALSE, to plot the digitized line.

In both of the `keep = TRUE` cases, the attribute `ppairs` of class `pointPairs` is present, containing the point pairs identified.

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org/>

See Also

[variogram](#), [plot.gstatVariogram](#), [plot.pointPairs](#), [identify](#), [locator](#)

Examples

```
data(meuse)
plot(variogram(log(zinc)~1, loc=~x+y, data=meuse, cloud=TRUE))
## commands that require interaction:
# x <- variogram(log(zinc)~1, loc=~x+y, data=meuse, cloud=TRUE)
# plot(plot(x, identify = TRUE), meuse)
# plot(plot(x, digitize = TRUE), meuse)
```

`point.in.polygon` *do point(s) fall in a given polygon?*

Description

verifies for one or more points whether they fall in a given polygon

Usage

```
point.in.polygon(point.x, point.y, pol.x, pol.y)
```

Arguments

<code>point.x</code>	numerical array of x-coordinates of points
<code>point.y</code>	numerical array of y-coordinates of points
<code>pol.x</code>	numerical array of x-coordinates of polygon
<code>pol.y</code>	numerical array of y-coordinates of polygon

Value

logical array; FALSE if a point is strictly exterior to the polygon, TRUE if not (point is strictly interior to polygon, point is a vertex of polygon, or point lies on the relative interior of an edge of polygon)

References

Uses the C function `InPoly()`, in `gstat` file `polygon.c`; `InPoly` is Copyright 1998 by Joseph O'Rourke. It may be freely redistributed in its entirety provided that this copyright notice is not removed.

Examples

```
# open polygon:
point.in.polygon(1:10,1:10,c(3,4,4,3),c(3,3,4,4))
# closed polygon:
point.in.polygon(1:10,1:10,c(3,4,4,3,3),c(3,3,4,4,3))
```

 predict.gstat

Multivariable Geostatistical Prediction and Simulation

Description

The function provides the following prediction methods: simple, ordinary, and universal kriging, simple, ordinary, and universal cokriging, point- or block-kriging, and conditional simulation equivalents for each of the kriging methods.

Usage

```
predict.gstat(object, newdata, block = numeric(0), nsim = 0,
              indicators = FALSE, BLUE = FALSE, debug.level = 1, mask,
              na.action = na.pass, ...)
```

Arguments

object	object of class <code>gstat</code> , see gstat and krige
newdata	data frame with prediction/simulation locations; should contain columns with the independent variables (if present) and the coordinates with names as defined in <code>locations</code>
block	block size; a vector with 1, 2 or 3 values containing the size of a rectangular in x-, y- and z-dimension respectively (0 if not set), or a data frame with 1, 2 or 3 columns, containing the points that discretize the block in the x-, y- and z-dimension to define irregular blocks relative to (0,0) or (0,0,0)—see also the details section below. By default, predictions or simulations refer to the support of the data values.
nsim	integer; if set to a non-zero value, conditional simulation is used instead of kriging interpolation. For this, sequential Gaussian or indicator simulation is used (depending on the value of <code>indicators</code>), following a single random path through the data.
indicators	logical; only relevant if <code>nsim</code> is non-zero; if TRUE, use indicator simulation, else use Gaussian simulation
BLUE	logical; if TRUE return the BLUE trend estimates only, if FALSE return the BLUP predictions (kriging)
debug.level	integer; set <code>gstat</code> internal debug level, see below for useful values. If set to -1 (or any negative value), a progress counter is printed
mask	not supported anymore – use <code>na.action</code> ; logical or numerical vector; pattern with valid values in <code>newdata</code> (marked as TRUE, non-zero, or non-NA); if mask is specified, the returned data frame will have the same number and order of rows in <code>newdata</code> , and masked rows will be filled with NA's.
na.action	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'. Missing values in coordinates and predictors are both dealt with.
...	ignored (but necessary for the S3 generic/method consistency)

Details

When a non-stationary (i.e., non-constant) mean is used, both for simulation and prediction purposes the variogram model defined should be that of the residual process, and not that of the raw observations.

For irregular block kriging, coordinates should discretize the area relative to 0, (0,0) or (0,0,0); the coordinates in newdata should give the centroids around which the block should be located. So, suppose the block is discretized by points (3,3) (3,5) (5,5) and (5,3), we should pass point (4,4) in newdata and pass points (-1,-1) (-1,1) (1,1) (1,-1) to the block argument. Although passing the uncentered block and (0,0) as newdata may work for global neighbourhoods, neighbourhood selection is always done relative to the centroid values in newdata.

The algorithm used by gstat for simulation random fields is the sequential simulation algorithm. This algorithm scales well to large or very large fields (e.g., more than 10^6 nodes). Its power lies in using only data and simulated values in a local neighbourhood to approximate the conditional distribution at that location, see `nmax` in [krige](#) and [gstat](#). The larger `nmax`, the better the approximation, the smaller `nmax`, the faster the simulation process. For selecting the nearest `nmax` data or previously simulated points, gstat uses a bucket PR quadtree neighbourhood search algorithm; see the reference below.

For sequential Gaussian or indicator simulations, a random path through the simulation locations is taken, which is usually done for sequential simulations. The reason for this is that the local approximation of the conditional distribution, using only the `nmax` nearest observed (or simulated) values may cause spurious correlations when a regular path would be followed. Following a single path through the locations, gstat reuses the expensive results (neighbourhood selection and solution to the kriging equations) for each of the subsequent simulations when multiple realisations are requested. You may expect a considerable speed gain in simulating 1000 fields in a single call to [predict.gstat](#), compared to 1000 calls, each for simulating a single field.

The random number generator used for generating simulations is the native random number generator of the environment (R, S); fixing randomness by setting the random number seed with `set.seed()` works.

When mean coefficients are not supplied, they are generated as well from their conditional distribution (assuming multivariate normal, using the generalized least squares BLUE estimate and its estimation covariance); for a reference to the algorithm used see Abrahamsen and Benth, *Math. Geol.* 33(6), page 742 and leave out all constraints.

Memory requirements for sequential simulation: let `n` be the product of the number of variables, the number of simulation locations, and the number of simulations required in a single call. the gstat C function `gstat_predict` requires a table of size `n * 12` bytes to pass the simulations back to R, before it can free `n * 4` bytes. Hopefully, R does not have to duplicate the remaining `n * 8` bytes when the coordinates are added as columns, and when the resulting matrix is coerced to a `data.frame`.

Useful values for `debug.level`: 0: suppress any output except warning and error messages; 1: normal output (default): short data report, program action and mode, program progress in %, total execution time; 2: print the value of all global variables, all files read and written, and include source file name and line number in error messages; 4: print OLS and WLS fit diagnostics; 8: print all data after reading them; 16: print the neighbourhood selection for each prediction location; 32: print (generalised) covariance matrices, design matrices, solutions, kriging weights, etc.; 64: print variogram fit diagnostics (number of iterations and variogram model in each iteration step) and order relation violations (indicator kriging values before and after order relation correction); 512: print block (or area) discretization data for each prediction location. To combine settings, sum their respective values. Negative values for `debug.level` are equal to positive, but cause the progress counter to work.

Value

a data frame containing the coordinates of `newdata`, and columns of prediction and prediction variance (in case of kriging) or the columns of the conditional Gaussian or indicator simulations

Note**Author(s)**

Edzer J. Pebesma

References

N.A.C. Cressie, 1993, *Statistics for Spatial Data*, Wiley.

<http://www.gstat.org/>

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30: 683-691.

For bucket PR quadtrees, excellent demos are found at <http://www.cs.umd.edu/~brabec/quadtrees/index.html>

See Also

[gstat](#), [krige](#)

Examples

```
# generate 5 conditional simulations
data(meuse)
data(meuse.grid)
v <- variogram(log(zinc)~1,~x+y, meuse)
m <- fit.variogram(v, vgm(1, "Sph", 300, 1))
plot(v, model = m)
set.seed(131)
sim <- krige(formula = log(zinc)~1, locations = ~x+y, model = m, data
             = meuse, newdata = meuse.grid, nmax = 15, beta = 5.9, nsim = 5)
# show all 5 simulation, using map.to.lev to rearrange sim:
if (require(sp) == FALSE) {
  library(lattice)
  levelplot(z~x+y|name, map.to.lev(sim, z=c(3:7)), aspect = mapasp(sim))

# calculate generalised least squares residuals w.r.t. constant trend:
g <- gstat(id = "log.zinc", formula = log(zinc)~1, locations = ~x+y,
           model = m, data = meuse)
blue0 <- predict(g, newdata = meuse, BLUE = TRUE)
blue0$blue.res <- log(meuse$zinc) - blue0$log.zinc.pred
bubble(blue0, zcol = "blue.res", main = "GLS residuals w.r.t. constant")

# calculate generalised least squares residuals w.r.t. linear trend:
m <- fit.variogram(variogram(log(zinc)~sqrt(dist.m),~x+y, meuse),
                  vgm(1, "Sph", 300, 1))
g <- gstat(id = "log.zinc", formula = log(zinc)~sqrt(dist.m),
           locations = ~x+y, model = m, data = meuse)
blue1 <- predict(g, newdata = meuse, BLUE = TRUE)
```

```

blue1$blue.res <- log(meuse$zinc) - blue1$log.zinc.pred
bubble(blue1, zcol = "blue.res",
       main = "GLS residuals w.r.t. linear trend")

# unconditional simulation on a 100 x 100 grid
xy <- expand.grid(1:100, 1:100)
names(xy) <- c("x", "y")
g.dummy <- gstat(formula = z~1, locations = ~x+y, dummy = TRUE, beta = 0,
                model = vgm(1, "Exp", 15), nmax = 20)
yy <- predict(g.dummy, newdata = xy, nsim = 4)
# show one realisation:
levelplot(sim1~x+y, yy, aspect = mapasp(yy))
# show all four:
levelplot(z~x+y|name, map.to.lev(yy, z=c(3:6)), aspect = mapasp(yy))
}

```

select.spatial *select points spatially*

Description

select a number of points by digitizing the area they fall in

Usage

```
select.spatial(x = data$x, y = data$y, data, pch = "+", n = 512)
```

Arguments

x	numerical array of x-coordinates of points
y	numerical array of y-coordinates of points
data	optional; data frame containing variables x and y
pch	plotting character to be used for points
n	default number of points to locate, can be overridden manually

Value

array with indexes (row numbers) of points inside the polygon digitized

See Also

[point.in.polygon](#), [locator](#)

Examples

```

data(meuse)
## the following command requires user interaction: left mouse
## selects points, right mouse ends digitizing
# select.spatial(data=meuse)

```

 show.vgms

Plot Variogram Model Functions

Description

Creates a trellis plot for a range of variogram models, possibly with nugget; and optionally a set of Matern models with varying smoothness.

Usage

```
show.vgms(min = 1e-12 * max, max = 3, n = 50, sill = 1, range = 1,
          models = as.character(vgm()[c(1:16)]), nugget = 0, kappa.range = 0.5,
          plot = TRUE)
```

Arguments

min	numeric; start distance value for semivariance calculation beyond the first point at exactly zero
max	numeric; maximum distance for semivariance calculation and plotting
n	integer; number of points to calculate distance values
sill	numeric; (partial) sill of the variogram model
range	numeric; range of the variogram model
models	character; variogram models to be plotted
nugget	numeric; nugget component for variogram models
kappa.range	numeric; if this is a vector with more than one element, only a range of Matern models is plotted with these kappa values
plot	logical; if TRUE, a plot is returned with the models specified; if FALSE, the data prepared for this plot is returned

Value

returns a (Trellis) plot of the variogram models requested; see examples. I do currently have strong doubts about the “correctness” of the “Hol” model. The “Spl” model does seem to need a very large range value (larger than the study area?) to be of some value.

If plot is FALSE, a data frame with the data prepared to plot is being returned.

Note

the min argument is supplied because the variogram function may be discontinuous at distance zero, surely when a positive nugget is present.

Author(s)

Edzer J. Pebesma

References

<http://www.gstat.org>

See Also

[vgm](#), [variogram.line](#),

Examples

```
show.vgms()
show.vgms(models = c("Exp", "Mat", "Gau"), nugget = 0.1)
# show a set of Matern models with different smoothness:
show.vgms(kappa.range = c(.1, .2, .5, 1, 2, 5, 10), max = 10)
```

 sic2004

Spatial Interpolation Comparison 2004 data set: Natural Ambient Radioactivity

Description

The text below is copied from <http://www.ai-geostats.org/events/sic2004/index.htm>, subsection Data.

The variable used in the SIC 2004 exercise is natural ambient radioactivity measured in Germany. The data, provided kindly by the German Federal Office for Radiation Protection (BfS), are gamma dose rates reported by means of the national automatic monitoring network (IMIS).

In the frame of SIC2004, a rectangular area was used to select 1008 monitoring stations (from a total of around 2000 stations). For these 1008 stations, 11 days of measurements have been randomly selected during the last 12 months and the average daily dose rates calculated for each day. Hence, we ended up having 11 data sets.

Prior information (sic.train): 10 data sets of 200 points that are identical for what concerns the locations of the monitoring stations have been prepared. These locations have been randomly selected (see Figure 1). These data sets differ only by their Z values since each set corresponds to 1 day of measurement made during the last 14 months. No information will be provided on the date of measurement. These 10 data sets (10 days of measurements) can be used as prior information to tune the parameters of the mapping algorithms. No other information will be provided about these sets. Participants are free of course to gather more information about the variable in the literature and so on.

The 200 monitoring stations above were randomly taken from a larger set of 1008 stations. The remaining 808 monitoring stations have a topology given in sic.pred. Participants to SIC2004 will have to estimate the values of the variable taken at these 808 locations.

The SIC2004 data (sic.val, variable dayx): The exercise consists in using 200 measurements made on a 11th day (THE data of the exercise) to estimate the values observed at the remaining 808 locations (hence the question marks as symbols in the maps shown in Figure 3). These measurements will be provided only during two weeks (15th of September until 1st of October 2004) on a web page restricted to the participants. The true values observed at these 808 locations will be released only at the end of the exercise to allow participants to write their manuscripts (sic.test, variables dayx and joker).

In addition, a joker data set was released (sic.val, variable joker), which contains an anomaly. The anomaly was generated by a simulation model, and does not represent measured levels.

Usage

```
data(sic2004) #
```

Format

The data frames contain the following columns:

record this integer value is the number (unique value) of the monitoring station chosen by us.

x X-coordinate of the monitoring station indicated in meters

y Y-coordinate of the monitoring station indicated in meters

day01 mean gamma dose rate measured during 24 hours, at day01. Units are nanoSieverts/hour

day02 same, for day 02

day03 ...

day04 ...

day05 ...

day06 ...

day07 ...

day08 ...

day09 ...

day10 ...

dayx the data observed at the 11-th day

joker the joker data set, containing an anomaly not present in the training data

Note

the data set `sic.grid` provides a set of points on a regular grid (almost 10000 points) covering the area; this is convenient for interpolation; see the function `makegrid` in package `gstat` (may be moved to `sp`).

The coordinates have been projected around a point located in the South West of Germany. Hence, a few coordinates have negative values as can be guessed from the Figures below.

References

<http://www.ai-geostats.org/>

<http://www.ai-geostats.org/events/sic2004/index.htm>

Examples

```
data(sic2004)
# FIGURE 1. Locations of the 200 monitoring stations for the 11 data sets.
# The values taken by the variable are known.
plot(y~x,sic.train,pch=1,col="red", asp=1)

# FIGURE 2. Locations of the 808 remaining monitoring stations at which
# the values of the variable must be estimated.
plot(y~x,sic.pred,pch="?", asp=1, cex=.8) # Figure 2

# FIGURE 3. Locations of the 1008 monitoring stations (exhaustive data sets).
# Red circles are used to estimate values located at the questions marks
plot(y~x,sic.train,pch=1,col="red", asp=1)
points(y~x, sic.pred, pch="?", cex=.8)
```

variogram.line *Semivariance Values For a Given Variogram Model*

Description

Generates a semivariance values given a variogram model

Usage

```
variogram.line(object, maxdist, n = 200, min = 1.0e-6 * maxdist,  
               dir = c(1,0,0), ...)
```

Arguments

object	variogram model for which we want semivariance function values
maxdist	maximum distance for which we want semivariance values
n	number of points
min	minimum distance; a value slightly larger than zero is usually used to avoid the discontinuity at distance zero if a nugget component is present
dir	direction vector: unit length vector pointing the direction in x (East-West), y (North-South) and z (Up-Down)
...	ignored

Value

a data frame of dimension (n x 2), with columns distance and gamma

Note

this function is used to plot a variogram model

Author(s)

Edzer J. Pebesma

See Also

[plot.gstatVariogram](#)

Examples

```
variogram.line(vgm(5, "Exp", 10, 5), 10, 10)  
# anisotropic variogram, plotted in E-W direction:  
variogram.line(vgm(1, "Sph", 10, anis=c(0,0.5)), 10, 10)  
# anisotropic variogram, plotted in N-S direction:  
variogram.line(vgm(1, "Sph", 10, anis=c(0,0.5)), 10, 10, dir=c(0,1,0))
```

 variogram

Calculate Sample or Residual Variogram or Variogram Cloud

Description

Calculates the sample variogram from data, or in case of a linear model is given, for the residuals, with options for directional, robust, and pooled variogram, and for irregular distance intervals.

Usage

```

variogram.formula(object, ...)
variogram.gstat(formula, locations, data, ...)
variogram.default(y, locations, X, cutoff, width = cutoff/15, alpha =
  0, beta = 0, tol.hor = 90/length(alpha), tol.ver =
  90/length(beta), cressie = FALSE, dX = numeric(0), boundaries =
  numeric(0), cloud = FALSE, trend.beta = NULL, debug.level = 1,
  cross = TRUE, grid, map = FALSE, g = NULL, ...)

print.gstatVariogram(v, ...)
print.variogramCloud(v, ...)

```

Arguments

object	object of class <code>gstat</code> ; in this form, direct and cross (residual) variograms are calculated for all variables and variable pairs defined in <code>object</code>
formula	formula defining the response vector and (possible) regressors, in case of absence of regressors, use e.g. <code>z~1</code>
data	data frame where the names in formula are to be found
locations	spatial data locations. For <code>variogram.formula</code> : a formula with only the coordinate variables in the right hand (explanatory variable) side e.g. <code>~x+y</code> ; see examples. For <code>variogram.default</code> : list with coordinate matrices, each with the number of rows matching that of corresponding vectors in <code>y</code> ; the number of columns should match the number of spatial dimensions spanned by the data (1 (x), 2 (x,y) or 3 (x,y,z)).
...	any other arguments that will be passed to <code>variogram.default</code>
y	list with for each variable the vector with responses
X	(optional) list with for each variable the matrix with regressors/covariates; the number of rows should match that of the corresponding element in <code>y</code> , the number of columns equals the number of regressors (including intercept)
cutoff	spatial separation distance up to which point pairs are included in semivariance estimates
width	the width of subsequent distance intervals into which data point pairs are grouped for semivariance estimates
alpha	direction in plane (x,y), in positive degrees clockwise from positive y (North): <code>alpha=0</code> for direction North (increasing y), <code>alpha=90</code> for direction East (increasing x); optional a vector of directions in (x,y)
beta	direction in z, in positive degrees up from the (x,y) plane;

<code>tol.hor</code>	horizontal tolerance angle in degrees
<code>tol.ver</code>	vertical tolerance angle in degrees
<code>crossie</code>	logical; if TRUE, use Cressie's robust variogram estimate; if FALSE use the classical method of moments variogram estimate
<code>dX</code>	include a pair of data points $y(s_1), y(s_2)$ taken at locations s_1 and s_2 for sample variogram calculation only when $\ x(s_1) - x(s_2)\ < dX$ with $x(s_i)$ the vector with regressors at location s_i , and $\ \cdot\ $ the 2-norm. This allows pooled estimation of within-strata variograms (use a factor variable as regressor, and <code>dX=0.5</code>), or variograms of (near-)replicates in a linear model (addressing point pairs having similar values for regressors variables)
<code>boundaries</code>	numerical vector with distance interval boundaries; values should be strictly increasing
<code>cloud</code>	logical; if TRUE, calculate the semivariogram cloud
<code>trend.beta</code>	vector with trend coefficients, in case they are known. By default, trend coefficients are estimated from the data.
<code>debug.level</code>	integer; set <code>gstat</code> internal debug level
<code>cross</code>	logical; if FALSE, no cross variograms are calculated when object is of class <code>gstat</code> and has more than one variable
<code>v</code>	object of class <code>variogram</code> or <code>variogramCloud</code> to be printed
<code>grid</code>	grid parameters, if data are gridded
<code>map</code>	logical; if TRUE, and <code>cutoff</code> and <code>width</code> are given, a variogram map is returned. This requires package <code>sp</code> . Alternatively, a map can be passed, of class <code>SpatialDataFrameGrid</code> (see <code>sp</code> docs)
<code>g</code>	NULL or object of class <code>gstat</code> ; may be used to pass settable parameters and/or variograms; see example

Value

If `map` is TRUE (or a map is passed), a grid map is returned containing the (cross) variogram map(s). See package `sp`.

In other cases, an object of class "gstatVariogram" with the following fields:

<code>np</code>	the number of point pairs for this estimate; in case of a <code>variogramCloud</code> see below
<code>dist</code>	the average distance of all point pairs considered for this estimate
<code>gamma</code>	the actual sample variogram estimate
<code>dir.hor</code>	the horizontal direction
<code>dir.ver</code>	the vertical direction
<code>id</code>	the combined id pair
<code>left</code>	for <code>variogramCloud</code> : data id (row number) of one of the data pair
<code>right</code>	for <code>variogramCloud</code> : data id (row number) of the other data in the pair

In the past, `gstat` returned an object of class "variogram"; however, this resulted in confusions for users of the package `geoR`: the `geoR` `variog` function also returns objects of class "variogram", incompatible to those returned by this function. That's why I changed the class name.

Note**Author(s)**

Edzer J. Pebesma

References

Cressie, N.A.C., 1993, Statistics for Spatial Data, Wiley.

<http://www.gstat.org/>

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences, 30: 683-691.

See Also

[print.gstatVariogram](#), [plot.gstatVariogram](#), [plot.variogramCloud](#); for variogram models: [vgm](#), to fit a variogram model to a sample variogram: [fit.variogram](#)

Examples

```
data(meuse)
# no trend:
variogram(log(zinc)~1, loc=~x+y, meuse)
# residual variogram w.r.t. a linear trend:
variogram(log(zinc)~x+y, loc=~x+y, meuse)
# directional variogram:
variogram(log(zinc)~x+y, loc=~x+y, meuse, alpha=c(0,45,90,135))

# GLS residual variogram:
v = variogram(log(zinc)~x+y,~x+y,meuse)
v.fit = fit.variogram(v, vgm(1, "Sph", 700, 1))
v.fit
set = list(gls=1)
v
g = gstat(NULL, "log-zinc", log(zinc)~x+y,~x+y, meuse, model=v.fit, set = set)
variogram(g)
```

vgm

Generate, or Add to Variogram Model

Description

Generates a variogram model, or adds to an existing model. `print.variogramModel` prints the essence of a variogram model.

Usage

```
vgm(psil1, model, range, nugget, add.to, anis, kappa = 0.5)
print.variogramModel(x, ...)
```

Arguments

<code>psill</code>	(partial) sill of the variogram model component
<code>model</code>	model type, e.g. "Exp", "Sph", "Gau", "Mat". Calling <code>vgm()</code> without a model argument returns the list with available models.
<code>range</code>	range of the variogram model component
<code>kappa</code>	smoothness parameter for the Matern class of variogram models
<code>nugget</code>	nugget component of the variogram (this basically adds a nugget component to the model)
<code>add.to</code>	a variogram model to which we want to add a component
<code>anis</code>	anisotropy parameters: see notes below
<code>x</code>	a variogram model to print
<code>...</code>	arguments that will be passed to <code>print</code> , e.g. <code>digits</code> (see examples)

Value

an object of class `variogramModel`, which extends `data.frame`. Called without a model argument returns a character list with available models.

Note

Geometric anisotropy can be modelled for each individual simple model by giving two or five anisotropy parameters, two for two-dimensional and five for three-dimensional data. In any case, the range defined is the range in the direction of the strongest correlation, or the major range. Anisotropy parameters define which direction this is (the main axis), and how much shorter the range is in (the) direction(s) perpendicular to this main axis.

In two dimensions, two parameters define an anisotropy ellipse, say `anis = c(45, 0.5)`. The first parameter, 30, refers to the main axis direction: it is the angle for the principal direction of continuity (measured in degrees, clockwise from positive Y, North). The second parameter, 0.5, is the anisotropy ratio, the ratio of the minor range to the major range (a value between 0 and 1). So, in our example, if the range in the major direction (North-East) is 100, the range in the minor direction (South-East) is 50.

In three dimensions, five values should be given in the form `anis = c(p,q,r,s,t)`. Now, p is the angle for the principal direction of continuity (measured in degrees, clockwise from Y, in direction of X), q is the dip angle for the principal direction of continuity (measured in positive degrees up from horizontal), r is the third rotation angle to rotate the two minor directions around the principal direction defined by p and q . A positive angle acts counter-clockwise while looking in the principal direction. Anisotropy ratios s and t are the ratios between the major range and each of the two minor ranges.

(Note that `anis = c(p,s)` is equivalent to `anis = c(p,0,0,s,1)`.)

The implementation in `gstat` for 2D and 3D anisotropy was taken from the `gslib` (probably 1992) code. I have seen a paper where it is argued that the 3D anisotropy code implemented in `gslib` (en then also in `gstat`) is in error, but I have not corrected anything afterwards.

Author(s)

Edzer J. Pebesma

References

Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30: 683-691.

See Also

[show.vgms](#) to view the available models, [fit.variogram](#), [variogram.line](#), [variogram](#) for the sample variogram.

Examples

```
vgm(10, "Exp", 300)
x <- vgm(10, "Exp", 300)
vgm(10, "Nug", 0)
vgm(10, "Exp", 300, 4.5)
vgm(10, "Mat", 300, 4.5, kappa = 0.7)
vgm( 5, "Exp", 300, add.to = vgm(5, "Exp", 60, nugget = 2.5))
vgm(10, "Exp", 300, anis = c(30, 0.5))
vgm(10, "Exp", 300, anis = c(30, 10, 0, 0.5, 0.3))
# Matern variogram model:
vgm(1, "Mat", 1, kappa=.3)
x <- vgm(0.39527463, "Sph", 953.8942, nugget = 0.06105141)
x
print(x, digits = 3);
# to see all components, do
print.data.frame(x)
```

zerodist

find point pairs with equal spatial coordinates

Description

find point pairs with equal spatial coordinates

Usage

```
zerodist(x, y, z, zero = 0.0)
```

Arguments

x	vector with x-coordinate
y	vector with y-coordinate (may be missing)
z	vector with z-coordinate (may be missing)
zero	value to be compared to for establishing when a distance is considered zero (default 0.0)

Value

pairs of row numbers with identical coordinates, numeric(0) if no such pairs are found; if zero is set to a positive value, the distance between point pairs is returned in the third column.

Note

Duplicate observations sharing identical spatial locations result in singular covariance matrices in kriging situations. This function may help identifying spatial duplications, so they can be removed. A matrix with all pair-wise distances is calculated, so if x, y and z are large this function is slow

Examples

```
if (require(sp) == FALSE) {
  data(meuse)
  # pick 10 rows
  n <- 10
  ran10 <- sample(nrow(meuse), size = n, replace = TRUE)
  meusedup <- rbind(meuse, meuse[ran10, ])
  zerodist(meusedup$x, meusedup$y)
  zd <- zerodist(meusedup$x, meusedup$y)
  sum(abs(zd[1:n,1] - sort(ran10))) # 0!
  # remove the duplicate rows:
  meusedup2 <- meusedup[-zd[,2], ]
  # find point pairs within 500 m distance of each other:
  zerodist(meuse$x, meuse$y, 500)
}
```

Index

- *Topic **color**
 - bpy.colors, 1
- *Topic **datasets**
 - fulmar, 7
 - meuse, 24
 - meuse.all, 21
 - meuse.alt, 22
 - meuse.grid, 23
 - ncp.grid, 25
 - oxford, 27
 - pcb, 28
 - sic2004, 40
- *Topic **dplot**
 - bubble, 2
 - image, 13
 - makegrid, 19
 - map.to.lev, 20
 - mapasp, 20
 - plot.gstatVariogram, 29
 - plot.pointPairs, 31
 - plot.variogramCloud, 33
 - show.vgms, 39
 - zerodist, 47
- *Topic **internal**
 - gstat-internal, 9
- *Topic **models**
 - fit.lmc, 4
 - fit.variogram, 5
 - fit.variogram.reml, 6
 - get.contr, 8
 - gstat, 10
 - krige, 16
 - krige.cv, 14
 - ossfim, 26
 - point.in.polygon, 34
 - predict.gstat, 35
 - select.spatial, 38
 - variogram, 43
 - variogram.line, 42
 - vgm, 45
- bpy.colors, 1
- bubble, 2
- cm.colors, 2
- cross.name (*gstat-internal*), 9
- fit.lmc, 4
- fit.variogram, 4, 5, 5, 7, 15, 17, 30, 31, 45, 47
- fit.variogram.reml, 6
- fulmar, 7, 25
- get.contr, 8
- gstat, 4, 10, 15–18, 35–37
- gstat-internal, 9
- gstat.cv (*krige.cv*), 14
- gstat.debug (*gstat-internal*), 9
- gstat.formula (*gstat-internal*), 9
- gstat.load.set (*gstat-internal*), 9
- gstat.set (*gstat-internal*), 9
- identify, 3, 34
- image, 13
- image.data.frame, 13, 20, 21
- image.default, 13
- krige, 12, 16, 16, 20, 21, 27, 35–37
- krige.cv, 14
- load.variogram.model (*gstat-internal*), 9
- locator, 34, 38
- makegrid, 19, 41
- map.to.lev, 20
- mapasp, 3, 20
- meuse, 22, 23, 24
- meuse.all, 21, 24
- meuse.alt, 22
- meuse.grid, 23, 24
- ncp.grid, 8, 25, 29
- ossfim, 26
- oxford, 27
- panel.pointPairs (*gstat-internal*), 9

pcb, 28
plot.gstatVariogram, 29, 33, 34, 42, 45
plot.pointPairs, 31, 34
plot.variogramCloud, 32, 33, 45
plot.variogramMap
 (plot.gstatVariogram), 29
point.in.polygon, 34, 38
predict.gstat, 8, 9, 12, 15–18, 20, 21,
 35, 36
print.gstat (gstat), 10
print.gstatVariogram, 45
print.gstatVariogram (variogram),
 43
print.variogramCloud (variogram),
 43
print.variogramModel (vgm), 45

rainbow, 2

select.spatial, 38
show.vgms, 39, 47
sic.grid (sic2004), 40
sic.pred (sic2004), 40
sic.test (sic2004), 40
sic.train (sic2004), 40
sic.val (sic2004), 40
sic2004, 40

variogram, 4–6, 30, 31, 34, 43, 47
variogram.default, 43
variogram.line, 31, 40, 42, 47
vgm, 4–6, 10, 15, 17, 30, 31, 40, 45, 45
vgm.dir.panel.xyplot
 (gstat-internal), 9
vgm.panel.xyplot
 (gstat-internal), 9

xvgm.panel.xyplot
 (gstat-internal), 9
xyz2img, 13
xyz2img (image), 13

zerodist, 47