Package ‘gstat’

March 12, 2017

Version 1.1-5
Title Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation
Description Variogram modelling; simple, ordinary and universal point or block (co)kriging; spatio-
temporal kriging; sequential Gaussian or indicator (co)simulation; variogram and vari-
ogram map plotting utility functions.
Depends R (>= 2.10)
Imports utils, stats, graphics, methods, lattice, sp (>= 0.9-72), zoo,
spacetime (>= 1.0-0), FNN
Suggests rgdal (>= 0.5.2), rgeos, fields, maps, mapdata, maptools, xts
License GPL (>= 2.0)
URL https://github.com/edzer/gstat/
Encoding UTF-8
BugReports https://github.com/edzer/gstat/issues/
NeedsCompilation yes
Author Edzer Pebesma [aut, cre],
Benedikt Graeler [aut]
Maintainer Edzer Pebesma <edzer.pebesma@uni-muenster.de>
Repository CRAN
Date/Publication 2017-03-12 09:02:30

R topics documented:

coalash ............................................................... 3
DE_RB_2005 .......................................................... 4
estiStAni ............................................................. 5
extractPar ............................................................ 6
fit.lmc ............................................................... 7
fit.StVariogram ..................................................... 8
fit.variogram ....................................................... 11
R topics documented:

fit.variogram.gls .......................................................... 13
fit.variogram.reml ....................................................... 14
fulmar ................................................................. 15
get.contr ............................................................... 16
gstat ................................................................. 17
hscat ................................................................. 21
image ................................................................. 22
jura ................................................................. 24
krige ................................................................. 26
krige.cv ............................................................... 30
krigeST ............................................................... 33
krigeTg ............................................................... 35
map.to.lev ............................................................. 37
meuse.all ............................................................. 38
meuse.alt ............................................................. 40
ncp.grid .............................................................. 40
ossfim ............................................................... 41
oxford ............................................................... 43
pcb ................................................................. 44
plot.gstatVariogram .................................................. 46
plot.pointPairs ......................................................... 48
plot.variogramCloud .................................................. 50
predict ............................................................... 51
progress .............................................................. 55
show.vgms ............................................................ 56
sic2004 ............................................................... 57
sic97 ................................................................. 59
spplot.vcov ............................................................. 60
tull ................................................................. 61
variogram ............................................................ 63
variogramLine ......................................................... 67
variogramST ........................................................... 68
variogramSurface ...................................................... 70
vgm ................................................................. 71
vgm.panel.xyplot ...................................................... 74
vgmArea ............................................................... 75
vgmST ............................................................... 76
vv ................................................................. 78
walker .............................................................. 79
wind .............................................................. 80

Index .......................................................... 83
coalash

Coal ash samples from a mine in Pennsylvania

Description

Data obtained from Gomez and Hazen (1970, Tables 19 and 20) on coal ash for the Robena Mine Property in Greene County Pennsylvania.

Usage

data(coalash)

Format

This data frame contains the following columns:

- x  a numeric vector; x-coordinate; reference unknown
- y  a numeric vector; x-coordinate; reference unknown
- coalash  the target variable

Note

data are also present in package fields, as coalash.

Author(s)

unknown; R version prepared by Edzer Pebesma; data obtained from http://www.stat.uiowa.edu/~dzimmer/spatialstats/, Dale Zimmerman’s course page

References


see also fields manual: http://www.image.ucar.edu/GSP/Software/Fields/fields.manual.coalashEX.Krig.shtml

Examples

data(coalash)
summary(coalash)
DE_RB_2005

Spatio-temporal data set with rural background PM10 concentrations in Germany 2005

Description

Spatio-temporal data set with rural background PM10 concentrations in Germany 2005 (airbase v6).

Usage

data("DE_RB_2005")

Format

The format is: Formal class 'STSDF' [package "spacetime"] with 5 slots .@ data :'data.frame': 23230 obs. of 2 variables: ..$ PM10 : num [1:23230] 16.7 31.7 5 22.4 26.8 ... ..$ logPM10: num [1:23230] 2.82 3.46 1.61 3.11 3.29 ... ..@ index : int [1:23230, 1:2] 1 2 3 4 5 6 7 8 9 10 ... ..@ sp :Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots ..@ data :'data.frame': 69 obs. of 9 variables: .. ..$ station_altitude : int [1:69] 8 3 700 15 35 50 343 339 45 45 .. ..$ station_european_code: Factor w/ 7965 levels "AD0942A","AD0944A",...: 1991 1648 1367 1113 1098 1437 2043 1741 1998 ... .. ..$ country_iso_code : Factor w/ 39 levels "AD","AL","AT",...: 10 10 10 10 10 10 10 10 10 ... .. ..$ station_start_date : Factor w/ 2409 levels "1900-01-01","1951-04-01",...: 152 1184 1577 1132 744 1202 1555 1148 407 ... .. ..$ station_end_date : Factor w/ 864 levels "","1975-02-06",...: 1 1 1 579 1 1 1 1 1 ... .. ..$ station_type_of_area : Factor w/ 4 levels "rural","suburban",...: 1 1 1 1 1 1 1 1 1 ... .. ..$ street_type : Factor w/ 5 levels "","Canyon street: L/H < 1.5",...: 4 1 1 1 1 1 1 1 ... .. ..$ annual_mean_PM10 : num [1:69] 20.9 21.8 16.5 20.3 23.3 ... ..@ coords.nrs : num(0) .. ..@ coords : num [1:69] 538709 545414 665711 551796 815738 ... .. .. - attr(*, "dimnames")=List of 2 .. .. .. ..$ : chr [1:69] "DESH001" "DENI063" "DEBY109" "DEUB038" ... .. .. .. ..$ : chr [1:2] "coords.x1" "coords.x2" .. .. .. ..@ bbox : num [1:2, 1:2] 307809 5295752 907375 6086661 ... .. .. - attr(*, "dimnames")=List of 2 .. .. .. ..$ : chr [1:2] "coords.x1" "coords.x2" ... .. .. ..$ : chr [1:2] "min" "max" .. .. .. ..@ proj4string: Formal class 'CRS' [package "sp"] with 1 slot .. .. .. ..@ projargs: chr +init=epsg:32632 +proj=utm +zone=32 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0" ..@ time :An ?xts? object on 2005-01-01/2005-12-31 containing: Data: int [1:365, 1] 5115 5116 5117 5118 5119 5120 5121 5122 5123 5124 ... - attr(*, "dimnames")=List of 2 ..$ : NULL ..$ : chr ".1" Indexed by objects of class: [POSIXct,POSIXt] TZ: GMT xts Attributes: NULL ..@ endTime: POSIXct[1:365], format: "2005-01-02" "2005-01-03" "2005-01-04" "2005-01-05" ...

Source

EEA, airbase v6
Examples

```r
data(DE_RB_2005)
str(DE_RB_2005)
```

---

**estiStAni**

*Estimation of the spatio-temporal anisotropy*

**Description**

Estimation of the spatio-temporal anisotropy without an underlying spatio-temporal model. Different methods are implemented using a linear model to predict the temporal gamma values or the ratio of the ranges of a spatial and temporal variogram model or a spatial variogram model to predict the temporal gamma values or the spatio-temporal anisotropy value as used in a metric spatio-temporal variogram.

**Usage**

```r
estiStAni(empVgm, interval, method = "linear", spatialVgm, 
temporalVgm, s.range=NA, t.range=NA)
```

**Arguments**

- `interval`: A search interval for the optimisation of the spatio-temporal anisotropy parameter.
- `method`: A character string determining the method to be used (one of `linear`, `range`, `vgm` or `metric`, see below for details).
- `spatialVgm`: A spatial variogram definition from the call to `vgm`. The model is optimised based on the pure spatial values in `empVgm`.
- `temporalVgm`: A temporal variogram definition from the call to `vgm`. The model is optimised based on the pure temporal values in `empVgm`.
- `s.range`: A spatial cutoff value applied to the empirical variogram `empVgm`.
- `t.range`: A temporal cutoff value applied to the empirical variogram `empVgm`.

**Details**

- **linear** A linear model is fitted to the pure spatial gamma values based on the spatial distances. An optimal scaling is searched to stretch the temporal distances such that the linear model explains best the pure temporal gamma values. This assumes (on average) a linear relationship between distance and gamma, hence it is advisable to use only those pairs of pure spatial (pure temporal) distance and gamma value that show a considerable increase (i.e. drop all values beyond the range by setting values for `s.range` and `t.range`).
- **range** A spatial and temporal variogram model is fitted to the pure spatial and temporal gamma values respectively. The spatio-temporal anisotropy estimate is the ratio of the spatial range over the temporal range.
A spatial variogram model is fitted to the pure spatial gamma values. An optimal scaling is used to stretch the temporal distances such that the spatial variogram model explains best the pure temporal gamma values.

**metric** A metric spatio-temporal variogram model is fitted with joint component according to the defined spatial variogram \( \text{spatialVgm} \). The starting value of \( \text{stAni} \) is the mean of the interval parameter (see \( \text{vgmST} \) for the metric variogram definition). The spatio-temporal anisotropy as estimated in the spatio-temporal variogram is returned. Note that the parameter interval is only used to set the starting value. Hence, the estimate might exceed the given interval.

**Value**
A scalar representing the spatio-temporal anisotropy estimate.

**Note**
Different methods might lead to very different estimates. All but the linear approach are sensitive to the variogram model selection.

**Author(s)**
Benedikt Graeler

**Examples**
```r
data(vv)
estiStAni(vv, c(10, 150))
estiStAni(vv, c(10, 150), "vgm", vgm(80, "Sph", 120, 20))```

---

**extractPar**

*Extracting parameters and their names from a spatio-temporal variogram model*

**Description**
All spatio-temporal variogram models have a different set of parameters. These functions extract the parameters and their names from the spatio-temporal variogram model. Note, this function is as well used to pass the parameters to the optim function. The arguments lower and upper passed to optim should follow the same structure.

**Usage**
```r
extractPar(model)
extractParNames(model)
```

**Arguments**
- **model** a spatio-temporal variogram model from \( \text{vgmST} \)
Value

A named numeric vector of parameters or a vector of characters holding the parameters’ names.

Author(s)

Benedikt Graeler

See Also

fit.StVariogram and vgmST

Examples

```r
summetricmodel <- vgmST("sumMetric",
    space=vgm(30, "Sph",  200, 6),
    time =vgm(30, "Sph",  15, 7),
    joint=vgm(60, "Exp",  84, 22),
    stAni=100)

extractPar(summetricModel)
extractParNames(summetricModel)
```

---

**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

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

Arguments

- **v** multivariable sample variogram, output of `variogram`
- **g** gstat object, output of `gstat`
- **model** variogram model, output of `vgm`; if supplied this value is used as initial value for each fit
- **fit.ranges** 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.
- **fit.lmc** logical; if TRUE, each coefficient matrices of partial sills is guaranteed to be positive definite
correct.diagonal

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

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 Pebesma

References

http://www.gstat.org/

See Also

variogram, vgm, fit.variogram, demo(cokriging)
**Arguments**

object The spatio-temporal sample variogram. Typically output from `variogramST`.
model The desired spatio-temporal model defined through `vgmST`.
... further arguments passed to `optim`. `extractParNames` provides the parameter structure of spatio-temporal variogram models that help to provide sensible upper and lower limits.
lower Lower limits used by optim. If missing, the smallest well defined values are used (mostly near 0).
upper Upper limits used by optim. If missing, the largest well defined values are used (mostly `Inf`).
method fit method, pass to `optim`
fit.method an integer between 0 and 13 determine the fitting routine (i.e. weighting of the squared residuals in the LSE). Values 0 to 6 correspond with the pure spatial version (see `fit.variogram`). See the details section for the meaning of the other values (partly experimental).

stAni The spatio-temporal anisotropy that is used in the weighting. Might be missing if the desired spatio-temporal variogram model does contain a spatio-temporal anisotropy parameter (this might cause bad convergence behaviour). The default is `NA` and will be understood as identity (1 temporal unit = 1 spatial unit). As this only in very few cases a valid assumption, a warning is issued.

wles Should be missing; only for backwards compatibility. `wles = TRUE` corresponds to `fit.method = 1` and `wles = FALSE` corresponds to `fit.method = 6`.

**Details**

The following list summarizes the meaning of the `fit.method` argument which is essential a weighting of the squared residuals in the least-squares estimation. Please note, that weights based on the models gamma value might fail to converge properly due to the dependence of weights on the variogram estimate:

- `fit.method = 0` no fitting, however the MSE between the provided variogram model and sample variogram surface is calculated.
- `fit.method = 1` Number of pairs in the spatio-temporal bin: $N_j$
- `fit.method = 2` Number of pairs in the spatio-temporal bin divided by the square of the current variogram model’s value: $N_j / \gamma(h_j, u_j)^2$
- `fit.method = 3` Same as `fit.method = 1` for compatibility with `fit.variogram` but as well evaluated in R.
- `fit.method = 4` Same as `fit.method = 2` for compatibility with `fit.variogram` but as well evaluated in R.
- `fit.method = 5` Reserved for REML for compatibility with `fit.variogram`, not yet implemented.
- `fit.method = 6` No weights.
- `fit.method = 7` Number of pairs in the spatio-temporal bin divided by the square of the bin’s metric distance. If `stAni` is not specified, the model’s parameter is used to calculate the metric distance across space and time: $N_j / (h_j^2 + stAni^2 \cdot u_j^2)$
fit.StVariogram

fit.method = 8 Number of pairs in the spatio-temporal bin divided by the square of the bin’s spatial distance. \( N_j/h_j^2 \). Note that the 0 distances are replaced by the smallest non-zero distances to avoid division by zero.

fit.method = 9 Number of pairs in the spatio-temporal bin divided by the square of the bin’s temporal distance. \( N_j/u_j^2 \). Note that the 0 distances are replaced by the smallest non-zero distances to avoid division by zero.

fit.method = 10 Reciprocal of the square of the current variogram model’s value: \( 1/\gamma(h_j, u_j)^2 \)

fit.method = 11 Reciprocal of the square of the bin’s metric distance. If stAni is not specified, the model’s parameter is used to calculate the metric distance across space and time: \( 1/(h_j^2 + stAni^2 \cdot u_j^2) \)

fit.method = 12 Reciprocal of the square of the bin’s spatial distance. \( 1/h_j^2 \). Note that the 0 distances are replaced by the smallest non-zero distances to avoid division by zero.

fit.method = 13 Reciprocal of the square of the bin’s temporal distance. \( 1/u_j^2 \). Note that the 0 distances are replaced by the smallest non-zero distances to avoid division by zero.

See also Table 4.2 in the gstat manual for the original spatial version.

Value

Returns a spatio-temporal variogram model, as S3 class StVariogramModel. It carries the temporal and spatial unit as attributes “temporal unit” and “spatial unit” in order to allow krigeST to adjust for different units. The units are obtained from the provided empirical variogram. Further attributes are the optim output “optim.output” and the always not weighted mean squared error “MSE”.

Author(s)

Benedikt Graeler

See Also

fit.variogram for the pure spatial case. extractParNames helps to understand the parameter structure of spatio-temporal variogram models.

Examples

# separable model: spatial and temporal sill will be ignored # and kept constant at 1-nugget respectively. A joint sill is used. ## Not run:
separableModel <- vgmST("separable",
    method = "Nelder-Mead", # no lower & upper needed space=vgm(0.9,"Exp", 123, 0.1),
    time =vgm(0.9,"Exp", 2.9, 0.1),
    sill=100)

data(vv)
separableModel <- fit.StVariogram(vv, separableModel,
    method="L-BFGS-B",
    lower=c(10,0.01,0,1),
    upper=c(10,0.01,0,1))
fit.variogram

Fit a Variogram Model to a Sample Variogram

Description

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

Usage

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

Arguments

- `object`: sample variogram, output of `variogram`
- `model`: variogram model, output of `vgm`; see Details below for details on how NA values in model are initialised.
- `fit.sills`: 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.
- `fit.ranges`: 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.
- `fit.method`: fitting method, used by gstat. The default method uses weights $N_j/h^2$ with $N_j$ the number of point pairs and $h$ the distance. This criterion is not supported by theory, but by practice. For other values of `fit.method`, see details.
- `debug.level`: integer; set gstat internal debug level
- `warn.if.neg`: logical; if TRUE a warning is issued whenever a sill value of a direct variogram becomes negative
- `fit.kappa`: logical; if TRUE, a sequence of 0.3, 0.4,...,5 will be searched for optimal fit; alternatively another sequence can be given to this argument

Details

If any of the initial parameters of `model` are NA, they are given default values as follows. The range parameter is given one third of the maximum value of `object$dist`. The nugget value is given the mean value of the first three values of `object$gamma`. The partial sill is given the mean of the last five values of `object$gamma`. 

Values for `fit.method` are 1: weights equal to $N_j$; 2: weights equal to $N_j/(gamma(h_j))^2$; 5 (ignore, use `fit.variogram.reml`); 6: unweighted (OLS); 7: $N_j/(h_j^2)$. (from: http://www.gstat.org/gstat.pdf, table 4.2).
Value

returns a fitted variogram model (of class variogramModel).
This is a data.frame with two attributes: (i) singular a logical attribute that indicates whether
the non-linear fit converged (FALSE), or ended in a singularity (TRUE), and (ii) SSErr a numerical
attribute with the (weighted) sum of squared errors of the fitted model. See Notes below.

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, which is generally the case for non-linear regression problems.
This function uses internal C code, which uses Levenberg-Marquardt.

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 the search does not move away from search space boundaries.

On singular model fits: If your variogram turns out to be a flat, horizontal or sloping line, then fitting
a three parameter model such as the exponential or spherical with nugget is a bit heavy: there’s an
infinite number of possible combinations of sill and range (both very large) to fit to a sloping line. In
this case, the returned, singular model may still be useful: just try and plot it. Gstat converges when
the parameter values stabilize, and this may not be the case. Another case of singular model fits
happens when a model that reaches the sill (such as the spherical) is fit with a nugget, and the range
parameter starts, or converges to a value smaller than the distance of the second sample variogram
estimate. In this case, again, an infinite number of possibilities occur essentially for fitting a line
through a single (first sample variogram) point. In both cases, fixing one or more of the variogram
model parameters may help you out.

Author(s)

Edzer Pebesma

References

http://www.gstat.org/


See Also

variogram, vgm

Examples

library(sp)
data(meuse)
coordinates(meuse) ~ x+y
vgm1 <- variogram(log(zinc)-1, meuse)
fit.variogram(vgm1, vgm(1, "Sph", 300, 1))
fit.variogram(vgm1, vgm("Sph"))
### fit.variogram.gls

GLS fitting of variogram parameters

#### Description

Fits variogram parameters (nugget, sill, range) to variogram cloud, using GLS (generalized least squares) fitting. Only for direct variograms.

#### Usage

```r
call <- fit.variogram.gls(formula, data, model, maxiter = 30, 
                         eps = .01, trace = TRUE, ignoreInitial = TRUE, cutoff = Inf, 
                         plot = FALSE)
```

#### Arguments

- **formula**: formula defining the response vector and (possible) regressors; in case of absence of regressors, use e.g. `z~1`  
- **data**: object of class Spatial  
- **model**: variogram model to be fitted, output of `vgm`  
- **maxiter**: maximum number of iterations  
- **eps**: convergence criterium  
- **trace**: logical; if TRUE, prints parameter trace  
- **ignoreInitial**: logical; if FALSE, initial parameter are taken from model; if TRUE, initial values of model are ignored and taken from variogram cloud: nugget: mean(y)/2, sill: mean(y)/2, range median(h0)/4 with y the semivariance cloud value and h0 the distances  
- **cutoff**: maximum distance up to which point pairs are taken into consideration  
- **plot**: logical; if TRUE, a plot is returned with variogram cloud and fitted model; else, the fitted model is returned.

#### Value

an object of class "variogramModel"; see `fit.variogram`; if `plot` is TRUE, a plot is returned instead.

#### Note

Inspired by the code of Mihael Drinovac, which was again inspired by code from Ernst Glatzer, author of package vardiag.
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 = 0)

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 set=list(iter=100) to set the max. number of iterations to 100.
- **degree**: order of trend surface in the location, between 0 and 3
**Value**

an object of class "variogramModel"; 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 demanding. I guess there is much more to likelihood variogram fitting in package geor, and probably also in nlme.

**Author(s)**

Edzer Pebesma

**References**


**See Also**

`fit.variogram`,

**Examples**

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

---

**fulmar**

*Fulmaris glacialis data*

**Description**

Airborne counts of *Fulmaris glacialis* during the Aug/Sept 1998 and 1999 flights on the Dutch (Netherlands) part of the North Sea (NCP, Nederlands Continentaal Plat).

**Usage**

```r
data(fulmar)
```
Format

This data frame contains the following columns:

- **year** year of measurement: 1998 or 1999
- **x** x-coordinate in UTM zone 31
- **y** y-coordinate in UTM zone 31
- **depth** sea water depth, in m
- **coast** distance to coast of the Netherlands, in km.
- **fulmar** observed density (number of birds per square km)

Author(s)

Dutch National Institute for Coastal and Marine Management (RIKZ), [http://www.rikz.nl/](http://www.rikz.nl/)

See Also

- `ncp.grid`

Examples

```r
data(fulmar)
summary(fulmar)
## Not run:
demo(fulmar)
## End(Not run)
```

---

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

Description

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

Usage

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

Arguments

- **data** data frame, output of `predict`
- **gstat.object** object of class `gstat`, used to extract ids; may be missing if ids is used
- **X** contrast vector or matrix; the number of variables in `gstat.object` should equal the number of elements in `X` if `X` is a vector, or the number of rows in `X` if `X` is a matrix.
- **ids** character vector with (selection of) id names, present in data
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 $C=X'y$ and its variance $\text{Var}(C)=X'VX$.

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, ...)

Author(s)

Edzer Pebesma

References

http://www.gstat.org/

See Also

predict

Usage

```r
gstat(g, id, formula, locations, data, model = NULL, beta, 
nmax = Inf, nmin = 0, omax = 0, maxdist = Inf, force = FALSE, 
dummy = FALSE, set, fill.all = FALSE, 
fill.cross = TRUE, variance = "identity", weights = NULL, merge, 
degree = 0, vdist = FALSE, lambda = 1.0) 
## S3 method for class 'gstat'
print(x, ...)
```
Arguments

**g**
gstat object to append to; if missing, a new gstat object is created

**id**
identifier of new variable; if missing, `varn` is used with `n` the number for this variable. If a cross variogram is entered, `id` should be a vector with the two `id` values, e.g. `c("zn", "cd")`, further only supplying arguments `g` and `model`. It is advisable not to use expressions, such as `log(zinc)`, as identifiers, as this may lead to complications later on.

**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`

**locations**
formula with only independent variables that define the spatial data locations (coordinates), e.g. `~x+y`; if data has a coordinates method to extract its coordinates this argument can be ignored (see package sp for classes for point or grid data).

**data**
data frame; contains the dependent variable, independent variables, and locations.

**model**
variogram model for this `id`; defined by a call to `vgm`; see argument `id` to see how cross variograms are entered

**beta**
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 expected value; for cross variogram computations: mean parameters to be used instead of the OLS estimates

**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

**nmin**
for local kriging: if the number of nearest observations within distance `maxdist` is less than `nmin`, a missing value will be generated, unless `force`==`TRUE`; see `maxdist`

**omax**
maximum number of observations to select per octant (3D) or quadrant (2D); only relevant if `maxdist` has been defined as well

**maxdist**
for local kriging: only observations within a distance of `maxdist` from the prediction location are used for prediction or simulation; if combined with `nmax`, both criteria apply

**force**
for local kriging, force neighbourhood selection: in case `nmin` is given, search beyond `maxdist` until `nmin` neighbours are found. A missing value is returned if this is not possible.

**dummy**
logical; if `TRUE`, consider this data as a dummy variable (only necessary for unconditional simulation)

**set**
named list with optional parameters to be passed to gstat (only `set` commands of gstat are allowed, and not all of them may be relevant; see the manual for gstat stand-alone, URL below)
gstat

gstat object to print

fill.all
logical; if TRUE, fill all of the direct variogram and, depending on the value of fill.cross also all cross variogram model slots in g with the given variogram model

fill.cross
logical; if TRUE, fill all of the cross variograms, if FALSE fill only all direct variogram model slots in g with the given variogram model (only if fill.all is used)

variance
character; variance function to transform to non-stationary covariances; "identity" does not transform, other options are "mu" (Poisson) and "mu(1-mu)" (binomial)

weights
numeric vector; if present, covariates are present, and variograms are missing weights are passed to OLS prediction routines resulting in WLS; if variograms are given, weights should be 1/variance, where variance specifies location-specific measurement error; see references section 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 c("id1", 1,"id2", 2). This merges the first parameter for variable id1 to the second of variable id2.

degree
order of trend surface in the location, between 0 and 3

vdist
logical; if TRUE, instead of Euclidian distance variogram distance is used for selecting the nmax nearest neighbours, after observations within distance maxdist (Euclidian/geographic) have been pre-selected

lambda
test feature; doesn’t do anything (yet)

... 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 . (e.g.: var1.var2)

set
list; named list, corresponding to set name=value; 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 colocated 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 were 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. There is also demo(pcb) which merges slopes across years, but with year-dependent intercept.

Author(s)

Edzer Pebesma

References


for kriging with known, varying measurement errors (weights), see e.g. Delhomme, J.P. Kriging in the hydrosciences. Advances in Water Resources, 1(5):251-266, 1978; see also the section Kriging with known measurement errors in the gstat user's manual. http://www.gstat.org/

See Also

predict, krige

Examples

library(sp)
data(meuse)
coordinates(meuse) = ~x+y
# let's do some manual fitting of two direct variograms and a cross variogram
g <- gstat(id = "ln.zinc", formula = log(zinc)-1, data = meuse)
g <- gstat(g, id = "ln.lead", formula = log(lead)-1, 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:
hscat

Produce h-scatterplot

Description

Produce h-scatterplots, where point pairs having specific separation distances are plotted. This function is a wrapper around xyplot.

Usage

hscat(formula, data, breaks, pch = 3, cex = .6, mirror = FALSE, variogram.alpha = 0, as.table = TRUE,...)

Arguments

formula specifies the dependent variable
data data where the variable in formula is resolved
breaks distance class boundaries
pch  
  plotting symbol

cex  
  plotting symbol size

mirror  
  logical; duplicate all points mirrored along x=y? (note that correlations are those of the points plotted)

variogram.alpha  
  parameter to be passed as alpha parameter to variogram; if alpha is specified it will only affect xyplot by being passed through ...

as.table  
  logical; if TRUE, panels plot top-to-bottom

...  
  parameters, passed to variogram and xyplot

Value

an object of class trellis; normally the h scatter plot

Note

Data pairs are plotted once, so the h-scatterplot are not symmetric.

Author(s)

Edzer Pebesma

References

http://www.gstat.org/


Examples

library(sp)
data(meuse)
coordinates(meuse) = ~x+y
hscat(log(zinc)-1, meuse, c(0, 80, 120, 250, 500, 1000))

Description

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

Usage

## S3 method for class 'data.frame'
image(x, zcol = 3, xcol = 1, ycol = 2, asp = 1, ...) 
xyz2img(xyz, zcol = 3, xcol = 1, ycol = 2, tolerance = 10 * .Machine$double.eps)
**image**

Arguments

- **x**: data frame (or matrix) with x-coordinate, y-coordinate, and z-coordinate in its columns
- **zcol**: column number or name of z-variable
- **xcol**: column number or name of x-coordinate
- **ycol**: column number or name of y-coordinate
- **asp**: aspect ratio for the x and y axes
- **xyz**: data frame (same as x)
- **tolerance**: maximum allowed 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 Pebesma

Examples

```r
library(sp)
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")
oldpin[2]
oldpin[1]
ratio <- length(unique(x$x))/length(unique(x$y))
par(par = c(oldpin[2]*ratio, oldpin[2]))
image(x, main="Exactly square cells, using par(pin)"
par(par = oldpin)
library(lattice)
levelplot(var~x+y, x, aspect = "iso", main = "kriging variance")

## jura

### Jura data set

**Description**

The jura data set from Pierre Goovaerts’ book (see references below). It contains four data frames: prediction.dat, validation.dat and transect.dat and juragrid.dat, and three data frames with consistently coded land use and rock type factors, as well as geographic coordinates. The examples below show how to transform these into spatial (sp) objects in a local coordinate system and in geographic coordinates, and how to transform to metric coordinate reference systems.

**Usage**

```r
data(jura)
```

**Format**

The data frames prediction.dat and validation.dat contain the following fields:

- **Xloc** X coordinate, local grid km
- **Yloc** Y coordinate, local grid km
- **Landuse** see book and below
- **Rock** see book and below
- **Cd** mg cadmium kg^-1 topsoil
- **Co** mg cobalt kg^-1 topsoil
- **Cr** mg chromium kg^-1 topsoil
- **Cu** mg copper kg^-1 topsoil
- **Ni** mg nickel kg^-1 topsoil
- **Pb** mg lead kg^-1 topsoil
- **Zn** mg zinc kg^-1 topsoil

The data frame juragrid.dat only has the first four fields. In addition the data frames jura.pred, jura.val and jura.grid also have inserted third and fourth fields giving geographic coordinates:

- **long** Longitude, WGS84 datum
- **lat** Latitude, WGS84 datum
Note

The points data sets were obtained from http://home.comcast.net/~pgoovaerts/book.html, which seems to be no longer available; the grid data were kindly provided by Pierre Goovaerts.

The following codes were used to convert prediction.dat and validation.dat to jura.pred and jura.val (see examples below):


Land uses: 1: Forest, 2: Pasture (Weide(land), Wiese, Grasland), 3: Meadow (Wiese, Flur, Matte, Anger), 4: Tillage (Ackerland, bestelltes Land)

Points 22 and 100 in the validation set (validation.dat[c(22,100),]) seem not to lie exactly on the grid originally intended, but are kept as such to be consistent with the book.

Georeferencing was based on two control points in the Swiss grid system shown as Figure 1 of Atteia et al. (see above) and further points digitized on the tentatively georeferenced scanned map. RMSE 2.4 m. Location of points in the field was less precise.

Author(s)

Data preparation by David Rossiter (dgr2@cornell.edu) and Edzer Pebesma; georeferencing by David Rossiter

References


Atteia, O., Dubois, J.-P., Webster, R., 1994, Geostatistical analysis of soil contamination in the Swiss Jura: Environmental Pollution 86, 315-327

Webster, R., Atteia, O., Dubois, J.-P., 1994, Coregionalization of trace metals in the soil in the Swiss Jura: European Journal of Soil Science 45, 205-218

Examples

data(jura)
summary(prediction.dat)
summary(validation.dat)
summary(transect.dat)
summary(juragrid.dat)

# the following commands were used to create objects with factors instead
# of the integer codes for Landuse and Rock:
## Not run:
    jura.pred = prediction.dat
    jura.val = validation.dat
    jura.grid = juragrid.dat

    jura.pred$Landuse = factor(prediction.dat$Landuse,
                              labels=levels(juragrid.dat$Landuse))
    jura.pred$Rock = factor(prediction.dat$Rock,
                            labels=levels(juragrid.dat$Rock))
    jura.val$Landuse = factor(validation.dat$Landuse,
labels=levels(juragrid.dat$Landuse))
jura.val$Rock = factor(validation.dat$Rock,
labels=levels(juragrid.dat$Rock))

## End(Not run)

# the following commands convert data.frame objects into spatial (sp) objects
#   in the local grid:
require(sp)
coordinates(jura.pred) = ~Xloc+Yloc
coordinates(jura.val) = ~Xloc+Yloc
coordinates(jura.grid) = ~Xloc+Yloc
gridded(jura.grid) = TRUE

# the following commands convert the data.frame objects into spatial (sp) objects
#   in WGS84 geographic coordinates
# example is given only for jura.pred, do the same for jura.val and jura.grid
# EPSG codes can be found by searching make_EPSG()
jura.pred <- as.data.frame(jura.pred)
coordinates(jura.pred) = ~ long + lat
proj4string(jura.pred) = CRS("+init=epsg:4326")

# display in Google Earth
if (require(maptools)) {
  kmlPoints(jura.pred, kmlfile="JuraPred.kml",
  kmlName="Jura Prediction Points", name=row.names(jura.pred@data),
  description=paste(jura.pred$Landuse, jura.pred$Rock, sep="/"))
}

if (require(rgdal)) {
  # transform to UTM 32N
  jura.pred.utm32n = spTransform(jura.pred, CRS("+init=epsg:32632"))
  coordnames(jura.pred.utm32n) = c("E","N")

  # transform to Swiss grid system CH1903 / LV03
  jura.pred.ch = spTransform(jura.pred, CRS("+init=epsg:21781"))
  coordnames(jura.pred.ch) = c("X","Y")
}

kridge 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 grid).
blocks), and conditional (Gaussian or indicator) simulation equivalents for all kriging varieties, and function for inverse distance weighted interpolation. For multivariable prediction, see gstat and predict

Usage

krige(formula, locations, ...)  
krige.layers(formula, locations, data, newdata, model, ..., beta, nmax  
= Inf, nmin = 0, omax = 0, maxdist = Inf, block, nsim = 0, indicators = FALSE,  
na.action = na.pass, debug.level = 1)  
krige.spatial(formula, locations, newdata, model, ..., beta, nmax  
= Inf, nmin = 0, omax = 0, maxdist = Inf, block, nsim = 0, indicators = FALSE,  
na.action = na.pass, debug.level = 1)  
krige0(formula, data, newdata, model, beta, y, ..., computeVar = FALSE,  
fullCovariance = FALSE)  
idw(formula, locations, ...)  
idw.layers(formula, locations, data, newdata, nmax = Inf,  
nmin = 0, omax = 0, maxdist = Inf, block, na.action = na.pass, idp = 2.0,  
debug.level = 1)  
idw.spatial(formula, locations, newdata, nmax = Inf, nmin = 0,  
omax = 0, maxdist = Inf, block = numeric(0), na.action = na.pass, idp = 2.0,  
debug.level = 1)  
idw0(formula, data, newdata, y, idp = 2.0)

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

locations object of class Spatial, or (deprecated) formula defines the spatial data locations (coordinates) such as ~x+y

data data frame: should contain the dependent variable, independent variables, and coordinates, should be missing if locations contains data.

newdata data frame or Spatial object with prediction/simulation locations; should contain attribute columns with the independent variables (if present) and (if locations is a formula) the coordinates with names as defined in locations

model variogram model of dependent variable (or its residuals), defined by a call to vgm or fit.variogram; for krige0 also a user-supplied covariance function is allowed (see example below)

beta 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 beta 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
**krige**

- **nmin**: for local kriging: if the number of nearest observations within distance `maxdist` is less than `nmin`, a missing value will be generated; see `maxdist`
- **omax**: see `gstat`
- **maxdist**: for local kriging: only observations within a distance of `maxdist` from the prediction location are used for prediction or simulation; if combined with `nmax`, both criteria apply
- **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 of `predict`. 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 `indicators`), following a single random path through the data.
- **indicators**: logical, only relevant if `nsim` is non-zero; if TRUE, use indicator simulation; else use Gaussian simulation
- **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.
- **debug.level**: debug level, passed to `predict`; use -1 to see progress in percentage, and 0 to suppress all printed information
- **...**: for krige: arguments that will be passed to `gstat`; for krige0: arguments that will be passe to `model`
- **idp**: numeric; specify the inverse distance weighting power
- **y**: matrix; to krige multiple fields in a single step, pass data as columns of matrix y. This will ignore the value of the response in `formula`.
- **computeVar**: logical; if TRUE, prediction variances will be returned
- **fullCovariance**: logical; if FALSE a vector with prediction variances will be returned, if TRUE the full covariance matrix of all predictions will be returned

**Details**

Function `krige` is a simple wrapper method around `gstat` and `predict` 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` directly.

Function `idw` performs just as `krige` without a model being passed, but allows direct specification of the inverse distance weighting power. Don’t use with predictors in the formula.

For further details, see `predict`. 
krige

Value

if locations is not a formula, object of the same class as newdata (deriving from Spatial); else a data frame containing the coordinates of newdata. Attributes columns contain prediction and prediction variance (in case of kriging) or the abs(nsim) columns of the conditional Gaussian or indicator simulations

krige and idw are alternative functions with reduced functionality and larger memory requirements; they return numeric vectors (or matrices, in case of multiple dependent) with predicted values only; in case computeVar is TRUE, a list with elements pred and var is returned, containing predictions, and (co)variances (depending on argument fullCovariance).

Methods

formula = "formula", locations = "formula" locations specifies which coordinates in data refer to spatial coordinates

formula = "formula", locations = "Spatial" Object locations knows about its own spatial locations

formula = "formula", locations = "NULL" used in case of unconditional simulations; newdata needs to be of class Spatial

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 Pebesma

References

http://www.gstat.org/

See Also

gstat, predict

Examples

library(sp)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
kriged(meuse.grid) = -x+y
m <- vgm(.59, "Sph", 874, .04)
# ordinary kriging:
x <- krige(log(zinc)-1, meuse, meuse.grid, model = m)
spplot(x["var1.pred"], main = "ordinary kriging predictions")
spplot(x["var1.var"], main = "ordinary kriging variance")
# simple kriging:
x <- krige(log(zinc)-1, meuse, meuse.grid, model = m, beta = 5.9)
# residual variogram:
m <- vgm(.4, "Sph", 954, .06)
# universal block kriging:
x <- krige(log(zinc)-x+y, meuse, meuse.grid, model = m, block = c(40,40))
spplot(x["var1.pred"], main = "universal kriging predictions")

# kriged, using user-defined covariance function and multiple responses in y:
# exponential variogram with range 500, defined as covariance function:
v = function(x, y = x) { exp(-spDists(coordinates(x),coordinates(y))/500) }
# krig two variables in a single pass (using 1 covariance model):
y = cbind(meuse$zinc,meuse$copper,meuse$lead,meuse$cadmium)
x <- krige0(zinc-1, meuse, meuse.grid, v, y = y)
meuse.grid$zinc = x[,1]
spplot(meuse.grid["zinc"], main = "zinc")
meuse.grid$copper = x[,2]
spplot(meuse.grid["copper"], main = "copper")

# the following has NOTHING to do with kriging, but --
# return the median of the nearest 11 observations:
x = krige(zinc-1, meuse, meuse.grid, set = list(method = "med"), nmax = 11)
# get 25%- and 75%-percentiles of nearest 11 obs, as prediction and variance:
x = krige(zinc-1, meuse, meuse.grid, nmax = 11,
set = list(method = "med", quantile = 0.25))
# get diversity (# of different values) and mode from 11 nearest observations:
x = krige(zinc-1, meuse, meuse.grid, nmax = 11, set = list(method = "div"))

---

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 = interactive(),
all.residuals = FALSE, ...)  
krige.cv(formula, locations, ...)
krige.cv.locations(formula, locations, data, model = NULL, ..., beta = NULL,
nmax = Inf, nmin = 0, maxdist = Inf, nfold = nrow(data),
verbose = interactive(), debug.level = 0)
krige.cv.spatial(formula, locations, model = NULL, ..., beta = NULL, nmax = Inf, nmin = 0, maxdist = Inf, nfold = nrow(locations), verbose = interactive(), debug.level = 0)

Arguments

object  
oBJECT of class gstat; see function gstat

nfold  
integer; if larger than 1, then apply n-fold cross validation; if nfold equals nrow(data) (the default), apply leave-one-out cross validation; if set to e.g. 5, five-fold cross validation is done. To specify the folds, pass an integer vector of length nrow(data) with fold indexes.

remove.all  
logical; if TRUE, remove observations at cross validation locations not only for the first, but for all subsequent variables as well

verbose  
logical; if FALSE, progress bar is suppressed

all.residuals  
logical; if TRUE, residuals for all variables are returned instead of for the first variable only

...  
other arguments that will be passed to predict in case of gstat.cv, or to gstat in case of krige.cv

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

locations  
formula with only independent variables that define the spatial data locations (coordinates), e.g. ~ x + y, OR data object deriving from class Spatial, which has a coordinates method to extract its coordinates.

data  
data frame; should contain the dependent variable, independent variables, and coordinates; only to be provided if locations is a formula

model  
variogram model of dependent variable (or its residuals), defined by a call to vgm or fit.variogram

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 maxdist is less than nmin, a missing value will be generated; see maxdist

maxdist  
for local kriging: only observations within a distance of maxdist from the prediction location are used for prediction or simulation; if combined with nmax, both criteria apply

debug.level  
print debugging information; 0 suppresses debug information
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.

Methods
`formula = "formula", locations = "formula"` locations specifies which coordinates in data refer to spatial coordinates
`formula = "formula", locations = "Spatial"` Object locations knows about its own spatial locations

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 Pebesma

References
http://www.gstat.org/

See Also
krige, gstat, predict

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

# multivariable; thanks to M. Rufino:
meuse.g <- gstat(id = "zn", formula = log(zinc) ~ 1, data = meuse)
meuse.g <- gstat(meuse.g, "cu", log(copper) ~ 1, meuse)
meuse.g <- gstat(meuse.g, model = vgm(1, "Sph", 900, 1), fill.all = TRUE)
x <- variogram(meuse.g, cutoff = 1000)
meuse.fit = fit.lmc(x, meuse.g)
out = gstat.cv(meuse.fit, nmax = 40, nfold = 5)
summary(out)
out = gstat.cv(meuse.fit, nmax = 40, nfold = c(rep(1,100), rep(2,55))
summary(out)
# mean error, ideally 0:
mean(out$residual)
# MSPE, ideally small
mean(out$residual^2)
# Mean square normalized error, ideally close to 1
mean(out$zscore^2)
# correlation observed and predicted, ideally 1
cor(out$observed, out$observed - out$residual)
# correlation predicted and residual, ideally 0
cor(out$observed - out$residual, out$residual)

krigeST

Ordinary global Spatio-Temporal Kriging

Description

Function for ordinary global and local and trans Gaussian spatio-temporal kriging on point support

Usage

krigeST(formula, data, newdata, modellist, y, beta, nmax = Inf, stAni = NULL,
computeVar = FALSE, fullCovariance = FALSE,
bufferNmax = 2, progress = TRUE)
kridgeSTg(formula, data, newdata, modellist, y, nmax = Inf, stAni = NULL,
bufferNmax = 2, progress = TRUE, lambda = 0)

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
data ST object: should contain the dependent variable and independent variables.
newdata ST object with prediction/simulation locations in space and time; should contain attribute columns with the independent variables (if present).
modellist: object of class StVariogramModel, created by vgmST; list with named elements: space, time and/or joint depending on the spatio-temporal covariance family, and an entry stModel. Currently implemented families that may be used for stModel are separable, productSum, metric, sumMetric and simpleSumMetric. See the examples section in fit.StVariogram or variogramSurface for details on how to define spatio-temporal covariance models. krigeST will look for a "temporal unit" attribute in the provided modellist in order to adjust the temporal scales.

y: matrix; to krig multiple fields in a single step, pass data as columns of matrix y. This will ignore the value of the response in formula.

beta: The (known) mean for simple kriging.

nmax: The maximum number of neighbouring locations for a spatio-temporal local neighbourhood.

stAni: a spatio-temporal anisotropy scaling assuming a metric spatio-temporal space. Used only for the selection of the closest neighbours. This scaling needs only to be provided in case the model does not have a stAni parameter, or if a different one should be used for the neighbourhood selection. Mind the correct spatial unit. Currently, no coordinate conversion is made for the neighbourhood selection (i.e. Lat and Lon require a spatio-temporal anisotropy scaling in degrees per second).

...: further arguments (currently unused)

computeVar: logical; if TRUE, prediction variances will be returned

fullCovariance: logical; if FALSE a vector with prediction variances will be returned, if TRUE the full covariance matrix of all predictions will be returned.

bufferNmax: factor with which nmax is multiplied for an extended search radius (default=2). Set to 1 for no extension of the search radius.

progress: whether a progress bar shall be printed for local spatio-temporal kriging; default=TRUE

lambda: The value of lambda used in the box-cox transformation.

Details

Function krigeST is a R implementation of the kriging function from gstat using spatio-temporal covariance models following the implementation of krige0. Function krigeST offers some particular methods for ordinary spatio-temporal (ST) kriging. In particular, it does not support block kriging or kriging in a distance-based neighbourhood, and does not provide simulation.

Value

An object of the same class as newdata (deriving from ST). Attributes columns contain prediction and prediction variance.

Author(s)

Edzer Pebesma, Benedikt Graeler
krigeTg

References


http://www.gstat.org/


See Also

krige0, gstat, predict, krigeTg

Examples

```r
library(sp)
library(spacetime)

sumMetricVgm <- vgmST("sumMetric",
    space=vgm( 4.4, "Lin", 196.6, 3),
    time =vgm( 2.2, "Lin", 1.1, 2),
    joint=vgm(34.6, "Exp", 136.6, 12),
    stAni=51.7)

data(air)

if (!exists("rural"))
rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))

rr <- rural[,"2005-06-01/2005-06-03"]
rr <- as(rr,"STFDF")

x1 <- seq(from=6,to=15,by=1)
x2 <- seq(from=48,to=55,by=1)

DE_gridded <- SpatialPoints(cbind(rep(x1,length(x2)), rep(x2,each=length(x1))),
    proj4string=CRS(proj4string(rr@sp)))
gridded(DE_gridded) <- TRUE
DE_pred <- STF(sp=as(DE_gridded,"SpatialPoints"), time=rr@time)
DE_kriged <- krigest(PM10~1, data=rr, newdata=DE_pred,
    model=list=sumMetricVgm)
gridded(DE_kriged@sp) <- TRUE
stplot(DE_kriged)
```

krigeTg  

TransGaussian kriging using Box-Cox transforms

Description

TransGaussian (ordinary) kriging function using Box-Cox transforms
Usage

krigeTg(formula, locations, newdata, model = NULL, ...,
        nmax = Inf, nmin = 0, maxdist = Inf, block = numeric(0),
        nsim = 0, na.action = na.pass, debug.level = 1,
        lambda = 1.0)

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 use a formula like z~1; the dependent variable should be NOT transformed.

locations object of class Spatial, with observations

newdata Spatial object with prediction/simulation locations; the coordinates should have names as defined in locations

model variogram model of the TRANSFORMED dependent variable, see vgm, or fit.variogram

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 maxdist is less than nmin, a missing value will be generated; see maxdist

maxdist for local kriging: only observations within a distance of maxdist from the prediction location are used for prediction or simulation; if combined with nmax, both criteria apply

block does not function correctly, afaiK

nsim does not function correctly, afaiK

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.

lambda value for the Box-Cox transform

debug.level debug level, passed to predict; use -1 to see progress in percentage, and 0 to suppress all printed information

... other arguments that will be passed to gstat

Details


As it uses the R/gstat krige function to derive everything, it needs in addition to ordinary kriging on the transformed scale a simple kriging step to find m from the difference between the OK and SK prediction variance, and a kriging/BLUE estimation step to obtain the estimate of μ.

For further details, see kriGe and predict.
map.to.lev

Value

an SpatialPointsDataFrame object containing the fields: m for the m (Lagrange) parameter for each location; var1SK.pred the $c_0 C^{-1}$ correction obtained by muhat for the mean estimate at each location; var1SK.var the simple kriging variance; var1.pred the OK prediction on the transformed scale; var1.var the OK kriging variance on the transformed scale; var1Tg.pred the transGaussian kriging predictor; var1Tg.var the transGaussian kriging variance, obtained by $\phi'(\hat{\mu}, \lambda)^2 \sigma_0^2$ OK.

Author(s)

Edzer Pebesma

References


http://www.gstat.org/

See Also

gstat, predict

Examples

library(sp)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
gridded(meuse.grid) = ~x+y
v = vgm(1, “Exp”, 300)
x1 = krigeTg(zinc~1, meuse, meuse.grid, v, lambda=1) # no transform
x2 = krige(zinc~1, meuse, meuse.grid, v)
summary(x2$var1.var-x1$var1Tg.var)
summary(x2$var1.pred-x1$var1Tg.pred)
lambda = -.25
m = fit.variogram(variogram((zinc^lambda-1)/lambda - 1, meuse), vgm(1, “Exp”, 300))
x = krigeTg(zinc~1, meuse, meuse.grid,m,lambda=-.25)
spplot(x[“var1Tg.pred”], col.regions=bp1.colors())
summary(meuse$x$zinc)
summary(x$var1Tg.pred)

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 krig or predict
- 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, krig; for examples see predict; 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

t 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.

Author(s)
The actual field data were collected by Ruud van Rijn and Mathieu Rikken; data compiled for R by Edzer Pebesma

References
http://www.gstat.org/

See Also
meuse.alt

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.all

Examples

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

ncp.grid  

Grid for the NCP, the Dutch part of the North Sea

Description

Gridded data for the NCP (Nederlands Continentaal Plat, the Dutch part of the North Sea), for a 5 km x 5 km grid; stored as data.frame.

Usage

data(ncp.grid)
ossfim

Format
This data frame contains the following columns:

- **x**: x-coordinate, UTM zone 31
- **y**: y-coordinate, UTM zone 31
- **depth**: sea water depth, m.
- **coast**: distance to the coast of the Netherlands, in km.
- **area**: identifier for administrative sub-areas

Author(s)
Dutch National Institute for Coastal and Marine Management (RIKZ); data compiled for R by Edzer Pebesma

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
- **spacings**: range of grid (data) spacings to be used
- **block.sizes**: range of block sizes to be used
- **model**: variogram model, output of vgm
- **nmax**: set the kriging neighbourhood size
- **debug**: 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 Pebesma

References


See Also

krige

Examples

```r
## Not run:
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)")

## End(Not run)
# 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.
```
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 external directory (see example below), shows an image of the soil map for the region

Author(s)

P.A. Burrough; compiled for R by Edzer Pebesma

References


Examples

data(oxford)
summary(oxford)
# open the following file with a jpg viewer:
system.file("external/oxford.jpg", package="gstat")

Description

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

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

data(pcb)

Format

This data frame contains the following columns:

- **year**: measurement year
- **x**: x-coordinate; UTM zone 31
- **y**: y-coordinate; UTM zone 31
- **coast**: distance to coast of the Netherlands, in km.
- **depth**: sea water depth, m.
- **PCB138**: PCB-138, measured on the sediment fraction smaller than 63 µ, in µg/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/


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, and possibly a fitted model*

**Description**

Creates a variogram plot

**Usage**

```r
## S3 method for class 'gstatVariogram'
plot(x, model = NULL, ylim, xlim, xlab = "distance", 
ylab = attr(x, "what"), panel = vgm.panel.xyplot, multipanel = TRUE, 
plot.numbers = FALSE, scales, ids = x$id, group.id = TRUE, skip, 
layout,...)
## S3 method for class 'variogramMap'
plot(x, np = FALSE, skip, threshold, ...)
## S3 method for class 'StVariogram'
plot(x, model = NULL, ..., col = bpy.colors(), xlab, ylab, 
map = TRUE, convertMonths = FALSE, as.table = TRUE, wireframe = FALSE, 
diff = FALSE, all = FALSE)
```

**Arguments**

- **x**: object obtained from the method `variogram`, possibly containing directional or cross variograms, space-time variograms and variogram model information
- **model**: in case of a single variogram: a variogram model, as obtained from `vgm` or `fit.variogram`, 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; in the spatio-temporal case, a single or a list of spatio-temporal models that will be plotted next to each other for visual comparison.
- **ylim**: numeric; vector of length 2, limits of the y-axis
- **xlim**: numeric; vector of length 2, limits of the x-axis
- **xlab**: character; x-axis label
- **ylab**: character; y-axis label
- **panel**: panel function
- **multipanel**: 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
- **plot.numbers**: logical or numeric; if TRUE, plot number of point pairs next to each plotted semivariance symbol, if FALSE these are omitted. If numeric, TRUE is assumed and the value is passed as the relative distance to be used between symbols and numeric text values (default 0.03).
- **scales**: optional argument that will be passed to `xyplot` in case of the plotting of variograms and cross variograms; use the value `list(relation = "same")` if y-axes need to share scales
plot.gstatVariogram

ids              ids of the data variables and variable pairs

ids              logical; control for directional multivariate variograms: if TRUE, panels di-

direction and colors indicate variables (ids), if FALSE panels divide vari-

ables/variable pairs and colors indicate direction

skip             logical; can be used to arrange panels, see xyplot

layout           integer vector; can be used to set panel layout: c(ncol,nrow)

np               logical (only for plotting variogram maps); if TRUE, plot number of point pairs,

if FALSE plot semivariances

threshold        semivariogram map values based on fewer point pairs than threshold will not be

plotted

...              any arguments that will be passed to the panel plotting functions (such as auto.key

in examples below)

col              colors to use

map              logical; if TRUE, plot space-time variogram map

convertMonths   logical; if TRUE, yearmon time lags will be unit converted and plotted as (integer)

months, and no longer match the numeric representation of yearmon, which

has years as unit

as.table        controls the plotting order for multiple panels, see xyplot for details.

wireframe       logical; if TRUE, produce a wireframe plot

diff            logical; if TRUE, plot difference between model and sample variogram; ignores

all.

all              logical; if TRUE, plot sample and model variogram(s) in single wireframes.

Details

Please note that in the spatio-temporal case the levelplot and wireframe plots use the spatial dis-

ances averaged for each time lag avgDist. For strongly varying spatial locations over time,

please check the distance columns dist and avgDist of the spatio-temporal sample variogram.

The lattice::cloud function is one option to plot irregular 3D data.

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 Pebesma
plot.pointPairs

Plot a point pairs, identified from a variogram cloud

Description

Plot a point pairs, identified from a variogram cloud

Usage

```r
## S3 method for class 'pointPairs'
plot(x, data, xcol = data$x, ycol = data$y, xlab = "x coordinate",
     ylab = "y coordinate", col.line = 2, line.pch = 0, main = "selected point pairs", ...)
```
**plot.pointPairs**

**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 `col.line`; the value passed to this argument will be used as plotting symbol (pch)
- `main` title of plot
- `...` arguments, further passed to `xyplot`

**Value**

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

**Author(s)**

Edzer Pebesma

**References**

http://www.gstat.org

**See Also**

`plot.variogramCloud`

**Examples**

```r
## The following requires interaction, and is therefore outcommented
#data(meuse)
#coordinates(meuse) = ~x+y
#vgm1 <- variogram(log(zinc)^-1, meuse, cloud = TRUE)
#pp <- plot(vgm1, 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

```r
## S3 method for class 'variogramCloud'
plot(x, identify = FALSE, digitize = FALSE, xlim, ylim, xlab, ylab,
     keep = FALSE, ...)
```

Arguments

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

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.
predict

Author(s)
Edzer Pebesma

References
http://www.gstat.org/

See Also
variogram, plot.gstatVariogram, plot.pointPairs, identify, locator

Examples
library(sp)
data(meuse)
coordinates(meuse) = ~x+y
plot(variogram(log(zinc)-1, 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)

pretect  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
## S3 method for class 'gstat'
predict(object, newdata, block = numeric(0), nsim = 0,
indicators = FALSE, BLUE = FALSE, debug.level = 1, mask,
na.action = na.pass, sps.ars = list(n = 500, type = "regular",
offset = c(.5, .5)), ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>object of class gstat, see gstat and krig</td>
</tr>
<tr>
<td>newdata</td>
<td>data frame with prediction/simulation locations; should contain columns with the independent variables (if present) and the coordinates with names as defined in locations; or: polygons, see below</td>
</tr>
</tbody>
</table>
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 indicators), following a single random path through the data.

indicators logical; only relevant if nsim 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 gstat 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 na.action; logical or numerical vector; pattern with valid values in newdata (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 newdata, 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.

sps.args when newdata is of class SpatialPolygons or SpatialPolygonsDataFrame this argument list gets passed to spsample to control the discretizing of polygons

... 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, 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.

If newdata is of class SpatialPolygons or SpatialPolygonsDataFrame, then the block average for each of the polygons or polygon sets is calculated, using spsample to discretize the polygon(s). Argument sps.args controls the parameters used for spsample. The "location" with respect to which neighbourhood selection is done is for each polygon the SpatialPolygons polygon label point; if you use local neighbourhoods you should check out where these points are—it may be well outside the polygon itself.

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 \texttt{nmax} in \texttt{krige} and \texttt{gstat}. The larger \texttt{nmax}, the better the approxi-
mation, the smaller \texttt{nmax}, the faster the simulation process. For selecting the nearest \texttt{nmax} data or
previously simulated points, \texttt{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 \texttt{nmax} nearest observed (or simulated)
values may cause spurious correlations when a regular path would be followed. Following a single
path through the locations, \texttt{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
\texttt{predict}, compared to 1000 calls, each for simulating a single field.

The random number generator used for generating simulations is the native random number gen-
erator of the environment (R, S); fixing randomness by setting the random number seed with
\texttt{set.seed()} works.

When mean coefficient are not supplied, they are generated as well from their conditional distri-
bution (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 \texttt{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
\texttt{gstat} C function \texttt{gstat\_predict} requires a table of size \texttt{n} * 12 bytes to pass the simulations back
to R, before it can free \texttt{n} * 4 bytes. Hopefully, R does not have to duplicate the remaining \texttt{n} * 8
bytes when the coordinates are added as columns, and when the resulting matrix is coerced to a
\texttt{data.frame}.

Useful values for \texttt{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 \texttt{debug\_level} are equal to positive, but cause the progress
counter to work.

For data with longitude/latitude coordinates (checked by \texttt{is\_projected}), \texttt{gstat} uses great circle
distances in km to compute spatial distances. The user should make sure that the semivariogram
model used is positive definite on a sphere.

**Value**

a data frame containing the coordinates of \texttt{newdata}, and columns of prediction and prediction
variance (in case of kriging) or the columns of the conditional Gaussian or indicator simulations
Author(s)

Edzer Pebesma

References

http://www.gstat.org/


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

See Also

gstat, kriged

Examples

# generate 5 conditional simulations
library(sp)
data(meuse)
coordinates(meuse) = ~x+y
v <- variogram(log(zinc)-1, meuse)
m <- fit.variogram(v, vgm(1, "Sph", 300, 1))
plot(v, model = m)
set.seed(131)
data(meuse.grid)
generated(meuse.grid) = ~x+y
sim <- krige(formula = log(zinc)-1, meuse, meuse.grid, model = m,
mm = 10, beta = 5.9, nsim = 5) # for speed -- 10 is too small!!
# show all 5 simulation
spplot(sim)

# calculate generalised least squares residuals w.r.t. constant trend:
g <- gstat(NULL, "log.zinc", log(zinc)-1, meuse, model = m)
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), meuse),
vgm(1, "Sph", 300, 1))
g <- gstat(NULL, "log.zinc", log(zinc)-sqrt(dist), meuse, model = m)
blue1 <- predict(g, 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)
```r
names(xy) <- c("x","y")
gridded(xy) = ~x+y
g.dummy <- gstat(formula = z~1, dummy = TRUE, beta = 0,
model = vgm(1,"Exp",nmax = 10) # for speed -- 10 is too small!!
yy <- predict(g.dummy, xy, nsim = 4)
# show one realisation:
spplot(yy[1])
# show all four:
spplot(yy)
```

---

### progress

*Get or set progress indicator*

**Description**

Get or set progress indicator

**Usage**

```r
get_gstat_progress()
set_gstat_progress(value)
```

**Arguments**

- `value` logical

**Value**

return the logical value indicating whether progress bars should be given

**Author(s)**

Edzer Pebesma

**Examples**

```r
set_gstat_progress(FALSE)
ge$t_gstat_progress()
```
**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**

```r
g <<- m <<- c <<- D <<- 0 <<- X <<- 0.5
```  

**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(s) of the variogram model
- `range`: numeric; range(s) of the variogram model
- `models`: character; variogram model(s) to be plotted
- `nugget`: numeric; nugget component(s) 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
- `...`: passed on to the call to `xyplot`
- `as.groups`: logical; if TRUE, different models are plotted with different lines in a single panel, else, in one panel per model

**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 Pebesma

References

http://www.gstat.org

See Also

vgm, variogramLine.

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)
# show a set of Exponential class models with different shape parameter:
show.vgms(kappa.range = c(.05, .1, .2, .5, 1, 1.5, 1.8, 1.9, 2), models = "Exc", max = 10)
# show a set of models with different shape parameter of M. Stein's representation of the Matern:
show.vgms(kappa.range = c(.01, .02, .05, .1, .2, .5, 1, 2, 5, 1000), models = "Ste", max = 2)

---

sic2004  

Spatial Interpolation Comparison 2004 data set: Natural Ambient Radioactivity

Description

The text below was copied from the original sic2004 event, which is no longer online available.

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 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.
**Author(s)**

Data: the German Federal Office for Radiation Protection (BfS), http://www.bfs.de/, data provided by Gregoire Dubois, R compilation by Edzer Pebesma.

**References**

https://wiki.52north.org/bin/view/AI_GEOSTATS/WebHome

**Examples**

```r
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)
```

---

**sic97**

*Spatial Interpolation Comparison 1997 data set: Swiss Rainfall*

---

**Description**

The text below is copied from the data item at ai-geostats, https://wiki.52north.org/bin/view/AI_GEOSTATS/WebHome

**Usage**

```r
data(sic97) #
```

**Format**

The data frames contain the following columns:

- **ID** this integer value is the number (unique value) of the monitoring station
- **rainfall** rainfall amount, in 10th of mm

**Note**

See the pdf that accompanies the original file for a description of the data. The .dxf file with the Swiss border is not included here.
spplot.vcov

Author(s)
Gregoire Dubois and others.

References
https://wiki.52north.org/bin/view/AGEOSTATS/WebHome

Examples
data(sic97)
image(demstd)
points(sic_full, pch=1)
points(sic_obs, pch=3)

spplot.vcov

Plot map matrix of prediction error variances and covariances

Description
Plot map matrix of prediction error variances and covariances

Usage
spplot.vcov(x, ...)

Arguments
x Object of class SpatialPixelsDataFrame or SpatialGridDataFrame, resulting from a kriging call with multiple variables (cokriging
... remaining arguments passed to spplot

Value
The plotted object, of class trellis; see spplot in package sp.

Author(s)
Edzer Pebesma
**Description**

The Südliche Tullnerfeld is a part of the Danube river basin in central Lower Austria and due to its homogeneous aquifer well suited for a model-oriented geostatistical analysis. It contains 36 official water quality measurement stations, which are irregularly spread over the region.

**Usage**

```r
data(tull)
```

**Format**

The data frames contain the following columns:

- `x` X location in meter
- `y` Y location in meter
- `S411` Station name
- `S429` Station name
- `S849` Station name
- `S854` Station name
- `S1502` Station name
- `S1584` Station name
- `S1591` Station name
- `S2046` Station name
- `S2047` Station name
- `S2048` Station name
- `S2049` Station name
- `S2051` Station name
- `S2052` Station name
- `S2053` Station name
- `S2054` Station name
- `S2055` Station name
- `S2057` Station name
- `S2058` Station name
- `S2059` Station name
- `S2060` Station name
- `S2061` Station name

**tull**  

| tull | NA |
S2062  Station name
S2063  Station name
S2064  Station name
S2065  Station name
S2066  Station name
S2067  Station name
S2070  Station name
S2071  Station name
S2072  Station name
S2128  Station name
S5319  Station name
S5320  Station name
S5321  Station name
S5322  Station name
S5323  Station name

Note

This data set was obtained on May 6, 2008 from http://www.ifas.jku.at/e5361/index_ger.html. The author of the book that uses it is found at: http://www.ifas.jku.at/e2571/e2604/index_ger.html

References

Werner G. Müller, Collecting Spatial Data, 3rd edition. Springer Verlag, Heidelberg, 2007

Examples

data(tull)

# TULLNREG = read.csv("TULLNREG.csv")

# I modified tulln36des.csv, such that the first line only contained: x,y
# resulting in row.names that reflect the station ID, as in
# tull36 = read.csv("tulln36des.csv")

# Chlorid92 was read & converted by:
#Chlorid92=read.csv("Chlorid92.csv")
#Chlorid92$Datum = as.POSIXct(strptime(Chlorid92$Datum, "%d.%m.%Y"))

summary(tull36)
summary(TULLNREG)
summary(Chlorid92)

# stack & join data to x,y,Date,Chloride form:
c1.st = stack(Chlorid92[-1])
names(cl.st) = c("Chloride", "Station")
c1.st$date = rep(Chloride92$Datum, length(names(Chloride92))-1)
c1.st$x = tull36[mvch(cl.st[, "Station"], row.names(tull36)), "x"]
c1.st$y = tull36[mvch(cl.st[, "Station"], row.names(tull36)), "y"]

# library(lattice)
# xyplot(Chloride~Date|Station, cl.st)
# xyplot(y~x|Date, cl.st, asp="iso", layout=c(16,11))
summary(cl.st)
plot(TULLNREG, pch=3, asp=1)
points(y~x, cl.st, add=TRUE, pch=16)

---

**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. In case spatio-temporal data is provided, the function `variogramST` is called with a different set of parameters.

**Usage**

```r
## S3 method for class 'gstat'
variogram(object, ...)   
## S3 method for class 'formula'
variogram(object, locations = coordinates(data), data, ...)    
## Default S3 method:
variogram(object, 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, ..., projected = TRUE, lambda = 1.0, verbose = FALSE, covariogram = FALSE, PR = FALSE, pseudo = -1)  
## S3 method for class 'gstatVariogram'
print(x, ...)  
## S3 method for class 'variogramCloud'
print(x, ...)  
```

**Arguments**

- **object**
  - object of class `gstat`: in this form, direct and cross (residual) variograms are calculated for all variables and variable pairs defined in object; in case of `variogram.formula`, formula defining the response vector and (possible) regressors, in case of absence of regressors use e.g. `z~1`; in case of `variogram.default`: list with for each variable the vector with responses (should not be called directly)
data frame where the names in formula are to be found

locations  
spatial data locations. For variogram.formula: a formula with only the coordinate variables in the right hand (explanatory variable) side e.g. \(~x+y\); see examples.

For variogram.default: list with coordinate matrices, each with the number of rows matching that of corresponding vectors in \(y\); 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 variogram.default (ignored)

\(X\)  
(optional) list with for each variable the matrix with regressors/covariates; the number of rows should match that of the corresponding element in \(y\), 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; as a default, the length of the diagonal of the box spanning the data is divided by three.

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): alpha=0 for direction North (increasing y), alpha=90 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; optional a vector of directions

tol.hor  
horizontal tolerance angle in degrees

tol.ver  
vertical tolerance angle in degrees

cressie  
logical; if TRUE, use Cressie’s robust variogram estimate; if FALSE use the classical method of moments variogram estimate

dX  
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 and \(x(s)\) the vector with regressors at location \(s\), and \(\|.\|\) the 2-norm. This allows pooled estimation of within-strata variograms (use a factor variable as regressor, and \(dX=0.5\)), or variograms of (near-)replicates in a linear model (addressing point pairs having similar values for regressors variables)

boundaries  
umerical vector with distance interval upper boundaries; values should be strictly increasing

cloud  
logical; if TRUE, calculate the semivariogram cloud

trend.beta  
vector with trend coefficients, in case they are known. By default, trend coefficients are estimated from the data.

debug.level  
integer; set gstat internal debug level

cross  
logical or character; if FALSE, no cross variograms are computed when object is of class gstat and has more than one variable; if TRUE, all direct and cross variograms are computed; if equal to “ST”, direct and cross variograms are computed for all pairs involving the first (non-time lagged) variable; if equal to “ONLY”, only cross variograms are computed (no direct variograms).
formula object of class variogram or variogramCloud to be printed
x object of class variogram or variogramCloud to be printed
grid grid parameters, if data are gridded (not to be called directly; this is filled automatically)
map logical; if TRUE, and cutoff and width are given, a variogram map is returned. This requires package sp. Alternatively, a map can be passed, of class SpatialDataFrameGrid (see sp docs)
g NULL or object of class gstat; may be used to pass settable parameters and/or variograms; see example
projected logical; if FALSE, data are assumed to be unprojected, meaning decimal longitude/latitude. For projected data, Euclidean distances are computed, for unprojected great circle distances (km). In variogram.formula or variogram.gstat, for data deriving from class Spatial, projection is detected automatically using is.projected
lambda test feature; not working (yet)
verbose logical; print some progress indication
pseudo integer; use pseudo cross variogram for computing time-lagged spatial variograms? -1: find out from coordinates – if they are equal then yes, else no; 0: no; 1: yes.
covariogram logical; compute covariogram instead of variogram?
PR logical; compute pairwise relative variogram (does NOT check whether variable is strictly positive)

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:

np the number of point pairs for this estimate; in case of a variogramCloud see below
dist the average distance of all point pairs considered for this estimate
gamma the actual sample variogram estimate
dir.hor the horizontal direction
dir.ver the vertical direction
id the combined id pair

If cloud is TRUE: an object of class variogramCloud, with the field np encoding the numbers of the point pair that contributed to a variogram cloud estimate, as follows. The first point is found by 1 + the integer division of np by the .BigInt attribute of the returned object, the second point by 1 + the remainder of that division. as.data.frame.variogramCloud returns no np field, but does the decoding into:

left for variogramCloud: data id (row number) of one of the data pair
right for variogramCloud: data id (row number) of the other data in the pair

In case of a spatio-temporal variogram is sought see variogramST for details.
Note

`variogram.default` should not be called by users directly, as it makes many assumptions about the organization of the data, that are not fully documented (but of course, can be understood from reading the source code of the other `variogram` methods).

Successfully setting `gridded()` <- `TRUE` may trigger a branch that will fail unless `dx` and `dy` are identical, and not merely similar to within machine epsilon.

Note

`variogram.line` is DEPRECATED; it is and was never meant as a variogram method, but works automatically as such by the R dispatch system. Use `variogramLine` instead.

Author(s)

Edzer Pebesma

References


http://www.gstat.org/


See Also

`print.gstatVariogram, plot.gstatVariogram, plot.variogramCloud`; for variogram models: `vgm`, to fit a variogram model to a sample variogram: `fit.variogram variogramST` for details on the spatio-temporal sample variogram.

Examples

```r
library(sp)
data(meuse)
# no trend:
coordinates(meuse) <- ~x+y
variogram(log(zinc)-1, meuse)
# residual variogram w.r.t. a linear trend:
variogram(log(zinc)-x+y, meuse)
# directional variogram:
variogram(log(zinc)-x+y, meuse, alpha=c(0,45,90,135))
variogram(log(zinc)-1, meuse, width=90, cutoff=1300)

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

```r
g = gstat(NULL, "log-zinc", log(zinc)-x+y, meuse, model=v.fit, set = set)
variogram(g)

if (require(rgdal)) {
  proj4string(meuse) = CRS("+init=epsg:28992")
  meuse.ll = spTransform(meuse, CRS("+proj=longlat +datum=WGS84 +ellps=WGS84"))
  # variogram of unprojected data, using great-circle distances, returning km as units
  variogram(log(zinc) - 1, meuse.ll)
}
```

### Description
Generates a semivariance values given a variogram model

### Usage
```r
variogramLine(object, maxdist, n = 200, min = 1.0e-6 * maxdist,
  dir = c(1,0,0), covariance = FALSE, ..., dist_vector, debug.level = 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)
- `covariance`: logical; if TRUE return covariance values, otherwise return semivariance values
- `...`: ignored
- `dist_vector`: numeric vector or matrix with distance values
- `debug.level`: gstat internal debug level

#### Value
A data frame of dimension (n x 2), with columns distance and gamma (semivariances or covariances), or in case `dist_vector` is a matrix, a conforming matrix with semivariance/covariance values is returned.

#### Note
`variogramLine` is used to generate data for plotting a variogram model.
variogramST

Calculate Spatio-Temporal Sample Variogram

Description

Calculates the sample variogram from spatio-temporal data.

Usage

variogramST(formula, locations, data, ..., tlags = 0:15, cutoff, width = cutoff/15, boundaries = seq(0, cutoff, width), progress = interactive(), pseudo = TRUE, assumeRegular=FALSE, na.omit=FALSE)

Arguments

formula formula, specifying the dependent variable.
locations A STFDF or STSDF containing the variable; kept for compatibility reasons with variogram, either locations or data must be provided.
data A STFDF, STSDF or STIDF containing the variable.
... any other arguments that will be passed to the underlying variogram function.
In case of using data of type STIDF, the argument tunit is recommended to set the temporal unit of the tlags. Additionally, twindow can be passed to control the temporal window used for temporal distance calculations. This builds on the property of xts being ordered and only the next twindow instances are considered. This avoids the need of huge temporal distance matrices. The default uses twice the number as the average difference goes into the temporal cutoff.
tlags integer; time lags to consider or in case data is of class STIDF the actual temporal boundaries with time unit given by tunit otherwise the same unit as diff on the index of the time slot will generate is assumed.
variogramST

- **cutoff**: spatial separation distance up to which point pairs are included in semivariance estimates; as a default, the length of the diagonal of the box spanning the data is divided by three.

- **width**: the width of subsequent distance intervals into which data point pairs are grouped for semivariance estimates, by default the cutoff is divided into 15 equal lags.

- **boundaries**: numerical vector with distance interval upper boundaries; values should be strictly increasing.

- **progress**: logical; if TRUE, show text progress bar.

- **pseudo**: integer; use pseudo cross variogram for computing time-lagged spatial variograms? -1: find out from coordinates – if they are equal then yes, else no; 0: no; 1: yes.

- **assumeRegular**: logical; whether the time series should be assumed regular. The first time step is assumed to be representative for the whole series. Note, that temporal lags are considered by index, and no check is made whether pairs actually have the desired separating distance.

- **na.omit**: shall all NA values in the spatio-temporal variogram be dropped? In case where complete rows or columns in the variogram consists of NA only, plot might produce a distorted picture.

**Value**

The spatio-temporal sample variogram contains besides the fields np, dist and gamma the spatio-temporal fields, timelag, spacelag and avgDist, the first of which indicates the time lag used, the second and third different spatial lags. spacelag is the midpoint in the spatial lag intervals as passed by the parameter boundaries, whereas avgDist is the average distance between the point pairs found in a distance interval over all temporal lags (i.e. the averages of the values dist per temporal lag.) To compute variograms for space lag $h$ and time lag $t$, the pseudo cross variogram $(Z_i(s)-Z_i+(s+h))^2$ is averaged over all time lagged observation sets $Z_i$ and $Z_i+t$ available (weighted by the number of pairs involved).

**Author(s)**

Edzer Pebesma, Benedikt Graeler

**References**


**See Also**

- `plot.StVariogram`, for variogram models: `vgmST`, to fit a spatio-temporal variogram model to a spatio-temporal sample variogram: `fit.StVariogram`
Examples

# The following spatio-temporal variogram has been calculated through
# vv = variogram_PM1_1, r5to10, width=20, cutoff = 200, tlags=0.5
# in the vignette "st".

data(vv)
str(vv)
plot(vv)

Description

Generates a surface of semivariance values given a spatio-temporal variogram model (one of separable, productSum, sumMetric, simpleSumMetric or metric)

Usage

variogramSurface(model, dist_grid, ...)

Arguments

model A spatio-temporal variogram model generated through vgmST or fit.StVariogram.
dist_grid A data.frame with two columns: spacelag and timelag.
... Additional arguments passed on to the underlying variogram functions.

Value

A data.frame with columns spacelag, timelag and gamma.

Author(s)

Benedikt Graeler

See Also

See variogramLine for the spatial version and fit.StVariogram for the estimation of spatio-temporal variograms.
vgm

Examples

separableModel <- vgmST("separable",
   space=vgm(0.86, "Exp", 476, 0.14),
   time =vgm(  1, "Exp",  3, 0),
   sill=113)

data(vv)

if(require(lattice)) {
   plot(vv, separableModel, wireframe=TRUE, all=TRUE)
}

# plotting of sample and model variogram
plot(vv, separableModel)

---

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(psill = NA, model, range = NA, nugget, add.to, anis, kappa = 0.5, ..., covtable, Err = 0)
## S3 method for class 'variogramModel'
print(x, ...)
as.vgm.variogramModel(m)

Arguments

- **psill**: (partial) sill of the variogram model component, or model: see Details
- **model**: model type, e.g. "Exp", "Sph", "Gau", "Mat". Calling vgm() without a model argument returns a data.frame with available models.
- **range**: range parameter of the variogram model component; in case of anisotropy: major range
- **kappa**: smoothness parameter for the Matern class of variogram models
- **nugget**: nugget component of the variogram (this basically adds a nugget component to the model); if missing, nugget component is omitted
- **add.to**: the variogram model to which we want to add a component (structure)
- **anis**: anisotropy parameters: see notes below
- **x**: a variogram model to print
arguments that will be passed to print, e.g. digits (see examples)

covtable

if model is Tab, instead of model parameters a one-dimensional covariance table can be passed here. See covtable.R in tests directory, and example below.

Err

numeric; if larger than zero, the measurement error variance component that will not be included to the kriging equations, i.e. kriging will now smooth the process Y instead of predict the measured Z, where Z = Y + e, and Err is the variance of e

Details

If only the first argument (psill) is given a character value indicating a model, as in vgm("Sph"), then this taken as a shorthand form of vgm(NA,"Sph",NA,NA), i.e. a spherical variogram with nugget and unknown parameter values; see examples below. Read fit.variogram to find out how NA variogram parameters are given initial values for a fitting a model, based on the sample variogram. Package automap gives further options for automated variogram modelling.

Value

If a single model is passed, an object of class variogramModel extending data.frame.

In case a vector of models is passed, an object of class variogramModelList which is a list of variogramModel objects.

When called without a model argument, a data.frame with available models is returned, having two columns: short (abbreviated names, to be used as model argument: "Exp", "Sph" etc) and long (with some description).

as.vgm.variomodel tries to convert an object of class variomodel (geoR) to vgm.

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(H, P). The first parameter, H, refers to the main axis direction: it is the angle for the principal direction of continuity (measured in degrees, clockwise from positive Y, i.e. North). The second parameter, P, 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 0.5 x 100 = 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. The anisotropy code was taken from GSLIB. Note that in http://www.gslib.com/sec_gb.html it is reported that this code has a bug. Quoting from this
site: “The third angle in all GSLIB programs operates in the opposite direction than specified in the GSLIB book. Explanation - The books says (pp27) the angle is measured clockwise when looking toward the origin (from the positive principal direction), but it should be counter-clockwise. This is a documentation error. Although rarely used, the correct specification of the third angle is critical if used.”

(Note that \( \text{anis} = c(p, s) \) is equivalent to \( \text{anis} = c(p, 0, 0, s, 1) \).)

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 (and so in gstat) is in error, but I have not corrected anything afterwards.

Author(s)

Edzer Pebesma

References

http://www.gstat.org/


For the validity of variogram models on the sphere, see Huang, Chunfeng, Haimeng Zhang, and Scott M. Robeson. On the validity of commonly used covariance and variogram functions on the sphere. Mathematical Geosciences 43.6 (2011): 721-733.

See Also

show.vgms to view the available models, fit.variogram, variogramLine, variogram for the sample variogram.

Examples

dq <- g(NA, "Sph")
dq( "Sph") # "Sph" is second argument: NO nugget in this case
dq(10, "Exp", 300)
x <- dq(10, "Exp", 300)
dq(10, "Nug", 0)
dq(10, "Exp", 300, 4.5)
dq(10, "Mat", 300, 4.5, kappa = 0.7)
dq( 5, "Exp", 300, add.to = dq(5, "Exp", 60, nugget = 2.5))
dq(10, "Exp", 300, anis = c(30, 0.5))
dx <- dq(10, "Exp", 300, anis = c(30, 10, 0, 0.5, 0.3))
# Matern variogram model:
dq(1, "Mat", 1, kappa=0.3)
x <- dq(0.39527463, "Sph", 953.8942, nugget = 0.06105141)
x
print(x, digits = 3);
# to see all components, do
print.data.frame(x)
vv=vgm(model = "Tab", covtable =
variogramLine(vgm(1, "Sph", 1), 1, n=1e4, min = 0, covariance = TRUE))
vgm(c("Mat", "Sph"))
vgm(, c("Mat", "Sph")) # no nugget

---

## vgm.panel.xyplot

panel functions for most of the variogram plots through lattice

### Description

Variogram plots contain symbols and lines; more control over them can be gained by writing your own panel functions, or extending the ones described here; see examples.

### Usage

```r
vgm.panel.xyplot(x, y, subscripts, type = "p", pch = plot.symbol$pch,
                   col, col.line = plot.line$col, col.symbol = plot.symbol$col,
                   lty = plot.line$lty, cex = plot.symbol$cex, ids, lwd = plot.line$lwd,
                   model = model, direction = direction, labels = shift, shift = shift, mode = mode, ...)
pseudo.pointPairs(x, y, type = "p", pch = plot.symbol$pch, col, col.line =
                   plot.line$col, col.symbol = plot.symbol$col, lty = plot.line$lty,
                   cex = plot.symbol$cex, lwd = plot.line$lwd, pairs = pairs,
                   line.pch = line.pch, ...)```

### Arguments

- `x`: x coordinates of points in this panel
- `y`: y coordinates of points in this panel
- `subscripts`: subscripts of points in this panel
- `type`: plot type: "l" for connected lines
- `pch`: plotting symbol
- `col`: symbol and line color (if set)
- `col.line`: line color
- `col.symbol`: symbol color
- `lty`: line type for variogram model
- `cex`: symbol size
- `ids`: gstat model ids
- `lwd`: line width
- `model`: variogram model
- `direction`: direction vector c(dir.horizontal, dir.ver)
- `labels`: labels to plot next to points
vgmArea

Description

Compute point-point, point-area or area-area variogram values from point model

Usage

vgmArea(x, y = x, vgm, ndiscr = 16, verbose = FALSE, covariance = TRUE)
vgmST

Constructing a spatio-temporal variogram

Arguments

- stModel: A string identifying the spatio-temporal variogram model (see details below). Only the string before an optional "_" is used to identify the model. This mechanism can be used to identify different fits of the same model (separable_A and separable_B will be interpreted as separable models, but carry different names).
- ...: unused, but ensure an exact match of the following parameters.
- space: A spatial variogram.
- time: A temporal variogram.

Value

semivariance or covariance matrix of dimension length(x) x length(y)

Author(s)

Edzer Pebesma

Examples

library(sp)
demo(meuse, ask = FALSE, echo = FALSE)
vgmArea(meuse[1:5], vgm = vgm(1, "Exp", 1000)) # point-point
vgmArea(meuse[1:5], meuse.area, vgm = vgm(1, "Exp", 1000)) # point-area
sill A joint spatio-temporal sill.
k The weighting of the product in the product-sum model.
nugget A joint spatio-temporal nugget.
stAni A spatio-temporal anisotropy; the number of space units equivalent to one time unit.
temporalUnits length one character vector, indicating the temporal units (like secs)

Details

The different implemented spatio-temporal variogram models have the following required parameters (see as well the example section)

separable: A variogram for space and time each and a joint spatio-temporal sill (variograms may have a separate nugget effect, but their joint sill will be 1) generating the call

\[ \text{vgmST("separable", space, time, sill)} \]

productSum: A variogram for space and time each, and the weighting of product \( k \) generating the call

\[ \text{vgmST("productSum", space, time, k)} \]

sumMetric: A variogram (potentially including a nugget effect) for space, time and joint each and a spatio-temporal anisotropy ratio \( \text{stAni} \) generating the call

\[ \text{vgmST("sumMetric", space, time, joint, stAni)} \]

simpleSumMetric: A variogram (without nugget effect) for space, time and joint each, a joint spatio-temporal nugget effect and a spatio-temporal anisotropy ratio \( \text{stAni} \) generating the call

\[ \text{vgmST("simpleSumMetric", space, time, joint, nugget, stAni)} \]

metric: A spatio-temporal joint variogram (potentially including a nugget effect) and \( \text{stAni} \) generating the call

\[ \text{vgmST("metric", joint, stAni)} \]

Value

Returns an S3 object of class \( \text{StVariogramModel} \).

Author(s)

Benedikt Graeler

See Also

\( \text{fit.StVariogram} \) for fitting, \( \text{variogramSurface} \) to plot the variogram and \( \text{extractParNames} \) to better understand the parameter structure of spatio-temporal variogram models.
Examples

# separable model: spatial and temporal sill will be ignored
# and kept constant at 1-nugget respectively. A joint sill is used.
separableModel <- vgmST("separable",
    space=vgm(0.9, "Exp", 147, 0.1),
    time=vgm(0.9, "Exp", 3.5, 0.1),
    sill=40)

# product sum model: spatial and temporal nugget will be ignored and kept
# constant at 0. Only a joint nugget is used.
prodSumModel <- vgmST("productSum",
    space=vgm(39, "Sph", 343, 0),
    time=vgm(36, "Exp", 3, 0),
    k=15)

# sum metric model: spatial, temporal and joint nugget will be estimated
sumMetricModel <- vgmST("sumMetric",
    space=vgm(6.9, "Lin", 200, 3.0),
    time=vgm(10.3, "Lin", 15, 3.5),
    joint=vgm(37.2, "Exp", 84, 11.7),
    stAni=77.7)

# simplified sumMetric model, only a overall nugget is fitted. The spatial,
# temporal and joint nuggets are set to 0.
simpleSumMetricModel <- vgmST("simpleSumMetric",
    space=vgm(20, "Lin", 150, 0),
    time=vgm(20, "Lin", 10, 0),
    joint=vgm(20, "Exp", 150, 0),
    nugget=1, stAni=15)

# metric model
metricModel <- vgmST("metric",
    joint=vgm(60, "Exp", 150, 10),
    stAni=60)

vv Precomputed variogram for PM10 in data set air

Description

Precomputed variogram for PM10 in data set air

Usage

data(vv)

Format

data set structure is explained in variogramST.
walker

Examples

```r
## Not run:
# obtained by:
library(spacetime)
library(gstat)
data(air)

if (!exists("rural"))
rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))
rr = rural[, "2005::2010"]
unsel = which(apply(as(rr, "xts"), 2, function(x) all(is.na(x))))
r5to10 = rr[-unsel,]
vv = variogram(PM10-1, r5to10, width=20, cutoff = 200, tlags=0:5)

## End(Not run)
```

---

**walker**

*Walker Lake sample and exhaustive data sets*

**Description**

This is the Walker Lake data sets (sample and exhaustive data set), used in Isaaks and Srivastava’s *Applied Geostatistics*.

**Usage**

data(walker)

**Format**

This data frame contains the following columns:

- **Id**: Identification Number
- **X**: X-location in meter
- **Y**: Y-location in meter
- **V**: V variable, concentration in ppm
- **U**: U variable, concentration in ppm
- **T**: T variable, indicator variable

**Note**

This data sets was obtained from the data sets on ai-geostats, [https://wiki.52north.org/bin/view/AI_GEOSTATS/WebHome](https://wiki.52north.org/bin/view/AI_GEOSTATS/WebHome)

**References**

*Applied Geostatistics* by Edward H. Isaaks, R. Mohan Srivastava; Oxford University Press.
Examples

library(sp)
data(walker)
summary(walker)
summary(walker.exh)

wind  
Ireland wind data, 1961-1978

Description

Daily average wind speeds for 1961-1978 at 12 synoptic meteorological stations in the Republic of Ireland (Haslett and raftery 1989). Wind speeds are in knots (1 knot = 0.5418 m/s), at each of the stations in the order given in Fig.4 of Haslett and Raftery (1989, see below)

Usage

data(wind)

Format

data.frame wind contains the following columns:

- **year**: year, minus 1900
- **month**: month (number) of the year
- **day**: day
- **RPT**: average wind speed in knots at station RPT
- **VAL**: average wind speed in knots at station VAL
- **ROS**: average wind speed in knots at station ROS
- **KIL**: average wind speed in knots at station KIL
- **SHA**: average wind speed in knots at station SHA
- **BIR**: average wind speed in knots at station BIR
- **DUB**: average wind speed in knots at station DUB
- **CLA**: average wind speed in knots at station CLA
- **MUL**: average wind speed in knots at station MUL
- **CLO**: average wind speed in knots at station CLO
- **BEL**: average wind speed in knots at station BEL
- **MAL**: average wind speed in knots at station MAL

data.frame wind.loc contains the following columns:

- **Station**: Station name
- **Code**: Station code
- **Latitude**: Latitude, in DMS, see examples below
- **Longitude**: Longitude, in DMS, see examples below
- **MeanWind**: mean wind for each station, metres per second
Note
This data set comes with the following message: “Be aware that the dataset is 532494 bytes long (that's over half a Megabyte). Please be sure you want the data before you request it.”
The data were obtained on Oct 12, 2008, from: http://www.stat.washington.edu/raftery/software.html
The data are also available from statlib.
Locations of 11 of the stations (ROS, Rosslare has been thrown out because it fits poorly the spatial correlations of the other stations) were obtained from: http://www.stat.washington.edu/research/reports/2005/tr475.pdf
Roslare lat/lon was obtained from google maps, location Roslare. The mean wind value for Roslare comes from Fig. 1 in the original paper.
Haslett and Raftery proposed to use a sqrt-transform to stabilize the variance.

Author(s)
Adrian Raftery; imported to R by Edzer Pebesma

References
These data were analyzed in detail in the following article:
and in many later papers on space-time analysis, for example:

Examples
data(wind)
summary(wind)
wind.loc
library(sp) # char2dms
wind.loc$x = as.numeric(char2dms(as.character(wind.loc[["Latitude"]])))
wind.loc$y = as.numeric(char2dms(as.character(wind.loc[["Longitude"]])))
coordinates(wind.loc) = ~x+y
# fig 1:
if (require(mapdata)) {
  map("worldHires", xlim = c(-11,11), ylim = c(-55.5,55.5))
  plot(wind.loc, add=TRUE, pch=16)
  text(coordinates(wind.loc), pos=1, label=wind.loc$Station)
}
wind$time = ISOdate(wind$year+1980, wind$month, wind$day)
# time series of e.g. Dublin data:
plot(DUB-time, wind, type='l', ylab = "windspeed (knots)", main = "Dublin")
# fig 2:
#wind = wind[(wind$month == 2 & wind$day == 29),]
wind$day = as.numeric(format(wind$time, '%j'))
```
windsqrt = sqrt(0.5148 * as.matrix(wind[4:15]))
Jday = 1:366
windsqrt = windsqrt - mean(windsqrt)
daymeans = sapply(split(windsqrt, wind$Jday), mean)
plot(daymeans ~ Jday)
lines(lowess(daymeans ~ Jday, f = 0.1))

# subtract the trend:
meanwind = lowess(daymeans ~ Jday, f = 0.1)$y[wind$Jday]
velocity = apply(windsqrt, 2, function(x) { x - meanwind })

# match order of columns in wind to Code in wind.loc:
pts = coordinates(wind.loc[match(names(wind[4:15]), wind.loc$Code),])

# fig 3, but not really yet...
dists = spdists(pts, longlat=TRUE)
corr = cor(velocity)
se = !(as.vector(dists) == 0)
plot(as.vector(corr[se]) ~ as.vector(dists[se]),
     xlim = c(0,500), ylim = c(.4, 1), xlab = "distance (km.)",
      ylab = "correlation")
# plots all points twice, ignores zero distance

# now really get fig 3:
ros = rownames(corr) == "ROS"
dists.nr = dists[!ros,ros]
corr.nr = corr[!ros,!ros]
se = !(as.vector(dists.nr) == 0)
plot(as.vector(corr.nr[se]) ~ as.vector(dists.nr[se]), pch = 3,
     xlim = c(0,500), ylim = c(.4, 1), xlab = "distance (km.)",
     ylab = "correlation")
# add outlier:
points(corr[ros,!ros] ~ dists[ros,!ros], pch=16, cex=.5)
xdiscr = 1:500
# add correlation model:
lines(xdiscr, .968 * exp(-.00134 * xdiscr))
```
Index

*Topic datasets
  coalash, 3
  DE_RB_2005, 4
  fulmar, 15
  jura, 24
  meuse.all, 38
  meuse.alt, 40
  ncp.grid, 40
  oxford, 43
  pcb, 44
  sic2004, 57
  sic97, 59
  tull, 61
  walker, 79
  wind, 80
*Topic dplot
  image, 22
  map.to.lev, 37
  plot.gstatVariogram, 46
  plot.pointPairs, 48
  plot.variogramCloud, 50
  show.vgms, 56
  spplot.vcov, 60
*Topic models
  fit.lmc, 7
  fit.StVariogram, 8
  fit.variogram, 11
  fit.variogram.gls, 13
  fit.variogram.reml, 14
  get.contr, 16
  gstat, 17
  hscat, 21
  krige, 26
  krige.cv, 30
  krigeST, 33
  krigeTg, 35
  osbfm, 41
  predict, 51
  progress, 55
  variogram, 63
  variogramLine, 67
  variogramST, 68
  variogramSurface, 70
  vgm, 71
  vgm-panel.xyplot, 74
  vgmArea, 75
  vgmST, 76
*Topic spatio-temporal
  variogramSurface, 70
  [.gstat (gstat), 17
  as.data.frame.variogramCloud, 65
  as.data.frame.variogramCloud
    (variogram), 63
  as.vgm.variomodel (vgm), 71
  Chlorid92 (tull), 61
  coalash, 3
  DE_RB_2005, 4
  demstd (sic97), 59
  diff, 68
  estiStAni, 5
  extractPar, 6
  extractParNames, 9, 10, 77
  extractParNames (extractPar), 6
  fit.lmc, 7
  fit.StVariogram, 7, 8, 34, 69, 70, 77
  fit.variogram, 8–10, 11, 13–15, 27, 31, 36, 46, 48, 66, 72, 73
  fit.variogram.gls, 13
  fit.variogram.reml, 11, 14
  fulmar, 15, 41
  get.contr, 16
  get_gstat_progress (progress), 55
  getGammas (variogramLine), 67
  gstat, 7, 17, 27–29, 31, 32, 34–37, 51, 53, 54
gstat.cv (krige.cv), 30

hscat, 21

identify, 51
idw (krige), 26
idw, formula, formula-method (krige), 26
idw, formula, Spatial-method (krige), 26
idw, formula, ST-method (krige), 26
idw-methods (krige), 26
idw.sp (krige), 26
image, 22
image.data.frame, 23, 38
image.default, 23

jura, 24
juragrid.dat (jura), 24

krige, 20, 26, 32, 36, 38, 42, 51, 53, 54
krige, formula, formula-method (krige), 26
krige, formula, NULL-method (krige), 26
krige, formula, Spatial-method (krige), 26
krige, formula, ST-method (krigeST), 33
krige-methods (krige), 26
krige.cv, 30
krige.cv, formula, formula-method (krige.cv), 30
krige.cv, formula, Spatial-method (krige.cv), 30
krige.cv.locations (krige.cv), 30
krige.cv.spatial (krige.cv), 30
krige.locations (krige), 26
krige.spatial (krige), 26
krige0, 34, 35
krige0 (krige), 26
krigeST, 10, 33
krigeSTTg (krigeST), 33
krigeTg, 35, 35

locator, 51
lpoints, 75

map.to.lev, 37
meuse.all, 38, 40
meuse.alt, 39, 40
ncp.grid, 16, 40, 45

optim, 9
ossfim, 41
oxford, 43

panel.pointPairs (vgm.panel.xyplot), 74
pcb, 44
plot.gstatVariogram, 46, 50, 51, 66, 68, 75
plot.pointPairs, 48, 51
plot.StVariogram, 69
plot.StVariogram (plot.gstatVariogram), 46
plot.variogramCloud, 49, 50, 66
plot.variogramMap
(plot.gstatVariogram), 46
predict, 16, 17, 20, 27–29, 31, 32, 35–38, 51, 53
prediction.dat (jura), 24
print.gstat (gstat), 17
print.gstatVariogram, 66
print.gstatVariogram (variogram), 63
print.variogramCloud (variogram), 63
print.variogramModel (vgm), 71
progress, 55

set_gstat_progress (progress), 55
show.vgms, 56, 73
sic.grid (sic2004), 57
sic.pred (sic2004), 57
sic.test (sic2004), 57
sic.train (sic2004), 57
sic.val (sic2004), 57
sic2004, 57
sic97, 59
sic_full (sic97), 59
sic_obs (sic97), 59
SpatialPoints, 76
SpatialPolygons, 52, 76
SpatialPolygonsDataFrame, 52
spplot.vcov, 60
spsample, 52, 76
ST, 34
STDF, 68
STIDF, 68
STSDF, 68

transect.dat (jura), 24
tull, 61
tull36 (tull), 61
TULLNREG (tull), 61
INDEX

validation.dat (jura), 24
variogram, 7, 8, 11, 12, 22, 46, 48, 51, 63, 68, 73
variogram.default, 64
variogramLine, 48, 57, 67, 70, 73
variogramST, 9, 63, 65, 66, 68, 78
variogramSurface, 34, 70, 77
vgm, 5, 7, 8, 11, 12, 18, 27, 31, 36, 46, 48, 57, 66, 71, 75, 76
vgm.panel.xyplot, 74
vgmArea, 75
vgmST, 6, 7, 9, 34, 69, 70, 76
vv, 78
walker, 79
wind, 80

xyplot, 47
xyz2img, 23
xyz2img (image), 22