

# Package ‘gstat’

March 14, 2022

**Version** 2.0-9

**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 variogram map plotting utility functions; supports sf and stars.

**Depends** R (>= 2.10)

**Imports** utils, stats, graphics, methods, lattice, sp (>= 0.9-72), zoo, spacetime (>= 1.0-0), FNN

**Suggests** fields, maps, mapdata, maptools, rgdal (>= 0.5.2), rgeos, sf (>= 0.7-2), stars (>= 0.3-0), xts, raster, future, future.apply

**License** GPL (>= 2.0)

**URL** <https://github.com/r-spatial/gstat/>

**Encoding** UTF-8

**BugReports** <https://github.com/r-spatial/gstat/issues/>

**NeedsCompilation** yes

**RoxygenNote** 6.1.1

**Author** Edzer Pebesma [aut, cre] (<<https://orcid.org/0000-0001-8049-7069>>),  
Benedikt Graeler [aut]

**Maintainer** Edzer Pebesma <edzer.pebesma@uni-muenster.de>

**Repository** CRAN

**Date/Publication** 2022-03-14 15:20:05 UTC

## R topics documented:

coalash	3
DE_RB_2005	4
estiStAni	5
extractPar	6
fit.lmc	7

fit.StVariogram . . . . .	8
fit.variogram . . . . .	11
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
krigeSimCE . . . . .	33
krigeST . . . . .	34
krigeSTSimTB . . . . .	36
krigeTg . . . . .	38
map.to.lev . . . . .	40
meuse.all . . . . .	40
meuse.alt . . . . .	42
ncp.grid . . . . .	43
ossfim . . . . .	44
oxford . . . . .	45
pcb . . . . .	47
plot.gstatVariogram . . . . .	48
plot.pointPairs . . . . .	50
plot.variogramCloud . . . . .	52
predict . . . . .	53
progress . . . . .	57
show.vgms . . . . .	58
sic2004 . . . . .	59
sic97 . . . . .	61
spplot.vcov . . . . .	62
tull . . . . .	63
variogram . . . . .	65
variogramLine . . . . .	69
variogramST . . . . .	70
variogramSurface . . . . .	72
vgm . . . . .	73
vgm.panel.xyplot . . . . .	76
vgmArea . . . . .	78
vgmAreaST . . . . .	79
vgmST . . . . .	80
vv . . . . .	82
walker . . . . .	83
wind . . . . .	84

---

`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://homepage.divms.uiowa.edu/~dzimmer/spatialstats/>, Dale Zimmerman's course page

**References**

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

Gomez, M. and Hazen, K. (1970). Evaluating sulfur and ash distribution in coal seams by statistical response surface regression analysis. U.S. Bureau of Mines Report RI 7377.

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 2350 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 10 ... .. .. ..\$ station\_start\_date : Factor w/ 2409 levels "1900-01-01","1951-04-01",...: 152 1184 1577 1132 744 328 1202 1555 1148 407 ... .. .. ..\$ station\_end\_date : Factor w/ 864 levels "", "1975-02-06",...: 1 1 1 579 1 1 1 1 1 1 ... .. .. ..\$ type\_of\_station : Factor w/ 5 levels "", "Background",...: 2 2 2 2 2 2 2 2 2 2 ... .. .. ..\$ station\_type\_of\_area : Factor w/ 4 levels "rural","suburban",...: 1 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 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, 1:2] 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**

```
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**

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

**Arguments**

empVgm	An empirical spatio-temporal variogram.
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 <code>vgm</code> . The model is optimised based on the pure spatial values in empVgm.
temporalVgm	A temporal variogram definition from the call to <code>vgm</code> . 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.

**vgm** 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 `spatialVgm`. The starting value of `stAni` is the mean of the interval parameter (see `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

```
data(vv)

estiStAni(vv, c(10, 150))
estiStAni(vv, c(10, 150), "vgm", vgm(80, "Sph", 120, 20))
```

---

extractPar	<i>Extracting parameters and their names from a spatio-temporal variogram model</i>
------------	---

---

### 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

```
extractPar(model)
extractParNames(model)
```

### Arguments

`model` a spatio-temporal variogram model from `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**

```
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**

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

**Arguments**

v	multivariable sample variogram, output of <a href="#">variogram</a>
g	gstat object, output of <a href="#">gstat</a>
model	variogram model, output of <a href="#">vgm</a> ; 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\(cokriging\)](#)

---

`fit.StVariogram`

*Fit a spatio-temporal sample variogram to a sample variogram*

---

**Description**

Fits a spatio-temporal variogram of a given type to spatio-temporal sample variogram.

**Usage**

```
fit.StVariogram(object, model, ..., method = "L-BFGS-B",
lower, upper, fit.method = 6, stAni=NA, wles)
```



**Arguments**

object	The spatio-temporal sample variogram. Typically output from <code>variogramST</code>
model	The desired spatio-temporal model defined through <code>vgmST</code> .
...	further arguments passed to <code>optim</code> . <code>extractParNames</code> provides the parameter structure of spatio-temporal variogram models that help to provide sensible upper and lower limits.
lower	Lower limits used by <code>optim</code> . If missing, the smallest well defined values are used (mostly near 0).
upper	Upper limits used by <code>optim</code> . If missing, the largest well defined values are used (mostly Inf).
method	fit method, pass to <code>optim</code>
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 <code>fit.variogram</code> ). 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, <code>wles = TRUE</code> corresponds to <code>fit.method = 1</code> and <code>wles = FALSE</code> corresponds to <code>fit.method = 6</code> .

**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:  $N_j / (h_j^2 + \text{stAni}^2 \cdot u_j^2)$

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:  $1/(h_j^2 + \text{stAni}^2 \cdot u_j^2)$

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(10,0,0.01,0,1),
```

```

                                upper=c(500,1,20,1,200))
plot(vv, separableModel)

## End(Not run) # dontrun

```

---

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

```
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 <a href="#">variogram</a>
model	variogram model, output of <a href="#">vgm</a> ; 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 <code>gstat</code> . The default method uses weights $N_h/h^2$ with $N_h$ the number of point pairs and $h$ the distance. This criterion is not supported by theory, but by practice. For other values of <code>fit.method</code> , see details.
debug.level	integer; set <code>gstat</code> 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/>

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

**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"))
```

```
# optimize the value of kappa in a Matern model, using ugly <- side effect:
f = function(x) attr(m.fit <- fit.variogram(vgm1, vgm("Mat",nugget=NA,kappa=x)),"SSErr")
optimize(f, c(0.1, 5))
plot(vgm1, m.fit)
# best fit from the (0.3, 0.4, 0.5. ... , 5) sequence:
(m <- fit.variogram(vgm1, vgm("Mat"), fit.kappa = TRUE))
attr(m, "SSErr")
```

---

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

```
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. $z \sim 1$
data	object of class <code>Spatial</code>
model	variogram model to be fitted, output of <code>vgm</code>
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: $\text{mean}(y)/2$ , sill: $\text{mean}(y)/2$ , range $\text{median}(h0)/4$ 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`.

**Author(s)**

Edzer Pebesma

**References**

Mueller, W.G., 1999: Least-squares fitting from the variogram cloud. *Statistics & Probability Letters*, 43, 93-98.

Mueller, W.G., 2007: *Collecting Spatial Data*. Springer, Heidelberg.

**See Also**

[fit.variogram](#),

**Examples**

```
library(sp)
data(meuse)
coordinates(meuse) = ~x+y
## Not run:
fit.variogram.gls(log(zinc)~1, meuse[1:40,], vgm(1, "Sph", 900,1))

## End(Not run)
```

---

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

**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**

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

**See Also**

[fit.variogram](#),

**Examples**

```
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**

```
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)

**See Also**

[ncp.grid](#)

E.J. Pebesma, R.N.M. Duin, P.A. Burrough, 2005. Mapping Sea Bird Densities over the North Sea: Spatially Aggregated Estimates and Temporal Changes. *Environmetrics* 16, (6), p 573-587.

**Examples**

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

## End(Not run)
```

---

get.contr

*Calculate contrasts from multivariable predictions*

---

**Description**

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

**Usage**

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

**Arguments**

data	data frame, output of <a href="#">predict</a>
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  $\hat{y}$ , 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  $\text{Var}(C)=X'V X$ .

**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](#)

---

gstat

*Create gstat objects, or subset it*

---

**Description**

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

**Usage**

```
gstat(g, id, formula, locations, data, model = NULL, beta,
      nmax = Inf, nmin = 0, 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**

<code>g</code>	gstat object to append to; if missing, a new gstat object is created
<code>id</code>	identifier of new variable; if missing, <code>varn</code> is used with <code>n</code> the number for this variable. If a cross variogram is entered, <code>id</code> should be a vector with the two <code>id</code> values, e.g. <code>c("zn", "cd")</code> , further only supplying arguments <code>g</code> and <code>model</code> . It is advisable not to use expressions, such as <code>log(zinc)</code> , as identifiers, as this may lead to complications later on.
<code>formula</code>	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name <code>z</code> , for ordinary and simple kriging use the formula <code>z~1</code> ; for simple kriging also define <code>beta</code> (see below); for universal kriging, suppose <code>z</code> is linearly dependent on <code>x</code> and <code>y</code> , use the formula <code>z~x+y</code>
<code>locations</code>	formula with only independent variables that define the spatial data locations (coordinates), e.g. <code>~x+y</code> ; if data has a <code>coordinates</code> method to extract its coordinates this argument can be ignored (see package <code>sp</code> for classes for point or grid data).
<code>data</code>	data frame; contains the dependent variable, independent variables, and locations.
<code>model</code>	variogram model for this <code>id</code> ; defined by a call to <code>vgm</code> ; see argument <code>id</code> to see how cross variograms are entered
<code>beta</code>	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
<code>nmax</code>	for local kriging: the number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations
<code>nmin</code>	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated, unless <code>force==TRUE</code> ; see <code>maxdist</code>
<code>omax</code>	maximum number of observations to select per octant (3D) or quadrant (2D); only relevant if <code>maxdist</code> has been defined as well
<code>maxdist</code>	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
<code>force</code>	for local kriging, force neighbourhood selection: in case <code>nmin</code> is given, search beyond <code>maxdist</code> until <code>nmin</code> neighbours are found. A missing value is returned if this is not possible.
<code>dummy</code>	logical; if <code>TRUE</code> , consider this data as a dummy variable (only necessary for unconditional simulation)
<code>set</code>	named list with optional parameters to be passed to <code>gstat</code> (only <code>set</code> commands of <code>gstat</code> are allowed, and not all of them may be relevant; see the manual for <code>gstat</code> stand-alone, URL below )

x	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 <code>c("id1", 1, "id2", 2)</code> . 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.: <code>var1.var2</code> )
)	
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. Sub-setted 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**

<http://www.gstat.org/> Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences, 30: 683-691.

for kriging with known, varying measurement errors (weights), see e.g. Delhomme, J.P. Kriging in the hydrosiences. 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:
```

```

plot(variogram(g), model = g$model, main = "models fitted by eye")
# see also demo(cokriging) for a more efficient approach
g["ln.zinc"]
g["ln.lead"]
g[c("ln.zinc", "ln.lead")]
g[1]
g[2]

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

# local universal kriging
gmeuse <- gstat(id = "log_zinc", formula = log(zinc)~sqrt(dist), data = meuse)
# variogram of residuals
vmeuse.res <- fit.variogram(variogram(gmeuse), vgm(1, "Exp", 300, 1))
# prediction from local neighbourhoods within radius of 170 m or at least 10 points
gmeuse <- gstat(id = "log_zinc", formula = log(zinc)~sqrt(dist),
data = meuse, maxdist=170, nmin=10, force=TRUE, model=vmeuse.res)
predmeuse <- predict(gmeuse, meuse.grid)
spplot(predmeuse)

```

---

hscat

*Produce h-scatterplot*


---

### Description

Produces 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 <a href="#">variogram</a> ; 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/>

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

**Examples**

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

---

image

*Image Gridded Coordinates in Data Frame*

---

**Description**

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

**Usage**

```
## 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)
```

**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
...	arguments, passed to image.default
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**

```
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 = "+")
```

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**

```
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<sup>-1</sup> topsoil

**Co** mg cobalt kg<sup>-1</sup> topsoil

**Cr** mg chromium kg<sup>-1</sup> topsoil

**Cu** mg copper kg<sup>-1</sup> topsoil

**Ni** mg nickel kg<sup>-1</sup> topsoil

**Pb** mg lead kg<sup>-1</sup> topsoil

**Zn** mg zinc kg<sup>-1</sup> 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):

Rock Types: 1: Argovian, 2: Kimmeridgian, 3: Sequanian, 4: Portlandian, 5: Quaternary.



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

Goovaerts, P. 1997. Geostatistics for Natural Resources Evaluation. Oxford Univ. Press, New-York, 483 p. Appendix C describes (and gives) the Jura data set.

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
## Not run:
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")
  }
}

## End(Not run)

```

---

krige

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

---

### Description

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

**Usage**

```

krige(formula, locations, ...)
krige.locations(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.locations(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 $z \sim 1$ ; for simple kriging also define $\beta$ (see below); for universal kriging, suppose $z$ is linearly dependent on $x$ and $y$ , use the formula $z \sim x + y$
locations	object of class <code>Spatial</code> or <code>sf</code> , or (deprecated) formula defines the spatial data locations (coordinates) such as $\sim x + y$
data	data frame: should contain the dependent variable, independent variables, and coordinates, should be missing if locations contains data.
newdata	object of class <code>Spatial</code> , <code>sf</code> or <code>stars</code> with prediction/simulation locations; should contain attributes with the independent variables (if present).
model	variogram model of dependent variable (or its residuals), defined by a call to <code>vgm</code> or <code>fit.variogram</code> ; for <code>krige0</code> 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
nmin	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
omax	see <code>gstat</code>

<code>maxdist</code>	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
<code>block</code>	block size; a vector with 1, 2 or 3 values containing the size of a rectangular in x-, y- and z-dimension respectively (0 if not set), or a data frame with 1, 2 or 3 columns, containing the points that discretize the block in the x-, y- and z-dimension to define irregular blocks relative to (0,0) or (0,0,0)—see also the details section of <code>predict</code> . By default, predictions or simulations refer to the support of the data values.
<code>nsim</code>	integer; if set to a non-zero value, conditional simulation is used instead of kriging interpolation. For this, sequential Gaussian or indicator simulation is used (depending on the value of indicators), following a single random path through the data.
<code>indicators</code>	logical, only relevant if <code>nsim</code> is non-zero; if TRUE, use indicator simulation; else use Gaussian simulation
<code>na.action</code>	function determining what should be done with missing values in 'newdata'. The default is to predict 'NA'. Missing values in coordinates and predictors are both dealt with.
<code>debug.level</code>	debug level, passed to <code>predict</code> ; use -1 to see progress in percentage, and 0 to suppress all printed information
<code>...</code>	for <code>krige</code> : arguments that will be passed to <code>gstat</code> ; for <code>krige0</code> : arguments that will be passed to <code>model</code>
<code>idp</code>	numeric; specify the inverse distance weighting power
<code>y</code>	matrix; to <code>krige</code> multiple fields in a single step, pass data as columns of matrix <code>y</code> . This will ignore the value of the response in formula.
<code>computeVar</code>	logical; if TRUE, prediction variances will be returned
<code>fullCovariance</code>	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`.

## 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

`krige0` and `idw0` 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

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

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

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

## See Also

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

## Examples

```
library(sp)
data(meuse)
coordinates(meuse) = ~x+y
data(meuse.grid)
gridded(meuse.grid) = ~x+y
m <- vgm(.59, "Sph", 874, .04)
# ordinary kriging:
x <- krige(log(zinc)~1, meuse, meuse.grid, model = m)
splot(x["var1.pred"], main = "ordinary kriging predictions")
splot(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")

# krige0, 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) }
# krige 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 <code>gstat</code> ; see function <a href="#">gstat</a>
nfold	integer; if larger than 1, then apply n-fold cross validation; if <code>nfold</code> equals <code>nrow(data)</code> (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 <code>nrow(data)</code> 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 <a href="#">predict</a> in case of <code>gstat.cv</code> , or to <a href="#">gstat</a> in case of <code>krige.cv</code>
formula	formula that defines the dependent variable as a linear model of independent variables; suppose the dependent variable has name <code>z</code> , for ordinary and simple kriging use the formula <code>z~1</code> ; for simple kriging also define <code>beta</code> (see below); for universal kriging, suppose <code>z</code> is linearly dependent on <code>x</code> and <code>y</code> , use the formula <code>z~x+y</code>
locations	data object deriving from class <code>Spatial</code> or <code>sf</code>
data	data frame (deprecated); should contain the dependent variable, independent variables, and coordinates; only to be provided if <code>locations</code> is a formula
model	variogram model of dependent variable (or its residuals), defined by a call to <a href="#">vgm</a> or <a href="#">fit.variogram</a>
beta	only for simple kriging (and simulation based on simple kriging); vector with the trend coefficients (including intercept); if no independent variables are defined the model only contains an intercept and this should be the simple kriging mean
nmax	for local kriging: the number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations. By default, all observations are used
nmin	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
maxdist	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
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")

# 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)
```

---

krigeSimCE

*Simulation based on circulant embedding*


---

## Description

Simulating a conditional/unconditional Gaussian random field via kriging and circulant embedding

## Usage

```
krigeSimCE(formula, data, newdata, model, n = 1, ext = 2)
```

## Arguments

formula	the formula of the kriging predictor
data	spatial data frame that conditions the simulation
newdata	locations in space where the Gaussian random field shall be simulated
model	a vgm model that defines the spatial covariance structure
n	number of simulations
ext	extension factor of the circulant embedding, defaults to 2

## Value

A spatial data frame as defined in newdata with n simulations.

## Author(s)

Benedikt Graeler

## References

Davies, Tilman M., and David Bryant: "On circulant embedding for Gaussian random fields in R." *Journal of Statistical Software* 55.9 (2013): 1-21. See i.e. the supplementary files at (retrieved 2018-05-25): <https://www.jstatsoft.org/index.php/jss/article/downloadSuppFile/v055i09/v55i09.R>

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

```
# see demo('circEmbeddingMeuse')
```

---

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, beta, y, ...,
        nmax = Inf, stAni = NULL,
        computeVar = FALSE, fullCovariance = FALSE,
        bufferNmax=2, progress=TRUE)
krigeSTG(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 <a href="#">vgmST</a> - see below or the function <a href="#">vgmAreaST</a> for area-to-point kriging. For the general kriging case: a 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 <a href="#">fit.StVariogram</a> or <a href="#">variogramSurface</a> 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 krige 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 used for instance to pass the model into vgmAreaST for area-to-point kriging
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

### References

Spatio-Temporal Geostatistics using `gstat`. Benedikt Graeler, Edzer Pebesma, Gerard Heuvelink. The R Journal, accepted.

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

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

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

**See Also**

[krige0](#), [gstat](#), [predict](#), [krigeTg](#)

**Examples**

```
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)
suppressWarnings(proj4string(stations) <- CRS(proj4string(stations)))
rural = STFDF(stations, dates, data.frame(PM10 = as.vector(air)))

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

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,
                    modelList=sumMetricVgm)
gridded(DE_kriged@sp) <- TRUE
stplot(DE_kriged)
```

---

krigeSTSimTB

*conditional/unconditional spatio-temporal simulation*

---

**Description**

conditional/unconditional spatio-temporal simulation based on turning bands

**Usage**

```
krigeSTSimTB(formula, data, newdata, modelList, nsim, progress = TRUE,
             nLyrs = 500, tGrid = NULL, sGrid = NULL, ceExt = 2, nmax = Inf)
```

**Arguments**

formula	the formula of the kriging predictor
data	conditioning data

<code>newdata</code>	locations in space and time where the simulation is carried out
<code>modellist</code>	the spatio-temporal variogram (from <a href="#">vgmST</a> ) defining the spatio-temporal covariance structure of the simulated Gaussian random field
<code>nsim</code>	number of simulations
<code>progress</code>	boolean; whether the progress should be shown in progress bar
<code>nLyrs</code>	number of layers used in the turning bands approach (default = 500)
<code>tGrid</code>	optional explicit temporal gridding that shall be used
<code>sGrid</code>	optional explicit spatial gridding that shall be used
<code>ceExt</code>	expansion in the circulant embedding, defaults to 2
<code>nmax</code>	number of nearest neighbours that shall be used, defaults to 'Inf' meaning all available points are used

**Value**

a spatio-temporal data frame with `nSim` simulations

**Author(s)**

Benedikt Graeler

**References**

Turning bands

Lantuejoul, C. (2002) Geostatistical Simulation: Models and Algorithms. Springer.

Matheron, G. (1973). The intrinsic random functions and their applications. *Adv. Appl. Probab.*, 5, 439-468.

Strokorb, K., Ballani, F., and Schlather, M. (2014) Tail correlation functions of max-stable processes: Construction principles, recovery and diversity of some mixing max-stable processes with identical TCF. *Extremes*, Submitted.

Turning layers

Schlather, M. (2011) Construction of covariance functions and unconditional simulation of random fields. In Porcu, E., Montero, J.M. and Schlather, M., *Space-Time Processes and Challenges Related to Environmental Problems*. New York: Springer.

**See Also**

[krigeSimCE](#)

**Examples**

```
# see demo('circEmbeddingMeuse')
```

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 <code>Spatial</code> , with observations
newdata	Spatial object with prediction/simulation locations; the coordinates should have names as defined in <code>locations</code>
model	variogram model of the TRANSFORMED dependent variable, see <a href="#">vgm</a> , or <a href="#">fit.variogram</a>
nmax	for local kriging: the number of nearest observations that should be used for a kriging prediction or simulation, where nearest is defined in terms of the space of the spatial locations. By default, all observations are used
nmin	for local kriging: if the number of nearest observations within distance <code>maxdist</code> is less than <code>nmin</code> , a missing value will be generated; see <code>maxdist</code>
maxdist	for local kriging: only observations within a distance of <code>maxdist</code> from the prediction location are used for prediction or simulation; if combined with <code>nmax</code> , both criteria apply
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 <a href="#">predict</a> ; use -1 to see progress in percentage, and 0 to suppress all printed information
...	other arguments that will be passed to <a href="#">gstat</a>

## Details

Function `krigeTg` uses `transGaussian` kriging as explained in <http://www.math.umd.edu/~bnk/bak/Splus/kriging.html>.

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  $\mu$ .

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

## 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  $\phi'(\hat{\mu}, \lambda)^2 \sigma_{OK}^2$

## Author(s)

Edzer Pebesma

## References

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

<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 = -0.25
m = fit.variogram(variogram((zinc^lambda-1)/lambda ~ 1,meuse), vgm(1, "Exp", 300))
x = krigeTg(zinc~1,meuse,meuse.grid,m,lambda=-.25)
splot(x["var1TG.pred"], col.regions=bpy.colors())
summary(meuse$zinc)
summary(x$var1TG.pred)
```

---

map.to.lev	<i>rearrange data frame for plotting with levelplot</i>
------------	---

---

**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 <a href="#">krige</a> or <a href="#">predict</a>
xcol	x-coordinate column number
ycol	y-coordinate column number
zcol	z-coordinate column number range
ns	names of the set of z-columns to be viewed

**Value**

data frame with the following elements:

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

**See Also**

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

---

meuse.all	<i>Meuse river data set – original, full data set</i>
-----------	---

---

**Description**

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

**Usage**

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



**Format**

This data frame contains the following columns:

**sample** sample number

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

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

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

**copper** topsoil copper concentration, ppm.

**lead** topsoil lead concentration, ppm.

**zinc** topsoil zinc concentration, ppm.

**elev** relative elevation

**om** organic matter, as percentage

**ffreq** flooding frequency class

**soil** soil type

**lime** lime class

**landuse** landuse class

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

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

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

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

**Note**

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

**Author(s)**

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

**References**

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

<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)
```

**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**

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

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

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

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

```
## 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.
```

---

oxford

*Oxford soil samples*

---

**Description**

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

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

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

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

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

**Usage**

data(oxford)

**Format**

This data frame contains the following columns:

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

**Note**

oxford.jpg, in the gstat package 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**

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

**Examples**

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

---

pcb	<i>PCB138 measurements in sediment at the NCP, the Dutch part of the North Sea</i>
-----	--

---

### Description

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  $63 \mu$ , in  $\mu g/kg$  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

Pebesma, E. J., & Duin, R. N. M. (2005). Spatial patterns of temporal change in North Sea sediment quality on different spatial scales. In P. Renard, H. Demougeot-Renard & R. Froidevaux (Eds.), *Geostatistics for Environmental Applications: Proceedings of the Fifth European Conference on Geostatistics for Environmental Applications* (pp. 367-378): Springer.

### 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**

```
## 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**

<code>x</code>	object obtained from the method <a href="#">variogram</a> , possibly containing directional or cross variograms, space-time variograms and variogram model information
<code>model</code>	in case of a single variogram: a variogram model, as obtained from <a href="#">vgm</a> or <a href="#">fit.variogram</a> , 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.
<code>ylim</code>	numeric; vector of length 2, limits of the y-axis
<code>xlim</code>	numeric; vector of length 2, limits of the x-axis
<code>xlab</code>	character; x-axis label
<code>ylab</code>	character; y-axis label
<code>panel</code>	panel function
<code>multipanel</code>	logical; if TRUE, directional variograms are plotted in different panels, if FALSE, directional variograms are plotted in the same graph, using color, colored lines and symbols to distinguish them



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 <code>xyplot</code> in case of the plotting of variograms and cross variograms; use the value <code>list(relation = "same")</code> if y-axes need to share scales
ids	ids of the data variables and variable pairs
group.id	logical; control for directional multivariate variograms: if TRUE, panels divide direction and colors indicate variables (ids), if FALSE panels divide variables/variable pairs and colors indicate direction
skip	logical; can be used to arrange panels, see <code>xyplot</code>
layout	integer vector; can be used to set panel layout: <code>c(ncol,nrow)</code>
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 <code>auto.key</code> in examples below)
col	colors to use
map	logical; if TRUE, plot space-time variogram map
convertMonths	logical; if TRUE, <code>yearmon</code> time lags will be unit converted and plotted as (integer) months, and no longer match the numeric representation of <code>yearmon</code> , which has years as unit
as.table	controls the plotting order for multiple panels, see <code>xyplot</code> 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 distances 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

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

```

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

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

# variogram maps:
plot(variogram(g, cutoff=1000, width=100, map=TRUE),
     main = "(cross) semivariance maps")
plot(variogram(g, cutoff=1000, width=100, map=TRUE), np=TRUE,
     main = "number of point pairs")

```

---

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

---

**Description**

Plot a point pairs, identified from a variogram cloud

**Usage**

```
## 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", ...)
```

**Arguments**

x	object of class "pointPairs", obtained from the function <a href="#">plot.variogramCloud</a> , 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 referred to by indices in) x

**Author(s)**

Edzer Pebesma

**References**

<http://www.gstat.org>

**See Also**

[plot.variogramCloud](#)

**Examples**

```
### 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

```
## 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 <a href="#">plot.gstatVariogram</a> (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.

**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)

```

---

predict

---

*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.args = list(n = 500, type = "regular",
  offset = c(.5, .5)), ...)

```

**Arguments**

object	object of class <code>gstat</code> , see <a href="#">gstat</a> and <a href="#">krige</a>
newdata	data frame with prediction/simulation locations; should contain columns with the independent variables (if present) and the coordinates with names as defined in <code>locations</code> ; or: polygons, see below

block	block size; a vector with 1, 2 or 3 values containing the size of a rectangular in x-, y- and z-dimension respectively (0 if not set), or a data frame with 1, 2 or 3 columns, containing the points that discretize the block in the x-, y- and z-dimension to define irregular blocks relative to (0,0) or (0,0,0)—see also the details section below. By default, predictions or simulations refer to the support of the data values.
nsim	integer; if set to a non-zero value, conditional simulation is used instead of kriging interpolation. For this, sequential Gaussian or indicator simulation is used (depending on the value of 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 <a href="#">spsample</a> 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 `nmax` in [krige](#) and [gstat](#). The larger `nmax`, the better the approximation, the smaller `nmax`, the faster the simulation process. For selecting the nearest `nmax` data or previously simulated points, `gstat` uses a bucket PR quadtree neighbourhood search algorithm; see the reference below.

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

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

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

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

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

For data with longitude/latitude coordinates (checked by `is.projected`), `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 `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**

- N.A.C. Cressie, 1993, *Statistics for Spatial Data*, Wiley.
- Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30: 683-691.

**See Also**

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

**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)
gridded(meuse.grid) = ~x+y
sim <- krige(formula = log(zinc)~1, meuse, meuse.grid, model = m,
nmax = 10, beta = 5.9, nsim = 5) # for speed -- 10 is too small!!
# show all 5 simulation
splot(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.m), meuse),
vgm(1, "Sph", 300, 1))
g <- gstat(NULL, "log.zinc", log(zinc)~sqrt(dist.m), 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)
names(xy) <- c("x","y")
gridded(xy) = ~x+y
g.dummy <- gstat(formula = z~1, dummy = TRUE, beta = 0,
model = vgm(1,"Exp",15), 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

```
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

```
set_gstat_progress(FALSE)
get_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

```
show.vgms(min = 1e-12 * max, max = 3, n = 50, sill = 1, range = 1,
  models = as.character(vgm()$short[c(1:17)]), nugget = 0, kappa.range = 0.5,
  plot = TRUE, ..., as.groups = FALSE)
```

### 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](https://wiki.52north.org/bin/view/AI_GEOSTATS/WebHome)

**Examples**

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

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

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

---

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

**Usage**

```
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.

**Author(s)**

Gregoire Dubois and others.

**References**

[https://wiki.52north.org/bin/view/AI\\_GEOSTATS/WebHome](https://wiki.52north.org/bin/view/AI_GEOSTATS/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

---

`tull`*Südliche Tullnerfeld data set*

---

**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**`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

**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](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](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:
cl.st = stack(Chlorid92[-1])

```



```

names(cl.st) = c("Chloride", "Station")
cl.st$Date = rep(Chlorid92$Datum, length(names(Chlorid92))-1)
cl.st$x = tull36[match(cl.st[, "Station"], row.names(tull36)), "x"]
cl.st$y = tull36[match(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, 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

```

## 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	data frame where the names in formula are to be found
locations	spatial data locations. For <code>variogram.formula</code> : a formula with only the coordinate variables in the right hand (explanatory variable) side e.g. $\sim x+y$ ; see examples. For <code>variogram.default</code> : list with coordinate matrices, each with the number of rows matching that of corresponding vectors in <code>y</code> ; the number of columns should match the number of spatial dimensions spanned by the data (1 (x), 2 (x,y) or 3 (x,y,z)).
...	any other arguments that will be passed to <code>variogram.default</code> (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 <code>y</code> , the number of columns equals the number of regressors (including intercept)
cutoff	spatial separation distance up to which point pairs are included in semivariance estimates; 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
crossie	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 $x(s_i)$ the vector with regressors at location $s_i$ , and $\ \cdot\ $ the 2-norm. This allows pooled estimation of within-strata variograms (use a factor variable as regressor, and $dX=0.5$ ), or variograms of (near-)replicates in a linear model (addressing point pairs having similar values for regressors variables)
boundaries	numerical 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 <code>gstat</code> internal debug level
cross	logical or character; if FALSE, no cross variograms are computed when object is of class <code>gstat</code> 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	formula, specifying the dependent variable and possible covariates
x	object of class <code>variogram</code> or <code>variogramCloud</code> to be printed
grid	grid parameters, if data are gridded (not to be called directly; this is filled automatically)
map	logical; if TRUE, and <code>cutoff</code> and <code>width</code> are given, a variogram map is returned. This requires package <code>sp</code> . Alternatively, a map can be passed, of class <code>SpatialDataFrameGrid</code> (see <code>sp</code> docs)
g	NULL or object of class <code>gstat</code> ; 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, Euclidian distances are computed, for unprojected great circle distances (km). In <code>variogram.formula</code> or <code>variogram.gstat</code> , for data deriving from class <code>Spatial</code> , projection is detected automatically using <code>is.projected</code>
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:

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

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:

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

In 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**

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

Cressie, N., C. Wikle, 2011, *Statistics for Spatio-temporal Data*, Wiley.

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

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

**See Also**

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

**Examples**

```
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
```

```

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"))
  # variogram of unprojected data, using great-circle distances, returning km as units
  variogram(log(zinc) ~ 1, meuse.ll)
}

```

---

variogramLine	<i>Semivariance Values For a Given Variogram Model</i>
---------------	--

---

### Description

Generates a semivariance values given a variogram model

### Usage

```

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.

**Author(s)**

Edzer Pebesma

**See Also**[plot.gstatVariogram](#)**Examples**

```

variogramLine(vgm(5, "Exp", 10, 5), 10, 10)
# anisotropic variogram, plotted in E-W direction:
variogramLine(vgm(1, "Sph", 10, anis=c(0,0.5)), 10, 10)
# anisotropic variogram, plotted in N-S direction:
variogramLine(vgm(1, "Sph", 10, anis=c(0,0.5)), 10, 10, dir=c(0,1,0))
variogramLine(vgm(1, "Sph", 10, anis=c(0,0.5)), dir=c(0,1,0), dist_vector = 0.5)
variogramLine(vgm(1, "Sph", 10, anis=c(0,0.5)), dir=c(0,1,0), dist_vector = c(0, 0.5, 0.75))

```

---

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, cores = 1)

```

**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 <a href="#">variogram</a> function. In case of using data of type STIDF, the argument tunit is recommended (and only used in the case of STIDF) 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.

<code>tlags</code>	integer; time lags to consider or in case data is of class <code>STIDF</code> the actual temporal boundaries with time unit given by <code>tunit</code> otherwise the same unit as <code>diff</code> on the index of the time slot will generate is assumed.
<code>cutoff</code>	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.
<code>width</code>	the width of subsequent distance intervals into which data point pairs are grouped for semivariance estimates, by default the <code>cutoff</code> is divided into 15 equal lags.
<code>boundaries</code>	numerical vector with distance interval upper boundaries; values should be strictly increasing
<code>progress</code>	logical; if TRUE, show text progress bar
<code>pseudo</code>	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.
<code>assumeRegular</code>	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.
<code>na.omit</code>	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.
<code>cores</code>	number of cores to use in parallel

### 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

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

Cressie, N., C. Wikle, 2011, Statistics for Spatio-temporal Data, Wiley.

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

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

**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(PM10~1, r5to10, width=20, cutoff = 200, tlags=0:5)  
# in the vignette "st".
```

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

---

variogramSurface	<i>Semivariance values for a given spatio-temporal variogram model</i>
------------------	--

---

**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, covariance = FALSE)
```

**Arguments**

model	A spatio-temporal variogram model generated through <a href="#">vgmST</a> or <a href="#">fit.StVariogram</a> .
dist_grid	A data.frame with two columns: spacelag and timelag.
covariance	Whether the covariance should be computed instead of the variogram (default: FALSE).

**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.



**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, ...)
## S3 method for class 'variogramModel'
plot(x, cutoff, ..., type = 'l')
as.vgm.variomodel(m)

```

**Arguments**

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

x	a variogram model to print or plot
...	arguments that will be passed to print, e.g. digits (see examples), or to variogramLine for the plot method
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
m	object of class variomodel, see <b>geoR</b>
cutoff	maximum distance up to which variogram values are computed
type	plot type

### 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(30, 0.5)`. The first parameter, 30, refers to the main axis direction: it is the angle for the principal direction of continuity (measured in degrees, clockwise from positive Y, i.e. North). The second parameter, 0.5, is the anisotropy ratio, the ratio of the minor range to the major range (a value between 0 and 1). So, in our example, if the range in the major direction (North-East) is 100, the range in the minor direction (South-East) is  $0.5 \times 100 = 50$ .

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),  $\theta_r$  is the third rotation angle to rotate the two minor directions around the principal direction defined by  $\theta_p$  and  $\theta_q$ . A positive angle acts counter-clockwise while looking in the principal direction. Anisotropy ratios  $\theta_s$  and  $\theta_t$  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](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 book 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  $\text{anis} = c(p, \theta, \theta, s, 1)$ .)

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/>

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

Deutsch, C.V. and Journel, A.G., 1998. *GSLIB: Geostatistical software library and user's guide*, second edition, Oxford University Press.

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

```
vgm()
vgm("Sph")
vgm(NA, "Sph", NA, NA)
vgm(, "Sph") # "Sph" is second argument: NO nugget in this case
vgm(10, "Exp", 300)
x <- vgm(10, "Exp", 300)
vgm(10, "Nug", 0)
vgm(10, "Exp", 300, 4.5)
vgm(10, "Mat", 300, 4.5, kappa = 0.7)
vgm( 5, "Exp", 300, add.to = vgm(5, "Exp", 60, nugget = 2.5))
vgm(10, "Exp", 300, anis = c(30, 0.5))
vgm(10, "Exp", 300, anis = c(30, 10, 0, 0.5, 0.3))
```

```

# Matern variogram model:
vgm(1, "Mat", 1, kappa=.3)
x <- vgm(0.39527463, "Sph", 953.8942, nugget = 0.06105141)
x
print(x, digits = 3);
# to see all components, do
print.data.frame(x)
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

```

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, mode = mode, ...)
panel.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 <code>c(dir.horizontal, dir.ver)</code>
labels	labels to plot next to points
shift	amount to shift the label right of the symbol
mode	to be set by calling function only
line.pch	symbol type to be used for point of selected point pairs, e.g. to highlight point pairs with distance close to zero
pairs	two-column matrix with pair indexes to be highlighted
...	parameters that get passed to <a href="#">lpoints</a>

**Value**

ignored; the enclosing function returns a plot of class `trellis`

**Author(s)**

Edzer Pebesma

**References**

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

**See Also**

[plot.gstatVariogram](#), [vgm](#)

**Examples**

```
library(sp)
data(meuse)
coordinates(meuse) <- c("x", "y")
library(lattice)
mypanel = function(x,y,...) {
  vgm.panel.xyplot(x,y,...)
  panel.abline(h=var(log(meuse$zinc)), color = 'red')
}
plot(variogram(log(zinc)~1,meuse), panel = mypanel)
```

---

vgmArea	<i>point-point, point-area or area-area semivariance</i>
---------	--

---

## 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)
```

## Arguments

x	object of class <a href="#">SpatialPoints</a> or <a href="#">SpatialPolygons</a>
y	object of class <a href="#">SpatialPoints</a> or <a href="#">SpatialPolygons</a>
vgm	variogram model, see <a href="#">vgm</a>
ndiscr	number of points to discretize an area, using <a href="#">spsample</a>
verbose	give progress bar
covariance	logical; compute covariances, rather than semivariances?

## 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
```

---

vgmAreaST                      *Function that returns the covariances for areas*

---

### Description

Function that returns the covariances for areas based on spatio-temporal point variograms for use in the spatio-temporal area-to-point kriging

### Usage

```
vgmAreaST(x, y = x, model, ndiscrSpace = 16, verbose = FALSE, covariance = TRUE)
```

### Arguments

x	spatio-temporal data frame
y	spatio-temporal data frame
model	spatio-temporal variogram model for point support
ndiscrSpace	number of discretisation in space
verbose	Boolean: default to FALSE, set to TRUE for debugging
covariance	Boolean: whether the covariance shall be evaluated, currently disfunction and set to TRUE

### Value

The covariance between 'x' and 'y'.

### Author(s)

Benedikt Graeler

### See Also

[vgmArea](#)

### Examples

```
# see demo('a2pinST')
```

vgmST

*Constructing a spatio-temporal variogram***Description**

Constructs a spatio-temporal variogram of a given type checking for a minimal set of parameters.

**Usage**

```
vgmST(stModel, ..., space, time, joint, sill, k, nugget, stAni, temporalUnit)
```

**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.
joint	A joint spatio-temporal variogram.
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.
temporalUnit	length one character vector, indicating the temporal unit (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

```
vgmST("separable", space, time, sill)
```

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

```
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 stAni generating the call



```
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 stAni generating the call

```
vgmST("simpleSumMetric", space, time, joint, nugget, stAni)
```

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

```
vgmST("metric", joint, stAni)
```

### Value

Returns an S3 object of class StVariogramModel.

### Author(s)

Benedikt Graeler

### See Also

[fit.StVariogram](#) for fitting, [variogramSurface](#) to plot the variogram and [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, 3.6),
  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](#).

## Examples

```

## Not run:
# obtained by:
library(spacetime)
library(gstat)
data(air)
suppressWarnings(proj4string(stations) <- CRS(proj4string(stations)))
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** Xlocation in meter

**Y** Ylocation 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 = 0.51418 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 (thats 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:

Haslett, J. and Raftery, A. E. (1989). Space-time Modelling with Long-memory Dependence: Assessing Ireland’s Wind Power Resource (with Discussion). *Applied Statistics* 38, 1-50.

and in many later papers on space-time analysis, for example:

Tilmann Gneiting, Marc G. Genton, Peter Guttorp: Geostatistical Space-Time Models, Stationarity, Separability and Full symmetry. Ch. 4 in: B. Finkenstaedt, L. Held, V. Isham, *Statistical Methods for Spatio-Temporal Systems*.

**Examples**

```
data(wind)
summary(wind)
wind.loc
library(sp) # char2dms
wind.loc$y = as.numeric(char2dms(as.character(wind.loc[["Latitude"]]))))
wind.loc$x = as.numeric(char2dms(as.character(wind.loc[["Longitude"]]))))
coordinates(wind.loc) = ~x+y

## Not run:
# fig 1:
library(maps)
library(mapdata)
map("worldHires", xlim = c(-11,-5.4), ylim = c(51,55.5))
points(wind.loc, pch=16)
text(coordinates(wind.loc), pos=1, label=wind.loc$Station)

## End(Not run)

wind$time = ISOdate(wind$year+1900, 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$jday = as.numeric(format(wind$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)
corv = cor(velocity)
sel = !(as.vector(dists) == 0)
plot(as.vector(corv[sel]) ~ as.vector(dists[sel]),
      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(corv) == "ROS"
dists.nr = dists[!ros,!ros]
corv.nr = corv[!ros,!ros]
sel = !(as.vector(dists.nr) == 0)
plot(as.vector(corv.nr[sel]) ~ as.vector(dists.nr[sel]), pch = 3,
      xlim = c(0,500), ylim = c(.4, 1), xlab = "distance (km.)",
      ylab = "correlation")
# add outlier:
points(corv[ros,!ros] ~ dists[ros,!ros], pch=16, cex=.5)
xdiscr = 1:500
# add correlation model:
lines(xdiscr, .968 * exp(- .00134 * xdiscr))

```

# Index

## \* datasets

- coalash, 3
- DE\_RB\_2005, 4
- fulmar, 15
- jura, 24
- meuse.all, 40
- meuse.alt, 42
- ncp.grid, 43
- oxford, 45
- pcb, 47
- sic2004, 59
- sic97, 61
- tull, 63
- walker, 83
- wind, 84

## \* dplot

- image, 22
- map.to.lev, 40
- plot.gstatVariogram, 48
- plot.pointPairs, 50
- plot.variogramCloud, 52
- show.vgms, 58
- spplot.vcov, 62

## \* 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, 34
- krigeTg, 38
- ossfim, 44
- predict, 53
- progress, 57

- variogram, 65
- variogramLine, 69
- variogramST, 70
- variogramSurface, 72
- vgm, 73
- vgm.panel.xyplot, 76
- vgmArea, 78
- vgmST, 80

## \* spatio-temporal

- variogramSurface, 72
- [.gstat (gstat), 17

- as.data.frame.variogramCloud, 67
- as.data.frame.variogramCloud  
(variogram), 65
- as.vgm.variomodel (vgm), 73

- Chlorid92 (tull), 63
- coalash, 3

- DE\_RB\_2005, 4
- demstd (sic97), 61
- diff, 71

- estiStAni, 5
- extractPar, 6
- extractParNames, 9, 10, 81
- extractParNames (extractPar), 6

- fit.lmc, 7
- fit.StVariogram, 7, 8, 34, 72, 81
- fit.variogram, 8–10, 11, 13–15, 27, 31, 38,  
48, 50, 68, 74, 75

- fit.variogram.gls, 13
- fit.variogram.reml, 11, 14
- fulmar, 15, 43

- get.contr, 16
- get\_gstat\_progress (progress), 57
- getGammas (variogramLine), 69

- gstat, [7](#), [17](#), [26–29](#), [31](#), [32](#), [35](#), [36](#), [38](#), [39](#), [53](#), [55](#), [56](#)
- gstat.cv (krige.cv), [30](#)
- hscat, [21](#)
- identify, [53](#)
- idw (krige), [26](#)
- idw, formula, formula-method (krige), [26](#)
- idw, formula, sf-method (krige), [26](#)
- idw, formula, Spatial-method (krige), [26](#)
- idw, formula, ST-method (krige), [26](#)
- idw-methods (krige), [26](#)
- idw.locations (krige), [26](#)
- idw.spatial (krige), [26](#)
- idw0 (krige), [26](#)
- image, [22](#)
- image.data.frame, [23](#), [40](#)
- image.default, [23](#)
- jura, [24](#)
- juragrid.dat (jura), [24](#)
- krige, [20](#), [26](#), [32](#), [39](#), [40](#), [45](#), [53](#), [55](#), [56](#)
- krige, formula, formula-method (krige), [26](#)
- krige, formula, NULL-method (krige), [26](#)
- krige, formula, sf-method (krige), [26](#)
- krige, formula, Spatial-method (krige), [26](#)
- krige, formula, ST-method (krigeST), [34](#)
- krige-methods (krige), [26](#)
- krige.cv, [30](#)
- krige.cv, formula, formula-method (krige.cv), [30](#)
- krige.cv, formula, sf-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, [35](#), [36](#)
- krige0 (krige), [26](#)
- krigeSimCE, [33](#), [37](#)
- krigeST, [10](#), [34](#)
- krigeSTSimTB, [34](#), [36](#)
- krigeSTTg (krigeST), [34](#)
- krigeTg, [36](#), [38](#)
- locator, [53](#)
- lpoints, [77](#)
- map.to.lev, [40](#)
- meuse.all, [40](#), [42](#)
- meuse.alt, [41](#), [42](#)
- ncp.grid, [16](#), [43](#), [47](#)
- optim, [9](#)
- ossfim, [44](#)
- oxford, [45](#)
- panel.pointPairs (vgm.panel.xyplot), [76](#)
- pcb, [47](#)
- plot.gstatVariogram, [48](#), [52](#), [53](#), [68](#), [70](#), [77](#)
- plot.pointPairs, [50](#), [53](#)
- plot.StVariogram, [72](#)
- plot.StVariogram (plot.gstatVariogram), [48](#)
- plot.variogramCloud, [51](#), [52](#), [68](#)
- plot.variogramMap (plot.gstatVariogram), [48](#)
- plot.variogramModel (vgm), [73](#)
- predict, [16](#), [17](#), [20](#), [26](#), [28](#), [29](#), [31](#), [32](#), [36](#), [38–40](#), [53](#), [55](#)
- prediction.dat (jura), [24](#)
- print.gstat (gstat), [17](#)
- print.gstatVariogram, [68](#)
- print.gstatVariogram (variogram), [65](#)
- print.variogramCloud (variogram), [65](#)
- print.variogramModel (vgm), [73](#)
- progress, [57](#)
- set\_gstat\_progress (progress), [57](#)
- show.vgms, [58](#), [75](#)
- sic.grid (sic2004), [59](#)
- sic.pred (sic2004), [59](#)
- sic.test (sic2004), [59](#)
- sic.train (sic2004), [59](#)
- sic.val (sic2004), [59](#)
- sic2004, [59](#)
- sic97, [61](#)
- sic\_full (sic97), [61](#)
- sic\_obs (sic97), [61](#)
- SpatialPoints, [78](#)
- SpatialPolygons, [54](#), [78](#)
- SpatialPolygonsDataFrame, [54](#)
- spplot.vcov, [62](#)
- spsample, [54](#), [78](#)



ST, [35](#)  
STFDF, [70](#)  
STIDF, [70](#), [71](#)  
STSDF, [70](#)

transect.dat (jura), [24](#)  
tull, [63](#)  
tull36 (tull), [63](#)  
TULLNREG (tull), [63](#)

validation.dat (jura), [24](#)  
variogram, [7](#), [8](#), [11](#), [12](#), [22](#), [48](#), [50](#), [53](#), [65](#), [70](#),  
[75](#)  
variogram.default, [66](#)  
variogramLine, [50](#), [59](#), [69](#), [72](#), [75](#)  
variogramST, [9](#), [65](#), [67](#), [68](#), [70](#), [82](#)  
variogramSurface, [34](#), [72](#), [81](#)  
vgm, [5](#), [7](#), [8](#), [11](#), [12](#), [18](#), [27](#), [31](#), [38](#), [48](#), [50](#), [59](#),  
[68](#), [73](#), [77](#), [78](#)  
vgm.panel.xyplot, [76](#)  
vgmArea, [78](#), [79](#)  
vgmAreaST, [34](#), [79](#)  
vgmST, [6](#), [7](#), [9](#), [34](#), [37](#), [72](#), [80](#)  
vv, [82](#)

walker, [83](#)  
wind, [84](#)

xyplot, [49](#)  
xyz2img, [23](#)  
xyz2img (image), [22](#)