

Package ‘RandomFields’

January 20, 2012

Version 2.0.54

Title Simulation and Analysis of Random Fields

Author Martin Schlather <martin.schlather@math.uni-goettingen.de>

Maintainer Martin Schlather <martin.schlather@math.uni-goettingen.de>

Depends R (>= 2.10)

Description Simulation of Gaussian and extreme value random fields; conditional simulation; kriging

License GPL

URL <http://www.stochastik.math.uni-goettingen.de/~schlather>

Repository CRAN

Date/Publication 2012-01-20 15:18:36

R topics documented:

Changings	2
CondSimu	3
CovarianceFct	6
DeleteRegister	16
Dev	17
EmpiricalVariogram	18
eval.parameters	21
FileExists	24
fitvario	25
Forest	36
fractal.dim	37
GaussRF	40
getactions	49
GetPracticalRange	50
GetRegisterInfo	51
host	53

hurst	54
Kriging	56
Locator	60
MaxStableRF	61
Papers	63
parameter.range	70
parampositions	71
Print	73
PrintModelList	74
RandomFields	75
Readline	77
regression	78
RFMethods	80
RFparameters	84
ShowModels	94
SimulateRF	99
sleep.milli	100
soil	101
Sophisticated Models	104
useraction	116
weather	119

Index	126
--------------	------------

Changings

Changings

Description

Here, only the important inconsistencies between Version 1.3 to 2.0 of RandomFields are listed.

Documentation of the extensions will be enhanced later.

Details

- output of `fitvario`
`$vario` is defunctioned. Use, for instance, `mlmodel` instead of `$vario$ml`.
- The options for `mle.methods` in `fitvario` are currently restricted to 'ml'. This options 'reml' and 'rml1' have not been tested yet.
- the list definition of sophisticated covariance models has become obsolete. Furthermore, the parameter kappa for the model parameters is not accepted anymore. Instead k should be used. See `help("sophisticated")` for the new definition of sophisticated covariance models
- Currently, `ShowModels` does not work, neither Havard Rue's Markov fields. Both should be available in summer 2011 again.
- Currently, the numerical evaluation of the covariance function in `tbm2` is not implemented. This should be available in summer 2011 again.

Please let the author know if further inconsistencies exist.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>;

 CondSimu

Conditional Simulation

Description

the function returns conditional simulations of a Gaussian random field

Usage

```
CondSimu(krige.method, x, y=NULL, z=NULL, T=NULL, grid,
         gridtriple=FALSE, model, param, method=NULL, given, data,
         trend, n=1, register=0,
         err.model=NULL, err.param=NULL, err.method=NULL,
         err.register=1, tol=1E-5, pch=".", paired=FALSE, na.rm=FALSE)
```

Arguments

krige.method	Assumptions on the random field which corresponds to the respective kriging method; currently 'S' (simple kriging) and 'O' (ordinary kriging) are implemented.
x	matrix or vector of x coordinates; points to be kriged.
y	vector of y coordinates.
z	vector of z coordinates.
T	vector in grid triple form for the time coordinates.
grid	logical; determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details.
gridtriple	logical. Only relevant if grid=TRUE. If gridtriple=TRUE then x, y, and z are of the form c(start,end,step); if gridtriple=FALSE then x, y, and z must be vectors of ascending values.
model	string; covariance model of the random field. See CovarianceFct , or type PrintModellist() to get all options for model. See CovarianceFct for model being a list.
param	parameter vector: param=c(mean, variance, nugget, scale,...); the parameters must be given in this order; further parameters are to be added in case of a parametrised class of covariance functions, see CovarianceFct ; the value of mean must be finite in the case of simple kriging, and is ignored otherwise. See CovarianceFct for param being NULL or list.
method	NULL or string; method used for simulating, see RFMethods , or type PrintMethodList() to get all options.

<code>given</code>	matrix or vector of locations where data are available; note that it is not possible to give the points in form of a grid definition.
<code>data</code>	the values measured.
<code>trend</code>	Not programmed yet. (used by universal kriging)
<code>n</code>	number of realisations to generate. If <code>paired=TRUE</code> then <code>n</code> must be even.
<code>register</code>	0:9; place where intermediate calculations are stored; the numbers are aliases for 10 internal registers; see GaussRF for further details.
<code>err.model</code>	covariance function for the error model. String or list. See <code>model</code> for details.
<code>err.param</code>	parameters for the error model. See also <code>param</code> .
<code>err.method</code>	Only relevant if <code>err.model</code> is not NULL. Then it must be given if and only if method is given; see <code>method</code> for details.
<code>err.register</code>	see <code>register</code> for details.
<code>tol</code>	considered only if <code>grid=TRUE</code> ; tolerated distances of a given point to the nearest grid point to be regarded as being zero; see <code>Details</code> .
<code>pch</code>	character. The included kriging procedure can be quite time consuming. The character <code>pch</code> is printed after roughly each 80th part of calculation.
<code>paired</code>	logical. If TRUE then every second simulation is obtained by only changing the signs of the standard Gaussian random variables, the simulation is based on (“antithetic pairs”).
<code>na.rm</code>	logical. If TRUE then NAs are removed from the given data.

Details

The same way as `GaussRF` the function `CondSimu` allows for simulating on grids or arbitrary locations. However simulation on a grid is sometimes performed as if the points were at arbitrary locations, what may imply a great reduction in speed. This happens when the given locations do not lay on the specified grid, since in an intermediate step simulation has to be performed simultaneously on both the grid defined by `x`, `y`, `z`, and the locations of `given`.

Comments on specific parameters

- `grid=FALSE` : the vectors `x`, `y`, and `z` are interpreted as vectors of coordinates
- `(grid=TRUE) && (gridtriple=FALSE)` : the vectors `x`, `y`, and `z` are increasing sequences with identical lags for each sequence. A corresponding grid is created (as given by `expand.grid`).
- `(grid=TRUE) && (gridtriple=TRUE)` : the vectors `x`, `y`, and `z` are triples of the form `(start,end,step)` defining a grid (as given by `expand.grid(seq(x$start,x$end,x$step), seq(y$start,yend,ystep), seq(z$start,z$end,z$step))`)

Value

The returned object depends on the parameters `n` and `grid`:

`n=1`:

* `grid=FALSE`. A vector of simulated values is returned (independent of the dimension of the random field)

* grid=TRUE. An array of the dimension of the random field is returned.

n>1:

* grid=FALSE. A matrix is returned. The columns contain the realisations.

* grid=TRUE. An array of dimension $d + 1$, where d is the dimension of the random field as given by x, y, and z, is returned. The last dimension contains the realisations.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Chiles, J.-P. and Delfiner, P. (1999) *Geostatistics. Modeling Spatial Uncertainty*. New York: Wiley.

Cressie, N.A.C. (1993) *Statistics for Spatial Data*. New York: Wiley.

Goovaerts, P. (1997) *Geostatistics for Natural Resources Evaluation*. New York: Oxford University Press.

Wackernagel, H. (1998) *Multivariate Geostatistics*. Berlin: Springer, 2nd edition.

See Also

[CovarianceFct](#), [GaussRF](#), [Kriging RandomFields](#),

Examples

```
## creating random variables first
## here, a grid is chosen, but any arbitrary points for which
## data are given are fine. Indeed if the data are given on a
## grid, the grid has to be expanded before calling 'CondSimu',
## see below.
## However, locations where values are to be simulated,
## should be given in form of a grid definition whenever
## possible
param <- c(0, 1, 0, 1)
model <- "exponential"

RFparameters(PracticalRange=FALSE)
p <- 1:7
data <- GaussRF(x=p, y=p, grid=TRUE, model=model, param=param)
for (i in 1:3) do.call(getOption("device"), list(height=4,width=4))

# another grid, where values are to be simulated
step <- 0.25 # or 0.3
x <- seq(0, 7, step)

# standardisation of the output
lim <- range( c(x, p) )
zlim <- c(-2.6, 2.6)
```

```

colour <- rainbow(100)

## visualise generated spatial data
dev.set(2)
image(p, p, data, xlim=lim, ylim=lim, zlim=zlim, col=colour)

#conditional simulation
krige.method <- "0" ## random field assumption corresponding to
                  ## those of ordinary kriging

cz <- CondSimu(krige.method, x, x, grid=TRUE,
              model=model, param=param,
              given=expand.grid(p,p),# if data are given on a grid
              # then expand the grid first
              data=data)

range(cz)
dev.set(3)
image(x, x, cz, col=colour, xlim=lim, ylim=lim, zlim=zlim)

#conditional simulation with error term
cze <- CondSimu(krige.method, x, x, grid=TRUE,
              model=model, param=c(0, 1/2, 0, 1),
              err.model="gauss", err.param=c(0, 1/2, 0, 1),
              given=expand.grid(p,p),
              data=data)

range(cze)
dev.set(4)
image(x, x, cze, col=colour, xlim=lim, ylim=lim, zlim=zlim)

```

CovarianceFct

Basic Covariance And Variogram Models

Description

CovarianceFct returns the values of a covariance function; see [Covariance](#) for sophisticated models

Variogram returns the values of a variogram model

Usage

```

Covariance(x, y=NULL, model, param=NULL, dim=ifelse(is.matrix(x),ncol(x),1),
          Distances, fctcall=c("Cov", "Variogram", "CovMatrix"))
CovarianceFct(...)
CovMatrix(...)

Variogram(x, model, param, dim=ifelse(is.matrix(x),ncol(x),1))

```

Arguments

x	vector or $(n \times \text{dim})$ -matrix. In particular, if the model is isotropic or $\text{dim}=1$ then x is a vector.
y	second vector or matrix in case of non-stationary covariance functions
model	for basic models, model is one of the names given in the Details.
param	The simplest form of param is the vector $\text{param}=\text{c}(\text{mean}, \text{variance}, \text{nugget}, \text{scale}, \dots)$, in this order; The dots ... stand for additional parameters of the model, e.g. the smoothing parameter in the whittle model. Within this function mean is not interpreted and can take an arbitrary value.
dim	dimension of the space in which the model is applied
Distances	for covariance matrices, the lower triangular part of the distance matrix can be given instead of the values x themselves
fctcall	internal. This parameter should not be considered by the user
...	The function CovarianceFct is identical to the function Covariance.

Details

Here, only the basic, isotropic models are listed; see [sophisticated models for nonisotropic and hyper models](#).

See [GetModel](#) for commands in R to get information about implemented models and currently used ones.

The implemented models are in standard notation for a covariance function (variance 1, nugget 0, scale 1) and for positive real arguments h :

- + see '[sophisticated](#)'
- * see '[sophisticated](#)'
- \$ see '[sophisticated](#)'
- ave1 see '[sophisticated](#)'
- ave2 see '[sophisticated](#)'
- besse1

$$C(h) = 2^\nu \Gamma(\nu + 1) h^{-\nu} J_\nu(h)$$

The parameter ν is greater than or equal to $\frac{d-2}{2}$, where d is the dimension of the random field.

- Brownian motion
see [fractalB](#)
- cardinal sine
see [wave](#)
- cauchy (normal scale mixture)

$$C(h) = (1 + h^2)^{-\beta}$$

The parameter β is positive. The model possesses two generalisations, the gencauchy model and the hyperbolic model. See also [nonstatcauchy](#) in [Covariance](#).

- cauchytbm

$$C(h) = (1 + (1 - \beta/\gamma)h^\alpha)(1 + h^\alpha)^{-\beta/\alpha - 1}$$

The parameter α is in $(0,2]$ and β is positive. The model is valid for dimensions $d \leq \gamma$; this has been shown for integer γ , but the package allows real values of γ .

It allows for simulating random fields where fractal dimension and Hurst coefficient can be chosen independently. It has negative correlations for $\beta > \gamma$ and large h .

This model is equivalent to the model `list("tbm3", n=gamma, list("gencauchy", alpha=alpha, beta=beta))`

- circular

$$C(h) = \left(1 - \frac{2}{\pi} \left(h\sqrt{1-h^2} + \arcsin(h)\right)\right) 1_{[0,1]}(h)$$

This isotropic covariance function is valid only for dimensions less than or equal to 2.

- cone

This model is used only for methods based on marked point processes (see [RFMethods](#)); it is defined only in two dimensions. The corresponding (boolean) function is a truncated cone with socle. The base has radius $\frac{1}{2}$. The model has three parameters, r , s , and h :

r gives the radius of the top circle of the cone, given as part of the socle radius; $r \in [0, 1)$.

s gives the height of the socle.

h gives the height of the truncated cone.

- coxisham see [sophisticated](#).
- cutoff see [sophisticated](#).
- cubic

$$C(h) = (1 - 7h^2 + 8.75h^3 - 3.5h^5 + 0.75h^7)1_{[0,1]}(h)$$

This model is valid only for dimensions less than or equal to 3. It is a 2 times differentiable covariance functions with compact support.

- dagum

$$C(h) = 1 - (1 + h^{-\beta})^{-\gamma/\beta}$$

RandomFields allows to vary the parameters β and γ within the intervals $(0, 1]$ and $(0, 1)$, respectively.

- dampedcosine (hole effect model)

$$C(h) = e^{-\lambda h} \cos(h), \quad h \geq 0$$

This model is valid for dimension 1 iff $\lambda \geq 1$, for dimension 2 iff $\lambda \geq 1$, and for dimension 3 iff $\lambda \geq \sqrt{3}$.

- DeWijsian

$$\gamma(h) = \log(\|h\|^\alpha + 1)$$

generalised version of the DeWijsian model with $\alpha \in (0, 2]$

- EAxxA and see [‘sophisticated’](#)
- EtAxxA and see [‘sophisticated’](#)

- exponential (normal scale mixture)

$$C(h) = e^{-h}, \quad h \geq 0$$

This model is a special case of the whittle model (for $\nu = \frac{1}{2}$ there) and the stable class (for $\alpha = 1$).

- FD

$$C(k) = \frac{(-1)^k \Gamma(1 - a/2)^2}{\Gamma(1 - a/2 + k) \Gamma(1 - a/2 - k)}, \quad k \in \mathbf{N}$$

and linearly interpolated otherwise. Here, Γ is the Gamma function and $a \in [-1, 1)$. The model is defined in 1 dimension only.

Remark: the fractionally differenced process stems from time series modelling where the grid locations are multiples of the scale parameter.

- fractalB (fractal Brownian motion)

$$\text{gamma}(h) = h^\alpha$$

Here, $\alpha \in (0, 2]$. (Implemented for up to three dimensions). See also genB.

- fractgauss

$$C(h) = 0.5(|h + 1|^\alpha - 2|h|^\alpha + |h - 1|^\alpha)$$

This model is the covariance function for the fractional Gaussian noise with Hurst parameter $H = \alpha/2$, $\alpha \in (0, 2]$. In particular, the model is valid only in one dimension.

- gauss (normal scale mixture)

$$C(h) = e^{-h^2}$$

This model is a special case of the stable class (for $\kappa = 2$ there). Note that the corresponding function for the random coins method (cf. the methods based on marked point processes in [RFMethods](#)) is

$$e^{-2h^2}.$$

See gneiting for an alternative model that does not have the disadvantages of the Gaussian model.

- genB (generalised fractal Brownian motion)

$$\gamma(h) = (h^\alpha + 1)^\delta - 1$$

Here, $\alpha \in (0, 2]$ and $\delta \in (0, 1)$. (Implemented for up to three dimensions). See also fractalB.

- gencauchy (generalised cauchy; normal scale mixture)

$$C(h) = (1 + h^\alpha)^{-\beta/\alpha}$$

The parameter α is in $(0, 2]$, and β is positive.

This model allows for simulating random fields where fractal dimension and Hurst coefficient can be chosen independently.

- gengneiting (generalised gneiting)

If $n = 1$ then

$$C(h) = (1 + (\alpha + 1)h) * (1 - h)^{\alpha+1} 1_{[0,1]}(h)$$

If $n = 2$ then

$$C(h) = (1 + (\alpha + 2)h + ((\alpha + 2)^2 - 1) h^2/3) (1 - h)^{\alpha+2} 1_{[0,1]}(h)$$

If $n = 3$ then

$$C(h) = (1 + (\alpha + 3)h + (2(\alpha + 3)^2 - 3) h^2/5 + ((\alpha + 3)^2 - 4) (\alpha + 3)h^3/15) (1-h)^{\alpha+3} 1_{[0,1]}(h)$$

The parameter n is a positive integer; here only the cases $n = 1, 2, 3$ are implemented. The parameter α is greater than or equal to $(d + 2n + 1)/2$ where d is the dimension of the random field.

- gneiting

$$C(h) = (1 + 8sh + 25(sh)^2 + 32(sh)^3) (1 - sh)^8 1_{[0,1]}(sh)$$

where $s = 0.301187465825$. This isotropic covariance function is valid only for dimensions less than or equal to 3. It is a 6 times differentiable covariance functions with compact support. It is an alternative to the gaussian model since its graph is visually hardly distinguishable from the graph of the Gaussian model, but possesses neither the mathematical and nor the numerical disadvantages of the Gaussian model.

This model is a special case of gengneiting (for $n = 3$ and $\alpha = 5$ there). Note that, in the original work by Gneiting (1999), $s = \frac{10\sqrt{2}}{47} \approx 0.3008965$, a numerical value slightly deviating from the optimal one.

- gneitingdiff is obsolete, see the last example in [Sophisticated](#) for a user's definition of gneitingdiff.

$$C(h) = (1 + 8h\alpha^{-1} + 25h^2\alpha^{-2} + 32h^3\alpha^{-3})(1 - h\alpha^{-1})^8 2^{1-\nu} (\Gamma(\nu))^{-1} h^\nu K_\nu(h) 1_{[0,\alpha]}(h)$$

This isotropic covariance function is valid only for dimensions less than or equal to 3. The parameters ν and α are positive.

This class of models with compact support allows for smooth parametrisation of the differentiability up to order 6.

- hyperbolic (normal scale mixture)

$$C(h) = \delta^{-\lambda} (K_\lambda(\nu\delta))^{-1} (\delta^2 + h^2)^{\lambda/2} K_\lambda(\nu[\delta^2 + h^2]^{1/2})$$

The parameters are such that

$\delta \geq 0, \nu > 0$ and $\lambda > 0$, or

$\delta > 0, \nu > 0$ and $\lambda = 0$, or

$\delta > 0, \nu \geq 0$, and $\lambda < 0$.

Note that this class is over-parametrised; always one of the three parameters ν, δ , and scale can be eliminated in the formula. Therefore, one of these parameters should be kept fixed in any simulation study.

The model contains as special cases the whittle model and the cauchy model, for $\delta = 0$ and $\nu = 0$, respectively.

See also nonstahyperbolic in [Covariance](#).

- iacocesare (non-separable space time model)

$$C(h, t) = (1 + \|h\|^\nu + |t|^\lambda)^{-\delta}$$

The parameters ν and λ take values in $[1, 2]$; the parameters δ must be greater than or equal to half the space-time dimension.

- J-Bessel
see `bessel`
- K-Bessel
see `whittle` and `matern`
- linear with sill
See `power` (`a=1` there).
- `lgd1` (local-global distinguisher)

$$C(h) = 1 - \frac{\beta}{\alpha + \beta} |h|^\alpha, |h| \leq 1 \quad \text{and} \quad \frac{\alpha}{\alpha + \beta} |h|^{-\beta}, |h| > 1$$

Here $\beta > 0$ and α is in $(0, (3 - d)/2]$ for dimension $d = 1, 2$. The random field has fractal dimension $d + 1 - \alpha/2$ and Hurst coefficient $1 - \beta/2$ for $\beta \in (0, 1]$

- `matern` (normal scale mixture)

$$C(x) = W_a(x) = 2^{1-\nu} \Gamma(\nu)^{-1} (\sqrt{2\nu}x)^\nu K_\nu(\sqrt{2\nu}x)$$

The parameter ν is positive.

This is the model of choice if the smoothness of a random field is to be parametrised: if $\nu > m$ then the graph is m times differentiable.

In contrast to the `whittle` model this model separates the effects of the scaling parameter and the shape parameter. For $\nu = 0.5$ we get the exponential model; for $\nu = \infty$ we get $C(x) = \exp(0.5x^2)$.

The model $C(x\sqrt{2})$ equals the Handcock-Wallis (1994) parameterisation.

The model allows further to replace `nu` by $1/\nu$, setting the second parameter `invnu=TRUE`.

See also `whittle`, and `nonstatwhittle` in [Covariance](#).

- `M` and see ‘[sophisticated](#)’
- `mastein` see ‘[sophisticated](#)’
- `mixed` see ‘[sophisticated](#)’
- `nugget`

$$C(h) = 1_{\{0\}}(h)$$

If the model is used in `param`-definition mode, either `param[2]`, the variance, or `param[3]`, the nugget, must be zero. If the model is used in the `list`-definition mode, the anisotropy matrix must be given in an anisotropic context, but not the scale parameter in an isotropic context. See also [sophisticated](#).

- `penta`

$$C(x) = \left(1 - \frac{22}{3}x^2 + 33x^4 - \frac{77}{2}x^5 + \frac{33}{2}x^7 - \frac{11}{2}x^9 + \frac{5}{6}x^{11}\right) 1_{[0,1]}(x)$$

valid only for dimensions less than or equal to 3. This is a 4 times differentiable covariance functions with compact support.

- power

$$C(x) = (1 - x)^a 1_{[0,1]}(x)$$

This covariance function is valid for dimension d if $a \geq (d + 1)/2$. For $\kappa = 1$ we get the well-known triangle (or tent) model, which is valid on the real line, only.

- powered exponential
See [stable](#).
- qexponential

$$C(x) = (2e^{-x} - \alpha e^{-2x}) / (2 - \alpha)$$

The parameter α takes values in $[0, 1]$.

- rational and see [‘sophisticated’](#)
- spherical

$$C(x) = (1 - 1.5x + 0.5x^3) 1_{[0,1]}(x)$$

This isotropic covariance function is valid only for dimensions less than or equal to 3.

- stable

$$C(x) = \exp(-x^\alpha)$$

The parameter α is in $(0, 2]$. See exponential and gaussian for special cases.

- Stein and see [‘sophisticated’](#)
- steinst1 and see [‘sophisticated’](#)
- symmetric stable
See [stable](#).
- tbm2 and see [‘sophisticated’](#)
- tbm3 and see [‘sophisticated’](#)
- tent model
See [power](#).
- triangle
See [power](#).
- wave

$$C(x) = \frac{\sin x}{x}, \quad x > 0 \quad \text{and } C(0) = 1$$

This isotropic covariance function is valid only for dimensions less than or equal to 3. It is a special case of the besel model (for $\kappa = 0.5$).

- whittle (normal scale mixture)

$$C(x) = W_\nu(x) = 2^{1-\nu} \Gamma(\nu)^{-1} x^\nu K_\nu(x)$$

The parameter ν is positive.

This is the model of choice if the smoothness of a random field is to be parametrised: if $\nu > m$ then the graph is m times differentiable.

The model is a special case of the hyperbolic model (for $\nu_3 = 0$ there).

See also [nonstWM](#) in [sophisticated](#).

Let `cov` be a model given in standard notation. Then the covariance model applied with arbitrary variance and scale equals

$$\text{variance} * \text{cov}((\cdot)/\text{scale}).$$

The parameters can be passed by the vector `param`, `param=c(mean, variance, nugget, scale, ...)`. Here ‘...’ stands for additional parameters such as ν in the `whittle` model. In case a model has several parameters, as in `hyperbolic`, the parameters must be given in the sequence they are explained above. However, it is strongly recommended to use the list notation explained in sophisticated. The list definition available in **RandomFields** V 1.x, is depreciated!

For a given covariance function `cov` the variogram γ equals

$$\gamma(x) = \text{cov}(0) - \text{cov}(x).$$

Note:

- The value of the covariance function or variogram depends also on `RFparameters()$PracticalRange`. If the latter is TRUE and the covariance model is isotropic then the covariance function is internally rescaled such that $\text{cov}(1) \approx 0.05$ for standard parameters (`scale=1`).
- Some models allow certain parameter combinations only for certain dimensions. As any model valid in d dimensions is also valid in 1 dimension, the default in `CovarianceFct` and `Variogram` is `dim=1`.

Value

`CovarianceFct` returns a vector of values of the covariance function.

`Variogram` returns a vector of values of the variogram model.

`CovMatrix` return a covariance matrix. Here a matrix of coordinates (`x`) or a vector or a matrix of Distances is expected. `CovMatrix` allows also for variogram models. Then negative of variogram matrix is returned.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Overviews:

- Chiles, J.-P. and Delfiner, P. (1999) *Geostatistics. Modeling Spatial Uncertainty*. New York: Wiley.
- Gneiting, T. and Schlather, M. (2004) Statistical modeling with covariance functions. *In preparation*.
- Schlather, M. (1999) *An introduction to positive definite functions and to unconditional simulation of random fields*. Technical report ST 99-10, Dept. of Maths and Statistics, Lancaster University.
- Schlather, M. (2002) Models for stationary max-stable random fields. *Extremes* **5**, 33-44.

- Yaglom, A.M. (1987) *Correlation Theory of Stationary and Related Random Functions I, Basic Results*. New York: Springer.
- Wackernagel, H. (2003) *Multivariate Geostatistics*. Berlin: Springer, 3rd edition.

Cauchy models, generalisations and extensions

- Gneiting, T. and Schlather, M. (2004) Stochastic models which separate fractal dimension and Hurst effect. *SIAM review* **46**, 269-282.

Dagum model

- Porcu, E., Zini, A. and Pini, R. (2007) Modelling spatio-temporal data: A new variogram and covariance structure proposal *Stats. Probab. Lett.*, **77**, 83-89.
- Berg, C., Mateu, J. and Porcu, E. (2008) The Dagum family of isotropic correlation functions *Bernoulli*, **14**, 1134-1149.

Generalised fractal Brownian motion

- Gneiting, T. (2002) Nonseparable, stationary covariance functions for space-time data, *JASA* **97**, 590-600.

Gneiting's models

- Gneiting, T. (1999) Correlation functions for atmospheric data analysis. *Q. J. Roy. Meteor. Soc., Part A* **125**, 2449-2464.

Holeeffect model

- Zastavnyi, V.P. (1993) Positive definite functions depending on a norm. *Russian Acad. Sci. Dokl. Math.* **46**, 112-114.

Hyperbolic model

- Shkarofsky, I.P. (1968) Generalized turbulence space-correlation and wave-number spectrum-function pairs. *Can. J. Phys.* **46**, 2133-2153.

fractalB

- Stein, M.L. (2002) Fast and exact simulation of fractional Brownian surfaces. *J. Comput. Graph. Statist.* **11**, 587-599.

genB

- Schlather, M. (2010) On some covariance models based on normal scale mixtures. *Bernoulli*, **16**, 780-797.

lgd

- Gneiting, T. and Schlather, M. (2004) Stochastic models which separate fractal dimension and Hurst effect. *SIAM review*

Power model

- Golubov, B.I. (1981) On Abel-Poisson type and Riesz means, *Analysis Mathematica* **7**, 161-184.
- Zastavnyi, V.P. (2000) On positive definiteness of some functions, *J. Multiv. Analys.* **73**, 55-81.

See Also

[sophisticated](#), [EmpiricalVariogram](#), [GetModel](#), [GetPracticalRange](#), [parameter.range](#), [RandomFields](#), [RFparameters](#), [ShowModels](#).

Examples

```
PrintModelList()
x <- 0:100

## the following five model definitions are the same!
##
## (1) very traditional form
(cv <- CovarianceFct(x, model="bessel", param=c(NA,2,1,5,0.5)))
plot(x, cv)

## (2) above model in the very general list definition
model <- list("+",
              list("$", var=2, scale=5, list("bessel", 0.5)),
              list("nugget"))
cv <- CovarianceFct(x, model=model)
points(x, cv, col="red", pch=20) ## no difference to first

## (3) nested model definition
## this kind of definiton models is depreciated from Version 2.0 on
cv <- CovarianceFct(x, model="bessel",
                    param=rbind(c(2, 5, 0.5), c(1, 0, 0)))
points(x, cv, col="blue", pch=20, cex=0.5)

## (4) anisotropic notation
model <- list("+",
              list("$", var=2, aniso=as.matrix(0.2),
                  list("bessel", nu=0.5)
              ),
              list("nugget")
            )
cv <- CovarianceFct(as.matrix(x), model=model)
points(x, cv, col="green", pch=4)

## Depreciated list defintions in Version 1.x
## this way of defining a model still works, but
## is not supported anymore
## (isotropic version)
model <- list(list(model="bessel", var=2, kappa=0.5, scale=5),
              "+",
```

```
list(model="nugget", var=1, scale=1))
cv <- CovarianceFct(x, model=model)
points(x, cv, col="black", pch=5)
```

DeleteRegister*Deleting Intermediate Results*

Description

DeleteRegister deletes an internal register of the random field simulation
DeleteAllRegisters deletes all internal registers

Usage

```
DeleteRegister(nr=0)
DeleteAllRegisters()
```

Arguments

nr number of the register to be deleted; in 0:9

Details

DeleteRegister and DeleteAllRegister should be used if

1. the simulations are finished and there is not enough memory available to continue the R session, and `RFparameters()$Storing` has been TRUE.
2. one of the parameters `RFparameters()$Storing` or `RFparameters()$PracticalRange` (or `RFparameters()$TBM.method`, once this parameter is available) has been changed by the user, and if these options should become global.

See [GaussRF](#) and [RFparameters](#) for further details.

Value

NULL.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de>, <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[GaussRF](#), [RandomFields](#) and [RFparameters](#).

Examples

```
DeleteAllRegisters() ## no effect visible for the user
```

 Dev *Choosing the device*

Description

Dev chooses between the graphical devices screen, postscript and pdf

Usage

```
Dev(on, dev, ps=NULL, cur.cex=TRUE, paper="special", width=5, height=5,
    quiet=FALSE, innerwidth, innerheight, mai, horizontal = FALSE, ...)
```

Arguments

on	logical. Indicates whether dev should be switched on or off
dev	see Details
ps	name of the pdf or postscript file
cur.cex	logical. If TRUE the par parameters of the current device are used, not the standard parameters
paper	kind of paper. Postscript parameter
width	width of figure. Postscript and pdf parameter
height	height of figure. Postscript and pdf parameter
quiet	logical. If FALSE additional information is given.
innerwidth	height of graphic without the margins; overwrites height if given – experimental state
innerheight	width of graphic without the margins; overwrites width if given – experimental state
mai	parameters of function <code>par</code> – experimental state
horizontal	logical. If FALSE the picture is not automatically turned if height is larger than width
...	further parameters for pdf or postscript

Details

The parameter dev might be

logical If suffix of ps is either "eps", "ps" or "pdf" the respective file is created. Otherwise, a postscript file ending with suffix "eps" is created if dev=TRUE and a pdf file with suffix "pdf" if dev=FALSE

character A function with name dev is called, and the suffix dev is added to ps

numeric Dev switches to the device with number dev; if such a device does not exist, a new X11 device is created. If dev<2 the device opened by Dev(on=TRUE,...) is closed by Dev(on=FALSE); otherwise only `par(new=F)` is applied to the device.

The parameters dev, ps, cur.cex, paper, width, height and ... are ignored if on=FALSE.

Value

NULL. Side effect is that the global variable `.dev.orig` is created when `Dev(on=TRUE,...)` is called.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

Examples

```
## first an eps-file test.eps is created, then a jpeg-file,
## finally the figure is plotted on the screen
dir(pattern="test*")
dev.list <- list(TRUE, 1)
if (interactive()) dev.list <- c(dev.list, "jpeg")
for (dev in dev.list) {
  Print(dev)
  size <- if (dev=="jpeg") 450 else 5
  err <- class(try(Dev(TRUE, dev, ps="test", height=size, width=size)))
  if (err!="try-error") {
    plot(0, 0, main=paste("dev=", dev))
    # readline("press return")
    Dev(FALSE)
  }
}
dir(pattern="test*")
```

EmpiricalVariogram *Empirical (Semi-)Variogram*

Description

`EmpiricalVariogram` calculates the empirical (semi-)variogram of a random field realisation

Usage

```
EmpiricalVariogram(x, y=NULL, z=NULL, T=NULL, data, grid, bin,
                  gridtriple=FALSE, phi, theta, deltaT)
```

Arguments

<code>x</code>	vector of x-coordinates, or matrix
<code>y</code>	vector of y-coordinates
<code>z</code>	vector of z-coordinates
<code>T</code>	vector of time components; here T is given in grid format, see GaussRF .

<code>data</code>	vector or matrix of data; if <code>data</code> has a multiple number of components as expected by the definition of the coordinates then it is assumed that the data stem from repeated, independent measurements at the given locations; the empirical variogram is calculated for the repeated data.
<code>grid</code>	logical; if TRUE then <code>x</code> , <code>y</code> , and <code>z</code> define a grid; otherwise <code>x</code> , <code>y</code> , and <code>z</code> are interpreted as points
<code>bin</code>	vector of ascending values giving the bin boundaries
<code>gridtriple</code>	logical. Only relevant if <code>grid=TRUE</code> . If <code>gridtriple=TRUE</code> then <code>x</code> , <code>y</code> , and <code>z</code> are of the form <code>c(start,end,step)</code> ; if <code>gridtriple=FALSE</code> then <code>x</code> , <code>y</code> , and <code>z</code> must be vectors of ascending values
<code>phi</code>	vector of two components. First component gives the angle for the first line of midpoints of an angular variogram. The second component gives the number of directions (on the half circle). The spatial dimension must be at least 2.
<code>theta</code>	vector of two components. First component gives the angle for the first line of midpoints of an angular variogram (angle is zero for the <code>xy</code> -plane). The second component gives the number of directions (on the half circle). The spatial dimension must be at least 3.
<code>deltaT</code>	vector of two components. First component gives the largest temporal distance; the second component the grid length, that must be a multiple of $T[3]$.

Details

Comments on specific parameters:

- `data`: the number of values must match the number of points (given by `x`, `y`, `z`, `grid`, and `gridtriple`). That is, it must equal the number of points or be a multiple of it. In case the number of data equals n times the number of points, the data are interpreted as n independent realisations for the given set of points.
- (`grid=FALSE`): the vectors `x`, `y`, and `z`, are interpreted as vectors of coordinates
- (`grid=TRUE`) && (`gridtriple=FALSE`): the vectors `x`, `y`, and `z` are increasing sequences with identical lags for each sequence. A corresponding grid is created (as given by `expand.grid`).
- (`grid=TRUE`) && (`gridtriple=TRUE`): the vectors `x`, `y`, and `z` are triples of the form `(start,end,step)` defining a grid (as given by `expand.grid(seq(x$start,x$end,x$step), seq(y$start,yend,ystep), seq(z$start,z$end,z$step))`)
- The bins are left open, right closed intervals, i.e., $(b_i, b_{i+1}]$ for $i = 1, \dots, \text{length}(\text{bin})-1$. Hence, to include zero, `bin[1]` must be negative.

Value

The function returns a list:

<code>centers</code>	central points of the bins
<code>emp.vario</code>	empirical variogram; vector or matrix or array, depending on the anisotropy definitions. The sequence is distances, <code>phi</code> , <code>theta</code> , <code>Tbins</code> . If <code>phi</code> , <code>theta</code> , or <code>Tbins</code> below are not given, the respective dimensions are missing.

sd sd of the variogram cloud within each bin
n.bin number of points within a bin
phi vector of angles in xy plane
theta vector of angles in the third dimensions
Tbins vector of temporal distances

The first four elements are vectors of length $(\text{length}(\text{bin})-1)$.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[GaussRF](#), [fitvario](#), and [RandomFields](#)

Examples

```
#####
## this example checks whether a certain simulation method ##
## works well for a specified covariance model and          ##
## a configuration of points                               ##
#####
x <- seq(0, 10, 0.5)
y <- seq(0, 10, 0.5)
gridtriple <- FALSE        ## see help("GaussRF")
model <- "whittle"        ## whittlematern
bins <- seq(0, 5, 0.001)
realisations <- 5 ## by far too small to get reliable results!!
              ## It should be of order 500, but then it will
              ## take some time to do the simulations
param <- c(mean=1, variance=10, nugget=5, scale=2, alpha=2)
f <- GaussRF(x=x, y=y, grid=TRUE, gridtriple=gridtriple,
            model=model, param=param, method="TBM3",
            n=realisations)
binned <- EmpiricalVariogram(x=x, y=y, data=f, grid=TRUE,
                            gridtriple=gridtriple, bin=bins)
truevariogram <- Variogram(binned$c, model, param)
matplot(binned$c, cbind(truevariogram,binned$e), pch=c("*","e"))
##black curve gives the theoretical values
```

eval.parameters	<i>Interactive menu</i>
-----------------	-------------------------

Description

eval.parameters provides an interactive menu on a X11 graphical device of R

Usage

```
eval.parameters(variable, entry, update, simulate, dev, create = TRUE,
  col.rect = "red", col.bg = "blue", col.sep = "grey",
  col.left = "red", col.mid = "white", col.right = "white",
  col.line = "red", col.txt = "black",
  cex=0.8, cex.i=cex, sep="-----", ...)
```

Arguments

variable	string. The name of the variable to be changed. The name consist of \$ and [[]] expression pointing to sublists of a list. The complete list must be given by name in ...
entry	A list of lists. See Details.
update	logical. If TRUE then simulate is called after each interactive input.
simulate	function that is called if simulations are to updated. The parameters must equal the variables given by ...; the function must return the complete list indicated by variable.
dev	Before calling eval.parameters() split.screen must have been called. dev gives the screen on which the interactive menu should be plotted.
create	logical. If TRUE missing list elements of variable are created.
col.rect	colour of the button for free input.

col.bg	colour of a interactive bar
col.sep	colour of the separating line
col.left	colour of preceding element
col.mid	colour for the message
col.right	colour of subsequent element
col.line	colour of the marking line in interactive bars of absolute choice.
col.txt	colour of headers
cex	font height of headers
cex.i	font height of text for elements
sep	style of added characters in separating line.
...	The input variables given by name; the names may not start with a dot; There the complete list to which variable refers must be given. Further additional parameters for the function simulate.

Details

eval.parameters shows a menu list on X11. Depending on the mode of the variables the menu bars have a different appearance and behave differently if the user clicks on the bar. Most of the menu bars have a small rectangle on the right hand side. If this rectangle is pressed the input of a variable is expected in the xterm where R is run.

entry is a list of lists. Each list may contain the following elements:

- name : header for menu button if var is not NULL; otherwise printed instead of a menu button
- var :
 - NULL : if val=NULL then it is a separating line in colour col.sep; name is surrounded by sep; all other elements of the list are ignored. If val is a string then val is interpreted as a function; a special string is "simulate", which entails the call of the function simulate with the appropriate parameters.
 - string : selected element of the list that is given by variable. *A special string "undo" will be installed to undo things.*
- val :
 - function(d, v) gives the update for var. If v is missing, a starting value (for d=1/2) is expected. Otherwise, v is the current value of var and d is the choice of the user, a value in [0, 1]
 - TRUE : logical bar.
 - FALSE : logical bar. The value for var is negated before shown. In the menu, the negative value for var is shown.
 - NULL : a string is read from the terminal inot var.
 - character (vector): if var is given then this vector of strings is interpreted as belonging to a categorical variable 1, ..., length(val) and var gives the number of the selected elements. If var=NULL then val is interpreted as a function; a special string is "simulate", which entails the call of the function simulate with the appropriate parameters.
- delta : logical. In the menu bar absolute values are plotted if delta=FALSE and increments otherwise. Only considered if val is a function.

- cond : The menu points is shown only if the given condition is satisfied. The condition must be expressed by named elements of the list variable, see example.
- col : colour that overwrites the standard colour for the rectangle or the separating text.
- update : logical. If not missing, its value overwrites locally the value of the global parameter update.
- ... : additional parameters for simulate that overwrite the values given in ... in the call of eval.parameters.

Value

The first variable given in "...", which is a list. To this list the entry .history is added.

If the users enters 'exit immediately' at any point, the program stops with an error message.

Note

To the list given by variable the element .history is added. .history is a list that contains the history of the user input. Each element is a list where the first entry is the number of the menu, the second and the third entries are the former and the new value. Exception: for entries with character val, the value of val is returned as second entry. Consequently, the name .history should not be used for other purposes in variable.

Further, any variable name given in ... must start with a letter.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[useraction](#)

Examples

```
## Not run:
## the following lines define a menu that does not make
## too much sense, but shows the various kinds of buttons

quadratic <- function(d, v, a, mini=0, maxi=Inf) {
  d <- pmin(1, pmax(0, d)) - 0.5
  d <- ((d>0) * 2 - 1) * d^2 * a * 4
  if (missing(v)) d else pmax(mini, pmin(maxi, v + d))
}

simulate <- function(H, par) { ## not a serious example
  Print(c(H$x$var, par, runif(1)))
  return(H) ## the function must return the first parameter
}

entry <- list(
```

```

list(name="Nonsense Menu"),
list(name="Simulate!", val="simulate", col="blue"),
list(name="show H", val="str(H)", col="blue"),
list(name="colx", var="colour",
      val=c("red", "green", "blue", "brown")),
list(name="open", var="closed", val=FALSE, par=4.5),
list(name="modifying", var="modify", val=TRUE, par=5),
list(name="probability", var="probab", delta=FALSE,
      val=function(d, v) pmin(1, pmax(0, d))),
list(name="variance", var="var", delta=TRUE,
      val=function(d, v) quadratic(d, v, 10)),
list(name="name", var="name", par=3, cond="modify")
)

scr <- split.screen(rbind(c(0, 0.45, 0, 1), c(0.5, 1, 0, 1)))
## before proceeding make sure that both the screen and the xterm
## are completely visible

H <- list(modify=5, x=list()) # note that in this example eval.parameters
##           will be called by H$x, hence modify=5 will be left
##           unchanged.
options(locatorBell=FALSE)

useraction("start.register") ## registering the user's input
Print(eval.parameters("H$x", entry, simulate, update=TRUE, dev=scr[2],
  H=H, par=17)) # do not forget to call by name
getactions()

## replay the user's input
useraction("replay")
Print(eval.parameters("H$x", entry, simulate, update=TRUE, dev=scr[2],
  H=H, par=17))

## End(Not run)

```

FileExists

Files

Description

The function FileExists checks whether a file or a lock-file exists

The function LockRemove removes a lock-file

Usage

```
FileExists(file, PrintLevel=RFparameters()$Print)
```

```
LockRemove(file)
```

Arguments

file name of the data file
 PrintLevel if PrintLevel<=1 no messages are displayed

Details

FileExists checks whether file or file.lock exists. If none of them exists file.lock is created and hostname and PID are written into file.lock. This is useful if several processes use the same directory. Further, it is checked whether another process has tried to create the same file in the same instance. In this case FileExists returns for at least one of the processes that file.lock has already been created.

Value

FileExists returns

1 if file already exists
 2 if file.lock already exists
 3 if file.lock was tried to be created, but another process inferred and got priority
 0 otherwise, file and file.lock did not exist and file.lock has been created

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

fitvario *LSQ and Maximum Likelihood Estimation of Random Field Parameters*

Description

The function estimates arbitrary parameters of a random field specification with various methods.

Usage

```
fitvario(x, y=NULL, z=NULL, T=NULL, data, model, param,
         lower=NULL, upper=NULL, sill=NA, grid=!missing(gridtriple),
         gridtriple, ...)
```

```
fitvario.default(x, y=NULL, z=NULL, T=NULL, data, model, param,
                 grid=!missing(gridtriple), gridtriple=FALSE,
                 trend = NULL,
                 BC.lambda, ## if missing then no BoxCox-Trafo
                 BC.lambdaLB=-10, BC.lambdaUB=10,
                 lower=NULL, upper=NULL, sill=NA,
```

```

use.naturalscaling=FALSE, PrintLevel,
optim.control=NULL, bins=20, nphi=1, ntheta=1, ntime=20,
distance.factor=0.5,
upperbound.scale.factor=3, lowerbound.scale.factor=3,
lowerbound.scale.LS.factor=5,
upperbound.var.factor=10, lowerbound.var.factor=100,
lowerbound.sill=1E-10, scale.max.relative.factor=1000,
minbounddistance=0.001, minboundreldist=0.02,
approximate.functioncalls=50, refine.onborder=TRUE,
minmixedvar=1/1000, maxmixedvar=1000,
pch=RFparameters()$pch,
transform=NULL, standard.style=NULL,
var.name="X", time.name="T",
lsq.methods=c("self", "plain", "sqrt.nr", "sd.inv", "internal"),
mle.methods=c("ml"),
cross.methods=NULL,

users.guess=NULL, only.users = FALSE,
Distances=NULL, truedim,
solvesigma = NA, # if NA then use algorithm -- ToDo
allowdistanceZero = FALSE,
na.rm = TRUE)

```

Arguments

x	($n \times 2$)-matrix of coordinates, or vector of x-coordinates. All locations must be given explicitly and cannot be passed via a grid definition as in <code>GaussRF</code>
y	vector of y coordinates
z	vector of z coordinates
T	vector of T coordinates; these coordinates are given in triple notation, see <code>GaussRF</code>
data	vector or matrix of values measured at coord; If a matrix is given then the columns are interpreted as independent realisations. If also a time component is given, then in the data the indices for the spatial components run the fastest. If an n-variate model is used, then each realisation is given as n consecutive columns of data.
model	string or list; covariance model, see <code>CovarianceFct</code> and <code>Covariance</code> , or type <code>PrintModellist()</code> to get all options. If model is a list, then the parameters with value NA are estimated. Parameters that have value NaN should be explicitly be defined by the function transform. An alternative to define NaN values and the function transform, is to replace the NaN by a real-valued function with solely parameter a list defining a covariance model. In case of the anisotropy matrix, the matrix must be replaced by a list if functions are introduced. Only the list elements variance, scale or anisotropy, and kappas can be used, and not the mean or the trend. Further, the mean or the trend cannot be set by such a function. See also transform below.

param	vector or matrix or NULL. If vector then param=c(mean, variance, nugget, scale, ...); the parameters must be given in this order. Further parameters are to be added in case of a parametrised class of covariance functions, see CovarianceFct and Covariance . Any components set to NA are estimated; the others are kept fix. See also model above.
lower	list or vector. Lower bounds for the parameters. If param is a vector, lower has to be a vector as well and its length must equal the number of parameters to be estimated. The order of param has to be maintained. A component being NA means that no manual lower bound for the corresponding parameter is set. If param is a list, lower has to be of (exactly) the same structure.
upper	list or vector. Upper bounds for the parameters. See also lower.
sill	If not NA the sill is kept fix. Only used if the standard format for the covariance model is given. See Details.
grid	boolean. Weather coordinates give a grid
gridtriple	boolean. Format, see GaussRF
BC.lambda	a vector of at most two numerical components (just one component corresponds to two identical ones) which are the parameters of the box-cox-transformation: $\frac{x_1^\lambda - 1}{\lambda} + \lambda_2$ If the model is univariate, the first parameter can be estimated by using NA.
BC.lambdaLB	lower bound for the first box-cox-parameter
BC.lambdaUB	upper bound for the first box-cox-parameter
trend	If a univariate model is used, the following trend types are possible: number: the constant mean (not to be estimated any more) NA: there is a constant mean to be estimated formula : uses X1, X2,... and T as internal parameters for the coordinates; all parameters are estimated list of matrices: length of the list must be the number of realisations; each matrix must have the same number of rows as x list of matrices and formula: trend is a list of matrices (see above) and one additional entry which is a formula In an n -variate model trend can be either a list of n trends for univariate models or a list of $n * d$ matrices (d : number of independent realisations) where each entry of trend corresponds to a column of data.
...	arguments as given in fitvario.default and listed in the following.
use.naturalscaling	logical. Only used if model is given in standard (simple) way. If TRUE then <i>internally</i> , rescaled covariance functions will be used for which cov(1)≈0.05. use.naturalscaling has the advantage that scale and the form parameters of the model get 'orthogonal', but use.naturalscaling does not work for all models. See Details.
PrintLevel	level to which messages are shown. See Details.
optim.control	control list for optim , which uses 'L-BFGS-B'. However 'parscale' may not be given.

<code>bins</code>	number of bins of the empirical variogram. See Details.
<code>nphi</code>	scalar or vector of 2 components. If it is a vector then the first component gives the first angle of the xy plane and the second one gives the number of directions on the half circle. If scalar then the first angle is assumed to be zero. Note that a good estimation of the variogram by LSQ with a anisotropic model a large value for <code>ntheta</code> might be needed (about 20).
<code>ntheta</code>	scalar or vector of 2 components. If it is a vector then the first component gives the first angle in the third direction and the second one gives the number of directions on the half circle. If scalar then the first angle is assumed to be zero. Note that a good estimation of the variogram by LSQ with a anisotropic model a large value for <code>ntheta</code> might be needed (about 20).
<code>ntime</code>	scalar or vector of 2 components. if <code>ntimes</code> is a vector, then the first component are the maximum time distance (in units of the grid length <code>T[3]</code>) and the second component gives the step size (in units of the grid length <code>T[3]</code>). If scalar then the step size is assumed to 1 (in units of the grid length <code>T[3]</code>).
<code>distance.factor</code>	relative right bound for the bins. See Details.
<code>upperbound.scale.factor</code>	relative upper bound for scale in LSQ and MLE. See Details.
<code>lowerbound.scale.factor</code>	relative lower bound for scale in MLE. See Details.
<code>lowerbound.scale.LS.factor</code>	relative lower bound for scale in LSQ. See Details.
<code>upperbound.var.factor</code>	relative upper bound for variance and nugget. See Details.
<code>lowerbound.var.factor</code>	relative lower bound for variance. See Details.
<code>lowerbound.sill</code>	absolute lower bound for variance and nugget. See Details.
<code>scale.max.relative.factor</code>	relative lower bound for scale below which an additional nugget effect is detected. See Details.
<code>minbounddistance</code>	absolute distance to the bounds below which a part of the algorithm is considered as having failed. See Details.
<code>minboundreldist</code>	relative distance to the bounds below which a part of the algorithm is considered as having failed. See Details.
<code>approximate.functioncalls</code>	approximate evaluations of the ML target function on a grid. See Details.
<code>refine.onborder</code>	logical. If <code>refine.onborder=TRUE</code> and if the result of any maximum likelihood method or cross validation method is on a borderline, then the optimisation is redone in a modified way (which takes about double extra time)
<code>minmixedvar</code>	lower bound for variance in a mixed model; so, the covariance model for mixed model part might be calibrated appropriately

maxmixedvar	upper bound for variance in a mixed model; so, the covariance model for mixed model part might be calibrated appropriately
pch	character shown before evaluating any method; if pch!="" then one or two additional steps in the MLE methods are marked by "+" and "#". Default: "*".
var.name	basic name for the coordinates in the formula of the trend. Default: 'X'
time.name	basic name for the time component in the formula of the trend. Default: 'X'
transform	<p>function. Essentially, transform allows for the definition of a parameter as a function of other estimated parameters. All the parameters are supposed to be in a vector called 'param' where the positions are given by parampositions. An example of transform is <code>function(param) {param[3] <- 5 - param[1]; param}</code>.</p> <p>Note that the mean and the trend of the model can be neither set nor used in transform. See also <code>standard.style</code>.</p> <p>Note further that many internal checks cannot be performed in case of the very flexible function transform. Hence, it is completely up to the user to get <code>users.guess</code>, lower and upper right. The parameter <code>users.guess</code> must be given; lower and upper should be given.</p> <p>Default: NULL</p>
standard.style	<p>logical or NULL. This variable should only be set by the advanced user. If NULL then <code>standard.style</code> will be TRUE if the covariance model allows for a 'standard' definition (see CovarianceFct) and transform is NULL.</p> <p>If a 'standard' definition is given and both the variance and the nugget are either not estimated or do not appear on the right hand side of the transform, then <code>standard.style</code> might be set to TRUE by the user. This accelerates the MLE algorithm. The responsibility is completely left to the user, then.</p>
lsq.methods	variants of the least squares fit of the variogram. See Details.
mle.methods	variants of the maximum likelihood fit of the covariance function. See Details.
cross.methods	Not implemented yet.
users.guess	User's guess of the parameters. All the parameters must be given using the same rules as for either param (except that no NA's should be contained) or model.
only.users	boolean. If true then only <code>users.guess</code> is used as a starting point for the fitting algorithms
Distances	alternatively to coordinates x, y, and z the distances themselves can be given. Then <code>truedim</code> must be indicated.
truedim	see Distances
solvesigma	Boolean – experimental stage! If a mixed effect part is present where the variance has to be estimated, then this variance parameter is solved iteratively within the profile likelihood function, if <code>solvesigma=TRUE</code> . This makes sense if the number of independent variables is very small. If <code>solvesigma=FALSE</code> then the variance parameter is treated as any other parameter to be estimated.
allowdistanceZero	boolean. If true, then multiple observations are allowed within a single data set. In this case, the coordinates are slightly scattered, so that the points have some tiny distances.

`na.rm` boolean – experimental stage. Only the data may have missing values. If `na.rm=TRUE` then lines of (repeated) data are deleted if at least one missing value appears. If `na.rm=FALSE` then the repetitions are treated sepeartely.

Details

The optimisations are performed using `optimize` if one parameter has to be estimated only and `optim`, otherwise.

First, by means of various control parameters, see below, the algorithm first tries to estimate the bounds for the parameters to be estimated, if the bounds for the parameters are not given.

Independently whether users `.guess` is given, the algorithm guesses initial values for the parameters. The automatic guess and the user’s guess will be called primitive methods in the following.

Second, the variogram model is fitted by various least squares methods (according to the value of `lsq.methods`) using the best parameter set among the primitive methods as initial value if the effective number of parameters is greater than 1.

[Remarks: (i) “best” with respect to the target value of the respective `lsq` method; (ii) the effective number of parameters in the optimisation algorithm can be smaller than the number of estimated parameters, since in some cases, some parameters can be calculated explicitley; relevant for the choice between `optimize` and `optim` is the effective number of parameters; (iii) `optim` needs]

Third, the model is fitted by various maximum likelihood methods (according to the value of `mle.methods`) using the best parameter set among the primitive methods and the `lsq` methods as initial value (if the effective number of parameters is greater than 1).

Comments on specific parameters:

- `BC.lambda` If you want to estimate `BC.lambda` you should assert that all data values are positive; otherwise errors will probably occur because of the box-cox-transformation. The second parameter of the box-cox-transformation cannot be estimated since it corresponds to the mean. So the mean should be estimated instead.
- `trend` Among the formes mentioned above it is possible to use just one matrix for the trend instead of a list of identical ones.
- `lower`
The lower bounds are technical bounds that should not really restrict the domaine of the value. However, if these values are too small the optimisation algorithm will frequently run into local minima or get stuck close the border of the parameter domain. It is advised to limit seriously the domain of the additional parameters of the covariance model and/or the total number of parameters to be estimated, if “many” parameters of the covariance model are estimated.
If the model is given in standard form, the user may supply the lower bounds for the whole parameter vector, or only for the additional form parameters of the model. The lower bound for the mean will be ignored. `lower` may contain NAs, then these values are generated by the
If a nested model is given, the bounds may again be supplied for all parameters or only for the additional form parameters of the model. The bounds given apply uniformly to all submodels of the nested model.
If the model is given in list format, then `lower` is a list, where components may be missing or NA. These are generated by the algorithm, then.
If `lower` is NULL all lower bounds are generated automatically.

- `upper.kappa`
See `lower.kappa`.
- `sill`
Additionally to estimating nugget and variance separately, they may also be estimated together under the condition that `nugget + variance = sill`. For the latter a finite value for `sill` has to be supplied, and `nugget` and `variance` are set to NA.
`sill` is only used for the standard model.
- `use.naturalscaling`
logical. If TRUE then internally, rescaled covariance functions will be used for which `cov(1)≈0.05`. However this parameter does not influence the output of `fitvario`: the parameter vector returned by `fitvario` refers *always* to the standard covariance model as given in `CovarianceFct`. (In contrast to `PracticalRange` in `RFparameters`.)
Advantages if `use.naturalscaling=TRUE`:
 - scale and the shape parameter of a parameterised covariance model can be estimated better if they are estimated simultaneously.
 - The estimated bounds calculated by means of `upperbound.scale.factor` and `lowerbound.scale.factor`, etc. might be more realistic.
 - in case of anisotropic models, the inverse of the elements of the anisotropy matrix should be in the above bounds.
 Disadvantages if `use.naturalscaling=TRUE`:
 - For some covariance models with additional parameters, the rescaling factor has to be determined numerically. Then, more time is needed to perform `fitvario`.
 Default: TRUE.
- `PrintLevel`
0 : no message
1 : error messages
2 : warnings
3 : minimum debugging information
5 : extended debugging information, including graphics
Default: 0.
- `trace.optim`
see control parameter `trace` of `optim`. Default: 0.
- `bins`
vector of explicit boundaries for the bins or the number of bins for the empirical variogram (used in the LSQ target function, which is described at the beginning of the Details). Note that for anisotropic models, the value of `bins` might be enlarged. Default: 20.
- `distance.factor`
right boundary of the last bin is calculated as `distance.factor * (maximum distance between all pairs of points)`. Only used if `bins` is a scalar. Default: 0.5.
- `upperbound.scale.factor`
The upper bound for the scale is determined as `upperbound.scale.factor * (maximum distance between all pairs of points)`. Default: 10.
- `lowerbound.scale.factor`
The lower bound for the scale is determined as

(minimum distance between different pairs of points)/`lowerbound.scale.factor`.
Default: 20.

- `lowerbound.scale.LS.factor`

For the LSQ target function a different lower bound for the scale is used. It is determined as (minimum distance between different pairs of points)/`lowerbound.scale.LS.factor`.
Default: 5.

- `upperbound.var.factor`

The upper bound for the variance and the nugget is determined as

$$\text{upperbound.var.factor} * \text{var}(\text{data}).$$

Default: 10.

- `lowerbound.var.factor`

The lower bound for the variance and the nugget is determined as

$$\text{var}(\text{data})/\text{lowerbound.var.factor}.$$

If a standard model definition is given and either the nugget or the variance is fixed, the parameter to be estimated must also be greater than `lowerbound.sill`. If a non-standard model definition is given then `lowerbound.var.factor` is only used for the first model; the other lower bounds for the variance are zero. Default: 100.

- `lowerbound.sill`

See `lowerbound.var.factor`. Default: 1E-10.

- `scale.max.relative.factor`

If the initial scale value for the ML estimation obtained by the LSQ target function is less than (minimum distance between different pairs of points)/`scale.max.relative.factor` a warning is given that probably a nugget effect is present. Note: if `scale.max.relative.factor` is greater than `lowerbound.scale.LS.factor` then no warning is given as the scale has the lower bound (minimum distance between different pairs of points)/`lowerbound.scale.LS.factor`.
Default: 1000.

- `minbounddistance`

If any value of the parameter vector returned from the ML estimation is closer than `minbounddistance` to any of the bounds or if any value has a relative distance smaller than `minboundreldist`, then it is assumed that the MLE algorithm has dropped into a local minimum, and it will be continued with evaluating the ML target function on a grid, cf. the beginning paragraphs of the Details. Default: 0.001.

- `minboundreldist`

See `minbounddistance`. Default: 0.02.

- `approximate.functioncalls`

In case the parameter vector is too close to the given bounds, the ML target function is evaluated on a grid to get a new initial value for the ML estimation. The number of points of the grid is approximately `approximate.functioncalls`. Default: 50.

- `lsq.methods`

Variogram fit by least squares methods; first, a preliminary trend is estimated by a simple regression; second, the variogram is fitted; third, the trend is fitted using the estimated covariance structure.

- "self" weighted lsq. Weights are the values of the fitted variogram to the power of -2
 - "plain" model fitted by least squares; trends are never taken into account
 - "sqrt.nr" weighted lsq. Weight is the square root of the number of points in the bin
 - "sd.inv" weighted lsq. Weight is the inverse the standard deviation of the variogram cloud within the bin
- mle.methods
Model fit by various maximum likelihood methods (according to the value of `mle.methods`) using the best parameter set among the primitive methods and the lsq methods as initial value (if the effective number of parameters is greater than 1). If the best parameter vector of the MLE found so far is too close to some given bounds, see the specific parameters above, it is assumed that `optim` ran into a local minimum because of a bad starting value. In this case and if `refine.onborder=TRUE` the MLE target function is calculated on a grid, the best parameter vector is taken, and the optimisation is restarted with this parameter vector.
 - "ml" maximum likelihood; since ML and REML give the same result if there are not any covariates, ML is performed in that case, independently whether it is given or not.
 - "reml" restricted maximum likelihood

Value

The function returns a list with the following elements

<code>ev</code>	list returned by EmpiricalVariogram
<code>table</code>	Matrix. The first rows contain the estimated parameters, followed by the target values of all methods for the given set of parameters; the last rows give the lower and upper bounds used in the estimations. The columns correspond to the various estimation methods for the parameters.
<code>lowerbounds</code>	lower bounds
<code>upperbounds</code>	upper bounds
<code>transform</code>	transformation function
<code>vario</code>	obsolete
<code>self</code>	list containing <ul style="list-style-type: none"> • <code>model</code> the variogram or covariance model • <code>residuals</code> NULL • <code>ml.value</code> the likelihood value for the model
<code>plain</code> , <code>sqrt.nr</code> , <code>sd.inv</code> , <code>internal</code> , <code>ml</code> , <code>reml</code>	see <code>self</code> ; exception is <code>ml</code> , where the <code>residuals</code> are given instead of NULL.

Acknowledgement

Thanks to Paulo Ribeiro for hints and comparing the preliminary versions of `fitvario` in `RandomFields V1.0` to `likfit` of the package `geoR` whose homepage is at <http://www.est.ufpr.br/geoR/>.

Note

This function does not depend on the value of `RFparameters()$PracticalRange`. The function `fitvario` always uses the standard specification of the covariance model as given in `CovarianceFct`.

Further, the function has implemented accelerations if the model is simple. E.g., if there is a common variance to estimated and the definition by lists is used, then the leading model should be ‘\$’ with `var=NA`.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

- Least squares and mle methods
Cressie, N.A.C. (1993) *Statistics for Spatial Data*. New York: Wiley.
- Related software
Ribeiro, P. and Diggle, P. (2001) Software for geostatistical analysis using R and S-PLUS: `geoR` and `geoS`, version 0.6.15. <http://www.maths.lancs.ac.uk/~ribeiro/geoR.html>.
- REML (rml)
LaMotte, L.R. (2007) A direct derivation of the REML likelihood function *Statistical Papers* **48**, 321-327.

See Also

[Covariance](#), [CovarianceFct](#), [GetPracticalRange](#), [parampositions](#) [RandomFields](#), [weather](#).

Examples

```

model <- "gencauchy"
param <- c(0, 1, 0, 1, 1, 2)
estparam <- c(0, NA, 0, NA, NA, 2) ## NA means: "to be estimated"
## sequence in 'estparam' is
## mean, variance, nugget, scale, (+ further model parameters)
## So, mean, variance, and scale will be estimated here.
## Nugget is fixed and equals zero.
points <- 100
x <- runif(points,0,3)
y <- runif(points,0,3) ## 300 random points in square [0, 3]^2
## simulate data according to the model:
d <- GaussRF(x=x, y=y, grid=FALSE, model=model, param=param, n=1000) #1000
## fit the data:

Print(fitvario(x=cbind(x,y), data=d, model=model, param=estparam,
```

```

lower=c(0.1, 0.1, 0.1), upper=c(1.9, 5, 2)))

#####
## The next two estimations give about the same result.
## For the first the sill is fixed to 1.5. For the second the sill
## is reached if the estimated variance is smaller than 1.5
estparam <- c(0, NA, NA, NA, NA, NA)
## Not run:
Print(v <- fitvario(x=cbind(x,y), data=d, model=model, param=estparam,
  sill=1, use.nat=FALSE)) ## gencauchy works better with use.nat=FALSE

## End(Not run)

estmodel <- list("+",
  list("$", var=NA, scale=NA,
    list("gencauchy", alpha=NA, beta=NA)
  ),
  list("$", var=NA, list("nugget"))
)
parampositions(model=estmodel, dim=2)
f <- function(variab) c(variab, max(0, 1.0 - variab[1]))
## Not run:
Print(v2 <- fitvario(x=cbind(x,y), data=d, model=estmodel,
  lower = c(TRUE, TRUE, TRUE, TRUE, FALSE),
  transform=f, use.nat=FALSE))

## End(Not run)

#####
## estimation of coupled parameters (alpha = beta, here)
# source("RandomFields/tests/source.R")
f <- function(param) param[c(1:3,3,4)]
## Not run:
Print(fitvario(x=cbind(x,y), data=d, model=estmodel,
  lower=c(TRUE, TRUE, TRUE, FALSE, TRUE),
  transform=f))

## End(Not run)

#####
## estimation in a anisotropic framework

x <- y <- (1:6)/4
model <- list("$", aniso=matrix(nc=2, c(4,2,-2,1)), var=1.5,
  list("exp"))
z <- GaussRF(x=x, y=y, grid=TRUE, model=model, n=10)
estmodel <- list("$", aniso=matrix(nc=2, c(NA,NA,-2,1)), var=NA,
  list("exp"))
Print(fitvario(as.matrix(expand.grid(x, y)), data=z,
  model=estmodel, nphi=20))

```

```
#####
## estimation with trend (formula)
model <- list("$", var=1, scale=2, list("gauss"))
estmodel <- list("$", var=NA, scale=NA, list("gauss"))
x <- seq(-pi,pi,pi/2)
n <- 5
data <- GaussRF(x, x, gridtri=FALSE, model=model,
               trend=function(X1,X2) sin(X1) + 2*cos(X2),n=n)
Print(v <- fitvario(x, x, data=data, gridtrip=FALSE,
                  model=estmodel,
                  trend=~sin(X1)+cos(X1)+sin(X2)+cos(X2)))

#####
## estimation of anisotropy matrix with two identical ##
## diagonal elements ##
## Not run:
x <- c(0, 5, 0.4)
model <- list("$", var=1, scale=1, list("exponential"))
z <- GaussRF(x, x, x, model=model, gridtriple=TRUE, n=10, Print=2)

est.model <- list("+",
                 list("$", var=NA, aniso=diag(c(NA, NA, NA)), list("exponen")),
                 list("$", var=NA, list("nugget")))
parampositions(est.model, dim=3)
trafo <- function(variab) {variab[c(1:2, 2:4)]}
lower <- c(TRUE, TRUE, FALSE, TRUE, TRUE) # which parameter to be estimated
fitlog <- fitvario(x, x, x, gridtriple=TRUE, data=z, model=est.model,
                  transform=trafo, lower=lower)

str(fitlog$m1)

## End(Not run)
```

Forest

Simulation study for Forest data

Description

– to be done – currently for internal use only

Usage

Forest(x, model, percent, ref.edge, coarse, gausspercent, debug=FALSE)

Arguments

x –
 model –
 percent –
 ref.edge –
 coarse –
 gausspercent –
 debug –

Details

–

Value

–

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

–

fractal.dim	<i>fractal dimension</i>
-------------	--------------------------

Description

The function estimates the fractal dimension of a process

Usage

```
fractal.dim(x, y = NULL, z = NULL, data,
            grid=TRUE, gridtriple = FALSE,
            bin= seq(min(ct$x[3, ]) / 2,
                    min((ct$x[2,]-ct$x[1,]) / 4, vario.n * min(ct$x[3,]) + 1),
                    min(ct$x[3,])),
            vario.n=5,
            sort=TRUE,
```

```

fft.m = c(65, 86), ## in % of range of l.lambda
fft.max.length=Inf,
fft.max.regr=150000,
fft.shift = 50, # in %; 50:WOSA; 100: no overlapping
method=c("variogram", "fft"),
mode=c("plot", "interactive"),
pch=16, cex=0.2, cex.main=0.85,
PrintLevel = RFparameters()$Print,
height=3.5,
...)
```

Arguments

x	matrix of coordinates, or vector of x coordinates; if x is not given a grid with unit grid length is assumed
y	vector of y coordinates
z	vector of z coordinates
data	the values measured.
grid	determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details. grid does not apply for T.
gridtriple	logical. Only relevant if grid=TRUE. If gridtriple=TRUE then x, y, and z are of the form c(start,end,step); if gridtriple=FALSE then x, y, and z must be vectors of ascending values.
bin	sequence of bin boundaries for the empirical variogram
vario.n	first vario.n value of the empirical variogram are used for the regression fit that are not NA.
sort	If TRUE then the coordinates are permuted such that the largest grid length is in x-direction; this is of interest for algorithms that slice higher dimensional fields into one-dimensional sections.
fft.m	numeric vector of two components; interval of frequencies for which the regression should be calculated; the interval is given in percent of the range of the frequencies in log scale.
fft.max.length	The first dimension of the data is cut into pieces of length fft.max.length. For each piece the FFT is calculated and then the average for all pieces is taken. The pieces may overlap, see the parameter fft.shift.
fft.max.regr	If the fft.m is too large, parts of the regression fit will take a very long time. Therefore, the regression fit is calculated only if the number points given by fft.m is less than fft.max.regr.
fft.shift	This parameter is given in percent [of fft.max.length] and defines the overlap of the pieces defined by fft.max.length. If fft.shift = 50 the WOSA estimator is given; if fft.shift = 100 no overlap exist.
method	list of implemented methods to calculate the fractal dimension; see Details
mode	character. A vector with components 'nographics', 'plot', or 'interactive':

'**nographics**' no graphical output
 '**plot**' the regression line is plotted
 '**interactive**' the regression domain can be chosen interactively

Usually only one mode is given. Two modes may make sense in the combination c("plot", "interactive") in which case all the results are plotted first, and then the interactive mode is called. In the interactive mode, the regression domain is chosen by two mouse clicks with the left mouse; a right mouse click leaves the plot.

pch	vector or scalar; sign by which data are plotted.
cex	vector or scalar; size of pch.
cex.main	The size of the title in the regression plots.
PrintLevel	integer. If PrintLevel is 0 nothing is printed. If PrintLevel=1 error messages are printed. If PrintLevel=2 warnings and the regression results are given. If PrintLevel>2 tracing information is given.
height	height of the graphics window
...	graphical parameters

Details

The function calculates the fractal dimension by various methods:

- variogram method
- Fourier transform

Value

The function returns a list with elements vario, fft corresponding to the 2 methods given in the Details.

Each of the elements is itself a list that contains the following elements.

x	the x-coordinates used for the regression fit
y	the y-coordinates used for the regression fit
regr	the return list of the lm
sm	smoothed curve through the (x,y) points
x.u	NULL or the restricted x-coordinates given by the user in the interactive plot
y.u	NULL or y-coordinates according to x.u
regr.u	NULL or the return list of lm for x.u and y.u
D	the fractal dimension
D.u	NULL or the fractal dimension corresponding to the user's regression line

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

variogram method

- Constantine, A.G. and Hall, P. (1994) Characterizing surface smoothness via estimation of effective fractal dimension. *J. R. Statist. Soc. Ser. B* **56**, 97-113.

fft

- Chan, Hall and Poskitt (1995)

See Also

[CovarianceFct](#), [hurst](#)

GaussRF

Gaussian Random Fields

Description

These functions simulate stationary spatial and spatio-temporal Gaussian random fields using turning bands/layers, circulant embedding, direct methods, and the random coin method.

Usage

```
GaussRF(x, y=NULL, z=NULL, T=NULL, grid=!missing(gridtriple), model, param,
        trend=NULL, method=NULL, n=1, register=0, gridtriple,
        paired=FALSE, ...)
```

```
InitGaussRF(x, y=NULL, z=NULL, T=NULL, grid=!missing(gridtriple), model,
            param, trend=NULL, method=NULL, register=0, gridtriple)
```

Arguments

x	matrix of coordinates, or vector of x coordinates
y	vector of y coordinates
z	vector of z coordinates
T	vector of time coordinates, may only be given if the random field is defined as an anisotropic random field, i.e. if <code>model=list(list(model=, var=, k=, aniso=), ...)</code> . T must always be given in the <code>gridtriple</code> format, independently how the spatial part is defined.
grid	logical; determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details. <code>grid</code> does not apply for T.
model	string or list; covariance or variogram model, see CovarianceFct , or type PrintModelList() to get the list of all implemented models; see Details.

param	vector or matrix of parameters or missing, see Details and CovarianceFct ; The simplest form is that param is vector of the form <code>param=c(NA, variance, nugget, scale, ...)</code> , in this order; The dots ... stand for additional parameters of the model.
trend	trend surface: number (mean) or a vector of length $d + 1$ (linear trend $a_0 + a_1x_1 + \dots + a_dx_d$), or function; you have the choice of using either x, y, z or X1, X2, X3, ... as spatial variables, as time variable T should be chosen
method	NULL or string; method used for simulating, see RFMethods , or type <code>PrintMethodList()</code> to get all options. If model is given as list then method may not be set if <code>model[[i]]\$method, i = 1, 3, ..</code> is given, and vice versa. However, a global parameter method and specific methods may be given, e.g. <code>list(list(model=..., method="TBM3"), ..., method="ci")</code> ; then the specific ones overwrite the global method.
n	number of realisations to generate
register	0:9; place where intermediate calculations are stored; the numbers are aliases for 10 internal registers
gridtriple	logical. Only relevant if <code>grid=TRUE</code> . If <code>gridtriple=TRUE</code> then x, y, and z are of the form <code>c(start, end, step)</code> ; if <code>gridtriple=FALSE</code> then x, y, and z must be vectors of ascending values
paired	logical. If TRUE then the second half of the simulations is obtained by only changing the signs of all the standard Gaussian random variables, on which the first half of the simulations is based. ("Antithetic pairs".)
...	RFparameters that are locally used only.

Details

GaussRF can use different methods for the simulation, i.e., circulant embedding, turning bands, direct methods, and random coin method. If `method=NULL` then GaussRF searches for a valid method. GaussRF may not find the fastest method neither the most precise one. It just finds any method among the available methods. (However it guesses what is a good choice.) See [RFMethods](#) for further information. Note that some of the methods do not work for all covariance or variogram models.

- An isotropic random field is created by GaussRF where model is the covariance or variogram model and the parameter is `param=c(mean, variance, nugget, scale, ...)`. Alternatively the trend can be given; then `param=c(variance, nugget, scale, ...)`.
- Nested models can be defined in the same way as a nested [CovarianceFct](#). If the trend is not given it is set to 0.
- An anisotropic random field (i.e. zonal anisotropy, geometrical anisotropy, separable models, non-separable space-time models) and a random field based on multiplicative or nested models is defined as in the case of an anisotropic [CovarianceFct](#). If the trend is not given it is set to 0. The method may be specified by the global method or for each model separately, as additional parameter method for each entry of the list; note that methods can not be mixed within a multiplicative part.

If `model=list(list(model=, var=, k=, aniso=), ...)` then a time component might be given. In case of `model="nugget"`, `aniso` must still be given as a matrix. Namely if `aniso` is a singular matrix then a zonal nugget effect is obtained.

GaussRF calls initially `InitGaussRF`, which does some basic checks on the validity of the parameters. Then, `InitGaussRF` performs some first calculations, like the first Fourier transform in the circulant embedding method or the matrix decomposition for the direct methods. Random numbers are not involved. GaussRF then calls `DoSimulateRF` which uses the intermediate results and random numbers to create a simulation.

When `InitGaussRF` checks the validity of the parameters, it also checks whether the previous simulation has had the same specification of the random field. If so (and if `RFparameters()$STORING==TRUE`), the stored intermediate results are used instead of being recalculated.

Comments on specific parameters:

- `grid=FALSE` : the vectors `x`, `y`, and `z` are interpreted as vectors of coordinates
- `(grid=TRUE) && (gridtriple=FALSE)` : the vectors `x`, `y`, and `z` are increasing sequences with identical lags for each sequence. A corresponding grid is created (as given by `expand.grid`).
- `(grid=TRUE) && (gridtriple=TRUE)` : the vectors `x`, `y`, and `z` are triples of the form `(start,end,step)` defining a grid (as given by `expand.grid(seq(x$start,x$end,x$step), seq(y$start,yend,ystep), seq(z$start,z$end,z$step))`)
- `register` is a parameter which may never be used by most of the users (please let me know if you use it!). In other words, the package will work fine if you ignore this parameter. The parameter `register` is of interest in the following situation. Assume you wish to create sequentially several realisations of two random fields Z_1 and Z_2 that have different specifications of the covariance/variogram models, i.e. $Z_1, Z_2, Z_1, Z_2, \dots$. Then, without using different registers, the algorithm will not be able to profit from already calculated intermediate results, as the specifications of the covariance/variogram model change every time. However, using different registers allows for profiting from up to 10 stored intermediate results.
- The strings for `model` and `method` may be abbreviated as long as the abbreviations match only one option. See also `PrintModelList()` and `PrintMethodList()`
- Further control parameters for the simulation are set by means of `RFparameters(...)`.

Value

`InitGaussRF` returns 0 if no error has occurred and a positive value if failed.

The object returned `GaussRF` and `DoSimulateRF` depends on the parameters `n` and `grid`:

if `vdim > 1` the `vdim`-variate vector makes the first dimension

if `grid=TRUE` an array of the dimension of the random field makes the next dimensions. Else if no time component is given, then the values are passed as a single vector. Else if the time component is given the next 2 dimensions give space and time.

if `n > 1` the repetitions make the last dimension

Note

The algorithms for all the simulation methods are controlled by additional parameters, see `RFparameters()`. These parameters have an influence on the speed of the algorithm and the precision of the result. The default parameters are chosen such that the simulations are fine for many models and their parameters. If in doubt modify the example in `EmpiricalVariogram()` to check the precision.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

Yindeng Jiang <jiangyindeng@gmail.com> (circulant embedding methods ‘cutoff’ and ‘intrinsic’)

References

See [RFMethods](#) for the references.

See Also

[Covariance](#), [CovarianceFct](#), [DeleteRegister](#), [DoSimulateRF](#), [GetPracticalRange](#), [EmpiricalVariogram](#), [fitvario](#), [MaxStableRF](#), [RFMethods](#), [RandomFields](#), [RFparameters](#), [ShowModels](#),

Examples

```
#####
##                                     ##
## Examples using the symmetric stable model, also called ##
## "powered exponential model"          ##
##                                     ##
#####
PrintModelList()  ## the complete list of implemented models
model <- "stable"
mean <- 0
variance <- 4
nugget <- 1
scale <- 10
alpha <- 1  ## see help("CovarianceFct") for additional
            ## parameters of the covariance functions
step <- 1   ## nicer, but also time consuming if step <- 0.1
x <- seq(0, 20, step)
y <- seq(0, 20, step)
f <- GaussRF(x=x, y=y, model=model, grid=TRUE,
            param=c(mean, variance, nugget, scale, alpha))
image(x, y, f)

#####
## ... using gridtriple
step <- 1   ## nicer, but also time consuming if step <- 0.1
x <- c(0, 20, step)  ## note: vectors of three values, not a
y <- c(0, 20, step)  ## sequence
f <- GaussRF(grid=TRUE, gridtriple=TRUE,
            x=x ,y=y, model=model,
            param=c(mean, variance, nugget, scale, alpha))
image(seq(x[1],x[2],x[3]), seq(y[1],y[2],y[3]), f)
```

```
#####
## arbitrary points
x <- runif(100, max=20)
y <- runif(100, max=20)
z <- runif(100, max=20) # 100 points in 3 dimensional space
(f <- GaussRF(grid=FALSE, Print=5,
              x=x, y=y, z=z, model=model,
              param=c(mean, variance, nugget, scale, alpha)))

#####
## usage of a specific method
## -- the complete list can be obtained by PrintMethodList()
x <- runif(100, max=20) # arbitrary points
y <- runif(100, max=20)
(f <- GaussRF(method="dir", # direct matrix decomposition
              x=x, y=y, model=model, grid=FALSE,
              param=c(mean, variance, nugget, scale, alpha)))

#####
## simulating several random fields at once
step <- 1 ## nicer, but also time consuming if step <- 0.1
x <- seq(0, 20, step) # grid
y <- seq(0, 20, step)
f <- GaussRF(n=3, # three simulations at once
            x=x, y=y, model=model, grid=TRUE,
            param=c(mean, variance, nugget, scale, alpha))
image(x, y, f[, ,1])
image(x, y, f[, ,2])
image(x, y, f[, ,3])

#####
##                               ##
##   Examples using the extended definition form   ##
##                               ##
##                               ##
#####

## library(RandomFields, lib=~"/TMP"); RFparameters(Print=6)
x <- (0:100)/10
m <- matrix(c(1,2,3,4),ncol=2)/5
model <- list("$", aniso=m,
            list("x",
                list("power", k=2),
                list("sph"))
            )
z <- GaussRF(x=x, y=x, grid=TRUE, model=model, me="TBM3")
Print(c(mean(as.double(z)),var(as.double(z))))
```

```

image(z,zlim=c(-3,3))

## to know more what GaussRF does, use Print
## TMB can be very slow. To trace the iteration, use every
##
z <- GaussRF(x=x, y=x, grid=TRUE, model=model, me="TBM3",
             Print=3, every=100)
image(z,zlim=c(-3,3))
## here, GaussRF uses 'direct decomp' to simulate on the line
## and the square root of the covariance matrix is
## calculated by the Cholesky decomposition

## non-separable space-time model applied for two space dimensions
## note that tbm method works in some special cases.
## library(RandomFields, lib=~/"TMP")
x <- y <- seq(0, 7, if (interactive()) 0.05 else 0.2)
T <- c(1,32,1) * 10      ## note necessarily gridtriple definition
model <- list("$", aniso=diag(c(3, 3, 0.02)),
             list("nsst", k1=2,
                 list("gauss"),
                 list("genB", k=c(1, 0.5))
             ))
z <- GaussRF(x=x, y=y, T=T, grid=TRUE, model=model,
            method="ci", CE.strategy=1,
            CE.trials=if (interactive()) 4 else 1)
r1 <- function() if (interactive()) readline("Press return")
for (i in 1:dim(z)[3]) { image(z[,,i], zlim=range(z)); r1();}
for (i in 1:dim(z)[2]) { image(z[,i,], zlim=range(z)); r1();}
for (i in 1:dim(z)[1]) { image(z[i,,], zlim=range(z)); r1();}

#####
##                                     ##
## Example of a 2d random field based on ##
## covariance functions valid in 1d only ##
##                                     ##
#####

x <- seq(0, 10, 1/10)
model <- list("*",
             list("$", aniso=matrix(nr=2, c(1, 0)),
                 list("fractgauss", k=0.5)),
             list("$", aniso=matrix(nr=2, c(0, 1)),
                 list("fractgauss", k=0.9))
             )
z <- GaussRF(x, x, grid=TRUE, gridtriple=FALSE, model=model)
image(x, x, z)

```

```
#####
##                                     ##
##           Brownian motion           ##
##           (using Stein's method)    ##
##                                     ##
#####
# 1d
kappa <- 1 # in [0,2)
z <- GaussRF(x=c(0, 10, 0.001), grid=TRUE, Print=5,
             model=list("fractalB", kappa))
plot(z, type="l")

# 2d
step <- 0.3 ## nicer, but also time consuming if step = 0.1
x <- seq(0, 10, step)
kappa <- 1 # in [0,2)
z <- GaussRF(x=x, y=x, grid=TRUE, model=list("fractalB", kappa))
image(z,zlim=c(-3,3))

# 3d
x <- seq(0, 3, step)
kappa <- 1 # in [0,2)
z <- GaussRF(x=x, y=x, z=x, grid=TRUE,
             model=list("fractalB", kappa))
r1 <- function() if (interactive()) readline("Press return")
for (i in 1:dim(z)[1]) { image(z[i,,]); r1();}

#####
## This example shows the benefits from stored, ##
## intermediate results: in case of the circulant ##
## embedding method, the speed is doubled in the second ##
## simulation. ##
#####

RFparameters(Storing=TRUE)
y <- x <- seq(0, 50, 0.2)
(p <- c(runif(3), runif(1)+1))
ut <- system.time(f <- GaussRF(x=x,y=y,grid=TRUE,model="exponen",
                              method="circ", param=p))

image(x, y, f)

cat("system time (first call)", format(ut,dig=3),"\n")

# second call with the same parameters can be much faster:
ut <- system.time(f <- DoSimulateRF())
image(x, y, f)
```

```

cat("system time (second call)", format(ut,dig=3),"\n")

#####
##                                     ##
##   Example how the cutoff method can be set   ##
##   explicitly using hypermodels               ##
##                                     ##
#####

## NOTE: this feature is still in an experimental stage
##       which has not been yet tested intensively;
## further: parameters and algorithms may change in
##       future.

## library(RandomFields, lib=~ /TMP");source("~ /R/cran/RandomFields/tests/source.R")
## simulation of the stable model using the cutoff method
#RFparameters(Print=8, Storing=FALSE)
x <- seq(0, 1, 1/24) # nicer pictures with 1/240
scale <- 1.0
modell <- list("$", scale=scale, list("stable", alpha=1.0))
rs <- get(".Random.seed", envir=.GlobalEnv, inherits = FALSE)
z1 <- GaussRF(x, x, grid=TRUE, gridtriple=FALSE,
              model=modell, n=1, meth="cutoff", Storing=TRUE)
(size <- GetRegisterInfo(meth=c("cutoff", "circ"))$S$size)
(cut.off.param <-
  GetRegisterInfo(meth=c("cutoff", "circ"), modelname="cutoff")$param)

image(x, x, z1)

## simulation of the same random field using the circulant
## embedding method and defining the hypermodel explicitly
modell2 <- list("$", scale = scale,
              list("cutoff", diam=cut.off.param$diam, a=cut.off.param$a,
                  list("stable", alpha=1.0))
            )
assign(".Random.seed", rs, envir=.GlobalEnv)
z2 <- GaussRF(x, x, grid=TRUE, gridtriple=FALSE, model=modell2,
              meth="circulant", n=1, CE.mmin=size, Storing=TRUE)
image(x, x, z2)
Print(range(z1-z2)) ## essentially no difference between the fields!

## library(RandomFields)

#####
## The cutoff method simulates on a torus and a (small) ##
## rectangle is taken as the required simulation.      ##
##                                                     ##
## The following code shows a whole such torus.        ##
#####

```

```

## The main part of the code sets local.dependent=TRUE and ##
## local.mmin to multiples of the basic rectangle lengths ##
#####

# definition of the realisation
RFparameters(CE.useprimes=FALSE)
x <- seq(0, 2, len=4) # better 20
y <- seq(0, 1, len=5) # better 40
grid.size <- c(length(x), length(y))
model <- list("$", var=1.1, aniso=matrix(nc=2, c(2, 1, 0.5, 1)),
           list(model="exp"))

# determination of the (minimal) size of the torus
InitGaussRF(x, y, model=model, grid=TRUE, method="cu")
ce.info.size <- GetRegisterInfo(meth=c("cutoff", "circ"))$Ssize
blocks <- ceiling(ce.info.size / grid.size / 4) *4
(size <- blocks * grid.size)

# simulation and plot of the torus
z <- GaussRF(x, y, model=model, grid=TRUE, method="cu", n=prod(blocks) * 2,
            local.dependent=TRUE, local.mmin=size)[,c(TRUE, FALSE)]
hei <- 8
do.call(getOption("device"),
        list(hei=hei, wid=hei / blocks[2] / diff(range(y)) *
            blocks[1] * diff(range(x))))

close.screen(close.screen())
sc <- matrix(nc=blocks[1], split.screen(rev(blocks)), byrow=TRUE)
sc <- as.vector(t(sc[nrow(sc):1, ]))

for (i in 1:prod(blocks)) {
  screen(sc[i])
  par(mar=rep(1, 4) * 0.0)
  image(z[, , i], zlim=c(-3, 3), axes=FALSE, col=rainbow(100))
}

#####
## Simulating with trend (as function) ##
#####

x <- seq(-5,5,0.1)
z <- GaussRF(x=x, y=x, model = "exponential", param=c(1,0,1), grid=TRUE,
            trend=function(x,y) 3*sin(x)*cos(y))
colors=heat.colors(1000)
image(x,x,z,col=colors)

#####

```

```
## Simulating with linear trend surface      ##
#####

x <- seq(-5,5,0.1)
##trend surface: 3x - y
z <- GaussRF(x=x, y=x, model = "cubic", param=c(1,0,2), grid=TRUE,
             trend=c(0,1,-1))
colors=heat.colors(1000)
persp(x,x,z, phi=30, theta=-3)
```

getactions

Get input behaviour

Description

The functions return values stored by [useraction](#), [Locator](#) and [Readline](#)

Usage

```
getactions()
getactionlist()
```

Value

`getactionlist` returns a list of stored values created by [Locator](#) and [Readline](#)

`getactions` returns a list of all parameters that influence the behaviour of [Locator](#) and [Readline](#).

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[Locator](#), [Readline](#), [useraction](#),

Examples

```
## see useraction
```

GetPracticalRange *Determination of the practical range*

Description

the function returns the practical range of a covariance model, i.e. the distance for which the model with standard parameters (variance=1, nugget=0, scale=1) values 0.05

Usage

```
GetPracticalRange(model, param, dim=1)
```

Arguments

model	covariance model in the new list not; the model must be primitive, i.e. may not call other models. See Covariance and CovarianceFct for details.
param	optional vector of parameters for the model, see CovarianceFct
dim	space-time dimension; except for some few anisotropic models, this parameter does not have an effect

Details

In case the practical range has to be determined numerically, only a rough approximation is provided.

The function can only be applied to isotropic models

Value

real number (the practical range)

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Goovaerts, P. (1997) *Geostatistics for Natural Resources Evaluation*. New York: Oxford University Press.

See Also

[CovarianceFct](#), [RandomFields](#), [RFparameters](#),

Examples

```

GetPracticalRange(list("exponential"))
GetPracticalRange(list("whittle", nu=2))
try(GetPracticalRange(list("bessel", nu=-0.5))) # leads to an error

```

GetRegisterInfo *Internal information*

Description

The function returns internal information about the simulation of a random field and the calculation of covariance functions

Usage

```

GetRegisterInfo(register, ignore.active = FALSE, max.elements=10^6,
               meth=NULL, modelname=NULL)
GetModelInfo(register, modelreg, level=3, gatter=FALSE)
GetModel(register, modelreg, modus=0)

```

Arguments

register	-2, -1, 0:9; (-1 and -2 only work with GetModelInfo) place where intermediate calculations are stored; the numbers 0:9 are aliases for 10 internal registers, see also GaussRF
modelreg	value in 0..4 with the following meaning: <ul style="list-style-type: none"> • 0 : register for the functions Covariance, CovarianceFct, ect • 1 : register for simulation • 2 : register for covariance calculation within Kriging • 3 : register for MLE (likelihood) • 4 : register for MLE (bounds) Positive values refer to the registers, see GaussRF .
ignore.active	logical. If FALSE and the register has non-active flag (because of an error or it is deleted) then a list is returned that contains only the element active=FALSE. Otherwise, a more complete list is returned – only for internal debugging purposes. Warning: ignore.active=TRUE may cause failure of R.
max.elements	integer; since GetRegisterInfo might be a dump of simulation that needs a huge amount of memory and since all entries are copied, the maximal amount of available memory might be easily exceeded. Therefore, only the size of the critical parts are returned and not the vector or matrix itself, if the number of elements exceeds max.elements.
meth	vector of strings. If meth is given GetRegisterInfo returns not the whole register info but only part method part.

modelName	string. If modelName is given then GetRegisterInfo returns the first appearance of the covariance model with name modelName. If meth is given then the model within the method is returned.
level	integer; level of details
modus	integer; see details
gatter	boolean; only for very advanced interests.

Details

GetRegisterInfo(register, ignore.active=TRUE) is useful for debugging and specialists' need to control the algorithm, see the examples in [RFparameters](#) and [GaussRF](#).

If [RFparameters\(\)](#)\$Storage=FALSE then values of the internal registers are not kept if [GaussRF](#) or [DoSimulateRF](#) has been called. Hence GetRegisterInfo cannot provide any information.

GetModelInfo returns the complete information on the internal model structure. It allows for register=-1 returning the model structure for the last use of [CovarianceFct](#), [sophisticated](#) models or [Variogram](#) or similar commands. register=-2 is for internal use only.

GetModel returns only the parts of the internal model structure that have been defined by the user. The modus is ignored if PracticalRange=FALSE in [RFparameters](#). In case PracticalRange=TRUE the modus has the following effects

1. 0the model is returned without the explicite scale transformation
2. 1the model is returned the way it is stored, including the scale transformation
3. 2the scale transformation of the the practical range is included, but a simplified model will be returned

Value

List of internal information is returned.

Note

Put Storing=TRUE, see [RFparameters](#) if you like to have more internal information in case of an expected failure of an initialisation of a random field simulation.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[GaussRF](#)

Examples

```
GaussRF(1:4, grid=TRUE, model="exp", param=c(1,2,3,4), Storing=TRUE)
Print(GetRegisterInfo(0))

# finally, a register that could not be successfully initialised
try(GaussRF(runif(4), grid=FALSE, model="exp", param=c(1,2,3,4),
           me="ci", Storing=TRUE))
if (interactive()) Print(GetRegisterInfo(0, TRUE))
```

host	<i>System calls</i>
------	---------------------

Description

The functions `hostname` and `pid` return the host name and the PID, respectively.

Usage

```
hostname()
```

```
pid()
```

Details

If R runs on a unix platform the host name and the PID are returned, otherwise the empty string and `naught`, respectively.

Value

`hostname` returns a string

`pid` returns an unsigned integer

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

hurst *Hurst coefficient*

Description

The function estimates the Hurst coefficient of a process

Usage

```
hurst(x, y = NULL, z = NULL, data,
      gridtriple = FALSE, sort = TRUE,
      block.sequ = unique(round(exp(seq(log(min(3000, dim[1] / 5)), log(dim[1]),
                                     len=min(100, dim[1]))))),
      fft.m = c(1, min(1000, (fft.len - 1) / 10)),
      fft.max.length = Inf, method=c("dfa", "fft", "var"),
      mode=c("plot", "interactive"), pch=16, cex=0.2, cex.main=0.85,
      PrintLevel=RFparameters()$Print,height=3.5,...)
```

Arguments

x	matrix of coordinates, or vector of x coordinates
y	vector of y coordinates
z	vector of z coordinates
data	the data
gridtriple	logical. Only relevant if grid=TRUE. If gridtriple=TRUE then x, y, and z are of the form c(start,end,step); if gridtriple=FALSE then x, y, and z must be vectors of ascending values
sort	logical. If TRUE then the coordinates are permuted such that the largest grid length is in x-direction; this is of interest for algorithms that slice higher dimensional fields into one-dimensional sections.
block.sequ	ascending sequences of block lengths for which the detrended fluctuation analysis and the variance method is performed.
fft.m	vector of 2 integers; lower and upper endpoint of indices for the frequency which are used in the calculation of the regression line for the periodogram near the origin.
fft.max.length	if the number of points in x-direction is larger than fft.max.length then the segments of length fft.max.length are considered, shifted by fft.max.length/2 (WOSA-estimator).
method	list of implemented methods to calculate the Hurst parameter; see Details
mode	character. A vector with components 'nographics', 'plot', or 'interactive': 'nographics' no graphical output 'plot' the regression line is plotted 'interactive' the regression domain can be chosen interactively

Usually only one mode is given. Two modes may make sense in the combination c("plot", "interactive") in which case all the results are plotted first, and then the interactive mode is called. In the interactive mode, the regression domain is chosen by two mouse clicks with the left mouse; a right mouse click leaves the plot.

pch	vector or scalar; sign by which data are plotted.
cex	vector or scalar; size of pch.
cex.main	font size for title in regression plot, see regression ; only used if mode includes 'plot' or 'interactive'
PrintLevel	integer. If PrintLevel is 0 or 1 nothing is printed. If PrintLevel=2 warnings and the regression results are given. If PrintLevel>2 tracing information is given.
height	height of the graphics window
...	graphical parameters

Details

The function is still in development. Several functionalities do not exist - see the code itself for the current stage.

The function calculates the Hurst coefficient by various methods:

- detrended fluctuation analysis (dfa)
- aggregated variation (var)
- periodogram or WOSA estimator (fft)

Value

The function returns a list with elements dfa, varmeth, fft corresponding to the three methods given in the Details.

Each of the elements is itself a list that contains the following elements.

x	the x-coordinates used for the regression fit
y	the y-coordinates used for the regression fit
regr	the coefficients of the lm
sm	smoothed curve through the (x,y) points
x.u	NULL or the restricted x-coordinates given by the user in the interactive plot
y.u	NULL or y-coordinates according to x.u
regr.u	NULL or the coefficients of lm for x.u and y.u
H	the Hurst coefficient
H.u	NULL or the Hurst coefficient corresponding to the user's regression line

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

detrended fluctuation analysis

- Peng, C.K., Buldyrev, S.V., Havlin, S., Simons, M., Stanley, H.E. and Goldberger, A.L. (1994) Mosaic organization of DNA nucleotides *Phys. Rev. E* **49**, 1685-1689

aggregated variation

- Taqqu, M.S. and Teverovsky, V. (1998) On estimating the intensity of long range dependence in finite and infinite variance time series. In: Adler, R.J., Feldman, R.E., and Taqqu, M.S. *A Practical Guide to Heavy Tails, Statistical Techniques and Applications*. Boston: Birkhaeuser
- Taqqu, M.S. and Teverovsky, V. and Willinger, W. (1995) Estimators for long-range dependence: an empirical study. *Fractals* **3**, 785-798

periodogram

- Percival, D.B. and Walden, A.T. (1993) *Spectral Analysis for Physical Applications: Multitaper and Conventional Univariate Techniques*, Cambridge: Cambridge University Press.
- Welch, P.D. (1967) The use of Fast Fourier Transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms *IEEE Trans. Audio Electroacoustics* **15**, 70-73.

See Also

[CovarianceFct](#), [fractal.dim](#)

Examples

```
x <- runif(1000)
if (interactive()) {
  h <- hurst(1:length(x), data=x)
} else {
  h <- hurst(1:length(x), data=x, mode = "nographics")
}
```

Kriging

Kriging methods

Description

The function allows for different methods of kriging.

Usage

```
Kriging(krige.method, x, y=NULL, z=NULL, T=NULL, grid,
        gridtriple=FALSE, model, param, given, data, trend=NULL, pch=".",
        return.variance=FALSE, allowdistanceZero = FALSE, cholesky=FALSE)
```

Arguments

<code>krige.method</code>	kriging method; currently only 'S' (simple kriging), 'O' (ordinary kriging), 'U' (universal kriging) and 'I' (intrinsic kriging) implemented.
<code>x</code>	$(n \times d)$ matrix or vector of x coordinates; coordinates of n points to be kriged
<code>y</code>	vector of y coordinates.
<code>z</code>	vector of z coordinates.
<code>T</code>	vector in grid triple form for the time coordinates.
<code>grid</code>	logical; determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details.
<code>gridtriple</code>	logical. Only relevant if <code>grid=TRUE</code> . If <code>gridtriple=TRUE</code> then x, y, and z are of the form <code>c(start,end,step)</code> ; if <code>gridtriple=FALSE</code> then x, y, and z must be vectors of ascending values.
<code>model</code>	string; covariance model, see CovarianceFct , or type PrintModelList() to get all options.
<code>param</code>	parameter vector: <code>param=c(mean, variance, nugget, scale, ...)</code> ; the parameters must be given in this order. Further parameters are to be added in case of a parametrised class of covariance functions, see CovarianceFct . The value of mean must be finite in the case of simple kriging, and is ignored otherwise.
<code>given</code>	matrix or vector of points where data are available.
<code>data</code>	the data values given at given; it might be a vector or a matrix. If a matrix is given multivariate data are assumed which are kriged <i>separately</i> .
<code>trend</code>	only used for universal and intrinsic kriging. In case of universal kriging trend is a non-negative integer (monomials up to order k as trend functions), a list of functions or a formula (the summands are the trend functions); you have the choice of using either x, y, z or X1, X2, X3,... as spatial variables; in case of intrinsic kriging trend should be a nonnegative integer which is the order of the underlying model.
<code>pch</code>	Kriging procedures are quite time consuming in general. The character pch is printed after roughly each 80th part of calculation.
<code>return.variance</code>	logical. If FALSE the kriged field is returned. If TRUE a list of two elements, <code>estim</code> and <code>var</code> , i.e. the kriged field and the kriging variances, is returned.
<code>allowdistanceZero</code>	if TRUE then identical locations are slightly scattered
<code>cholesky</code>	if TRUE cholesky decomposition is used instead of LU.

Details

- `grid=FALSE` : the vectors x, y, and z are interpreted as vectors of coordinates
- `(grid=TRUE) && (gridtriple=FALSE)` : the vectors x, y, and z are increasing sequences with identical lags for each sequence. A corresponding grid is created (as given by `expand.grid`).
- `(grid=TRUE) && (gridtriple=TRUE)` : the vectors x, y, and z are triples of the form `(start,end,step)` defining a grid (as given by `expand.grid(seq(x$start,x$end,x$step), seq(y$start,yend,ystep), seq(z$start,z$end,z$step))`)

Value

If `variance.return=FALSE` Kriging returns a vector or matrix of kriged values corresponding to the specification of `x`, `y`, `z`, and `grid`, and `data`.

`data`: a vector or matrix with *one* column

* `grid=FALSE`. A vector of simulated values is returned (independent of the dimension of the random field)

* `grid=TRUE`. An array of the dimension of the random field is returned (according to the specification of `x`, `y`, and `z`).

`data`: a matrix with *at least two* columns

* `grid=FALSE`. A matrix with the `ncol(data)` columns is returned.

* `grid=TRUE`. An array of dimension $d+1$, where d is the dimension of the random field, is returned (according to the specification of `x`, `y`, and `z`). The last dimension contains the realisations.

If `variance.return=TRUE` a list of two elements, `estim` and `var`, i.e. the kriged field and the kriging variances, is returned. The format of `estim` is the same as described above. The format of `var` is accordingly.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Chiles, J.-P. and Delfiner, P. (1999) *Geostatistics. Modeling Spatial Uncertainty*. New York: Wiley.

Cressie, N.A.C. (1993) *Statistics for Spatial Data*. New York: Wiley.

Goovaerts, P. (1997) *Geostatistics for Natural Resources Evaluation*. New York: Oxford University Press.

Wackernagel, H. (1998) *Multivariate Geostatistics*. Berlin: Springer, 2nd edition.

See Also

[CondSimu](#), [Covariance](#), [CovarianceFct](#), [EmpiricalVariogram](#), [RandomFields](#),

Examples

```
###Example 1: Ordinary Kriging
## creating random variables first
## here, a grid is chosen, but does not matter
step <- 0.25
x <- seq(0,7,step)
param <- c(0,1,0,1)
model <- "exponential"
RFparameters(PracticalRange=FALSE)
p <- 1:7
points <- as.matrix(expand.grid(p,p))
data <- GaussRF(points, grid=FALSE, model=model, param=param)
```

```
## visualise generated spatial data
zlim <- c(-2.6,2.6)
colour <- rainbow(100)
image(p, p, xlim=range(x), ylim=range(x),
      matrix(data,ncol=length(p)),
      col=colour,zlim=zlim)

## now: kriging
krige.method <- "0" ## ordinary kriging
z <- Kriging(krige.method=krige.method,
            x=x, y=x, grid=TRUE,
            model=model, param=param,
            given=points, data=data)
image(x,x,z,col=colour,zlim=zlim)
```

 Locator

Graphical Input

Description

Reads the position of the graphics cursor when the (first) mouse button is pressed, or replays from storage.

Usage

```
Locator(n, type="n", info=NULL, ...)
```

Arguments

n	the maximum number of points to locate; the value of n is ignored if
type	One of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines.
info	arbitrary object; tracing information that is useful in case the user likes to edit the stored sequence of inputs; the value of info is considered only if useraction has been called by <code>action="start.register"</code> or <code>action="continue.register"</code>
...	additional graphics parameters

Details

The behaviour of Locator depends on the the value of action set by [useraction](#), see there for more information.

See also [locator](#) for further information on the parameters.

Value

A list containing 'x' and 'y' components which are the coordinates of the identified points in the user coordinate system, i.e., the one specified by `'par("usr")'`.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[getactions](#), [Readline](#), [useraction](#), [userinput](#)

Examples

```
## see useraction
```

MaxStableRF

Max-Stable Random Fields

Description

These functions simulate stationary and isotropic max-stable random fields with unit Frechet margins.

Usage

```
MaxStableRF(x, y=NULL, z=NULL, grid, model, param, maxstable,
            method=NULL, n=1, register=0, gridtriple=FALSE,...)
```

```
InitMaxStableRF(x, y=NULL, z=NULL, grid, model, param, maxstable,
                method=NULL, register=0, gridtriple=FALSE)
```

Arguments

x	matrix of coordinates, or vector of x coordinates
y	vector of y coordinates
z	vector of z coordinates
grid	logical; determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details.
model	string; see CovarianceFct , or type PrintModelList() to get all options; interpretation depends on the value of maxstable, see Details.
param	parameter vector: param=c(mean, variance, nugget, scale,...); the parameters must be given in this order; further parameters are to be added in case of a parametrised class of covariance functions, see CovarianceFct , or be given in one of the extended forms, see Details
maxstable	string. Either 'extremalGauss' or 'BooleanFunction'; see Details.
method	NULL or string; method used for simulating, see RFMethods , or type PrintMethodList() to get all options; interpretation depends on the value of maxstable.
n	number of realisations to generate

register	0:9; place where intermediate calculations are stored; the numbers are aliases for 10 internal registers
gridtriple	logical; if gridtriple=FALSE ascending sequences for the parameters x, y, and z are expected; if gridtriple=TRUE triples of form c(start,end,step) expected; this parameter is used only if grid=TRUE
...	RFparameters that are locally used only.

Details

There are two different kinds of models for max-stable processes implemented:

- maxstable="extremalGauss"
Gaussian random fields are multiplied by independent random factors, and the maximum is taken. The random factors are such that the resulting random field has unit Frechet margins; the specification of the random factor is uniquely given by the specification of the random field. The parameter vector param, the model, and the method are interpreted in the same way as for Gaussian random fields, see [GaussRF](#).
- maxstable="BooleanFunction"
Deterministic or random, upper semi-continuous L_1 -functions are randomly centred and multiplied by suitable, independent random factors; the pointwise maximum over all these functions yields a max-stable random field. The simulation technique is related to the random coin method for Gaussian random field simulation, see [RFMethods](#). Hence, only models that are suitable for the random coin method are suitable for this technique, see [PrintModelList\(\)](#) for a complete list of suitable covariance models.
The only value allowed for method is 'max.MPP' (and NULL), see [PrintMethodList\(\)](#). In the parameter list param the first two entries, namely mean and variance, are ignored. If the nugget is positive, for each point an additional independent unit Frechet variable with scale parameter nugget is involved when building the maximum over all functions.
The model may be defined alternatively in one of the two extended ways as introduced in [CovarianceFct](#) and [GaussRF](#). However only a single model may be given! The model may be anisotropic.

Value

InitMaxStableRF returns 0 if no error has occurred, and a positive value if failed.

MaxStableRF and [DoSimulateRF](#) return NULL if an error has occurred; otherwise the returned object depends on the parameters:

n=1:

* grid=FALSE. A vector of simulated values is returned (independent of the dimension of the random field)

* grid=TRUE. An array of the dimension of the random field is returned.

n>1:

* grid=FALSE. A matrix is returned. The columns contain the realisations.

* grid=TRUE. An array of dimension $d+1$, where d is the dimension of the random field, is returned. The last dimension contains the realisations.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Schlather, M. (2002) Models for stationary max-stable random fields. *Extremes* **5**, 33-44.

See Also

[CovarianceFct](#), [sophisticated](#), [GaussRF](#), [RandomFields](#), [RFMethods](#), [RFparameters](#), [DoSimulateRF](#),
.

Examples

```
n <- 30 ## nicer, but time consuming if n <- 100
x <- y <- 1:n
ms0 <- MaxStableRF(x, y, grid=TRUE, model="exponen",
                  param=c(0,1,0,40), maxstable="extr",
                  CE.force = TRUE)
image(x,y,ms0)
```

Papers

Code used in the papers co-authored by M. Schlather

Description

Code is provided that provides the results in the references below.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>;

References

- Gneiting, T., Kleiber, W. and Schlather, M. (2011) Matern cross-covariance functions for multivariate random fields *J. Amer. Statist. Assoc.* **105**, 1167-1177.
- Gneiting, T., Sevcikova, H., Percival, D.B., Schlather, M., Jiang, Y. (2006) Fast and Exact Simulation of Large Gaussian Lattice Systems in R2: Exploring the Limits. *J. Comput. Graph. Stat.*, **15**, 483-501.
- Schlather, M. (2002) Models for stationary max-stable random fields. *Extremes* **5**, 33-44.

- Scheuerer, M. and Schlather, M. (2011) Covariance Models for Random Vector Fields.
- Schlather, M. (2010) On some covariance models based on normal scale mixtures. *Bernoulli*, **16**, 780-797.

Examples

```
#####
##                                     ##
##      M. Schlather, Extremes 5:1, 33-44, 2002      ##
##                                     ##
#####

col=grey((100:0)/100)
pts <- if (interactive()) 512 else 32
x <- (1:pts) / pts * 10
scalegauss <- 1.5
runif(1)
root <- 1/2
RFparameters(mpp.plus=2*scalegauss, CE.force=TRUE)

save.seed <- .Random.seed
ms1 <- MaxStableRF(x, x, model="gauss", param=c(0, 1, 0, scalegauss),
                  maxstable="Bool", grid=TRUE, Print=3)
image(x, x, ms1^root, col=col)

.Random.seed <- save.seed
scalecirc<- 3.3 # 0.16 0.00 0.18 0.00 0.00
ms2 <- MaxStableRF(x, x, model="sph", param=c(0, 1, 0, scalecirc),
                  maxstable="Bool", grid=TRUE, Print=3)
image(x, x, ms2^root, col=col)

.Random.seed <- save.seed
ms3 <- MaxStableRF(x, x, model="exponen", param=c(0, 1, 0, 1),
                  maxstable="extr", grid=TRUE, Print=3, method="ci")
image(x, x, ms3^root, col=col)

.Random.seed <- save.seed
ms4 <- MaxStableRF(x, x, model="gauss", param=c(0, 1, 0, 1),
                  maxstable="extr", grid=TRUE, Print=3, method="ci")
image(x, x, ms4^root, col=col)

#####
##                                     ##
##      M. Schlather, Bernoulli, 16, 780-797, 2010      ##
##                                     ##
#####
```

```
#####

# % library(RandomFields, lib=~"/TMP")

jpg <- jpeg
Plot <- function(basicname, x, y, z, pixels=200, col=col, delay=1,
                 basicn=10000, T, redo=TRUE, height=5, zi=1, speed=0.3,
                 zlim = range(z)) {
  Print(try(dir.create(basicname)))
  if (length(T)==3) T <- seq(T[1], T[2], T[3])
  cex = 2
  par(mar=c(4,4.3, 0.8, 0.8), cex=0.7)
  for (file in list(TRUE, "jpg")) {
    h <- if (is.logical(file)) height else pixels
    args <- list(TRUE, file, ps=paste(basicname, "/X2", basicname, sep=""),
                quiet=FALSE, height=h, width=h)

    do.call("Dev", args)
    image(x, T, z[,zi,], zlim=zlim, col=col, cex.axis=cex, cex.lab=cex,
          ylab="time")
    Dev(FALSE)

    args <- list(TRUE, file, ps=paste(basicname, "/X3", basicname, sep=""),
                quiet=FALSE, height=h, width=h)
    do.call("Dev", args)
    image(x, y, z[, ,1], zlim=zlim, col=col, cex.axis=cex, cex.lab=cex)
    Dev(FALSE)
  }
  k <- 0
  time <- length(T)
  if (redo) {
    system(paste("rm ", basicname, "/", basicname, "*.jpg", sep=""))

    par(mar=rep(0, 4))
    Dev(TRUE, "jpg", ps=paste(basicname, "/", basicname, sep=""),
        quiet=FALSE, height=pixels * height, width=pixels * height)
    plot(Inf, Inf, axes=FALSE, xlim=c(0,1), ylim=c(0,1))
    Dev(FALSE)

    for (i in 1:(time)) {
      for (j in 1:delay) {
        k <- k + 1
        name <- paste(basicname, "/", basicname, k + basicn, sep="")
        Dev(TRUE, "jpg", ps=name, quiet=FALSE, height=pixels, width=pixels)
        image(x, y, z[, ,i], zlim=zlim, col=col, axes=FALSE)
        Dev(FALSE)
      }
    }
  }
  system(paste("cd ", basicname, ";mencoder -mf fps=30 -ffourcc DX50 -ovc lavc ",
              " -speed ", speed,
              " -lavcopts vcodec=mpeg4:vbitrate=9800:aspect=4/3:vhq:keyint=15",
              " -vf scale=720:576 -o ", basicname, ".avi mf://",
```

```

        basicname, "*.jpg &", sep="")
    }

### Example 10 in Schlather (2010)
basicname <- "coxisham"
y <- x <- seq(0, 10, len=if (interactive()) 256 else 5)
T <- c(0, 25.5, if (interactive()) 0.1 else 5)
col <- c(topo.colors(300)[1:100], cm.colors(300)[c((1:50) * 2, 101:150)])
model <- list("coxisham", mu=c(1, 1), D=matrix(nr=2, c(1, 0.5, 0.5, 1)),
             list("whittle", nu=1)
            )
z <- GaussRF(x, y, T=T, grid =TRUE, spectral.lines=1500, every=10,
            model = model, Print=2)

zlim <- range(z)
time <- dim(z)[3]
for (i in 1:time) {
  cat(i, "\n")
  image(x, y, z[, , i], add=i>1, col=col, zlim=zlim)
}
# load(paste(basicname, ".dat", sep=""))
Plot(basicname, x, y, z, col=col, T=T)
save(file=paste(basicname, "/", basicname, ".dat", sep=""), z)

### Example 13 in Schlather (2010)
basicname <- "moving"
y <- x <- seq(0, 10, len = if (interactive()) 256 else 10)
T <- c(0, 25.5, if (interactive()) 0.1 else 10)
col <- c(topo.colors(300)[1:100], cm.colors(300)[c((1:50) * 2, 101:150)])
model <- list("ave1",
             A = matrix(nc=2, c(0.5, 0, 0, 1)), z= c(2,0),
             list("whittle", nu=1)
            )
z <- GaussRF(x, x, T=T, model=model, grid=TRUE, Print=2, me="av",
            every = 10, CE.trial=2, CE.force=TRUE, CE.maxmem=16777216*8)

zlim <- range(z)
time <- dim(z)[3]
for (i in 1:time) {
  Print(i)
  image(x, y, z[, , i], add=i>1, col=col, zlim=zlim)
}
# load(paste(basicname, ".dat", sep=""))
Plot(basicname, x, y, z, col=col, T=T)
save(file=paste(basicname, "/", basicname, ".dat", sep=""), z)

### Example 16 in Schlather (2010)

```

```

intens <- if (interactive()) 100 else 3 ## 1000 takes a huge amount
##                               ## of time; take smaller values
basicname <- "cyclone"
len <- if (interactive()) 81 else 10
y <- x <- seq(-3, 3, len=len)
T <- seq(0, 6, len=len)
col <- c(topo.colors(300)[1:100], cm.colors(300)[c((1:50) * 2, 101:150)])
model <- list("stp",
             M=matrix(nc=3, rep(0, 9)),
             Sx=list("EtAxxA",
                    E=c(1, 1, 1),
                    alpha = -2 * pi,
                    A=t(matrix(nc=3, c(2, 0, 0, 1, 1, 0, 0, 0, 0))),
                    H = list("Rotat", phi= -2 * pi),
                    phi = list("nonstWM", nu = 1)
             )
z <- GaussRF(x, x, z=T, model=model, grid=TRUE, me="coin", every = 10,
            mpp.intens=intens, mpp.p = 0.1, mpp.beta=3.5, mpp.plus = 8, Print=3)
zlim <- c(-3.5, 3.5)
time <- dim(z)[3]
for (i in 1:time) {Print(i);image(x, y, z[, ,i], add=i>1, col=col, zlim=zlim)}
# load(paste(basicname, ".dat", sep=""))
Plot(basicname, x, y, z, col=col, T=T, pixels=256, zi=0.5 + dim(z)[2]/2,
     speed=0.2, zlim=zlim)
save(file=paste(basicname, "/", basicname, ".dat", sep=""), z)

```

```

#####
##                               ##
##   T. Gneiting, W. Kleiber, M. Schlather, JASA 2011   ##
##                               ##
#####

```

```
## see help("weather")
```

```

#####
##                               ##
##                               ##
##           Figure 1 in           ##
##           Gneiting, T., Sevcikova, H.,           ##
## Percival, D.B., Schlather, M. and Jiang, Y. (2005) ##
##                               ##
#####

```

```

stabletest <- function(alpha, theta, size=if (interactive()) 512 else 20) {
  RFparameters(CE.trials=1, CE.tolIm = 1e-8, CE.tolRe=0, CE.force = FALSE,
              CE.useprimes=TRUE, CE.strategy=0, skipchecks=!FALSE,
              Print=2, Storing=TRUE)
  model <- list("cutoff", diameter=theta, a=1,
              list(model="stable", alpha=alpha))

```

```

CovarianceFct(0, model=model, dim=2)
r <- GetModelInfo(-1)$sub[[1]]$internalq[5] # theor R
x <- c(0, r, r / (size - 1)) * theta
err <- try(InitGaussRF(x, x, grid=TRUE, model=model, meth="circulant"))
return(if (!is.numeric(err) || err != 0) NA else r)
}

alphas <- seq(1.52, 2.0, if (interactive()) 0.02 else 0.1)
thetas <-
  if (interactive()) seq(0.05, 3.5, 0.05) else seq(0.1, 3.5, 0.2)

m <- matrix(NA, nrow=length(thetas), ncol=length(alphas))
for (it in 1:length(thetas)) {
  theta <- thetas[it]
  for (ia in 1:length(alphas)) {
    alpha <- alphas[ia]
    cat("alpha=", alpha, "theta=", theta, "\n")
    print(unix.time(m[it, ia] <- stabletest(alpha=alpha, theta=theta))
    if (is.na(m[it, ia])) break
  }
  if (any(is.finite(m))) image(thetas, alphas, m, col=rainbow(100))
}

#####
##                                     ##
##          M. Scheuerer, M. Schlather, 2011          ##
##                                     ##
#####
# % library(RandomFields, lib=~ /TMP")

my.legend <- function(lu.x, lu.y, zlim, col, cex=1) {
  ## uses already the legend code of R-1.3.0
  cn <- length(col)
  filler <- vector("character", length=(cn-3)/2)
  legend(lu.x, lu.y, y.i=0.03, x.i=0.1,
         legend=c(format(zlim[2], dig=2), filler,
                       format(mean(zlim), dig=2), filler,
                       format(zlim[1], dig=2)),
         lty=1, col=rev(col), cex=cex)
}

my.arrows <- function(xy, z, r, thinning) {
  startx <- as.vector(xy[,1] - r/2*z[as.integer(dim(z)[1]/3) + 1,,])
  starty <- as.vector(xy[,2] - r/2*z[as.integer(dim(z)[1]/3) + 2,,])
  endx <- as.vector(xy[,1] + r/2*z[as.integer(dim(z)[1]/3) + 1,,])
  endy <- as.vector(xy[,2] + r/2*z[as.integer(dim(z)[1]/3) + 2,,])
  startx[c(rep(TRUE, thinning), FALSE)] <- NA
  starty[c(rep(TRUE, thinning), FALSE)] <- NA
  endx[c(rep(TRUE, thinning), FALSE)] <- NA
  endy[c(rep(TRUE, thinning), FALSE)] <- NA
  arrows(x0=startx, y0=starty, x1=endx, y1=endy, length=0.03)
}

```

```

}

x <- c(-3, 3, if (interactive()) 0.049 else 0.5)
nu <- 3
col <- grey(seq(0, 1, 0.01))
thinning <- 21
length.arrow <- 1.5 / thinning
runif(1)
seed <- .Random.seed
eps <- interactive() # true falls eps/pdf drucken

for (modelname in c("divfree", "curlfree")) {
  cat(modelname, "\n")
  model <- list(modelname, list("matern", nu=nu))
  xx <- seq(x[1], x[2], x[3])
  RFparameters(Print=2)

  if (!eps) {
    cf <- Covariance(x=cbind(xx,0), model=model)
    do.call(getOption("device"), list(height=5, width=5))
    par(mfcol=c(1,1))
    j <- 3
    plot(xx, cf[(j-1) * length(xx) + (1 : length(xx)), j])
  }

  .Random.seed <- seed
  z <- GaussRF(x, x,
               grid=TRUE, gridtr=TRUE, model=model, n=1, CE.trial=2,
               Stor=TRUE, me="ci", Print=3, CE.force=!TRUE)
  ## z[1,,] : Potentialfeld
  ## z[2:3,,] : vectorfeld
  ## z[4,,] : div bzw. rot

  if (eps) {
    do.call(getOption("device"), list(height=5, width=5))
    par(mfcol=c(1,1))
  } else {
    do.call(getOption("device"), list(height=5, width=10))
    par(mfcol=c(1,2))
  }
  par(mar=c(2.2,2.5,0.5,0.5), cex.axis=2, bg="white")
  for (no.vectors in c(TRUE,FALSE)) for (i in c(1,2,4)) {
    ## image i=1: Potential feld + Vektorfeld
    ## image i=4: div/rot feld + Vektorfeld
    if (i==1 || i==4) {
      image(xx, xx, col=col, z[i,,] )
      if (no.vectors)
        my.legend(max(xx) - 0.3 * diff(range(xx)),
                  min(xx) + 0.3 * diff(range(xx)),
                  xlim=range(z[i,,]), col=col, cex=1.5)
    } else plot(Inf, Inf, xlim=range(xx), ylim=range(xx))

    if (!no.vectors || i!=1 && i!=4) {

```

```

xy <- as.matrix(expand.grid(xx, xx))
my.arrows(xy, z, length.arrow, thinning)
}

if (all(par()$mfc01==1)) {
  name <- paste(modelname, "_", nu, "_", !no.vectors, "_", i, sep="")
  cat(name, "\n")
  dev.copy2eps(file=paste(name, ".eps", sep=""))
  dev.copy2pdf(file=paste(name, ".pdf", sep=""))
}
}
}

```

parameter.range

Range of model specific parameters

Description

parameter.range returns the range of the parameters specific to a parametrised variogram model or a covariance function

— NOT PROGRAMMED YET —

Usage

```
parameter.range(model, param, dim=1)
```

Arguments

model	the variogram model or covariance function, see CovarianceFct
param	see CovarianceFct
dim	the dimension of the random field

Value

If the model is not valid or not specified for the given dimension, NaN is returned.

If the model does not have additional parameters, NULL is returned. Otherwise a list is returned.

If the parameter space is simple, a list of the following elements is returned

theoretical	The theoretical ranges of the parameters. Note that open and closed intervals are not distinguished here.
practical	The ranges of the parameters usually not exceeded in practical applications.

Both elements are matrices with two rows and the number of columns being the number of parameters.

If the parameter space is complex, the parameter space is divided into rectangular areas and `theoretical` and `practical` are lists where each element is a matrix as above.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[CovarianceFct](#)

Examples

```
parameter.range("exponential", param=NA, dim=1) # NULL
try(parameter.range("power", param=NA, dim=1))
try(parameter.range("power", param=c(NA, NA, NA, NA, 2), dim=2))
Print(parameter.range("gengneiting", param=NA, dim=2))
Print(parameter.range("nsst", param=NA, dim=1))
Print(parameter.range("hyper", param, dim=1))
```

parampositions	<i>Position of the parameters</i>
----------------	-----------------------------------

Description

The function returns the internal positions of the model parameters

Usage

```
parampositions(model, param, trend=NULL, dim, print=1)
```

Arguments

model	see CovarianceFct
param	see CovarianceFct
trend	trend
dim	dimension of the field
print	if 0 only an invisible list is returned

Value

The model is printed and returned where the values of the parameters are the positions in the internal representation.

An error appears if the model can be substantially simplified.


```

trafo <- function(param) {param[2] <- exp(param[2]); param}
lower <- list("+",
             list("$", var=NA, scale=-3, list("exponential")),
             list("$", var=NA, list("nugget")))
upper <- list("+",
             list("$", var=NA, scale=3, list("exponential")),
             list("$", var=NA, list("nugget")))
guess <- list("+",
             list("$", var=1, scale=0, list("exponential")),
             list("$", var=1, list("nugget")))
fitlog <- fitvario(x, gridtriple=TRUE, data=z, model=est.model,
                 transform=trafo, lower=lower, upper=upper,
                 users.guess=guess)
str(fitlog$m1) ## note that scale is returned, not log(scale) !

```

Print

Nice print function returning also the names automatically

Description

prints variable names and the values

Usage

```
Print(...)
```

Arguments

... any object that can be print-ed

Value

prints the names and the values; for vectors cat is used and for lists str

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

Examples

```

a <- 4
b <- list(c=5, g=7)
m <- matrix(1:4, nc=2)
Print(a, b, m)

```

PrintModelList

Information about the implemented covariance models

Description

PrintModelList prints the list of currently implemented models including the corresponding simulation methods

GetModelList returns a matrix of currently implemented models and their simulation methods

GetModelNames returns a list of currently implemented models

Usage

```
PrintModelList(operators=FALSE, internal=FALSE)
```

```
GetModelList(abbr=TRUE, internal=FALSE)
```

```
GetModelNames(stationary = c("any", "stationary", "variogram", "irf",
                             "auxmatrix", "auxvector", "gencovariance",
                             "nonstationary", "genvariogram", "prev.model"),
               isotropy = c("any", "isotropic", "spaceisotropic",
                             "zerospaceiso", "anisotropic", "prev.model"),
               multivariate = -9999:9999,
               operator, normalmix, finiterange, methods, internal=FALSE,
               dim = 1)
```

Arguments

operators	logical. Flag whether operators should be also considered.
internal	logical. Flag whether internal models should be also considered. In case of PrintModelList and internal=2, variants of internal models are also printed.
abbr	logical or numerical. If TRUE the names for the methods are abbreviated. If numerical, abbr gives the number of letters.
stationary	Selection criterion for the models
isotropy	Selection criterion for the models
multivariate	the dimension of the multivariate model
operator	logical. If missing any covariance model is returned; otherwise, the (non)primitive models are returned if FALSE (TRUE); see Covariance for details.
normalmix	logical. If missing any covariance model is returned; otherwise all normal scale mixture models are returned if TRUE
finiterange	logical. If missing any covariance model is returned; otherwise all models with finite range are returned if TRUE; note that some option are excluding, e.g. no normal scale model can have a finite range.

methods	If missing any covariance model is returned; otherwise the models with respective attribute are returned; see PrintMethodList for a complete list of methods
dim	dimension in which the model should be valid; there are some models where the validity in a given dimension depends on the some parameter; those models are returned partially only.

Details

See [CovarianceFct](#) and **sophisticated** models for a description of the models and their use

Value

`PrintModelList` prints a table of the currently implemented covariance functions and the matching methods. `PrintModelList` returns `NULL`.

`GetModelNames` returns a list of implemented models

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[sophisticated](#), [CovarianceFct](#), [GetModel](#)

Examples

```
PrintModelList()
GetModelList(abbr=TRUE)
GetModelNames("stationary", "spaceisotropic")
```

Description

The package `RandomFields` allows for simulating various kinds of random fields, including anisotropic processes. Furthermore, algorithms for conditional simulation and simulation of max-stable random fields are provided.

Additionally, the package includes tools for analysing spatial data: Hurst parameter, fractal dimension, empirical variogram, interactive fitting of parameters, LSQ and MLE estimation of parameters. Basic kriging procedures are also provided.

Starting with version 2.0, it also allows for the simulation of random fields that are non-stationary or multivariate or sophisticated space-time fields. `fitvario` allows for multivariate models and mixed effect models.

There are some changings in the definitions and in the output, see `help("changings")`

Details

The following random fields and related functionalities are provided by the package.

1. stationary and isotropic Gaussian random fields
 - [CondSimu](#) : conditional simulation
 - [CovarianceFct](#), [sophisticated](#) models: covariance functions and variogram models
 - [EmpiricalVariogram](#) : empirical variogram
 - [GaussRF](#) : simulation of Gaussian random fields; nice examples to get familiar with the simulation features of the package;
 - [Kriging](#) : simple and ordinary kriging
 - [fitvario](#) : variogram/covariance function fit by least squares, maximum likelihood and cross validation techniques
 - [PrintMethodList](#) : list of implemented simulation methods
 - [ShowModels](#) : interactive, graphical choice of models – currently not available.
Use ShowModels in version 1.3.x
 - [soil](#) : Soil physical and chemical data; the [example](#) gives a simple geostatistical analysis using features of the package
2. stationary (and isotropic) max-stable random fields
 - [CovarianceFct](#) : covariance models for extremal Gaussian random fields
 - [MaxStableRF](#) : simulation of max-stable random fields
Note: Simulation algorithm for Brown-Resnick processes will come up soon.
3. stationary and isotropic Poisson random fields (not implemented yet)

Data and example code:

- [soil](#) : soil physical data
- [papers](#) : code used in the papers published by the author
- [weather](#) : UWME weather data

Functions used in diverse simulation methods:

- [DeleteRegister](#) : deleting internal registers
- [RFparameters](#) : control parameters (advanced settings)

Functions of general purpose:

- [fractal.dim](#) : estimation of the fractal dimension
- [hurst](#) : estimation of the Hurst parameter
- [Print](#) : nice print function
- [regression](#) : interactive regression plot
- [Locator](#), [Readline](#) : [locator](#) and [readline](#), respectively, with storage and replay functionality

- `sleep.milli` : sets the process into sleeping status

In the beta version, the following functionalities are currently not available:

- `ShowModels`
- numerical evaluation of the covariance function in `tbm2`
- Harvard Rue's Markov fields

Acknowledgement

Many thanks to

- Peter Menck implemented the multivariate circulant embedding for version 2.0.
- Yindeng Jiang <jiangyindeng@gmail.com> implemented the circulant embedding methods 'cutoff' and 'intrinsic' in 2004 for the versions 1.2.
- Martin Maechler, Paulo Ribeiro, and Tilmann Gneiting were proof-reading parts of the code and the help text for the versions 1.0.

Financial support

- V1.0 has been financially supported by the German Federal Ministry of Research and Technology (BMFT) grant PT BEO 51-0339476C during 2000-03.
- V1.0 has been financially supported by the EU TMR network ERB-FMRX-CT96-0095 on "Computational and statistical methods for the analysis of spatial data" in 1999.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather;>

References

This package is announced in and can be cited by:

Schlather, M. (2001) Simulation of stationary and isotropic random fields. *R-News* **1** (2), 18-20.

Readline

Read a Line

Description

Readline reads a line from the terminal or from storage

Usage

```
Readline(prompt="", info=NULL)
```

Arguments

prompt	the string printed when prompting the user for input. Should usually end with a space " "
info	arbitrary object; tracing information that is useful in case the user likes to edit the stored sequence of inputs; the value of info is considered only if <code>useraction</code> has been called by <code>action="start.register"</code> or <code>action="continue.register"</code>

Details

The behaviour of `Readline` depends on the the value of `action` set by `useraction`, see there for more information.

The prompt string will be truncated to a maximum allowed length, normally 256 chars (but can be changed in the source code), since the function is based on `readline`.

Value

A character vector of length one.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[getactions](#), [Locator](#), [readline](#) [useraction](#), [userinput](#),

Examples

```
## see useraction
```

regression

Regression plot

Description

Interactive regression plot

Usage

```
regression(x, y, main, scr, value=function(regr) regr$coeff[2],
  variable="var", mode=c("nographics", "plot", "interactive"),
  regr.select=c("coefficients"), averaging=FALSE, cex.main=0.85,
  col.passive = "grey", col.active = "green",
  col.main.passive = "black", col.main.active = "red",
  col.points.chosen = "lightblue", col.abline = "yellow",
  col.abline.chosen = "darkblue", col.smooth = "red", ...)
```

Arguments

<code>x</code>	x-coordinates of the regression plot
<code>y</code>	y-coordinates
<code>main</code>	main title
<code>scr</code>	<code>split.screen</code> must have been called beforehand; <code>scr</code> gives the screen number
<code>value</code>	function; the characteristic of interest is a function of the regression output.
<code>variable</code>	name of the characteristic of interest, plotted in the graphs
<code>mode</code>	A scalar character of one of the following values 'silent', 'plot', or 'interactive': 'silent' : no output except the return of the result 'plot' : the regression line is plotted 'interactive' : the regression domain can be chosen interactively Any mode includes all the functionalities of the modes that precede in the above enumeration. In the interactive mode, the regression domain is chosen by two mouse clicks with the left mouse; a right mouse click leaves the plot.
<code>regr.select</code>	vector of characters that select elements from the <code>lm</code> return list; these elements are returned by regression.
<code>averaging</code>	logical. If TRUE and <code>y</code> is matrix then the values of <code>y</code> are averaged row-wise beforehand
<code>cex.main</code>	font size of the title
<code>col.passive</code>	colour of the data points if plot is not active
<code>col.active</code>	colour of the data points if plot is active
<code>col.main.passive</code>	colour of the title if plot is not active
<code>col.main.active</code>	colour of the title if plot is active
<code>col.points.chosen</code>	colour of the selected points
<code>col.abline</code>	colour of the regression line for all points
<code>col.abline.chosen</code>	colour of the regression line for the selected points
<code>col.smooth</code>	colour for the smoothed data
<code>...</code>	further graphical parameters

Value

The function returns a list with the following components

<code>regr</code>	part of the return list of <code>lm</code>
<code>val</code>	the calculated value of the variable of interest
<code>regr.u</code>	NULL or part of the return list of <code>lm</code> for <code>x.u</code> and <code>y.u</code>
<code>val.u</code>	NULL or the calculated value of the variable of interest for the restricted domain
<code>x.u</code>	NULL or the restricted x-coordinates given by the user in the interactive plot
<code>y.u</code>	NULL or y-coordinates according to <code>x.u</code>
<code>sm</code>	smoothed curve through the (x,y) points

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

 RFMethods

Simulation Techniques

Description

PrintMethodList prints the list of currently implemented methods for simulating random fields

GetMethodNames returns a list of currently implemented methods

Usage

PrintMethodList()

GetMethodNames()

Details

- (random spatial) Averages
 - details soon
- Boolean functions.
 - See marked point processes.
- circulant embedding.
 - Introduced by Dietrich & Newsam (1993) and Wood and Chan (1994).
 - Circulant embedding is a fast simulation method based on Fourier transformations. It is guaranteed to be an exact method for covariance functions with finite support, e.g. the spherical model.
 - See also `cutoff` embedding and `intrinsic` embedding for variants of the method.
- `cutoff` embedding.
 - Modified circulant embedding method so that exact simulation is guaranteed for further covariance models, e.g. the whittle matern model. In fact, the circulant embedding is called with the `cutoff` hypermodel, see `CovarianceFct`, and $A = B$ there. `cutoff` embedding halves the maximum number of elements models used to define the covariance function of interest (from 10 to 5).
 - Here multiplicative models are not allowed (yet).
- direct matrix decomposition.
 - This method is based on the well-known method for simulating any multivariate Gaussian distribution, using the square root of the covariance matrix. The method is pretty slow and limited to about 8000 points, i.e. a 20x20x20 grid in three dimensions. This implementation can use the Cholesky decomposition and the singular value decomposition. It allows for arbitrary points and arbitrary grids.

- hyperplane method.
The method is based on a tessellation of the space by hyperplanes. Each cell takes a spatially constant value of an i.i.d. random variables. The superposition of several such random fields yields approximatively a Gaussian random field.
- intrinsic embedding.
Modified circulant embedding so that exact simulation is guaranteed for further *variogram* models, e.g. the fractal brownian one. Note that the simulated random field is always *non-stationary*. In fact, the circulant embedding is called with the Stein hypermodel, see [CovarianceFct](#), and $A = B$ there.
Here multiplicative models are not allowed (yet).

- Marked point processes.

Some methods are based on marked point process $\Pi = \bigcup [x_i, m_i]$ where the marks m_i are deterministic or i.i.d. random functions on R^d .

- add.MPP (Random coins).

Here the functions are elements of the intersection $L_1 \cap L_2$ of the Hilbert spaces L_1 and L_2 . A random field Z is obtained by adding the marks:

$$Z(\cdot) = \sum_{[x_i, m_i] \in \Pi} m_i(\cdot - x_i)$$

In this package, only stationary Poisson point fields are allowed as underlying unmarked point processes. Thus, if the marks m_i are all indicator functions, we obtain a Poisson random field. If the intensity of the Poisson process is high we obtain an approximative Gaussian random field by the central limit theorem - this is the add.mpp method.

- max.MPP (Boolean functions).

If the random functions are multiplied by suitable, independent random values, and then the maximum is taken, a max-stable random field with unit Frechet margins is obtained - this is the max.mpp method.

- Markov Gaussian Random Fields.

The method is due to Havard Rue and uses the property that the precision matrix (i.e. the inverse of the covariance matrix) of a Markov Gaussian random field has a small band width.

Note: This method is only available on Intel Linux systems when installed as follows:

1. install GMRFLib by Havard Rue from www.math.ntnu.no/~hrue/GMRFsim
2. install all external libraries given in GMRFLib/EXTLIBS/ (it may happen that precompiled libraries included in your distribution work better)
3. install RandomFields with option `-enable-GMRF` and `-with-GMRF-lib=LIB_PATH` and `-with-GMRF-EXT-lib=LIB_PATH` for the path of GMRFLib and the EXTLIBS, respectively. E.g., `'R CMD INSTALL RandomFields_* --configure-args="-enable-GMRF -with-GMRF-EXT-lib=/home/schlather/local/lib'`

- nugget.

The method allows for generating a random field of independent Gaussian random variables. In the isotropic case and if the simple notation of a model (with `model` and `param`) is used, this method is called automatically if the nugget effect is positive except the method "circulant embedding" or "direct" have been explicitly.

The method has been extended to zonal anisotropies, see also parameter `nugget.tol` in [RFparameters](#).

- particular method
 - details missing –
- Random coins.
 - See marked point processes.
- sequential This method is programmed for spatio-temporal models where the field is modelled sequentially in the time direction conditioned on the previous k instances. For $k = 5$ the method has its limits for about 1000 spatial points. It is an approximative method. The larger k the better. It also works for certain grids where the last dimension should contain the highest number of grid points.
- spectral TBM (Spectral turning bands).
 - The principle of spectral TBM does not differ from the other turning bands methods. However, line simulations are performed by a spectral technique (Mantoglou and Wilson, 1982). The standard method allows for the simulation of 2-dimensional random fields defined on arbitrary points or arbitrary grids. Here realisation is given as the cosine with random amplitude and random phase.
- TBM2, TBM3 (Turning bands methods; turning layers).
 - It is generally difficult to use the turning bands method (TBM2) directly in the 2-dimensional space. Instead, 2-dimensional random fields are frequently obtained by simulating a 3-dimensional random field (using TBM3) and taking a 2-dimensional cross-section. TBM3 allows for multiplicative models; in case of anisotropy the anisotropy matrices must be multiples of the first matrix or the anisotropy matrix consists of a time component only (i.e. all components are zero except the very last one).
 - TBM2 and TBM3 allow for arbitrary points, and arbitrary grids (arbitrary number of points in each direction, arbitrary grid length for each direction).
 - Note:** Both the precision and the simulation time depend heavily on `TBM*.linesimustep` and `TBM*.linesimufactor` that can be set by [RFparameters](#). For covariance models with larger values of the scale parameter, `TBM*.linesimufactor=2` is too small.
 - The turning layers are used for the simulations with time component. Here, if the model is a multiplicative covariance function then the product may contain matrices with pure time component. All the other matrices must be equal up to a factor and the temporal part of the anisotropy matrix (right column) may contain only zeros, except the very last entry.

Automatic selection algorithm

— details coming soon —

Note

Most methods possess additional parameters, see [RFparameters\(\)](#) that control the precision of the result. The default parameters are chosen such that the simulations are fine for many models and their parameters. The example in [EmpiricalVariogram\(\)](#) shows a way of checking the precision.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

Yindeng Jiang <jiangyindeng@gmail.com> (circulant embedding methods ‘cutoff’ and ‘intrinsic’)

References

Gneiting, T. and Schlather, M. (2004) Statistical modeling with covariance functions. *In preparation*.

Lantuejoul, Ch. (2002) *Geostatistical simulation*. **New York:** Springer.

Schlather, M. (1999) *An introduction to positive definite functions and to unconditional simulation of random fields*. Technical report ST 99-10, Dept. of Maths and Statistics, Lancaster University.

Original work:

- Circulant embedding:
 - Chan, G. and Wood, A.T.A. (1997) An algorithm for simulating stationary Gaussian random fields. *J. R. Stat. Soc., Ser. C* **46**, 171-181.
 - Dietrich, C.R. and Newsam, G.N. (1993) A fast and exact method for multidimensional Gaussian stochastic simulations. *Water Resour. Res.* **29**, 2861-2869.
 - Dietrich, C.R. and Newsam, G.N. (1996) A fast and exact method for multidimensional Gaussian stochastic simulations: Extensions to realizations conditioned on direct and indirect measurement *Water Resour. Res.* **32**, 1643-1652.
 - Wood, A.T.A. and Chan, G. (1994) Simulation of stationary Gaussian processes in $[0, 1]^d$ *J. Comput. Graph. Stat.* **3**, 409-432.

The code used in *RandomFields* is based on Dietrich and Newsam (1996).
- Intrinsic embedding and Cutoff embedding:
 - Stein, M.L. (2002) Fast and exact simulation of fractional Brownian surfaces. *J. Comput. Graph. Statist.* **11**, 587-599.
 - Gneiting, T., Sevcikova, H., Percival, D.B., Schlather, M. and Jiang, Y. (2005) Fast and Exact Simulation of Large Gaussian Lattice Systems in R^2 : Exploring the Limits *J. Comput. Graph. Statist.* Submitted.
- Markov Gaussian Random Field:
 - Rue, H. (2001) Fast sampling of Gaussian Markov random fields. *J. R. Statist. Soc., Ser. B*, **63** (2), 325-338.
 - Rue, H., Held, L. (2005) *Gaussian Markov Random Fields: Theory and Applications*. Monographs on Statistics and Applied Probability, no **104**, Chapman & Hall.
- Turning bands method (TBM), turning layers:
 - Dietrich, C.R. (1995) A simple and efficient space domain implementation of the turning bands method. *Water Resour. Res.* **31**, 147-156.
 - Mantoglou, A. and Wilson, J.L. (1982) The turning bands method for simulation of random fields using line generation by a spectral method. *Water. Resour. Res.* **18**, 1379-1394.
 - Matheron, G. (1973) The intrinsic random functions and their applications. *Adv. Appl. Probab.* **5**, 439-468.
 - Schlather, M. (2004) Turning layers: A space-time extension of turning bands. *Submitted*
- Random coins:
 - Matheron, G. (1967) *Elements pour une Theorie des Milieux Poreux*. Paris: Masson.

See Also

[CovarianceFct](#), [GaussRF](#), [MaxStableRF](#), [PrintModelList](#), [RandomFields](#), [sophisticated](#).

RFparameters

*Control Parameters***Description**

RFparameters sets and returns control parameters for the simulation of random fields

Usage

```
RFparameters(..., no.readonly=FALSE)
```

Arguments

... arguments in tag = value form, or a list of tagged values.

no.readonly If RFparameters is called without parameter then all parameters are returned in a list. If no.readonly=TRUE then only rewritable parameters are returned.

Details

The possible parameters are

General options

PracticalRange logical or . If not FALSE the range of the **primitive** covariance functions is adjusted so that cov(1) is zero for models with finite range. The value of cov(1) is about 0.05 (for scale=1) for models without range. See [CovarianceFct](#) or type `PrintModelList()` for the list of primitive models.

- FALSE : the practical range adjustment is not used.
- TRUE : PracticalRange is applicable only if the value is known exactly, or, at least, can be approximated by a closed formula.
- 2 : if the practical range is not known exactly it is approximated numerically.

Default: FALSE [init].

PrintLevel If `PrintLevel ≤ 0` there is not any output on the screen. The higher the number the more tracing information is given. Default: 1 [init, do].

1 : error messages
 2 : messages about partial failures of the algorithm
 >2 : additional informations

Note that `PrintLevel` is also used in other packages as a default, for example in `SoPhy` (`risk.index` and `create.roots`). The changing of `PrintLevel` here may cause some unexpected effects in these functions. See the documentation there.

General options for simulating

`pch` Character or empty string. The character is printed after each performed simulation if more than one simulation is performed at once. If `pch='!`' then an absolute counter is shown instead of the character. If `pch='%'` then a counter of percentages is shown instead of the character. Note that also `'^H'`s are printed in the last two cases, which may have undesirable interactions with some few other R functions, e.g. [Sweave](#). Default: `'*' [do]`.

`Storing` Logical. If `FALSE` then the intermediate results are destroyed after the simulation of the random field(s) or if an error had occurred.

On the other hand, if `Storing=TRUE`, then several simulations performed with `DoSimulateRF` for the same model parameters are performed faster.

See also `CE.several`, `TBMCE.several` and `local.several` for related parameters.

Default: `FALSE [do]`.

`skipchecks` logical. If `TRUE`, the check whether the given parameter values and the dimension are within the allowed range is skipped. Do not change the value of this variable except you really know what you do.

Default: `FALSE [init]`.

`stationary.only` Logical or NA. Used for the automatic choice of methods. See also [RFMethods](#).

- `TRUE`: the simulation of non-stationary random fields is refused. In particular, the intrinsic embedding method is excluded and the simulation of Brownian motion is rejected.
- `FALSE`: intrinsic embedding is always allowed, actually it's the first one considered in the automatic selection algorithm.
- `NA`: the simulation of the Brownian motion allowed, but intrinsic embedding is not used for stationary random fields.

Default: `NA [init]`.

`exactness` logical or NA.

- `TRUE`: `add.MPP`, `hyperplanes` and all turning bands methods are excluded. If the circulant embedding method is considered as badly behaved, then the matrix decomposition methods are preferred.
- `FALSE`: if the circulant embedding method is considered as badly behaved or the number of points to be simulated is large, the turning bands methods are rather preferred.
- `NA`: approximative non-exact methods are excluded, i.e. `TBM2` if the Abel transform of the covariance function cannot be given explicitly.

Default: `NA [init]`.

`every` integer. if greater than zero, then every `everyth` iteration is printed if simulated by `TBM` or `random coin` method. Default: `0 [do]`.

`aniso` logical. If `TRUE` missing anisotropy parameter or scale parameter in model `$` are interpreted as identity matrix. If `FALSE` missing anisotropy or scale parameter is interpreted as scale parameter of value 1.0. The latter coding tends to faster simulation, but also to more error messages. The parameter should be changed only by advanced users. Default: `TRUE [init]`.

Options for simulating with the standard circulant embedding method

`CE.force` Logical. Circulant embedding does not work if a certain circulant matrix has negative eigenvalues. Sometimes it is convenient to replace all the negative eigenvalues by zero (`CE.force=TRUE`) after `CE.trials` number of trials. Default: `FALSE [init]`.

- `CE.mmin` Scalar or vector, integer if positive. `CE.mmin` determines the initial size of the circulant matrix. If `CE.mmin=0` the minimal starting size is determined automatically according to the dimensions of the grid. If `CE.mmin>0` then the absolute starting size is given. If `CE.mmin<0` then the automatically determined matrix size is multiplied by $|\text{CE.mmin}|$; here `CE.mmin` must be smaller than -1; the value -1 takes over the minimal starting size.
Note: in any cases, the initial size might be increased according to `CE.useprimes`.
Default: 0 [init].
- `CE.strategy` 0 : if the circulant matrix has negative eigenvalues then the size in each direction is doubled;
1 : the size is enhanced only in one direction, namely that one where the covariance function has the largest value at the end point of the grid — note that the default value of `CE.trials` is probably too small in that case.
In some cases `CE.strategy=0` works better, in other cases `CE.strategy=1`. Just try.
Default: 0 [init].
- `CE.maxmem` maximal total size of the circulant matrix. The total amount of memory needed for the internal calculations is about 16 ($=2 * \text{sizeof}(\text{double})$) times as large as `CE.maxmem` if `RFparameters()$Storing=FALSE` and 32 ($=4 * \text{sizeof}(\text{double})$) time as large if `Storing=TRUE`. Note that `CE.maxmem` can be used to control the automatic choice of the simulation algorithm. Namely, in case of huge circulant matrices, other simulation methods (TBM) are faster and might be preferred by the user.
Default: $4096^2 = 16777216$ [init].
- `CE.tolIm` If the modulus of the imaginary part is less than `CE.tolIm` then the eigenvalue is considered as real. Default: $1\text{E-}3$ [init].
- `CE.tolRe` Eigenvalues between `CE.tolRe` and 0 are considered as 0 and set 0. Default: $-1\text{E-}7$ [init].
- `CE.trials` A larger circulant matrix is likely to make more eigenvalues non-negative. If at least one of the thresholds `CE.tolRe` and `CE.tolIm` are missed then the matrix size is doubled according to `CE.strategy`, and the matrix is checked again. This procedure is repeated up to `CE.trials-1` times. If there are still negative eigenvalues, the simulation method fails if `CE.force=FALSE`. Default: 3 [init].
- `CE.several` logical. If `FALSE` only half the memory is need, but only a single independent realisation can be created. Default: `TRUE` [init].
- `CE.useprimes` Logical. If `FALSE` the columns of the circulant matrix have length 2^k for some k . Otherwise the algorithm tries to find a nicely factorizable number close to the size of the given matrix. Default: `TRUE` [init].
- `CE.dependent` Logical. If `FALSE` then independent random fields are created. If `TRUE` then at least 4 non-overlapping rectangles are taken out of the the expanded grid defined by the circulant matrix. These simulations are dependent. See below for an example. See `CE.trials` for some more information on the circulant matrix. Default: `FALSE` [init].
- `CE.method` 0 or 1. Decomposition of the covariance matrix. This parameter is only relevant if multivariate random fields are simulated.
Default (`CE.method=0`): If `.Random.seed` is fixed, Cholesky decomposition allows for fixing the random field of the first component whilst parameters for the second field are changed. If `CE.method=0` and approximate circulant embedding is allowed, i.e. `CE.force=TRUE`, SVD is tried for the last failed attempt of Cholesky decomposition. (SVD, in contrast to Cholesky decomposition, allows for approximate circulant embedding.)

If `CE.method=1`, Cholesky decomposition will not be attempted, but singular value decomposition used instead in all attempts. (SVD is slower, but more precise.)

Default: 0 [init].

Options for simulating by simple matrix decomposition

`direct.bestvariables` integer. When searching for an appropriate simulation method the matrix decomposition method (see ‘direct method’ below) is preferred if the number of variables is less than or equal to `direct.bestvariables`. Default: 800 [init]

`direct.maxvariables` If the number of variables to generate is greater than `direct.maxvariables`, then any matrix decomposition method is rejected. It is important that this option is set conveniently to avoid great losses of time during the automatic search of a simulation method (method=NULL in `GaussRF`). Default: 4096 [init]

`direct.method` Decomposition of the covariance matrix. If `direct.method=1`, Cholesky decomposition will not be attempted, but singular value decomposition used instead if `direct.svdtolerance` positive. In case of a multivariate random field, `direct.method = 2` or `3` orders the covariance such that first all components are considered for the first variable, then all components for the second one, and so on. If `direct.method = 0` or `1` it starts with the first component of all locations, then the second components follow, etc.

Default: 0 [init].

`direct.svdtolerance` If SVD decomposition is used for calculating the square root of the covariance matrix then the absolute componentwise difference between the covariance matrix and square of the square root must be less than `direct.svdtolerance`. No check is performed if `direct.svdtolerance` is negative. Default: 1e-12 [init].

Options for simulating hyperplane tessellations

`hyper.superpos` integer. number of superposed hyperplane tessellations. Default: 300 [do].

`hyper.maxlines` integer. Maximum number of allowed lines. Default: 1000 [init].

`hyper.mar.distr` integer. code for the marginal distribution used in the simulation:

0 uniform distribution

1 Frechet distribution with form parameter `hyper.mar.param`

2 Bernoulli distribution (Binomial with $n = 1$) with parameter `hyper.mar.param`

The parameter should not be changed yet. Default: 0 [do].

`hyper.mar.param` Parameter used for the marginal distribution. The parameter should not be changed yet. Default: 0 [do].

Options for simulating with the local ce methods (cutoff, intrinsic)

`local.force` see `CE.force` above. Default: FALSE [init].

`local.mmin` see `CE.mmin` above.

Difference: if `local.mmin=0` the automatic determination of the initial size of the circulant matrix takes into account an expansion factor. This expansion factor is intended to make the circulant matrix positive definite and is either theoretically or numerically known, or guessed.

If the usual strategy of circulant embedding (doubling the grid sizes) should be taken over then `local.mmin` must be set to `-1`.

Default: 0 [init].

`local.maxmem` see `CE.maxmem` above. Default: 20000000 [init].

`local.tolIm` see `CE.tolIm` above. Default: 1E-7 [init].

`local.tolRe` see `CE.tolRe` above. Default: -1E-9 [init].

`local.several` see `CE.several` above. Default: 1 [init].

`local.useprimes` see `CE.useprimes` above. Default: TRUE [init].

`local.dependent` see `CE.dependent` above. Default: FALSE [init].

Options for simulating a Gaussian Markov random fields

`markov.neighbours` 2 or 3. Number of neighbours in each direction. Default: 2 [init].

`markov.precision` See also the GMRF manual. Default: 1.0 [init].

`markov.cyclic` logical. simulation on a torus; should not be changed except the user knows what he/she does. (The covariance function will be (slightly?!) changed then without further notice!) Default: FALSE [init].

`markov.maxmem` integer. maximum number of points of the grid. Default: 250000 [init].

Options for simulating by random coins

`mpp.locations` integer. Way of defining the point process.

-1 : automatic choice

0 : grid

1 : Poisson point process

2 : uniform distribution

3 : single point follows d dimensional Gaussian distribution

Default: -1 [init].

`mpp.intensity` real. Number of superposed realisations (to approximate the normal distribution; total number for all (additive) components with same anisotropy); if `mpp.intensity` ≤ 0.0 then only a single value is simulated (for checking). Default: 100 [do].

`mpp.plus` In order avoid edge effects, the simulation area is enlarged by a constant r so that all marks have their (supposed) support in the ball with radius r centred at the origin; see also `mpp.approxzero`. If `mpp.plus` > 0 the true radius r is replaced by `mpp.radius`. Default: 0.0 [init].

`mpp.relRadius` `mpp.relRadius/V` is added to the `effectiveRadius`. The latter gives the approximate radius for the support of the grain. Default: 2.5 [init].

`mpp.approxzero` Functions that do not have compact support are set to zero outside the ball outside for which the function has absolute values less than `MPP.approxzero`. Default: 0.001 [init].

`mpp.samplingdist` `mpp.samplingdist` gives the grid distance to numerically approximate the volume of the grain. `[-mpp.samplingr, mpp.sampling]^d` gives the domain over which the approximation is calculated.

Default: 0.01 [init].

`mpp.samplingr` see `mpp.samplingdist` Default: 5 [init].

Options for simulating nugget effects

Simulating a nugget effect seems trivial. It gets complicated and best methods (including direct and circulant embedding!) fail if zonal anisotropies are considered, where sets of points have to be identified that belong to the same subspace of eigenvalue 0 of the anisotropy matrix.

`nugget.tol` points at a distance less than or equal to `nugget.tol` are considered as being identical.

This strategy applies to the simulation method and the covariance function itself. Hence, the covariance function is only positive definite if `nugget.tol=0.0`. However, if the anisotropy matrix does not have full rank and `nugget.tol=0.0` then, the simulations are likely to be odd. The value of `nugget.tol` should be of order 10^{-15} . Default: 0.0 [init].

`nugget.meth` logical. If TRUE any method given the user for the simulation of the nugget effect is replaced by the method 'nugget' whenever appropriate (zonal nugget or a method different from circulant embedding, direct and sequential).

Options for using the sequential method

The method works only for spatio-temporal settings (and grids).

`sequ.max` integer. maximum number of allowed variables on which the conditional distribution is conditioned. Default: 5000 [init].

`sequ.back` integer. number of previous instances on which the algorithm should condition. If less than one then the number of previous instances equals `sequ.max / (number of spatial points)`. Default: 5 [init].

`sequ.initial` First, $N=(\text{number of spatial points}) * \text{sequ.back}$ number of points are simulated. Then, sequentially, all spatial points for the next time instance are simulated at once, based on the previous `sequ.back` instances. The distribution of the first N points is the correct distribution, but differs, in general, from the distribution of the sequentially simulated variables. We prefer here to have the same distribution all over (although only approximatively the correct one), hence do some initial sequential steps first. If `sequ.initial` is non-negative, then `sequ.initial` first steps are performed. If `sequ.initial` is negative, then `sequ.back - sequ.initial` initial steps are performed. The latter ensures that none of the very first N variables are returned. Default: -10 [init].

Options for special simulation methods

Special methods exist for the following covariance functions

spatialMA1 this covariance function is simulated by a certain moving average of a spatially independent, but temporally dependent random field Y .

`spec.MA.r` gives the radius beyond which the bivariate standard normal density is considered as being zero Default: 2.5 [init].

`spec.MA.dist` the random field Y is approximated by a grid; the grid length is given by `spec.MA.dist`. Default: 0.1 [init].

Options for simulating with a turning bands method

Currently, there are 3 variants of the turning bands method implemented:

`spectral` The spectral turning bands method is implemented for 2 (and 1) dimensions only.

`TBM2` It is based on the two dimensional turning bands operator and is applicable for 1 and 2 dimensions. As an additional dimension the time dimension can be added.

`TBM3` It is based on the three dimensional turning bands operator and is applicable for 1,2,3 dimensions. As an additional dimension the time dimension can be added.

The following parameters are used.

- `spectral.grid` Logical. The angle of the lines is random if `spectral.grid=FALSE`, and $k\pi/\text{spectral.lines}$ for k in $1:\text{spectral.lines}$, otherwise. This parameter is only considered if the spectral measure, not the density is used. Default: TRUE [do].
- `spectral.ergodic` In case of an additive model and `spectral.ergodic=FALSE`, the additive component are chosen proportional to their variance. In total `spectral.lines` are simulated. If `spectral.ergodic=TRUE`, the components are simulated separately and then added. Default: FALSE [do].
- `spectral.lines` Number of lines used (in total for all additive components of the covariance function). Default: 500 [do].
- `spectral.metro` Logical. If TRUE then preference is given to the (slower) metropolis sampling algorithm of the spectral density. Default: FALSE [init].
- `spectral.nmetro` integer. Considered if the Metropolis algorithm is used. It gives the number of metropolis steps before returning the next draw from the spectral density. If `spectral.nmetro` is not positive then `RandomFields` tries to find a good choice for `spectral.nmetro` by itself. Default: 0 [init].
- `spectral.sigma` real. Considered if the Metropolis algorithm is used. It gives the standard deviation of the multivariate normal distribution of the proposing distribution. If `spectral.sigma` is not positive then `RandomFields` tries to find a good choice for `spectral.sigma` itself. Default: 0 [init].
- `TBM.method` character. The preferred method to simulate on the line for TBM2 and TBM3; If ‘Nothing’ then automatic choice. Default: “Nothing” [init].
- `TBM.center` Scalar or vector. If not NA, the `TBM.center` is used as the center of the turning bands for TBM2 and TBM3. Otherwise the center is determined automatically such that the line length is minimal. See also `TBM.points` and the examples below. Default: NA [init].
- `TBM.points` integer. If greater than 0, `TBM.points` gives the number of points simulated on the TBM line, hence must be greater than the minimal number of points given by the size of the simulated field and the two paramters `TBMx.linesimufactor` and `TBMx.linesimustep`. If `TBM.points` is not positive the number of points is determined automatically. The use of `TBM.center` and `TBM.points` is highlighted in an example below. Default: 0 [init].
- `TBM2.every` If `TBM2.every>0` then every `TBM2.every`th iteration is announced. Default: 0 [do].
- `TBM2.lines` Number of lines used. Default: 60 [do].
- `TBM2.linesimufactor` `TBM2.linesimufactor` or `TBM2.linesimustep` must be non-negative; if `TBM2.linesimustep` is positive then `TBM2.linesimufactor` is ignored. If both parameters are naught then `TBM.points` is used (and must be positive). The grid on the line is `TBM2.linesimufactor`-times finer than the smallest distance. See also `TBM2.linesimustep`. Default: 2.0 [init].
- `TBM2.linesimustep` If `TBM2.linesimustep` is positive the grid on the line has lag `TBM2.linesimustep`. See also `TBM2.linesimufactor`. Default: 0.0 [init].
- `TBM2.layers` Logical or integer. If TRUE then the turning layers are used whenever a time component is given. If FALSE the turning layers are used only when the traditional TBM is not applicable. If negative then turning layers may never be used. If greater than 1 then only turning layers may be used. Default: FALSE [init].
- `TBM3.every` If `TBM3.every>0` then every `TBM3.every`th iteration is announced. Default: 0 [do].

TBM3.lines Number of lines used. Default: 500 [do].

TBM3.linesimufactor See TBM2.linesimufactor for the meaning. Default: 2.0 [init].

TBM3.linesimustep See TBM2.linesimustep for the meaning. Default: 0.0 [init].

TBM3.layers See TBM2.layers for the meaning. Default: FALSE [init].

TBMCE.force see TBM.method and CE.force Default: FALSE [init].

TBMCE.mmin see TBM.method and CE.mmin. Default: 0 [init].

TBMCE.strategy see TBM.method and CE.strategy. Default: 0 [init].

TBMCE.maxmem see TBM.method and CE.maxmem. Default: 10000000 [init].

TBMCE.tolIm see TBM.method and CE.tolIm. Default: 1E-3 [init].

TBMCE.tolRe see TBM.method and CE.tolRe. Default: -1E-7 [init].

TBMCE.trials see TBM.method and CE.trials. Default: 3 [init].

TBMCE.useprimes see TBM.method and CE.useprimes. Default: TRUE [init].

TBMCE.dependent see TBM.method and CE.dependent. Default: FALSE [init].

Options specific to simulating max-stable random fields

maxstable.maxGauss Max-stable random fields. The simulation of the max-stable process based on random fields uses a stopping rule that necessarily needs a finite upper endpoint of the marginal distribution of the random field. In the case of extremal Gaussian random fields, see [MaxStableRF](#), the upper endpoint is approximated by maxstable.maxGauss. Default: 3.0 [init].

General comments on the options

The following refers to the simulation of Gaussian random fields ([InitGaussRF](#), [GaussRF](#)), but most parts also apply for the simulation of max-stable random fields ([InitMaxStableRF](#), [MaxStableRF](#)).

Some of the global parameters determine the basic settings of a simulation, e.g. `direct.method` (which chooses a square root of a positive definite matrix). The values of such parameters are read by [InitGaussRF](#) and stored in an internal register. Changing such a parameter between calling [InitGaussRF](#) and calling [DoSimulateRF](#) or between subsequent calls of [GaussRF](#) will not have any effect. These parameters have the flag "[init]".

Parameters like `TBM2.lines` (which determines the number of i.i.d. processes to be simulated on the line) are only relevant when generating random numbers. These parameters are read by [DoSimulateRF](#) (or by the second part of [GaussRF](#)), and are marked by "[do]".

Storing has an influence on both, [InitGaussRF](#) and [DoSimulateRF](#). [InitGaussRF](#) may reserve more memory if `Storing=TRUE`. [DoSimulateRF](#) will free the register if `Storing=FALSE`, whatever the value of `Storing` was when [InitGaussRF](#) was called.

The distinction between [init] and [do] is also relevant if [GaussRF](#) is used and called a second time with the same parameters for the random field and if `RFparameters()$Storing=TRUE`. Then [GaussRF](#) realises that the second call has the same random field parameters, and takes over the stored intermediate results (that have been calculated with the `RFparameters()` at that time). To prevent the use of stored intermediate results or to take into account intermediate changes of `RFparameters` set `RFparameters(Storing=FALSE)` or use [DeleteRegister\(\)](#) between calls of [GaussRF](#).

A programme that checks whether the parameters are well adapted to a specific simulation problem is given as an example of [EmpiricalVariogram\(\)](#).

For further details on the implemented methods, see [RFMethods](#).

Value

If any parameter has been given RFparameters returns an invisible list of the given parameters in full name.

Otherwise the complete list of parameters is returned. Further the values of the following internal readonly variables are returned:

- * covmaxchar: max. name length for variogram/covariance models
- * covnr: number of currently implemented variogram/covariance models
- * distrmaxchar: max. name length for a distribution
- * distrnr: number of currently implemented distributions
- * maxdim: maximum number of dimensions for a random field
- * maxmodels: maximum number of elementary models in definition of a complex covariance model
- * methodmaxchar: max. name length for methods
- * methodnr: number of currently implemented simulation methods

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Schlather, M. (1999) *An introduction to positive definite functions and to unconditional simulation of random fields*. Technical report ST 99-10, Dept. of Maths and Statistics, Lancaster University.

See Also

[GaussRF](#), [GetPracticalRange](#), [MaxStableRF](#), [RandomFields](#), and [RFMethods](#).

Examples

```
RFparameters(Storing=TRUE)
str(RFparameters())

#####
##                                     ##
##      use of TBM.points and TBM.center   ##
##                                     ##
#####
## The following example shows that the same realisation
## can be obtained on different grid geometries (or point
## configurations, i.e. grid, non-grid) using TBM3 (or TBM2)

x1 <- seq(-150,150,1)
y1 <- seq(-15, 15, 1)
x2 <- seq(-50, 50, 1)
model <- "exponential"
param <- c(0, 1, 0, 10)
meth <- "TBM3"
```

```
##### simulation of a random field on long thin stripe
runif(1)
rs <- get(".Random.seed", envir=.GlobalEnv, inherits = FALSE)
z1 <- GaussRF(x1, y1, model=model, param=param, grid=TRUE,
              method=meth, TBM.center=0, Storing=TRUE)
do.call(getOption("device"), list(height=1.55, width=12))
par(mar=c(2.2, 2.2, 0.1, 0.1))
image(x1, y1, z1, col=rainbow(100))
polygon(range(x2)[c(1,2,2,1)], range(y1)[c(1,1,2,2)], border="red", lwd=3)

##### definition of a random field on a square of shorter diagonal
assign(".Random.seed", rs, envir=.GlobalEnv)
tbm.points <- length(GetRegisterInfo(meth="TBM")$$line)
z2 <- GaussRF(x2, x2, model=model, param=param, grid=TRUE, register=1,
              method=meth, TBM.center=0, TBM.points=tbm.points)
do.call(getOption("device"), list(height=4.3, width=4.3))
par(mar=c(2.2, 2.2, 0.1, 0.1))
image(x2, x2, z2, zlim=range(z1), col=rainbow(100))
polygon(range(x2)[c(1,2,2,1)], range(y1)[c(1,1,2,2)], border="red", lwd=3)

#####
##
##          use of exactness          ##
##
#####

x <- seq(0, 1, 1/30)
model <- list("+",
              list("stable", alpha=1.0),
              list(model="gencauchy", alpha=1.0, beta=2.0)
              )

for (exactness in c(NA, FALSE, TRUE)) {
  readline(paste("\n\nexactness: ", exactness, "; press return"))
  z <- GaussRF(x, x, grid=TRUE, gridtriple=FALSE,
               model=model, exactness=exactness,
               stationary.only=NA, Print=1, n=1,
               TBM2.linesimustep=1, Storing=TRUE)
  Print(GetRegisterInfo()$meth$name)
}

#####
## The following gives a tiny example on the advantage of ##
## local.dependent=TRUE (and CE.dependent=TRUE) if in a ##
## study most of the time is spent with simulating the ##
## Gaussian random fields. Here, the covariance at a pair ##
## of points is estimated for n independent repetitions ##
## and 2*n locally dependent dependent repetitions .      ##
```

```

## To get the precision, the procedure is repeated m times.##
#####

# In the example below, local.dependent speeds up the simulation
# by about factor 16 at the price of an increased variance of
# factor 1.5

x <- seq(0, 1, len=10)
y <- seq(0, 1, len=10)
grid.size <- c(length(x), length(y))
model <- list("$", var=1.1, aniso=matrix(nc=2, c(2,1,0.5,1)),
            list(model="exp"))
(CovarianceFct(matrix(c(1, -1), ncol=2), model=model)) ## true value

m <- if (interactive()) 100 else 2
n <- if (interactive()) 100 else 10

# using local.dependent=FALSE (which is the default)
c1 <- numeric(m)
unix.time(
for (i in 1:m) {
  cat("", i)
  z <- GaussRF(x, y, model=model, grid=TRUE, method="cu", n=n,
               local.dependent=FALSE, pch="")
  c1[i] <- cov(z[1,length(y), ], z[length(x), 1, ])
}) # many times slower than with local.dependent=TRUE
mean(c1)
sd(c1)

# using local.dependent=TRUE...
c2 <- numeric(m)
unix.time(
for (i in 1:m) {
  cat("", i)
  z <- GaussRF(x, y, model=model, grid=TRUE, method="cu", n=2 * n,
               local.dependent=TRUE, pch="")
  c2[i] <- cov(z[1,length(y),], z[length(x), 1 , ])
})
mean(c2)
sd(c2) # the sd is smaller (using more locally dependent realisations)
##      but it is (much) faster! For n=n2 instead of n=2 * n, the
##      value of sd(c2) would be larger due to the local dependencies
##      in the realisations.

```

Description

ShowModels is an interactive plot for the selection of models and their one- or two-dimensional simulations; it also allows for the fitting of variogram models by eye.

CURRENTLY NOT AVAILAIBLE IN VERSION 2 OF RANDOMFIELDS. PLEASE USE VERSION 1.3.47 INSTEAD.

Usage

```
ShowModels(x, y=NULL,
           covx=ifelse(is.null(empirical), diff(range(x))/5,
                      max(empirical$c)),
           fixed.rs=TRUE, method=NULL, empirical=NULL,
           model=NULL, param=NULL, anisotropy = FALSE, all.param=NULL,
           legends = TRUE,
           register=0, Mean=NULL, erase=TRUE,
           x.fraction=0.60, cex.names=1, covx.default = 100,
           link.fct=NULL, Zlim=NULL,
           Col.rect="red", Col.bg="blue", Col.sep="grey",
           Col.line="red", Col.txt="black", Col.flash="red",
           Col.vario="blue", Col.main="black",
           Col.model=c("red", "black"), vario.lty=c(1,2),
           cex.leg = 0.7 * cex.names, cex.eval = 0.8 * cex.names,
           update=TRUE, screen.new=TRUE, use.outer.RFparameters=FALSE,
           debug=FALSE, ...)
```

Arguments

x	if NULL simulations are not performed; otherwise it gives the x coordinates of a grid as a sequence of increasing numbers
y	if NULL at most one-dimensional simulations are performed (depending on the value of x); otherwise y gives the y coordinates of a two-dimensional grid (as a sequence of increasing numbers).
covx	if a single value is given, it is the largest distance for which the covariance functions or the variograms are plotted; otherwise the models are plotted for the given values, and the origin.
fixed.rs	if TRUE then the same random seed is used for all simulations until the user clicks on the formula, the title or the subtitles.
method	simulation method, see RFMethods ; if NULL then a suitable simulation method is chosen automatically.
empirical	empirical variogram; a list as returned by EmpiricalVariogram . Also empirical variograms with a pair number of anisotropy directions may be passed. Then the first and the middle one are taken.
model	covariance model, see CovarianceFct , or type PrintModellist() to get all options. If given, this model is shown at the beginning. Additive or multiplicative models are not allowed.

However, model can also be given by a simple list definition, see [CovarianceFct](#). Then param must not be given. In this case also anisotropic models can be defined.

param	parameter vector: param=c(mean, variance, nugget, scale,...); the parameters must be given in this order; see CovarianceFct for more details. Only considered if model is given. If given, model is initialised by param.
anisotropy	logical. If TRUE then an isotropic model is considered.
all.param	all.param=c(mean, variance, nugget, scale); the parameters must be given in this order; If all.param is given then the parameters of all covariance functions are set to the given values. The values are overwritten for a specific model if model and param are given. Note that it is not possible to set the values of additional (form) parameters of a parametrised class by means of all.param. In case of an anisotropic model the anisotropy matrix is by default diagonal with both entries equal to 1/all.param[4].
legends	if TRUE then a legend is added to the two-dimensional plot.
register	register where intermediate results of the simulations are stored, see also GaussRF .
Mean	mean of the random field
erase	parameter of split.screen , which is called at the very beginning
x.fraction	the current screen is split into 2 x 2 screens. The parameter x.fraction gives the size of the left screens in the x directions as part of 1. See also the Details.
cex.names	font size for model names
covx.default	if length(cov.x)==1 then [0, cov.x] is covered by covx.default points of equal distance
link.fct	NULL or function(values) or "MaxStable". Transformation of the Gaussian random field. If link.fct="MaxStable" then max-stable random fields are simulated for the given covariance function and the extremal coefficient function is given (up the constant -1) instead of the variogram or the covariance function
Zlim	Vector of two elements or list of two vectors of two elements. Graphical limits for the Gaussian random process (and the transformed field).
Col.rect	colour for interactive plot; see eval.parameters .
Col.bg	colour for interactive plot; see eval.parameters .
Col.sep	colour for interactive plot; see eval.parameters .
Col.line	colour for interactive plot; see eval.parameters .
Col.txt	colour for interactive plot; see eval.parameters .
Col.flash	colour for the previously chosen model
Col.vario	colour for the empirical variogram plot
Col.main	colour for the title of the random field
Col.model	vector of two colours for plotting the variogram of the Gaussian random field and the transformed field
vario.lty	vector of two line types for primary and secondary axis of the variogram

<code>cex.leg</code>	font size used in the legends
<code>cex.eval</code>	font size used in the menu entries
<code>update</code>	logical. If TRUE the plots are updated after each interactive change of the values. Otherwise, the bottom 'simulate' is added in the menu.
<code>screen.new</code>	logical. If FALSE the screen is erased before a simulation and completely rebuild; otherwise the screen is updated. If FALSE flickering appears during the update of the current screen, otherwise it may happen during the reorganisation of any window (and may take quite a lot of time).
<code>use.outer.RFparameters</code>	logical. If FALSE the following parameters usually set by RFparameters are internally set <ul style="list-style-type: none"> • <code>PracticalRange=FALSE</code> • <code>PrintLevel=1</code> if <code>debug=FALSE</code> and 5 otherwise. • <code>maxstable.maxGauss=2</code> • <code>CE.force=TRUE</code> • <code>CE.trials=1</code> • <code>CE.mmin=-4</code> • <code>CE.useprimes=TRUE</code>
<code>debug</code>	logical. If TRUE then internally the <code>RFparameter()\$PrintLevel</code> is set to 5.
<code>...</code>	additional graphics options for the plot of the one- or two-dimensional simulations, see plot and image .

Details

The interactive plot consists of 3 parts:

- top left: graph of the covariance function or the variogram. In case `empirical` is given the empirical variogram is also plotted. If `link.fct` is given, then also the covariance function or the variogram is plotted. If the correlation model is for a non-stationary random field, the variogram for the transformed random field is not estimated in a primitive way – this is indicated with a star in the legend
- bottom left: one- or two-dimensional simulation
- right:
 - list of implemented models; a specific model is chosen by the left mouse button, or:
 - menu for the parameters of the chosen model. The list includes the variance, a nugget effect, the mean and the scale or the anisotropy parameters. Further, some global parameters can be changed. They are the `PracticalRange` (see [RFparameters](#) for details) and the angle of the main variogram direction (or NA, then it follows the angle of the anisotropy). Finally, the user can choose between the plot of the covariance and the corresponding variogram.

The interactive plot is left by clicking any mouse button different from the left when the top right part is active.

Value

list of the last model and its parameters.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[CovarianceFct](#), [eval.parameters](#), [GaussRF](#), [RFMethods](#), [RandomFields](#).

Examples

```
## Not run:

# first example: one-dimensional simulations

# library(RandomFields)

options(locatorBell=FALSE)
x <- seq(1,10,0.1)
ShowModels(x=x, covx=10, cov.def=100, type="1")

x <- seq(1,10,0.1)
ShowModels(x=x, y=x, covx=10, cov.def=100)

# second example: two-dimensional simulations and
#                   empirical variogram
dx <- runif(300,0,8)
dy <- runif(300,0,8)
dz <- GaussRF(x=dx, y=dy, grid=FALSE, model="gaus",
              param=c(1,2,1,2))
ev <- EmpiricalVariogram(x=dx, y=dy, data=dz, grid=FALSE,
                        bin=(-1:20)/4)
x <- seq(1,5,0.1);
ShowModels(x=x, y=x, empirical=ev)

# third example: two-dimensional anistropic simulations and
#                   link function

x <- seq(1,10,0.1)
ShowModels(x=x, y=x, link=function(x) exp(x),
           model=list(list(model="spheric", var=1, aniso=c(1,0,0,5))))

x <- seq(1,10,0.1)
ShowModels(x=x, link=function(x) exp(x),
           model=list(list(model="spheric",var=1, scale=1)))

x <- seq(1,10,0.1)
```

```
ShowModels(x=x, link="MaxStable", fixed.rs=TRUE,
           model=list(list(model="gauss",var=1, scale=1)), type="l")

## End(Not run)
```

 SimulateRF

Simulation of Random Fields

Description

DoSimulateRF performs an already initialised simulation.

InitSimulateRF internal function; use [InitGaussRF](#) and [InitMaxStableRF](#), instead.

Usage

```
DoSimulateRF(n=1, register=0, paired=FALSE, trend=NULL)
```

```
InitSimulateRF(x, y=NULL, z=NULL, T=NULL, grid=!missing(gridtriple),
              model, param, trend, method=NULL, register=0, gridtriple,
              distribution=NA)
```

Arguments

x	matrix of coordinates, or vector of x coordinates
y	vector of y coordinates
z	vector of z coordinates
T	time instances
grid	logical; determines whether the vectors x, y, and z should be interpreted as a grid definition, see Details .
model	string; covariance or variogram model, see CovarianceFct , or type PrintModelList() to get all options
param	vector or list. param=c(mean, variance, nugget, scale, ...), param=list(c(variance, scale, ...), ..., c(variance,scale,...)), param=matrix(...), or param=list(list(variance, anisotropy, kappa),..., list(variance, anisotropy, kappa)); the parameters must be given in this order; further parameters are to be added in case of a parametrised class of models, see CovarianceFct
method	NULL or string; Method used for simulating, see RFMethods , or type PrintMethodList() to get all options
register	0:9; place where intermediate calculations are stored; the numbers are aliases for 10 internal registers
gridtriple	logical; if gridtriple=FALSE ascending sequences for the parameters x, y, and z are expected; if gridtriple=TRUE triples of form c(start,end,step) expected; this parameter is used only if grid=TRUE

distribution	marginal distribution: 'Gauss', 'Poisson', or 'MaxStable'
n	number of realisations to generate; if paired=TRUE then n must be even.
paired	logical. paired may be TRUE only for the simulation of Gaussian random fields. If TRUE then every second simulation is obtained by only changing the signs of the standard Gaussian random variables, the simulation is based on ("antithetic pairs").
trend	only used for universal and intrinsic kriging. In case of universal kriging trend is a non-negative integer (monomials up to order k as trend functions), a list of functions or a formula (the summands are the trend functions); you have the choice of using either x, y, z or X1, X2, X3,... as spatial variables; in case of intrinsic kriging trend should be a nonnegative integer which is the order of the underlying model.

Value

InitSimulateRF returns 0 if no error has occurred during the initialisation process, and a positive value if failed.

DoSimulateRF returns NULL if an error has occurred; otherwise the returned object depends on the parameters n and grid:

n=1:

* grid=FALSE. A vector of simulated values is returned (independent of the dimension of the random field)

* grid=TRUE. An array of the dimension of the random field is returned.

n>1:

* grid=FALSE. A matrix is returned. The columns contain the realisations.

* grid=TRUE. An array of dimension $d+1$, where d is the dimension of the random field, is returned. The last dimension contains the realisations.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[GaussRF](#), [MaxStableRF](#), [RandomFields](#)

sleep.milli

Sleep

Description

Process sleeps for a given amount of time

Usage

```
sleep.milli(milli)
```

Arguments

`milli` sleeping time in milliseconds

Value

No value is returned.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

soil

Soil data of North Bavaria, Germany

Description

Soil physical and chemical data collected on a field in the Weissenstaedter Becken, Germany

Usage

```
data(soil)
```

Format

This data frame contains the following columns:

x.coord x coordinates given in cm

y.coord y coordinates given in cm

nr number of the samples, which were taken in this order

moisture moisture content [Kg/Kg * 100%]

NO3.N nitrate nitrogen [mg/Kg]

Total.N total nitrogen [mg/Kg]

NH4.N ammonium nitrogen [mg/Kg]

DOC dissolved organic carbon [mg/Kg]

N20N nitrous oxide [mg/Kg dried substance]

Details

For technical reasons some of the data were obtained as differences of two measurements (which are not available anymore). Therefore, some of the data have negative values.

Source

The data were collected by Wolfgang Falk, Soil Physics Group, <http://www.geo.uni-bayreuth.de/bodenphysik/Welcome.html>, University of Bayreuth, Germany.

References

Falk, W. (2000) *Kleinskalige räumliche Variabilität von Lachgas und bodenchemischen Parametern [Small Scale Spatial Variability of Nitrous Oxide and Pedo-Chemical Parameters]*, Master thesis, University of Bayreuth, Germany.

Examples

```
#####
##                                     ##
##      a geostatistical analysis that demonstrates      ##
##      features of the package 'RandomFields'          ##
##                                                     ##
#####

data(soil)
names(soil)
pts <- soil[,c(1,2)]
d <- soil$moisture

## define some graphical parameters first
close.screen(close.screen())
maxbin <- max(dist(pts)) / 2
(zlim <- range(d))
cn <- 100
colour <- rainbow(cn)
par(bg="white",cex=1, cex.lab=1.4, cex.axis=1.4, mar=c(4.3,4.3,0.8,0.8))
lu.x <- min(soil$x)
lu.y <- max(soil$y)
y <- x <- seq(min(soil$x), max(soil$x), l=121)

## ... and a certain appearance of the legend
my.legend <- function(lu.x, lu.y, zlim, col, cex=1) {
  ## uses already the legend code of R-1.3.0
  cn <- length(col)
  filler <- vector("character", length=(cn-3)/2)
  legend(lu.x, lu.y, y.i=0.03, x.i=0.1,
         legend=c(format(zlim[2], dig=2), filler,
                    format(mean(zlim), dig=2), filler,
                    format(zlim[1], dig=2)),
         lty=1, col=rev(col),cex=cex)
}
```

```

## plot the data first
plot(pts, col=colour[1+(cn-1)*((d-zlim[1])/diff(zlim))], pch=16,
      xlab="x [cm]", ylab="y [cm]", cex.axis=1.5, cex.lab=1.5)
my.legend(lu.x, lu.y, zlim=zlim, col=colour, cex=1.5)

## empirical variogram
ev <- EmpiricalVariogram(pts, data=d, grid=FALSE,
                         bin=c(-1,seq(0,maxbin,l=30)))

## NOTE: the next command requires a selection of the
##       model by mouse clicks in the graphics window
##       see help("Showmodels") for further details

by.eye <- ShowModels(x=x, y=y, emp=ev, col=colour, Zlim=zlim,
                    Mean=mean(d), me="ci")

## fit parameters of the whittlematern model by MLE

fit <- fitvario(x=pts, data=d, model="whittle", par=rep(NA,5),
               mle.m="ml", cross.m=NULL)
str(fit)

## plot the fitted model and the empirical variogram
plot(ev$c, ev$emp.var, ylim=c(0,11), ylab="variogram", xlab="lag")
gx <- seq(0.001, max(ev$c), l=100)
if(!is.null(by.eye)) lines(gx, Variogram(gx, model=by.eye))
lines(gx, Variogram(gx, model=fit$ml$model), col=2)
lines(gx, Variogram(gx, model=fit$plain$model), col=3)
lines(gx, Variogram(gx, model=fit$sqrt.nr$model), col=4)
lines(gx, Variogram(gx, model=fit$sd.inv$model), col=6, lty=2)
legend(120, 4, c("empirical", "by eye", "ML", "lsq", "sqrt(n) lsq",
               "sd^-1 lsq"),
      lty=c(-1, rep(1, 5)), pch=c(1, rep(-1, 5)),
      col=c(1, 1, 2, 3, 4, 6), cex=1.4)

## map of expected values

k <- Kriging("0", x=x, y=y, grid=TRUE, model=fit$ml$model, given=pts, data=d)
par(mfrow=c(1,2))
plot(pts, col=colour[1+(cn-1)*((d-zlim[1])/diff(zlim))],
      pch=16, xlab="x [cm]", ylab="y [cm]")
my.legend(lu.x, lu.y, zlim=zlim, col=colour, cex=1)
image(x, y, k, col=colour, zlim=zlim, xlab="x [cm]", ylab="y [cm]")
par(bg="white")

```

```

## what is the probability that at no point of the
## grid given by x and y the moisture is greater than 24 percent?

RFparameters(Print=1, CE.force=FALSE, CE.trials=3, CE.useprimes=TRUE)
cs <- CondSimu("0", x=x, y=y, grid=TRUE, model=fit$m1$model, given=pts,
              data=d, n=10) # better n=100 or n=1000
par(mfcol=c(2,3))
plot(pts, col=colour[1+(cn-1)*((d-zlim[1])/diff(zlim))], pch=16,
     xlab="x [cm]", ylab="y [cm]", cex.axis=1.5, cex.lab=1.5)
my.legend(lu.x, lu.y, zlim=zlim, col=colour, cex=0.5)
image(x, y, k, col=colour, zlim=zlim, xlab="x [cm]", ylab="y [cm]")
for (i in 1:4)
  image(x, y, cs[, , i], col=colour, zlim=zlim,
       xlab="x [cm]", ylab="y [cm]")

mean(apply(cs<=24, 3, all)) ## about 40 percent ...

```

Description

Covariance returns the values of complex stationary and nonstationary covariance functions; see [CovarianceFct](#) for basic isotropic models

Details

Here only the non-isotropic and hyper models are listed; see [CovarianceFct](#) for basic isotropic models.

The implemented models are in standard notation for a covariance function (variance 1, nugget 0, scale 1) and for positive real arguments h (and t) for the stationary models or parts:

- +
Operator that adds up at most 10 submodels
- *
Operator that multiplies at most 10 submodels
- \$

$$C(x, y) = vC(x/s, y/s)$$

$$C(x, y) = vC(xa, ya)$$

$$C(x, y) = vC(Ax, Ay)$$

$$C(x, y) = vC(px, py)$$

Operator that modifies the the variance ($v = \text{var}$) and the coordinates or distances by

- the scale ($s = \text{scale}$) or
- the anisotropy matrix $a = \text{anisoT}$ multiplied from the right or
- the anisotropy matrix A multiplied from the left or
- $p = \text{proj}$ on a lower dimensional space along the coordinate axis

The parameter $scale$ is positive, $aniso$ and A are matrices, and $proj$ is a vector indices with between 1 and the dimension of x . Note, at most one of the parameters, $anisoT$, A , $proj$ may be given at the same time.

The operator $\$$ has 1 submodel. If the dimension of the field is 1 or $aniso$ is not given, the operator allows for derivatives.

- `ave1`

$$C(h, u) = |E + 2Ahh^tA|^{-1/2} \phi(\sqrt{(\|h\|^2/2 + (z^th + u)^2(1 - 2h^tA(E + 2Ahh^tA)^{-1}Ah))})$$

where E is the identity matrix.

A is a symmetric positive definite $(d - 1) \times (d - 1)$ and z is a $d - 1$ dimensional vector. The function ϕ is normal mixture model, e.g. `whittle` model, see `CovarianceFct` and `PrintModellist()`.

- `ave2` (nonstationary)

Here $C(h) = C_0(h, 0)$ where C_0 is the `ave1` model.

- `biWM` (bivariate model)

$$C_{ij}(h) = c_{ij}W_{\nu_{ij}}(h/s_{ij})$$

where $W_n u$ is the `whittle` model and $i, j = 1, 2$. For ($i=j$) the constants $\nu_{ii}, s_{ii}, c_{ii} > 0$. For the offdiagonal elements with have $C_{12} = C_{21}, s_{12} = s_{21} > 0, \nu_{12} = \nu_{21} = 0.5(\nu_{11} + \nu_{22})/\nu_{red}$ for some constant $\nu_{red} \in (0, 1]$. The scalar $c_{12} = c_{21} = \rho_{red} \sqrt{f m c_{11} c_{22}}$ where

$$f = \Gamma(\nu_{11} + d/2) * \Gamma(\nu_{22} + d/2) / \Gamma(\nu_{11}) / \Gamma(\nu_{22}) * (\Gamma(\nu_{12}) / \Gamma(\nu_{12} + d/2))^2 * (s_{12}^{2*\nu_{12}} / s_{11}^{\nu_{11}} / s_{22}^{\nu_{22}} /)^2$$

and Γ is the Gamma function and d is the dimension of the space. The constant m is the infimum of the function g on $[0, \infty)$,

$$g(t) = (1/s_{12}^2 + t^2)^{2\nu_{12} + d} (1/s_{11}^2 + t^2)^{-\nu_{11} - d/2} (1/s_{22}^2 + t^2)^{-\nu_{22} - d/2}$$

see the reference below for details on the infimum.

The model now has the parameters

$$\text{nu} = (\text{nu}_{11}, \text{nu}_{22})$$

$$\text{nured12} = \nu_{red}$$

$$s = (s_{11}, s_{22})$$

$$s12 = s_{12} = s_{21} \setminus c = (c_{11}, c_{22})$$

$$\text{rhored} = \rho_{red} \text{ See also } \text{parsbiWM}.$$

- `constant`

This model is designed for the use in `fitvario` as a part of a linear model definition. Its only parameter is a covariance matrix of appropriate size to match the number of (non-repeated) observations or the number of columns of parameters X in model `mixed`, see `sophisticated`.

- coxisham

$$C(h, u) = |E + u^\beta D|^{-1/2} \phi([(h - u\mu)^t (E + u^\beta D)^{-1} (h - u\mu)]^{1/2})$$

Here u is vector; E is the identity matrix and D is a correlation matrix with $|D| > 0$. Currently implementation is done only for $d = 2$. The parameter β is in $(0, 2]$ and equals 2 by default.

- curlfree (multivariate)

$$(-\nabla_x \nabla_x^T) C_0(x, t)$$

C_0 is a univariate covariance model that is motion invariant and at least twice differentiable in the first component. The operator is applied to the first component only. The model returns the potential field in the first component, the corresponding curlfree field and field of sources and sinks in the last component. The above formula for the covariance function only gives the part for the curlfree field. The complete matrix-valued correlation function, including all components, is more complicated.

C_0 is either a spatiotemporal model (then t is the time component) or it is an isotropic model. Then, the first $Dspace$ coordinates are considered as x coordinates and the remaining ones as t coordinates. By default, $Dspace$ equals the dimension of the field (and t is identically 0).

See also the models `divfree` and `vector`.

- cutoff

$$\begin{aligned} C(h) &= \phi(h), 0 \leq h \leq d \\ C(h) &= b_0((dr)^a - h^a)^{2a}, d \leq h \leq dr \\ C(h) &= 0, dr \leq h \end{aligned}$$

The cutoff model is a functional of the covariance function ϕ .

Here, $d > 0$ should be the diameter of the domain on which simulation is done. The parameter $a > 0$ has been shown to be optimal for $a = 1/2$ or $a = 1$.

The parameters r and b_0 are chosen internally such that C is a smooth function.

NOTE: The algorithm that checks the given parameters knows only about some few necessary conditions. Hence it is not ensured that the cutoff-model is a valid covariance function for any choice of ϕ and the parameters.

For certain models ϕ , i.e. stable, whittle and gencauchy, some sufficient conditions are known.

- delayeffect (bivariate)

$$C_{11}(h) = C_{22}(h) = C_0(h) \quad C_{12}(h) = C_0(h + r), C_{21}(h) = C_0(-h + r)$$

Here r is a vector of the dimension of the random field, and C_0 is a translation invariant, univariate covariance model.

- divfree (multivariate)

$$(-\Delta E + \nabla \nabla^T) C_0(x, t)$$

C_0 is a univariate covariance model that is motion invariant and at least twice differentiable in the first component. The operator is applied to the first component only. The model returns the potential field in the first component, the corresponding divfree field and the field of curl strength in the last component. The above formula for the covariance function only gives

the part for the divfree field. The complete matrix-valued correlation function, including all components, is more complicated.

C_0 is either a spatiotemporal model (then t is the time component) or it is an isotropic model. Then, the first $Dspace$ coordinates are considered as x coordinates and the remaining ones as t coordinates. By default, $Dspace$ equals the dimension of the field (and t is identically 0).

See also the models `curlfree` and `vector`.

- `EtAxxA` (auxiliary function)

$$S(x) = E + R^t A^t x x^t A R, \quad x \in R^3$$

where E and A are arbitrary 3×3 matrices and R is a rotation matrix,

$$R = \begin{pmatrix} \cos(\alpha x_3) & -\sin(\alpha x_3) & 0 \\ \sin(\alpha x_3) & \cos(\alpha x_3) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This is not a covariance function, but can be used as a submodel for certain classes of non-stationary covariance functions.

- `Exp`

$$C(h) = \exp(-\gamma(h))$$

where γ is a valid variogram. If a stationary covariance model C is given in stead of γ , this is automatically turned into a variogram model, i.e. $C(h) = \exp(-C(0) + C(h))$.

- `M`

$$C(h) = M^t \phi(h) M$$

Here ϕ is a k -variate variogram or covariance, and M is any $m \times k$ matrix.

- `ma1`

$$C(h) = (\theta / (1 - (1 - \theta) * C_0(h)))^\alpha$$

Here, C_0 is any correlation function, $\alpha \in (0, \infty)$ and $\theta \in (0, 1)$.

- `ma2`

$$C(h) = (1 - \exp(-\gamma(h))) / \gamma(h)$$

Here γ is a variogram model.

- `mastein`

$$C(h, t) = \frac{\Gamma(\nu + \gamma(t)) \Gamma(\nu + \delta)}{\Gamma(\nu + \gamma(t) + \delta) \Gamma(\nu)} W_{\nu + \gamma(t)}(\|h - Vt\|)$$

Γ is the Gamma function; $\gamma(t)$ is a variogram on the real axis; W is the Whittle-Matern model. Here, the names of covariance models can also be used; the algorithm chooses the corresponding variograms then. The parameter ν is the smoothness parameter of the Whittle-Matern model (for $t = 0$) and must be positive. Finally, δ must be greater than or equal to half the dimension of h . Instead of the velocity parameter V in original model description, a preceding anisotropy matrix is chosen appropriately:

$$\begin{pmatrix} A & -V \\ 0 & 1 \end{pmatrix}$$

A is a spatial transformation matrix. (I.e. (x, t) is multiplied from left on the above matrix and the first elements of the obtained vector are interpreted as new spatial components and only

these components are used to form the argument in the Whittle-Matern function.) The last component in the new coordinates is the time which is passed to γ . (Velocity is assumed to be zero in the new coordinates.)

Note, that for numerical reasons, $\nu + \gamma + d$ may not exceed the value 80.0. If exceeded the algorithm fails.

- **mixed** This model is designed for the use in `fitvario` to build up linear regression models with fixed effects, mixed effects, including geoadditive parts.

The model has two parameters. The first, X is a matrix of independent variables. The second, b , is a vector of regression coefficients. Furthermore a submodel, `covb`, may give the covariance structure for b .

Let n the number of (non-repeated) observations. The following combinations are allowed:

- only X is given. Then X is a scalar or a vector of length n , and X defines a known mean.
- X and b are given. Then X is a $(n \times m)$ matrix where m is the length of the vector b . Then a fixed effect is defined.
- X and `covb` are given.
 - * if `covb` is the model *constant*, then we have a random model (maybe with preceding model \$).
 - * if `covb` is any other model then we have a geoadditive part

The data in the `fitvario` may contain NAs, but not X .

- **mcam** (multivariate quasi-arithmetic mean)

$$C_{ij}(h) = \rho_{ij} \phi(\theta \phi^{-1}(C_i(h)) + (1 - \theta) \phi^{-1}(C_j(h)))$$

where ϕ is a completely monotone function and C_i are suitable covariance functions.

The submodel ϕ is given (by name) as first submodel. Since ϕ is completely monotone if and only if $\phi(\|\cdot\|^2)$ is a valid covariance function for all dimensions, e.g. `stable`, `gauss`, `exponential`, ϕ is given by the name of the corresponding covariance function C , i.e. `phi(.) = C(sqrt(.))`.

Warning: `RandomFields` cannot check whether the combination of ϕ and C_i is valid.

- **natsc**

$$C(h) = C_0(h/s)$$

Where C_0 is any stationary and isotropic model. The parameter s is chosen by `natsc` such that the practical range (or the mathematical range, if finite) is 1.

- **nonstWM**

$$\begin{aligned} C(x, y) &= \Gamma(\mu) \Gamma(\nu(x))^{-1/2} \Gamma(\nu(y))^{-1/2} W_\mu(\|x - y\|) \\ &= 2^{1-\mu} \Gamma(\nu(x))^{-1/2} \Gamma(\nu(y))^{-1/2} \|x - y\|^\mu K_\nu(\|x - y\|) \end{aligned}$$

where $\mu = [\nu(x) + \nu(y)]/2$ and ν is a positive function. If ν is a scalar use the variable `nu`. If ν is a function, use the submodel `Nu`. Note that for `Nu` the usual list structure applies and only the defined covariance models can be used.

- **nsst** (Non-Separable Space-Time model)

$$C(h, u) = (\psi(u) + 1)^{-\delta/2} \phi(h/\sqrt{\psi(u) + 1})$$

The parameter δ must be greater than or equal to the spatial dimension of the field. ϕ is normal mixture model and ψ is a variogram.

This model is used for space-time modelling where the spatial component is isotropic.

- nugget (multivariate model)

$$C(h) = \text{diag}(1, \dots, 1)1_{\{0\}}(h)$$

The components of the multivariate vector are always independent. The model adapts the multivariate dimension to the calling model.

- parsbiWM (bivariate model)

$$C_{ij}(h) = c_{ij}W_{\nu_{ij}}(h/s)$$

where $W_n u$ is the whittle model and $i, j = 1, 2$. For ($i=j$) the constants $\nu_{ii}, c_{ii} \geq 0$ and $s > 0$. For the offdiagonal elements with have $C_{12} = C_{21}$. Furthermore, $\nu_{12} = \nu_{21} = 0.5(\nu_{11} + \nu_{22})$ and the scalar $c_{12} = c_{21} = \rho_{red} \sqrt{f m c_{11} c_{22}}$ where

$$f = \Gamma(\nu_{11} + d/2) * \Gamma(\nu_{22} + d/2) / \Gamma(\nu_{11}) / \Gamma(\nu_{22}) * (\Gamma(\nu_{12}) / \Gamma(\nu_{12} + d/2))^2$$

and Γ is the Gamma function and d is the dimension of the space. The constant m is the infimum of the function g on $[0, \infty)$,

$$g(t) = (1/s_{12}^2 + t^2)^{2\nu_{12}+d} (1/s_{11}^2 + t^2)^{-\nu_{11}-d/2} (1/s_{22}^2 + t^2)^{-\nu_{22}-d/2}$$

see the reference below for details on the infimum.

The model now has the parameters

$$\text{nu} = (\nu_{11}, \nu_{22})$$

$$\text{s} = (s_{11}, s_{22})$$

$$\text{s12} = s_{12} = s_{21}$$

$$\text{c} = (c_{11}, c_{22})$$

$$\text{rhored} = \rho_{red}$$

See also biWM.

- Pow

$$\gamma(h) = (\gamma_0(h))^\alpha$$

or

$$C(h) = C_0(0) - [C_0(0) - C_0(h)]^\alpha$$

where γ_0 is a valid variogram or C_0 is a valid covariance function, and $\alpha \in [0, 1]$.

- qam (Quasi-arithmetic mean)

$$C(h) = \phi\left(\sum_i \theta_i \phi^{-1}(C_i(h))\right)$$

where ϕ is a completely monotone function and C_i are suitable covariance functions.

The submodel ϕ is given (by name) as first submodel. Since ϕ is completely monotone if and only if $\phi(\|\cdot\|^2)$ is a valid covariance function for all dimensions, e.g. `stable`, `gauss`, `exponential`, ϕ is given by the name of the corresponding covariance function C , i.e. $\text{phi}(\cdot) = C(\text{sqrt}(\cdot))$.

Warning: `RandomFields` cannot check whether the combination of ϕ and C_i is valid.

- rational (auxiliary)

$$S(x) = (a_0 + a_1 * x^t AA^t x) / (1 + x^t AA^t x)$$

where is some $d \times d$ matrix and $a = (a_0, a_1)$ is a 2-dimensional vector.

- Rotat(auxiliary function)

$$S^t(x) = x^t R, \quad x \in R^3$$

where and R is a rotation matrix,

$$R = \begin{pmatrix} \cos(\alpha x_3) & -\sin(\alpha x_3) & 0 \\ \sin(\alpha x_3) & \cos \alpha x_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This is not a covariance function, but can be used a submodel for certain classes of non-stationary covariance functions.

- Stein

$$C(h) = a_0 + a_2(h)^2 + \phi(h), 0 \leq h \leq D$$

$$C(h) = b_0(rD - h)^3 / (h), r \leq h \leq rD$$

$$C(h) = 0, rD \leq h$$

The Stein model is a functional of the covariance function ϕ .

Here, $D > 0$ should be the diameter of the domain on which simulation is done, $r \geq 1$. The parameters a_0, a_2 and b_0 are chosen internally such that C becomes a smooth function.

NOTE: The algorithm that checks the given parameters knows only about some few necessary conditions. Hence it is not ensured that the Stein-model is a valid covariance function for any choice of phi and the parameters.

For certain models ϕ , i.e. stable, whittle, gencauchy, and the variogram model fractalB some sufficient conditions are known.

- steinst1 (non-separable space time model)

$$C(h, t) = W_\nu(y) - \frac{\langle h, z \rangle t}{(\nu - 1)(2\nu + d)} W_{\nu-1}(y)$$

Here, W_ν is the Whittle-Matern model with smoothness parameter ν ; $y = \|(h, t)\|$. z is a vector whose norm must less than or equal to 1.

- stp

$$C(x, y) = |S_x|^{1/4} |S_y|^{1/4} |A|^{-1/2} \phi(Q(x, y)^{1/2})$$

where

$$Q(x, y) = c^2 - m^2 + h^t (S_x + 2(m + c)M) A^{-1} (A_y + 2(m - c)M) h,$$

$$c = -z^t h + \xi_2(x) - \xi_2(y),$$

$$A = S_x + S_y + 4M h h^t M$$

$$m = h^t M h.$$

$$h = H(x) - H(y)$$

The parameters are

- S_x (strictly) positive definite matrices for $x \in R^d$
- M an arbitrary $d \times d$ matrix
- $z \in R^d$ arbitrary
- H arbitrary d-variate function on R^d
- ξ arbitrary univariate function on R^d
- ϕ a normal mixture model

The model allows for mimicking cyclonic behaviour.

- `tbm2`

$$C(h) = \frac{d}{dh} \int_0^h \frac{u\phi(u)}{\sqrt{h^2 - u^2}} du$$

for some stationary and isotropic covariance ϕ that is valid in at least 2 dimensions.

This operator is currently only designed for internal use!

- `tbm3`

$$C(h) = \phi(h) + h\phi'(h)/n$$

which, for $n=1$ reduced to the standard TBM operator

$$C(h) = \frac{d}{dh} h\phi(h)$$

for some stationary and isotropic covariance ϕ that is valid in at least $n + 2$ dimensions. n should be an integer.

This operator is currently only designed for internal use!

- `vector` (multivariate)

$$(-0.5 * (a + 1)\Delta E + a\nabla\nabla^T)C_0(x, t)$$

C_0 is a univariate covariance model that is motion invariant and at least twice differentiable in the first component. The operator is applied to the first component only. The parameter a is in $[-1, 1]$. If $a = -1$ then the field is curl free; if $a = 1$ then the field is divergence free.

C_0 is either a spatiotemporal model (then t is the time component) or it is an isotropic model. Then, the first $Dspace$ coordinates are considered as x coordinates and the remaining ones as t coordinates. By default, $Dspace$ equals the dimension of the field (and t is identically 0).

See also the models `divfree` and `curlfree`

See [CovarianceFct](#) for comments on the use of a covariance model.

However, for the above sophisticated models, the following differences should be considered:

- `RFparameters()`\$`PracticalRange` is usually not defined for the above models
- only the list notation can be used, but not the simple model definitions with `model="name"` and `param=c(mean, variance, nugget, scale, ...)`.
- the use of `Covariance` is obligatory if the model is non-stationary.
- the anisotropy matrix belonging to a hypermodel is applied first to the coordinates before any call of the submodels.

To use the above models, a new, very flexible, straight forward list notation is needed. Background of this notation is that we have ‘primitives’, i.e. functions that are positive definite. And we have ‘operators’, i.e. functionals that make out of given variograms, covariance functions etc. new models. Examples are "+", "*", or Gneiting's "nsst". Consequently, we need also an operator, called "\$", that changes the variance and the scale.

E.g. a standard exponential model (variance=1, scale=1, nugget=0) is now simply written as

```
list("exponential")
```

(And no param must be given!)

Further, a standard exponential model with a nugget effect, nugget variance 3, is now written as

```
list("+",
list("exponential"),
list("$", var=3, list("nugget"))
)
```

Here, only the relevant parameters need to be given; the missing parameters get standard values whenever standard values exist, e.g. variance equals 1 if not given. Further, the parameters can (and must) be called by names, which makes complex models much more readable. Submodels, as `list("exponential")` in the second example above, can (but need not) be called by name.

Value

CovarianceFct and Covariance return a vector of values of the covariance function.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

References

Overviews:

- see reference list in [CovarianceFct](#)

ave1, ave2

- Schlather, M. (2010) On some covariance models based on normal scale mixtures. *Bernoulli*, **16**, 780-797. (Example 13)

biWM, parsbiWM

- Gneiting, T., Kleiber, W., Schlather, M. (2011) Matern covariance functions for multivariate random fields *JASA*

coxisham

- Cox, D.R., Isham, V.S. (1988) A simple spatial-temporal model of rainfall. *Proc. R. Soc. Lond. A*, **415**, 317-328.
- Schlather, M. (2010) On some covariance models based on normal scale mixtures. *Bernoulli*, **16**, 780-797.

curlfree

- see vector

cutoff

- Gneiting, T., Sevecikova, H, Percival, D.B., Schlather M., Jiang Y. (2006) Fast and Exact Simulation of Large Gaussian Lattice Systems in \mathbb{R}^2 : Exploring the Limits. *J. Comput. Graph. Stat.* **15**, 483-501.
- Stein, M.

delayeffect

- Wackernagel, H. (2003) *Multivariate Geostatistics*. Berlin: Springer, 3rd edition.

divfree

- see vector

Iaco-Cesare model

- de Cesare, L., Myers, D.E., and Posa, D. (2002) FORTRAN programs for space-time modeling. *Computers & Geosciences* **28**, 205-212.
- de Iaco, S., Myers, D.E., and Posa, D. (2002) Nonseparable space-time covariance models: some parameteric families. *Math. Geol.* **34**, 23-42.

vector

- Fuselier, E.J. (2006) *Refined Error Estimates for Matrix-Valued Radial Basis Functions* PhD thesis. Texas A&M University
- Scheuerer, M. and Schlather, M. (2011) Covariance Models for Random Vector Fields *Submitted*

Ma-Stein model

- Ma, C. (2003) Spatio-temporal covariance functions generated by mixtures. *Math. Geol.*, **34**, 965-975.
- Stein, M.L. (2005) Space-time covariance functions. *JASA*, **100**, 310-321.

ma1/ma2

-

mixed

- Ober, U., Erbe, M., Porcu, E., Schlather, M. and Simianer, H. (2011) Kernel-Based Best Linear Unbiased Prediction with Genomic Data. *Submitted*.

nonstWM/hyperbolic/cauchy

- Stein, M. (2005) Nonstationary Spatial Covariance Functions. Tech. Rep., 2005

nsst

- Gneiting, T. (1997) Normal scale mixtures and dual probability densities, *J. Stat. Comput. Simul.* **59**, 375-384.
- Gneiting, T. (2002) Nonseparable, stationary covariance functions for space-time data, *JASA* **97**, 590-600.
- Gneiting, T. and Schlather, M. (2001) Space-time covariance models. In El-Shaarawi, A.H. and Piegorsch, W.W.: *The Encyclopedia of Environmetrics*. Chichester: Wiley.
- Zastavnyi, V. and Porcu, E. (2011) Characterization theorems for the Gneiting class space-time covariances. *Bernoulli*, ??.
- Schlather, M. (2010) On some covariance models based on normal scale mixtures. *Bernoulli*, **16**, 780-797.

Quasi-arithmetic means (qam, mqam)

- Porcu, E., Mateu, J. & Cchristakos, G. (2007) Quasi-arithmetic means of covariance functions with potential applications to space-time data. Submitted to Journal of Multivariate Analysis.
-

Paciorek-Stein (steinst1)

- Stein, M. (2005) Nonstationary Spatial Covariance Functions. Tech. Rep., 2005
- Paciorek, C. (2003) *Nonstationary Gaussian Processes for Regression and Spatial Modelling*, Carnegie Mellon University, Department of Statistics, PhD thesis.

Stein

- Stein, M.

stp

- Schlather, M. (2008) On some covariance models based on normal scale mixtures. *Submitted*

tbm

- Gneiting, T. (1999) On the derivatives of radial positive definite function. *J. Math. Anal. Appl.*, **236**, 86-99
- Matheron, G. (1973). The intrinsic random functions and their applications. *Adv . Appl. Probab.*, **5**, 439-468.

See Also

[CovarianceFct](#), [EmpiricalVariogram](#), [GetPracticalRange](#), [parameter.range](#), [RandomFields](#), [RFparameters](#), [ShowModels](#).

Examples

```

PrintModelList(op=TRUE)

## the subsequent model can be used to model rainfall...
y <- x <- seq(0, 10, len=25) # better 256 -- but will take a while
T <- c(0, 10, 1) # better 0.1
col <- c(topo.colors(300)[1:100], cm.colors(300)[c((1:50) * 2, 101:150)])

model <- list("coxisham", mu=c(1, 1), D=matrix(nr=2, c(1, 0.5, 0.5, 1)),
             list("whittle", nu=1)
            )

system.time(z <- GaussRF(x, y, T=T, grid =TRUE, spectral.lines=1500,
                       model = model))

zlim <- range(z)
time <- dim(z)[3]
for (i in 1:time) {
  Print(i)
  sleep.milli(100)
  image(x, y, z[, , i], add=i>1, col=col, zlim=zlim)
}

#####
#####

# the following five model definitions are the same!
## (1) very traditional form
(cv <- CovarianceFct(x, model="bessel", param=c(NA, 2 , 1, 5, 0.5)))

## (2) traditional form in list notation
model <- list(model="bessel", param=c(NA, 2, 1, 5, 0.5))
cv - CovarianceFct(x, model=model)

## (3) nested model definition
cv - CovarianceFct(x, model="bessel",
                  param=rbind(c(2, 5, 0.5), c(1, 0, 0)))

#### most general notation in form of lists
## (4) isotropic notation
model <- list("+",
             list("$", var=2, scale=5, list("bessel", 0.5)),
             list("nugget"))
cv - CovarianceFct(x, model=model)

## (5) anisotropic notation
model <- list("+",
             list("$", var=2, aniso=0.2, list("bessel", 0.5)),
             list("nugget"))

```

```

cv - CovarianceFct(as.matrix(x), model=model)

#####
#####

# The model gneitingdiff was defined in RandomFields v1.0.
# This isotropic covariance function is valid for dimensions less
# than or equal to 3 and has two positive parameters.
# It is a class of models with compact support that allows for
# smooth parametrisation of the differentiability up to order 6.
# The former model 'gneitingdiff' should now be coded as

gneitingdiff <- function(p){
  list("+",
    list("$", var=p[3], list("nugget")),
    list("$", scale=p[4],
      list("*",
        list("$", var=p[2], scale=p[6], list("gneiting")),
        list("whittle", nu=p[5])
      )
    )
  )
}

# and then
param <- c(NA, runif(5, max=10))
CovarianceFct(0:100, model=gneitingdiff(param))
## instead of formerly CovarianceFct(x,"gneitingdiff",param)

```

useraction

Set input behaviour

Description

The functions store values that influence the behaviour of [Locator](#) and [Readline](#)

Usage

```

useraction(action=c("none", "start.register", "continue.register",
  "replay", "endless", "delete"),
  sleep, wait, PrintLevel, actionlist)
putactions(l)

```

Arguments

action string of the following values

	<p>“none” the behaviour of <code>Readline</code> and <code>Locator</code> is that of <code>readline</code> and <code>locator</code>, respectively. The values are not stored.</p> <p>“start.register” the behaviour of <code>Readline</code> and <code>Locator</code> is that of <code>readline</code> and <code>locator</code>, respectively. The current list of stored values is deleted and new inputs by <code>Readline</code> and <code>Locator</code> are stored.</p> <p>“continue.register” the behaviour of <code>Readline</code> and <code>Locator</code> is that of <code>readline</code> and <code>locator</code>, respectively. The current list of stored values is continued with inputs from <code>Readline</code> and <code>Locator</code>.</p> <p>“replay” <code>Locator</code> and <code>Readline</code> do not get the data from the mouse or terminal, but read them from the stored list. If the end of storage list is reached, the user is informed and the action is switched to “none”.</p> <p>“endless” similar to “replay”, but storage list is looped.</p> <p>“delete” The current list of stored values is deleted and action is set to “none”</p>
actionlist	list of stored values by <code>Readline</code> and <code>Locator</code>
sleep	sleeping time before the blinking starts (replay mode of <code>Locator</code>); value is kept until explicit changing; starting value: 2.5
wait	regulated the blinking speed of the characters (<code>Locator</code>) and of the speed of appearance of characters (<code>Readline</code>); value is kept until explicit changing; starting value: 0.1
PrintLevel	if <code>Printlevel</code> >1 and <code>Readline</code> or <code>Locator</code> is in replay mode then the current pointer position within the storage list and the tracing information (if any) is shown. The value is kept until explicit changing; starting value: 0
1	list of all parameters relevant to the behaviour of <code>Readline</code> and <code>Locator</code> .

Details

The function `putactions` restores the state taken by `getactions`.

Concerning the function `useraction`, the parameter `action` must be given. If any of the other parameters is not given, its current value is kept. If `action="start.register"` or `action="delete"`, the value of `actionlist` is ignored.

If replaying, `Readline` shows the input values on the screen and `Locator` shows blinking characters as side effects.

Value

`useraction` returns `NULL`.

`putactions` returns invisibly the state before the new state is set.

Author(s)

Martin Schlather, <martin.schlather@math.uni-goettingen.de> <http://www.stochastik.math.uni-goettingen.de/~schlather>

See Also

[eval.parameters](#), [getactions](#), [Locator](#), [Readline](#),

Examples

```
sample.interaction <- function(read.r2=TRUE, type="p", pch=16) {
  r1 <- Readline(prompt="first input: ")
  r2 <- if (read.r2) Readline(prompt="second input: ", info="2nd input")
  else " -- no input -- "
  cat("Mark some points with the right mouse key, then leave with the left mouse key\n")
  do.call(getOption("device"), list())
  plot(Inf, Inf, xlim=c(-1,1), ylim=c(-1,1), xlab="", ylab="")
  l <- Locator(100, info="locator input", type=type, pch=pch)
  r3 <- Readline(prompt="third input: ", info="last input")
  return(list(r1=r1, r2=r2, l=l, r3=r3))
}
```

```
## Not run:
```

```
#####
```

```
## user terminal input ##
```

```
#####
```

```
{
  useraction("start.register")
  str(sample.interaction())
  str(l <- getactionlist())
}
```

```
#####
```

```
## just replay ##
```

```
#####
```

```
useraction("replay", sleep=2, wait=interactive() * 0.2 * 5,
  PrintLevel=2, actionlist=l)
str(sample.interaction(type="l"))
```

```
#####
```

```
## modify first the input list, then replay ##
```

```
#####
```

```
l2 <- l[-2]
l2[[1]] <- list("some other words", info="changed input")
str(l2)
useraction("replay", sleep=1, wait=interactive() * 0.05,
  PrintLevel=0, l2)
str(sample.interaction(read.r2=FALSE, type="o")) # input now from l2
Readline(prompt="new input: ", info="?!") # switch to terminal
##                               since end of stored list
# str(getactionlist()) # new input has not been not stored ...
```

```
#####
```

```
## use of the two lists, l and l2, in a mixed way ##
```

```
#####
useraction("replay", sleep=2, wait=interactive() * 0.05,
           PrintLevel=0, actionlist=1)
Readline(prompt="first input of 1: ")
dump <- getactions()
useraction("replay", sleep=0.5, wait=interactive() * 0.05,
           PrintLevel=0, actionlist=12)
Readline(prompt="first input of 2: ")
dump2 <- putactions(dump)
Readline(prompt="second input of 1: ")
## locator call reading from list 1:
plot(Inf, Inf, xlim=c(-1,1), ylim=c(-1,1), xlab="", ylab="")
Locator(100, info="locator input", type="p")
dump <- putactions(dump2)
##locator call reading from list 2:
Locator(100, info="locator input", type="p", pch=20, col="blue")
Readline(prompt="last input of 2: ")
putactions(dump)
Readline(prompt="last input of 1: ")

#####
## see help("eval.parameters") for another example ##
#####

## End(Not run)
```

weather

Pressure and temperature forecast errors over the Pacific Northwest

Description

Meteorological dataset, which consists of difference between forecasts and observations (forecasts minus observations) of temperature and pressure at 157 locations in the North American Pacific Northwest.

Usage

```
data(weather)
```

Format

This data frame contains the following columns:

pressure in units of Pascal

temperature in units of degree Celcius

lon longitudinal coordinates of the locations

lat latitude coordinates of the locations

Details

The forecasts are from the GFS member of the University of Washington regional numerical weather prediction ensemble (UWME; Gritmit and Mass 2002; Eckel and Mass 2005); they were valid on December 18, 2003 at 4 pm local time, at a forecast horizon of 48 hours.

Source

The data were obtained from Cliff Mass and Jeff Baars in the University of Washington Department of Atmospheric Sciences.

References

- Eckel, A. F. and Mass, C. F. (2005) Aspects of effective mesoscale, short-range ensemble forecasting *Wea. Forecasting* **20**, 328-350.
- Gneiting, T., Kleiber, W. and Schlather, M. (2011) Matern cross-covariance functions for multivariate random fields *J. Amer. Statist. Assoc.* **105**, 1167-1177.
- Gritmit, E. P. and Mass, C. F. (2002) Initial results of a mesoscale short-range forecasting system over the Pacific Northwest *Wea. Forecasting* **17**, 192-205.

Examples

```
## Not run:

#

#####
##                               ##
##   T. Gneiting, W. Kleiber, M. Schlather, JASA 2011   ##
##                               ##
#####

## Here the analysis in the above mentioned paper is replicated.
## The results differ slightly from those in the paper, as the algorithm
## has been improved, and the estimation has been nearly fully automated.
## In particular, user supplied lower and upper bound for estimating
## the independent model are no longer required.
## As a result, the corresponding MLE fit deteriorates, whereas
## the other fits improve slightly.

#####
## get the data                               ##
#####
library(fields)
miles2km <- 1.608

data(weather)
```

```

## P and T are so different in scale so that they have
## to be normalized first:
sdP <- sd(weather[, 1])
sdT <- sd(weather[, 2])
PT <- cbind( weather[, 1] / sdP, weather[, 2] / sdT )

Dist.mat <- rdist.earth(weather[,3:4])
Dist.mat <- Dist.mat[lower.tri(Dist.mat)] ## in miles

#####
## first: marginal estimation ##
#####
uni.est <-
  list("+",
    list("$", var=NA, list("nugget")),
    list("$", var=NA, scale=NA, list("whittle", nu=NA))
  )
fvP <- fitvario(Distances=Dist.mat, truedim=2, data=PT[, 1],
  model=uni.est, grid=FALSE, m1="m1", Print=1)$m1 # -153.9

fvT <- fitvario(Distances=Dist.mat, truedim=2, data=PT[, 2],
  model=uni.est, grid=FALSE, m1="m1", Print=1)$m1 # -138.45

#

#####
## second: parsimonious model ##
#####
puni2bi <- function(mP, mT, lower) {
  nugg1 <- mP[[2]]$var
  nugg2 <- mT[[2]]$var
  nu1 <- mP[[3]][[4]]$nu
  nu2 <- mT[[3]][[4]]$nu
  s <- mean(c(mP[[3]]$scale, mT[[3]]$scale))
  c1 <- mP[[3]]$var
  c2 <- mT[[3]]$var
  if (missing(lower)) {
    rhored <- 0
    f <- 1
  } else if (lower) {
    rhored <- -1
    f <- 0.2
    nugg1 <- nugg2 <- 0
  } else {
    rhored <- 1
    f <- 4
    nugg1 <- c1 / 2
    nugg2 <- c2 / 2
  }
  return(list("+",

```

```

    list("M", M=matrix(nc=2, c(sqrt(nugg1), 0, 0, sqrt(nugg2))),
          list("nugget")),
    list("parsbiWM",
          nu = c(nu1 * f, nu2 * f),
          s = s * f,
          c = c(c1 * f, c2 * f), rhored=rhored
        )
  )
}

p.est.model <-
  list("+",
        list("M", M=matrix(nc=2, c(NA, 0, 0, NA)),
              list("nugget")),
        list("parsbiWM",
              nu = c(NA, NA),
              s = NA,
              c = c(NA, NA), rhored=NA
            )
  ))

## takes a rather long time (15 min in 2011)
pars <- fitvario(Distances=Dist.mat, truedim = 2, data=PT,
                model=p.est.model, grid=FALSE, trend=list(0,0),
                lower= puni2bi(fvP$model, fvT$model, lower=TRUE),
                upper= puni2bi(fvP$model, fvT$model, lower=FALSE),
                users.guess=puni2bi(fvP$model, fvT$model),
                Print=2, ml="ml")$ml ## -280.9791

#
#

#####
## third: full biwm model ##
#####
pars2full <- function(model, lower){
  s <- model[[3]]$s
  nuP <- model[[3]]$nu[1]
  nuT <- model[[3]]$nu[2]
  tauP <- model[[2]]$M[1]
  tauT <- model[[2]]$M[4]
  cP <- model[[3]]$c[1]
  cT <- model[[3]]$c[2]
  Rhored <- model[[3]]$rhored
  nured <- 1

  if (missing(lower)) {
    f <- 1
  } else if (lower) {
    nured <- 0.1
    f <- 0.5
    Rhored <- -1
    tauP <- tauT <- 0
  }
}

```

```

} else {
  f <- 2
  tauP <- max(f * tauP, cP / 10)
  tauT <- max(f * tauT, cT / 10)
  Rhored <- 0 ## as estimated Rhored is negativ
}

list("+",
  list("M", M=matrix(nc=2, c(tauP, 0, 0, tauT)),
    list("nugget")),
  list("biWM",
    nu = c(nuP * f, nuT * f), nured=nured,
    s = c(s * f, s * f), s12=s * f,
    c = c(cP * f, cT * f), rhored=Rhored
  )
)

est.model <-
list("+",
  list("M", M=matrix(nc=2, c(NA, 0, 0, NA)),
    list("nugget")),
  list("biWM",
    nu = c(NA, NA), nured=NA,
    s = c(NA, NA), s12=NA,
    c = c(NA, NA), rhored=NA
  ))

fv <- fitvario(Distances=Dist.mat, truedim = 2, data=PT, Print=2,
  model=est.model, grid=FALSE, trend=list(0,0),
  lower=pars2full(pars$model, lower=TRUE),
  upper=pars2full(pars$model, lower=FALSE),
  users.guess=pars2full(pars$model), ml="ml")$ml # -280.6910

##
#

#####
## output ##
#####

Factor <- function(nu1, nu2, nu12, a1, a2, a12, d) {
  gamma(nu1 + d/2) * gamma(nu2 + d/2) / gamma(nu1) / gamma(nu2) *
  (gamma(nu12) / gamma(nu12+d/2))^2 * (a1^nu1 * a2^nu2 / a12^(2*nu12) )^2
}

InfQ <- function(nu1, nu2, nu12, a1, a2, a12, d) {
  gamma <- (2*nu12+d) * a1^2 * a2^2 - (nu2 +d/2)*a1^2*a12^2 -
  (nu1 +d/2) *a2^2*a12^2
  beta <- (2 * nu12 - nu2 + d/2) * a1^2 + (2 * nu12 - nu1 + d/2) * a2^2 -
  (nu1 + nu2 + d) *a12^2
  alpha <- 2 * nu12 - nu1 -nu2
}

```

```

rednu12 <- 0.5 * (nu1 + nu2) / nu12

if (rednu12 == 1.0) {
  t1sq <- t2sq <- max(0, if (beta==0) 0 else -gamma / beta)
  inf <- 1
} else {
  inf <- Inf
  discr <- beta^2 - 4 * alpha * gamma
  t1sq <- t2sq <- 0
  if (discr >= 0) {
    discr <- sqrt(discr)
    t1sq = max(0, (-beta + discr) / (2.0 * alpha))
    t2sq = max(0, (-beta - discr) / (2.0 * alpha))
  }
}
tsq <- c(0, t1sq, t2sq)
return(min(inf, (a1^2 + tsq)^(2.0 * nu12 + d) /
           (a1^2 + tsq)^(nu1 + d/2) / (a2^2 + tsq)^(nu2 + d/2) ))
}

```

```

uni2full <- function(mP, mT) {
  nuggP <- mP[[2]]$var
  nuggT <- mT[[2]]$var
  nuP <- mP[[3]][[4]]$nu
  nuT <- mT[[3]][[4]]$nu
  sP <- mP[[3]]$scale
  sT <- mT[[3]]$scale
  c1 <- mP[[3]]$var
  c2 <- mT[[3]]$var
  return(list("+",
             list("M", M=matrix(nc=2, c(sqrt(nuggP), 0, 0, nuggT)),
                  list("nugget")),
             list("biWM",
                  nu = c(nuP, nuT), nured=1,
                  s = c(sP, sT), s12=1,
                  c = c(c1, c2), rhored=0)
             ) )
}

```

```

PaperOutput <- function(m, sdP, sdT) {
  m[[2]]$M <- m[[2]]$M * c(sdP, 0, 0, sdT)
  m[[3]]$c <- m[[3]]$c * c(sdP, sdT)^2
  sP <- m[[3]]$s[1] * miles2km
  sT <- m[[3]]$s[2] * miles2km
  sPT <- m[[3]]$s12 * miles2km
  m[[3]]$s <- c(sP, sT)
  m[[3]]$s12 <- sPT
  d <- 2
}

```

```

ml <- fitvario(Distances=Dist.mat * miles2km,
              truedim = d, data=t(t(PT) * c(sdP, sdT)),
              m=m, grid=FALSE, trend=list(0,0), ml="ml")$ml$ml

nuP <- m[[3]]$nu[1]
nuT <- m[[3]]$nu[2]
nuPT <- 0.5 * (nuP + nuT) / m[[3]]$nured

f <- Factor(nuP, nuT, nuPT, 1/sP, 1/sT, 1/sPT, d)
infQ <- InfQ(nuP, nuT, nuPT, 1/sP, 1/sT, 1/sPT, d)
return(list(
  model = m,
  sigmaP = sqrt(m[[3]]$c[1]),
  sigmaT = sqrt(m[[3]]$c[2]),
  nuP = nuP,
  nuT = nuT,
  nuPT = nuPT,
  inv.aP = sP,
  inv.aT = sT,
  inv.aPT= sPT,
  rhoPT = m[[3]]$rhored * sqrt(f * infQ),
  tauP = m[[2]]$M[1],
  tauT = m[[2]]$M[4],
  ml = ml
))
}

Print(PaperOutput(uni2full(fvP$model, fvT$model), sdP, sdT)) # -1277.11
Print(PaperOutput(pars2full(pars$model), sdP, sdT)) # -1265.73
Print(PaperOutput(fv$model, sdP, sdT)) # -1265.30

## End(Not run)

```

Index

- *Topic **datasets**
 - soil, [101](#)
 - weather, [119](#)
- *Topic **device**
 - Dev, [17](#)
- *Topic **file**
 - FileExists, [24](#)
- *Topic **iplot**
 - Locator, [60](#)
- *Topic **misc**
 - sleep.milli, [100](#)
- *Topic **print**
 - Print, [73](#)
- *Topic **spatial**
 - Changings, [2](#)
 - CondSimu, [3](#)
 - CovarianceFct, [6](#)
 - DeleteRegister, [16](#)
 - EmpiricalVariogram, [18](#)
 - eval.parameters, [21](#)
 - fitvario, [25](#)
 - Forest, [36](#)
 - fractal.dim, [37](#)
 - GaussRF, [40](#)
 - GetPracticalRange, [50](#)
 - GetRegisterInfo, [51](#)
 - hurst, [54](#)
 - Kriging, [56](#)
 - MaxStableRF, [61](#)
 - Papers, [63](#)
 - parameter.range, [70](#)
 - parampositions, [71](#)
 - PrintModellist, [74](#)
 - RandomFields, [75](#)
 - regression, [78](#)
 - RFMethods, [80](#)
 - RFparameters, [84](#)
 - ShowModels, [94](#)
 - SimulateRF, [99](#)
 - Sophisticated Models, [104](#)
- *Topic **sysdata**
 - host, [53](#)
- *Topic **utilities**
 - Dev, [17](#)
 - FileExists, [24](#)
 - getactions, [49](#)
 - host, [53](#)
 - Locator, [60](#)
 - Readline, [77](#)
 - sleep.milli, [100](#)
 - useraction, [116](#)
- Changings, [2](#)
- changings (Changings), [2](#)
- CondSimu, [3](#), [58](#), [76](#)
- Covariance, [6](#), [7](#), [10](#), [11](#), [26](#), [27](#), [34](#), [43](#), [50](#), [58](#), [74](#)
- Covariance (CovarianceFct), [6](#)
- CovarianceFct, [3](#), [5](#), [6](#), [26](#), [27](#), [29](#), [31](#), [34](#), [40](#), [41](#), [43](#), [50](#), [52](#), [56–58](#), [61](#), [63](#), [71](#), [72](#), [75](#), [76](#), [80](#), [81](#), [83](#), [84](#), [95](#), [96](#), [98](#), [99](#), [104](#), [105](#), [111](#), [112](#), [114](#)
- CovMatrix (CovarianceFct), [6](#)
- DeleteAllRegisters (DeleteRegister), [16](#)
- DeleteRegister, [16](#), [43](#), [76](#), [91](#)
- Dev, [17](#)
- DoSimulateRF, [42](#), [43](#), [52](#), [62](#), [63](#), [91](#)
- DoSimulateRF (SimulateRF), [99](#)
- EmpiricalVariogram, [15](#), [18](#), [33](#), [42](#), [43](#), [58](#), [76](#), [82](#), [91](#), [95](#), [114](#)
- eval.parameters, [21](#), [96](#), [98](#), [117](#)
- FileExists, [24](#)
- fitvario, [20](#), [25](#), [43](#), [72](#), [76](#), [105](#), [108](#)
- Forest, [36](#)
- fractal (fractal.dim), [37](#)
- fractal.dim, [37](#), [56](#), [76](#)

- GaussRF, 4, 5, 16, 18, 20, 26, 27, 40, 51, 52, 62, 63, 76, 83, 87, 91, 92, 96, 98, 100
- getactionlist (getactions), 49
- getactions, 49, 61, 78, 117
- GetMethodNames (RFMethods), 80
- GetModel, 7, 15, 75
- GetModel (GetRegisterInfo), 51
- GetModelInfo (GetRegisterInfo), 51
- GetModelList (PrintModelList), 74
- GetModelNames (PrintModelList), 74
- GetPracticalRange, 15, 34, 43, 50, 92, 114
- GetRegisterInfo, 51
- host, 53
- hostname (host), 53
- Hurst (hurst), 54
- hurst, 40, 54, 76
- image, 97
- InitGaussRF, 91, 99
- InitGaussRF (GaussRF), 40
- InitMaxStableRF, 91, 99
- InitMaxStableRF (MaxStableRF), 61
- InitSimulateRF (SimulateRF), 99
- Kriging, 5, 56, 76
- likfit, 33
- lm, 39, 55, 79
- Locator, 49, 60, 76, 78, 116, 117
- locator, 60, 76, 117
- LockRemove (FileExists), 24
- long memory dependence (hurst), 54
- MaxStableRF, 43, 61, 76, 83, 91, 92, 100
- mleRF (fitvario), 25
- optim, 27, 30, 31, 33
- optimize, 30
- Papers, 63
- papers, 76
- papers (Papers), 63
- par, 17
- parameter.range, 15, 70, 114
- parampositions, 29, 34, 71
- pid (host), 53
- plot, 97
- PracticalRange (GetPracticalRange), 50
- Print, 73, 76
- PrintMethodList, 3, 41, 42, 61, 62, 75, 76, 99
- PrintMethodList (RFMethods), 80
- PrintModelList, 3, 26, 40, 42, 57, 61, 62, 74, 83, 95, 99, 105
- putactions (useraction), 116
- RandomFields, 5, 15, 16, 20, 34, 43, 50, 58, 63, 75, 83, 92, 98, 100, 114
- Readline, 49, 61, 76, 77, 116, 117
- readline, 76, 78, 117
- regression, 55, 76, 78
- RFMethods, 3, 8, 9, 41, 43, 61–63, 80, 85, 91, 92, 95, 98, 99
- RFparameters, 13, 15, 16, 31, 34, 41–43, 50, 52, 62, 63, 76, 81, 82, 84, 86, 97, 111, 114
- Schlather (Papers), 63
- schlather (Papers), 63
- ShowModels, 15, 43, 76, 77, 94, 114
- SimulateRF, 99
- sleep (sleep.milli), 100
- sleep.milli, 77, 100
- soil, 76, 101
- Sophisticated, 10
- Sophisticated (Sophisticated Models), 104
- sophisticated, 7, 8, 11, 12, 15, 52, 63, 72, 75, 76, 83, 105
- sophisticated (Sophisticated Models), 104
- Sophisticated Models, 104
- split.screen, 96
- Sweave, 85
- useraction, 23, 49, 60, 61, 78, 116
- userinput, 61, 78
- Variogram, 52
- Variogram (CovarianceFct), 6
- weather, 34, 76, 119