Obsolete Functions of RandomFields Version 2

December 1, 2013

`RandomFields` *Simulation and Analysis of Random Fields VERSION 2*

Description

THIS IS SOME PARTIAL DOCUMENTATION OF THE FORMER VERSION 2. THIS VERSION IS OUT OF DATE AND NOT MAINTAINED ANYMORE.

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

2. stationary (and isotropic) max-stable random fields

   - `CovarianceFct` : covariance models for extremal Gaussian random fields
   - `MaxStableRF` : simulation of max-stable random fields

3. Special Functions

   - `FileExists` : used for simple parallel evaluation
   - `hostname` : hostname of the computer
   - `pid` : PID of the R process
   - `sleep` : sleeping/waiting for a certain period

Acknowledgement

Many thanks to

- R Core Team making available the algoithm for fft (fft.c) by Richard Singleton and advicing
- Ben Pfaff, 12167 Airport Rd, DeWitt MI 48820, USA making available an algorithm for AVL trees (avltr*)
- 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de;

References

Singleton, R.C. (1979). In *Programs for Digital Signal Processing* Ed.: Digital Signal Processing Committee and IEEE Acoustics, Speech, and Signal Processing Committe (1979) IEEE press.

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.
given	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.
data	the values measured.
trend	Not programmed yet. (used by universal kriging)

n	number of realisations to generate. If paired=TRUE then n must be even.
register	0:9; place where intermediate calculations are stored; the numbers are aliases for 10 internal registers; see GaussRF for further details.
err.model	covariance function for the error model. String or list. See model for details.
err.param	parameters for the error model. See also param.
err.method	Only relevant if err.model is not NULL. Then it must be given if and only if method is given; see method for details.
err.register	see register for details.
tol	considered only if grid=TRUE; tolerated distances of a given point to the nearest grid point to be regarded as being zero; see Details.
pch	character. The included kriging procedure can be quite time consuming. The character pch is printed after roughly each 80th part of calculation.
paired	logical. 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").
na.rm	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,y$end,y

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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 <- "O" ## 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)
```

`Sophisticated Models` *Sophicated Covariance And Variogram Models*

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 ($v =$`var`) and the coordinates or distances by

- the scale (s =`scale`) or
- the anisotropy matrix a =`anisoT` multiplied from the right or
- the anisotropy matrix A multiplied from the left or
- p =`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_nu 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 $c_{12} = c_{21} = \rho_{red}\sqrt{fmc_{11}c_{22}}$ 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
`nu` $= (nu_{11}, nu_{22})$
`nured12` $= \nu_{red}$
`s` $= (s_{11}, s_{22})$
`s12` $= s_{12} = s_{21}$\ `c` $= (c_{11}, c_{22})$
`rhored` $= \rho_{red}$ See also `parsbiWM`.

- `constant`
This model is designes 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 $\$mu$ 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`

$$C(h) = \phi(h), 0 \le h \le d$$
$$C(h) = b_0((dr)^a - h^a)^{2a}, d \le h \le dr$$
$$C(h) = 0, dr \le h$$

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 phi and the parameters.

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

- `delayeffect` (bivariate)

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

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)

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

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 phi 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 preceeding 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 intepreted 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 preceeding model `$`).
    - if `covb` is any other model then we have a geoadditive part

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

- `mqam` (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 $\phi(\|.\|^2)$ 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 covriance 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` (multivariat model)

$$C(h) = \mathrm{diag}(1, \ldots, 1) 1_{\{0\}}(h)$$

The components of the multivariate vector are always independent. The models 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
`nu` $= (\nu_{11}, \nu_{22})$
`s` $= (s_{11}, s_{22})$
`s12` $= s_{12} = s_{21}$
`c` $= (c_{11}, c_{22})$
`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)

$$C(h) = \phi(\sum_i \theta_i \phi^{-1}(C_i(h)))$$

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 $\phi(\|.\|^2)$ 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.

- `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)

$$S^t(x) = x^t R, \qquad x \in R^3$$

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`

$$C(h) = a_0 + a_2(h)^2 + \phi(h), 0 \le h \le D$$

$$C(h) = b_0(rD - h)^3/(h), r \le h \le rD$$

$$C(h) = 0, rD \le h$$

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,$r \ge 1$. 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-separabel 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

$$Q(x,y) = c^2 - m^2 + h^t(S_x + 2(m+c)M)A^{-1}(A_y + 2(m-c)M)h,$$
$$c = -z^t h + \xi_2(x) - \xi_2(y),$$
$$A = S_x + S_y + 4Mhh^tM$$
$$m = h^tMh.$$
$$h = H(x) - H(y)$$

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 sophicated 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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. (2010) 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 R^2: Exploring the Limits. *J. Comput. Graph. Stat.* **15**, 483-501.
- Stein, M.

delayeffect

- Wackernagel, H. (2003) *Multivariate Geostatistics.* Berlin: Springer, 3nd 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 densitites, *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) Caracterization 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)
```

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 $(n \times \texttt{dim})$-matrix. In particular, if the model is isotropic or 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 param=c(mean,variance,nugget,scale,...), 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'
- bessel
$$C(h) = 2^{\nu}\Gamma(\nu+1)h^{-\nu}J_{\nu}(h)$$
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)
$$C(h) = \left(1+h^2\right)^{-\beta}$$
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
$$C(k) = \frac{(-1)^k \Gamma(1-a/2)^2}{\Gamma(1-a/2+k)\Gamma(1-a/2-k),} \qquad k \in \mathbf{N}$$
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)
$$gamma(h) = h^\alpha$$
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
$$C(h) = \left(1 + (\alpha+2)h + \left((\alpha+2)^2 - 1\right) h^2/3\right) (1-h)^{\alpha+2} 1_{[0,1]}(h)$$
If $n = 3$ then
$$C(h) = \left(1 + (\alpha+3)h + \left(2(\alpha+3)^2 - 3\right) h^2/5 + \left((\alpha+3)^2 - 4\right)(\alpha+3)h^3/15\right) (1-h)^{\alpha+3} 1_{[0,1]}(h)$$
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`
$$C(h) = \left(1 + 8sh + 25(sh)^2 + 32(sh)^3\right)(1 - sh)^8 1_{[0,1]}(sh)$$
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 `nonstathyperbolic` in `Covariance`.
- `iacocesare` (non-separabel 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)
$$C(h) = 1 - \frac{\beta}{\alpha + \beta}|h|^\alpha, |h| \leq 1 \qquad \text{and} \qquad \frac{\alpha}{\alpha + \beta}|h|^{-\beta}, |h| > 1$$

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`

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

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`
$$C(x) = \frac{\sin x}{x}, \quad x > 0 \qquad \text{and } C(0) = 1$$
This isotropic covariance function is valid only for dimensions less than or equal to 3. It is a special case of the `bessel` 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
$$\texttt{variance} * \texttt{cov}((\cdot)/\texttt{scale}).$$
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 aboved. 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
$$\gamma(x) = cov(0) - cov(x).$$

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 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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, 3nd 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 differnce 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)
```

`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

x	vector of x-coordinates, or matrix
y	vector of y-coordinates

z	vector of z-coordinates
T	vector of time components; here T is given in grid format, see GaussRF.
data	vector or matrix of data; if data 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.
grid	logical; if TRUE then x, y, and z define a grid; otherwise x, y, and z are interpreted as points
bin	vector of ascending values giving the bin boundaries
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
phi	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.
theta	vector of two components. First component gives the angle for the first line of midpoints of an angular variogram (angle is zero for the xy-plane). The second component gives the number of directions (on the half circle). The spatial dimension must be at least 3.
deltaT	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 \ codex, 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), s
- The bins are left open, right closed intervals, i.e., $(b_i, b_{i+1}]$ for $i = 1, \ldots,$length(bin)-1. Hence, to include zero, bin[1] must be negative.

Value

The function returns a list:

centers	central points of the bins

emp.vario	empirical variogram; vector or matrix or array, depending on the anisotropy definitions. The sequence is distances, phi, theta, Tbins. If phi, theta, or Tbins 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 (length(bin)-1).

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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
```

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

---

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 explicitely and cannot be passed via a grid definition as in GaussRF
y	vector of y coordinates
z	vector of z coordinates
T	vector of T coordinates; these coordinates are given in triple notation, see GaussRF
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 CovarianceFct and Covariance, or type PrintModelList() to get all options. If model is a list, then the parameters with value NA are estimated. Parameters that have value NaN should be explicitely 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: $\frac{x_1^{\lambda}-1}{\lambda}_1 + \lambda_2$ 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 *internally*, 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.
bins	number of bins of the empirical variogram. See Details.
nphi	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 variogramm by LSQ with a anisotropic model a large value for ntheta might be needed (about 20).
ntheta	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 variogramm by LSQ with a anisotropic model a large value for ntheta might be needed (about 20).
ntime	scalar or vector of 2 components. if ntimes is a vector, then the first component are the maximum time distance (in units of the grid length T[3]) and the second

component gives the step size (in units of the grid length `T[3]`). If scalar then the step size is assumed to 1 (in units of the grid length `T[3]`).

`distance.factor`
: relative right bound for the bins. See Details.

`upperbound.scale.factor`
: relative upper bound for scale in LSQ and MLE. See Details.

`lowerbound.scale.factor`
: relative lower bound for scale in MLE. See Details.

`lowerbound.scale.LS.factor`
: relative lower bound for scale in LSQ. See Details.

`upperbound.var.factor`
: relative upper bound for variance and nugget. See Details.

`lowerbound.var.factor`
: relative lower bound for variance. See Details.

`lowerbound.sill`
: absolute lower bound for variance and nugget. See Details.

`scale.max.relative.factor`
: relative lower bound for scale below which an additional nugget effect is detected. See Details.

`minbounddistance`
: absolute distance to the bounds below which a part of the algorithm is considered as having failed. See Details.

`minboundreldist`
: relative distance to the bounds below which a part of the algorithm is considered as having failed. See Details.

`approximate.functioncalls`
: approximate evaluations of the ML target function on a grid. See Details.

`refine.onborder`
: logical. If `refine.onborder=TRUE` 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)

`minmixedvar` 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` 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 `function(param) {param[3] <- 5 - param[1]; param}`. Note that the mean and the trend of the model can be neither set nor used in `transform`. See also `standard.style`.

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 `users.guess`, `lower` and `upper` right. The parameter `users.guess` must be given; `lower` and `upper` should be given.

Default: `NULL`

`standard.style` logical or `NULL`. This variable should only be set by the advanced user. If `NULL` then `standard.style` will be `TRUE` if the covariance model allows for a 'standard' definition (see `CovarianceFct`) and `transform` is `NULL`.

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 `standard.style` might be set to `TRUE` by the user. This accelerates the MLE algorithm. The responsibility is completely left to the user, then.

`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 users.guess 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 `truedim` 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 `solvesigma=TRUE`.This makes sense if the number of independent variables is very small. If `solvesigma=FALSE` 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 explicitely; 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 `NA`s, 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 uniformely 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

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

  Default: 10.

- lowerbound.var.factor
  The lower bound for the variance and the nugget is determined as

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

  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

ev	list returned by EmpiricalVariogram
table	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.
lowerbounds	lower bounds
lowerbounds	upper bounds
transform	transformation function
vario	obsolete
self	list containing • modelthe variogram or covariance model • residualsNULL • ml.valuethe likelihood value for the model
plain, sqrt.nr, sd.inv, internal, ml, reml	see self; excepetion is ml, where the residuals are given instead of NULL.

Acknowledgement

Thanks to Paulo Ribeiro for hints and comparing the perliminary 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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$ml)

## End(Not run)
```

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 model=list(list(model=,var=,k=,aniso=),...). T must always be given in the gridtriple 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. grid 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 param=c(NA,variance,nugget,scale,...), 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 $a_0 + a_1x_1 + \ldots + a_dx_d$), 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 PrintMethodList() to get all options. If model is given as list then method may not be set if model[[i]]$method, $i = 1, 3, ..$ is given, and vice versa. However, a global parameter method and specific methods may be given, e.g. list(list(model=..., method="TBM3"), ... 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 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
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$step), seq(y$
- 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,... 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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("*",
                   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)
rl <- function() if (interactive()) readline("Press return")
for (i in 1:dim(z)[3]) { image(z[,,i], zlim=range(z)); rl();}
for (i in 1:dim(z)[2]) { image(z[,i,], zlim=range(z)); rl();}
for (i in 1:dim(z)[1]) { image(z[i,,], zlim=range(z)); rl();}



 ############################################################
 ##                                                        ##
 ##      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 Steins 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))
rl <- function() if (interactive()) readline("Press return")
for (i in 1:dim(z)[1]) { image(z[i,,]); rl();}



 ###############################################################
 ## 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")
## simuation 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
model1 <- list("$", scale=scale, list("stable", alpha=1.0))
rs <- get(".Random.seed", envir=.GlobalEnv, inherits = FALSE)
z1 <- GaussRF(x, x,  grid=TRUE, gridtriple=FALSE,
             model=model1, n=1, meth="cutoff", Storing=TRUE)
(size <- GetRegisterInfo(meth=c("cutoff", "circ"))$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 explicitely
model2 <- 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=model2,
              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"))$S$size
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)
```

---

host *System calls*

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

Examples

```
hostname()
```

`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

`krige.method`	kriging method; currently only 'S' (simple kriging), 'O' (ordinary kriging), 'U' (universal kriging) and 'I' (intrinsic kriging) implemented.
`x`	$(n \times d)$ matrix or vector of `x` coordinates; coordinates of n 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, see `CovarianceFct`, or type `PrintModelList()` to get all options.
`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.
`given`	matrix or vector of points where data are available.
`data`	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 *separately*.

`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.

`pch` Kriging procedures are quite time consuming in general. The character `pch` is printed after roughly each 80th part of calculation.

`return.variance`
logical. If `FALSE` the kriged field is returned. If `TRUE` a list of two elements, `estim` and `var`, i.e. the kriged field and the kriging variances, is returned.

`allowdistanceZero`
if `TRUE` then identical locations are slightly scattered

`cholesky` 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,y$end,y`

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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 <- "O" ## 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)
```

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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)
```

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, param=matrix(...), or param=list(list(variance, anisotropy, kappa),..., list(varianc 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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

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, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de

Examples

```
sleep(1000) # 1 sec
```

Index