Package 'DiceKriging'

April 24, 2015

Title Kriging Methods for Computer Experiments

Version 1.5.5

Date 2015-04-23

Author Olivier Roustant, David Ginsbourger, Yves Deville. Contributors: Clement Chevalier, Yann Richet.

Maintainer Olivier Roustant <roustant@emse.fr>

Description Estimation, validation and prediction of kriging models.
Important functions : km, print.km, plot.km, predict.km.

Depends methods

Suggests rgenoud, foreach, doParallel, testthat

License GPL-2 | GPL-3

URL http://dice.emse.fr/

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-04-24 11:22:56

R topics documented:

`DiceKriging-package` *Kriging Methods for Computer Experiments*

Description

Estimation, validation and prediction of kriging models.

Details

Package:	DiceKriging
Type:	Package
Version:	1.5.5
Date:	2015-04-23
License:	GPL-2 | GPL-3

Note

A previous version of this package was conducted within the frame of the DICE (Deep Inside Computer Experiments) Consortium between ARMINES, Renault, EDF, IRSN, ONERA and TOTAL S.A. (http://dice.emse.fr/).

The authors wish to thank Laurent Carraro, Delphine Dupuy and Celine Helbert for fruitful discussions about the structure of the code, and Francois Bachoc for his participation in validation and estimation by leave-one-out. They also thank Gregory Six and Gilles Pujol for their advices on practical implementation issues, as well as the DICE members for useful feedbacks.

Package `rgenoud` >=5.3.3. is recommended.

Important functions or methods:

`km`	Estimation (or definition) of a kriging model with unknown (known) parameters
`predict`	Prediction of the objective function at new points using a kriging model (Simple and Universal Kriging)
`plot`	Plot diagnostic for a kriging model (leave-one-out)
`simulate`	Simulation of kriging models

Author(s)

Olivier Roustant, David Ginsbourger, Yves Deville. Contributors: C. Chevalier, Y. Richet.

(maintainer: Olivier Roustant <roustant@emse.fr>)

References

F. Bachoc (2013), Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics and Data Analysis*, **66**, 55-69. http://www.lpma.math.upmc.fr/pageperso/bachoc/publications.html

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

O. Dubrule (1983), Cross validation of Kriging in a unique neighborhood. *Mathematical Geology*, **15**, 687-699.

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009.

D. Ginsbourger, D. Dupuy, A. Badea, O. Roustant, and L. Carraro (2009), A note on the choice and the estimation of kriging models for the analysis of deterministic computer experiments, *Applied Stochastic Models for Business and Industry*, **25** no. 2, 115-131.

A.G. Journel and C.J. Huijbregts (1978), *Mining Geostatistics*, Academic Press, London.

A.G. Journel and M.E. Rossi (1989), When do we need a trend model in kriging ?, *Mathematical Geology*, **21** no. 7, 715-739.

D.G. Krige (1951), A statistical approach to some basic mine valuation problems on the witwatersrand, *J. of the Chem., Metal. and Mining Soc. of South Africa*, **52** no. 6, 119-139.

R. Li and A. Sudjianto (2005), Analysis of Computer Experiments Using Penalized Likelihood in Gaussian Kriging Models, *Technometrics*, **47** no. 2, 111-120.

K.V. Mardia and R.J. Marshall (1984), Maximum likelihood estimation of models for residual covariance in spatial regression, *Biometrika*, **71**, 135-146.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

G. Matheron (1963), Principles of geostatistics, *Economic Geology*, **58**, 1246-1266.

G. Matheron (1969), Le krigeage universel, *Les Cahiers du Centre de Morphologie Mathematique de Fontainebleau*, **1**.

W.R. Mebane, Jr., J.S. Sekhon (2011). Genetic Optimization Using Derivatives: The rgenoud Package for R. *Journal of Statistical Software*, **42**(11), 1-26. http://www.jstatsoft.org/v42/i11/

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

B.D. Ripley (1987), *Stochastic Simulation*, Wiley.

O. Roustant, D. Ginsbourger and Yves Deville (2012), DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization, *Journal of Statistical Software*, **51**(1), 1-55, http://www.jstatsoft.org/v51/i01/.

J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn (1989), Design and analysis of computer experiments, *Statistical Science*, **4**, 409-435.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

M.L. Stein (1999), *Interpolation of spatial data, some theory for kriging*, Springer.

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

`affineScalingFun` *Scaling function (affine case)*

Description

Parametric transformation of the input space variables. The transformation is obtained coordinate-wise by integrating piecewise affine marginal "densities" parametrized by a vector of knots and a matrix of density values at the knots. See references for more detail.

Usage

```
affineScalingFun(X, knots, eta)
```

Arguments

X	an n*d matrix standing for a design of n experiments in d-dimensional space
knots	a (K+1) vector of knots parametrizing the transformation. The knots are here the same in all dimensions.
eta	a d*(K+1) matrix of coefficients parametrizing the d marginal transformations. Each line stands for a set of (K+1) marginal density values at the knots defined above.

Value

The image of X by a scaling transformation of parameters knots and eta

References

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

See Also

scalingFun

---

affineScalingGrad *Gradient of the Scaling function (affine case)*

Description

Gradient of the Scaling function (marginal in dimension k) of Xiong et al. with respect to eta

Usage

```
affineScalingGrad(X, knots, k)
```

Arguments

X	an n*d matrix standing for a design of n experiments in d-dimensional space
knots	a (K+1) vector of knots parametrizing the transformation. The knots are here the same in all dimensions.
k	dimension of the input variables for which the gradient is calculated

Value

Gradient at X of the scaling transformation of Xiong et al. with respect to eta. In the present version of the function, the gradient calculation is restricted to the case of 2 parameters by dimension (corresponding to the CovAffineScaling class).

References

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

See Also

scalingGrad

---

checkNames *Consistency test between the column names of two matrices*

Description

Tests if the names of a second matrix are equal to a given matrix up to a permutation, and permute its columns accordingly. When the second one has no column names, the names of the first one are used in the same order.

Usage

```
checkNames(X1, X2, X1.name = "X1", X2.name = "X2")
```

Arguments

X1	a matrix containing column names.
X2	a matrix containing the same number of columns.
X1.name	,
X2.name	optional names for the matrix X1 and X2 theirselves (useful for error messages).

Details

If X2 does not contain variable names, then the names of X1 are used in the same order, and X2 is returned with these names. Otherwise, if the column names of X1 and X2 are equal up to a permutation, the column of X2 are permuted according to the order of X1' names.

Value

The matrix X2, with columns possibly permuted. See details.

Author(s)

O. Roustant

See Also

predict,km-method, simulate,km-method

Examples

```
X1 <- matrix(1, 2, 3)
X2 <- matrix(1:6, 2, 3)

colnames(X1) <- c("x1", "x2", "x3")
checkNames(X1, X2)
# attributes the same names for X2, and returns X2

colnames(X2) <- c("x1", "x2", "x5")
## Not run: checkNames(X1, X2)
# returns an error since the names of X1 and X2 are different

colnames(X2) <- c("x2", "x1", "x3")
checkNames(X1, X2)
# returns the matrix X2, but with permuted columns
```

`coef` *Get coefficients values*

Description

Get or set coefficients values.

Usage

```
coef(object, ...)
```

Arguments

`object`	an object specifying a covariance structure or a km object.
`...`	other arguments (undocumented at this stage).

Note

The replacement method `coef<-` is not available.

Author(s)

Y. Deville, O. Roustant

`computeAuxVariables` *Auxiliary variables for kriging*

Description

Computes or updates some auxiliary variables used for kriging (see below). This is useful in several situations : when all parameters are known (as for one basic step in Bayesian analysis), or when some new data is added but one does not want to re-estimate the model coefficients. On the other hand, `computeAuxVariables` is not used during the estimation of covariance parameters, since this function requires to compute the trend coefficients at each optimization step; the alternative given by (Park, Baek, 2001) is preferred.

Usage

```
computeAuxVariables(model)
```

Arguments

`model`	an object of class `km` with missing (or non updated) items.

Value

An updated `km` objet, where the changes concern the following items:

`T`	a matrix equal to the upper triangular factor of the Choleski decomposition of `C`, such that `t(T)*T = C` (where C is the covariance matrix).
`z`	a vector equal to `inv(t(T))*(y - F*beta)`, with `y`, `F`, `beta` are respectively the response, the experimental matrix and the trend coefficients specified in `model@trend.coef`. If `model@trend.coef` is empty, `z` is not computed.
`M`	a matrix equal to `inv(t(T))*F`.

Note

`T` is computed with the base function `chol`. `z` and `M` are computed by solving triangular linear systems with `backsolve`. `z` is not computed if `model@trend.coef` is empty.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne

References

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

See Also

`covMatrix`, `chol`, `backsolve`.

covAffineScaling-class
Class "covAffineScaling"

Description

Composition of isotropic kernels with coordinatewise non-linear scaling obtained by integrating affine functions

Objects from the Class

In 1-dimension, the covariance kernels are parameterized as in (Rasmussen, Williams, 2006). Denote by `theta` the range parameter, `p` the exponent parameter (for power-exponential covariance), `s` the standard deviation, and `h=|x-y|`. Then we have `C(x,y) = s^2 * k(x,y)`, with:

Gauss	`k(x,y) = exp(-1/2*(h/theta)^2)`
Exponential	`k(x,y) = exp(-h/theta)`
Matern(3/2)	`k(x,y) = (1+sqrt(3)*h/theta)*exp(-sqrt(3)*h/theta)`
Matern(5/2)	`k(x,y) = (1+sqrt(5)*h/theta+(1/3)*5*(h/theta)^2) *exp(-sqrt(5)*h/theta)`
Power-exponential	`k(x,y) = exp(-(h/theta)^p)`

Here, in every dimension, the corresponding one-dimensional stationary kernel `k(x,y)` is replaced by `k(f(x),f(y))`, where `f` is a continuous monotonic function indexed by 2 parameters (see the references for more detail).

Slots

`d`: Object of class `"integer"`. The spatial dimension.

`knots`: Object of class `"numeric"`. A vector specifying the position of the two knots, common to all dimensions.

`eta`: Object of class `"matrix"`. A d*2 matrix of scaling coefficients, parametrizing the coordinatewise transformations in the d dimensions.

`name`: Object of class `"character"`. The covariance function name. To be chosen between `"gauss"`, `"matern5_2"`, `"mate` and `"powexp"`

`paramset.n`: Object of class `"integer"`. 1 for covariance depending only on the ranges parameters, 2 for "powexp" which also depends on exponent parameters.

`var.names`: Object of class `"character"`. The variable names.

`sd2`: Object of class `"numeric"`. The variance of the stationary part of the process.

`known.covparam`: Object of class `"character"`. Internal use. One of: "None", "All".

`nugget.flag`: Object of class `"logical"`. Is there a nugget effect?

`nugget.estim`: Object of class `"logical"`. Is the nugget effect estimated or known?

nugget: Object of class "numeric". If there is a nugget effect, its value (homogeneous to a variance).

param.n: Object of class "integer". The total number of parameters.

Extends

Class "covKernel", directly.

Methods

coef signature(object = "covAffineScaling"): ...

covMat1Mat2 signature(object = "covAffineScaling"): ...

covMatrix signature(object = "covAffineScaling"): ...

covMatrixDerivative signature(object = "covAffineScaling"): ...

covParametersBounds signature(object = "covAffineScaling"): ...

covparam2vect signature(object = "covAffineScaling"): ...

vect2covparam signature(object = "covAffineScaling"): ...

inputnames signature(x = "covAffineScaling"): ...

kernelname signature(x = "covAffineScaling"): ...

ninput signature(x = "covAffineScaling"): ...

nuggetflag signature(x = "covAffineScaling"): ...

nuggetvalue signature(x = "covAffineScaling"): ...

show signature(object = "covAffineScaling"): ...

summary signature(object = "covAffineScaling"): ...

Author(s)

O. Roustant, D. Ginsbourger

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

M.L. Stein (1999), *Interpolation of spatial data, some theory for kriging*, Springer.

See Also

km covTensorProduct

Examples

```
showClass("covAffineScaling")
```

covIso-class *Class of tensor-product spatial covariances with isotropic range*

Description

S4 class of isotropic spatial covariance kerlnes based upon the covTensorProduct class

Objects from the Class

In 1-dimension, the covariance kernels are parameterized as in (Rasmussen, Williams, 2006). Denote by `theta` the range parameter, `p` the exponent parameter (for power-exponential covariance), `s` the standard deviation, and `h=||x-y||`. Then we have `C(x,y) = s^2 * k(x,y)`, with:

Gauss	`k(x,y) = exp(-1/2*(h/theta)^2)`
Exponential	`k(x,y) = exp(-h/theta)`
Matern(3/2)	`k(x,y) = (1+sqrt(3)*h/theta)*exp(-sqrt(3)*h/theta)`
Matern(5/2)	`k(x,y) = (1+sqrt(5)*h/theta+(1/3)*5*(h/theta)^2) *exp(-sqrt(5)*h/theta)`
Power-exponential	`k(x,y) = exp(-(h/theta)^p)`

Slots

`d`: Object of class `"integer"`. The spatial dimension.

`name`: Object of class `"character"`. The covariance function name. To be chosen between `"gauss"`, `"matern5_2"`, `"mate` and `"powexp"`

`paramset.n`: Object of class `"integer"`. 1 for covariance depending only on the ranges parameters, 2 for "powexp" which also depends on exponent parameters.

`var.names`: Object of class `"character"`. The variable names.

`sd2`: Object of class `"numeric"`. The variance of the stationary part of the process.

`known.covparam`: Object of class `"character"`. Internal use. One of: "None", "All".

`nugget.flag`: Object of class `"logical"`. Is there a nugget effect?

`nugget.estim`: Object of class `"logical"`. Is the nugget effect estimated or known?

`nugget`: Object of class `"numeric"`. If there is a nugget effect, its value (homogeneous to a variance).

`param.n`: Object of class `"integer"`. The total number of parameters.

`range.names`: Object of class `"character"`. Names of range parameters, for printing purpose. Default is "theta".

`range.val`: Object of class `"numeric"`. Values of range parameters.

Extends

Class `"covKernel"`, directly.

Methods

coef `signature(object = "covIso")`: ...

covMat1Mat2 `signature(object = "covIso")`: ...

covMatrix `signature(object = "covIso")`: ...

covMatrixDerivative `signature(object = "covIso")`: ...

covParametersBounds `signature(object = "covIso")`: ...

covparam2vect `signature(object = "covIso")`: ...

vect2covparam `signature(object = "covIso")`: ...

covVector.dx `signature(object = "covIso")`: ...

inputnames `signature(x = "covIso")`: ...

kernelname `signature(x = "covIso")`: ...

ninput `signature(x = "covIso")`: ...

nuggetflag `signature(x = "covIso")`: ...

nuggetvalue `signature(x = "covIso")`: ...

show `signature(object = "covIso")`: ...

summary `signature(object = "covIso")`: ...

Author(s)

O. Roustant, D. Ginsbourger

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

M.L. Stein (1999), *Interpolation of spatial data, some theory for kriging*, Springer.

See Also

`km` `covTensorProduct`

Examples

```
showClass("covIso")
```

covKernel-class *Class "covKernel"*

Description

Union of classes including "covTensorProduct", "covIso", "covAffineScaling", "covScaling" and "covUser"

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "covKernel" in the signature.

Author(s)

Olivier Roustant, David Ginsbourger, Yves Deville

Examples

```
showClass("covKernel")
```

covMat1Mat2 *Cross covariance matrix*

Description

Computes the cross covariance matrix between two sets of locations for a spatial random process with a given covariance structure. Typically the two sets are a learning set and a test set.

Usage

```
covMat1Mat2(object, X1, X2, nugget.flag=FALSE)
```

Arguments

object	an object specifying the covariance structure.
X1	a matrix whose rows represent the locations of a first set (for instance a set of learning points).
X2	a matrix whose rows represent the locations of a second set (for instance a set of test points).
nugget.flag	an optional boolean. If TRUE, the covariance between 2 equal locations takes into account the nugget effect (if any). Locations are considered equal if their euclidian distance is inferior to 1e-15. Default is FALSE.

Value

a matrix of size (nb of rows of X1 * nb of rows of X2) whose element (i1,i2) is equal to the covariance between the locations specified by row i1 of X1 and row i2 of X2.

Author(s)

Olivier Roustant, David Ginsbourger, Ecole des Mines de St-Etienne.

See Also

covMatrix

---

covMatrix *Covariance matrix*

Description

Computes the covariance matrix at a set of locations for a spatial random process with a given covariance structure.

Usage

```
covMatrix(object, X, noise.var = NULL)
```

Arguments

object	an object specifying the covariance structure.
X	a matrix whose columns represent locations.
noise.var	for noisy observations : an optional vector containing the noise variance at each observation

Value

a list with the following items :

C	a matrix representing the covariance matrix for the locations specified in the X argument, including a possible nugget effect or observation noise.
vn	a vector of length n (X size) containing a replication of the nugget effet or the observation noise (so that C-diag(vn) contains the covariance matrix when there is no nugget effect nor observation noise)

Author(s)

Olivier Roustant, David Ginsbourger, Ecole des Mines de St-Etienne.

See Also

covMatrixDerivative

covParametersBounds *Boundaries for covariance parameters*

Description

Default boundaries for covariance parameters.

Usage

```
covParametersBounds(object, X)
```

Arguments

object	an object specifying the covariance structure.
X	a matrix representing the design of experiments.

Details

The default values are chosen as follows :

Range parameters (all covariances)	lower=1e-10, upper=2 times the difference between the max. and min. values of X for each coordinate
Shape parameters (powexp covariance)	lower=1e-10, upper=2 for each coordinate

Value

a list with items lower, upper containing default boundaries for the covariance parameters.

Author(s)

Olivier Roustant, David Ginsbourger, Ecole des Mines de St-Etienne.

See Also

km

covScaling-class *Class "covScaling"*

Description

Composition of isotropic kernels with coordinatewise non-linear scaling obtained by integrating piecewise affine functions

Objects from the Class

In 1-dimension, the covariance kernels are parameterized as in (Rasmussen, Williams, 2006). Denote by `theta` the range parameter, `p` the exponent parameter (for power-exponential covariance), `s` the standard deviation, and `h=|x-y|`. Then we have `C(x,y) = s^2 * k(x,y)`, with:

Gauss	`k(x,y) = exp(-1/2*(h/theta)^2)`
Exponential	`k(x,y) = exp(-h/theta)`
Matern(3/2)	`k(x,y) = (1+sqrt(3)*h/theta)*exp(-sqrt(3)*h/theta)`
Matern(5/2)	`k(x,y) = (1+sqrt(5)*h/theta+(1/3)*5*(h/theta)^2) *exp(-sqrt(5)*h/theta)`
Power-exponential	`k(x,y) = exp(-(h/theta)^p)`

Here, in every dimension, the corresponding one-dimensional stationary kernel `k(x,y)` is replaced by `k(f(x),f(y))`, where `f` is a continuous monotonic function indexed by a finite number of parameters (see the references for more detail).

Slots

`d`: Object of class `"integer"`. The spatial dimension.

`knots`: Object of class `"list"`. The j-th element is a vector containing the knots for dimension j.

`eta`: Object of class `"list"`. In correspondance with knots, the j-th element is a vector containing the scaling coefficients (i.e. the derivatives of the scaling function at the knots) for dimension j.

`name`: Object of class `"character"`. The covariance function name. To be chosen between `"gauss"`, `"matern5_2"`, `"mate` and `"powexp"`

`paramset.n`: Object of class `"integer"`. 1 for covariance depending only on the ranges parameters, 2 for "powexp" which also depends on exponent parameters.

`var.names`: Object of class `"character"`. The variable names.

`sd2`: Object of class `"numeric"`. The variance of the stationary part of the process.

`known.covparam`: Object of class `"character"`. Internal use. One of: "None", "All".

`nugget.flag`: Object of class `"logical"`. Is there a nugget effect?

`nugget.estim`: Object of class `"logical"`. Is the nugget effect estimated or known?

`nugget`: Object of class `"numeric"`. If there is a nugget effect, its value (homogeneous to a variance).

`param.n`: Object of class `"integer"`. The total number of parameters.

Extends

Class `"covKernel"`, directly.

Methods

coef `signature(object = "covScaling")`: ...

covMat1Mat2 `signature(object = "covScaling")`: ...

covMatrix `signature(object = "covScaling")`: ...

covMatrixDerivative `signature(object = "covScaling")`: ...

covParametersBounds `signature(object = "covScaling")`: ...

covparam2vect `signature(object = "covScaling")`: ...

vect2covparam `signature(object = "covScaling")`: ...

show `signature(object = "covScaling")`: ...

inputnames `signature(x = "covAffineScaling")`: ...

kernelname `signature(x = "covAffineScaling")`: ...

ninput `signature(x = "covAffineScaling")`: ...

nuggetflag `signature(x = "covAffineScaling")`: ...

nuggetvalue `signature(x = "covAffineScaling")`: ...

nuggetvalue<- `signature(x = "covAffineScaling")`: ...

summary `signature(object = "covAffineScaling")`: ...

Author(s)

Olivier Roustant, David Ginsbourger, Yves Deville

References

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

See Also

`km` `covTensorProduct` `covAffineScaling` `covIso` `covKernel`

Examples

```
showClass("covScaling")
```

`covTensorProduct-class`

Class of tensor-product spatial covariances

Description

S4 class of tensor-product (or separable) covariances.

Value

`covTensorProduct`

separable covariances depending on 1 set of parameters, such as Gaussian, exponential, Matern with fixed nu... or on 2 sets of parameters, such as power-exponential.

Objects from the Class

A d-dimensional tensor product (or separable) covariance kernel `C(x,y)` is the tensor product of 1-dimensional covariance kernels : `C(x,y) = C(x1,y1)C(x2,y2)...C(xd,yd)`.

In 1-dimension, the covariance kernels are parameterized as in (Rasmussen, Williams, 2006). Denote by `theta` the range parameter, `p` the exponent parameter (for power-exponential covariance), `s` the standard deviation, and `h=|x-y|`. Then we have `C(x,y) = s^2 * k(x,y)`, with:

Gauss	`k(x,y) = exp(-1/2*(h/theta)^2)`
Exponential	`k(x,y) = exp(-h/theta)`
Matern(3/2)	`k(x,y) = (1+sqrt(3)*h/theta)*exp(-sqrt(3)*h/theta)`
Matern(5/2)	`k(x,y) = (1+sqrt(5)*h/theta+(1/3)*5*(h/theta)^2) *exp(-sqrt(5)*h/theta)`
Power-exponential	`k(x,y) = exp(-(h/theta)^p)`

Slots

`d`: Object of class `"integer"`. The spatial dimension.

`name`: Object of class `"character"`. The covariance function name. To be chosen between `"gauss"`, `"matern5_2"`, `"mate` and `"powexp"`

`paramset.n`: Object of class `"integer"`. 1 for covariance depending only on the ranges parameters, 2 for "powexp" which also depends on exponent parameters.

`var.names`: Object of class `"character"`. The variable names.

`sd2`: Object of class `"numeric"`. The variance of the stationary part of the process.

`known.covparam`: Object of class `"character"`. Internal use. One of: "None", "All".

`nugget.flag`: Object of class `"logical"`. Is there a nugget effect?

`nugget.estim`: Object of class `"logical"`. Is the nugget effect estimated or known?

`nugget`: Object of class `"numeric"`. If there is a nugget effect, its value (homogeneous to a variance).

`param.n`: Object of class `"integer"`. The total number of parameters.

`range.n`: Object of class `"integer"`. The number of range parameters.

`range.names`: Object of class `"character"`. Names of range parameters, for printing purpose. Default is "theta".

`range.val`: Object of class `"numeric"`. Values of range parameters.

`shape.n`: Object of class `"integer"`. The number of shape parameters (exponent parameters in "powexp").

`shape.names`: Object of class `"character"`. Names of shape parameters, for printing purpose. Default is "p".

`shape.val`: Object of class `"numeric"`. Values of shape parameters.

Methods

show `signature(x = "covTensorProduct")` Print covariance function. See `show,km-method`.

coef `signature(x = "covTensorProduct")` Get the coefficients of the covariance function.

Author(s)

O. Roustant, D. Ginsbourger

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, `http://www.GaussianProcess.org/gpml`

M.L. Stein (1999), *Interpolation of spatial data, some theory for kriging*, Springer.

See Also

`covStruct.create` to construct a covariance structure.

`covUser-class` *Class "covUser"*

Description

An arbitrary covariance kernel provided by the user

Objects from the Class

Any valid covariance kernel, provided as a 2-dimensional function (x,y) -> k(x,y). At this stage, no test is done to check that k is positive definite.

Slots

`kernel`: Object of class `"function"`. The new covariance kernel.

`nugget.flag`: Object of class `"logical"`. Is there a nugget effect?

`nugget`: Object of class `"numeric"`. If there is a nugget effect, its value (homogeneous to a variance).

Extends

Class `"covKernel"`, directly.

Methods

coef `signature(object = "covUser")`: ...

covMat1Mat2 `signature(object = "covScaling")`: ...

covMatrix `signature(object = "covScaling")`: ...

show `signature(object = "covScaling")`: ...

nuggetflag `signature(x = "covAffineScaling")`: ...

nuggetvalue `signature(x = "covAffineScaling")`: ...

nuggetvalue<- `signature(x = "covAffineScaling")`: ...

Author(s)

Olivier Roustant, David Ginsbourger, Yves Deville

See Also

`km` `covTensorProduct` `covAffineScaling` `covIso` `covKernel`

Examples

```
showClass("covUser")
```

---

inputnames *Get the input variables names*

Description

Get the names of the input variables.

Usage

```
inputnames(x)
```

Arguments

`x`	an object containing the covariance structure.

Value

A vector of character strings containing the names of the input variables.

`kernelname` *Get the kernel name*

Description

Get the name of the underlying tensor-product covariance structure.

Usage

```
kernelname(x)
```

Arguments

`x`	an object containing the covariance structure.

Value

A character string.

`km` *Fit and/or create kriging models*

Description

`km` is used to fit kriging models when parameters are unknown, or to create `km` objects otherwise. In both cases, the result is a `km` object. If parameters are unknown, they are estimated by Maximum Likelihood. As a beta version, Penalized Maximum Likelihood Estimation is also possible if some penalty is given, or Leave-One-Out for noise-free observations.

Usage

```
km(formula=~1, design, response, covtype="matern5_2",
   coef.trend = NULL, coef.cov = NULL, coef.var = NULL,
   nugget = NULL, nugget.estim=FALSE, noise.var=NULL, estim.method="MLE",
   penalty = NULL, optim.method = "BFGS", lower = NULL, upper = NULL,
   parinit = NULL, multistart = 1, control = NULL, gr = TRUE,
   iso=FALSE, scaling=FALSE, knots=NULL, kernel=NULL)
```

Arguments

formula	an optional object of class "formula" specifying the linear trend of the kriging model (see lm). This formula should concern only the input variables, and not the output (response). If there is any, it is automatically dropped. In particular, no response transformation is available yet. The default is ~1, which defines a constant trend.
design	a data frame representing the design of experiments. The ith row contains the values of the d input variables corresponding to the ith evaluation
response	a vector (or 1-column matrix or data frame) containing the values of the 1-dimensional output given by the objective function at the design points.
covtype	an optional character string specifying the covariance structure to be used, to be chosen between "gauss", "matern5_2", "matern3_2", "exp" or "powexp". See a full description of available covariance kernels in covTensorProduct-class. Default is "matern5_2". See also the argument kernel that allows the user to build its own covariance structure.
coef.trend,	(see below)
coef.cov,	(see below)
coef.var	optional vectors containing the values for the trend, covariance and variance parameters. For estimation, 4 cases are implemented: 1. (All unknown) If all are missing, all are estimated. 2. (All known) If all are provided, no estimation is performed; 3. (Known trend) If coef.trend is provided but at least one of coef.cov or coef.var is missing, then BOTH coef.cov and coef.var are estimated; 4. (Unknown trend) If coef.cov and coef.var are provided but coef.trend is missing, then coef.trend is estimated (GLS formula).
nugget	an optional variance value standing for the homogeneous nugget effect.
nugget.estim	an optional boolean indicating whether the nugget effect should be estimated. Note that this option does not concern the case of heterogeneous noisy observations (see noise.var below). If nugget is given, it is used as an initial value. Default is FALSE.
noise.var	for noisy observations : an optional vector containing the noise variance at each observation. This is useful for stochastic simulators. Default is NULL.
estim.method	a character string specifying the method by which unknown parameters are estimated. Default is "MLE" (Maximum Likelihood). At this stage, a beta version of leave-One-Out estimation (estim.method="LOO") is also implemented for noise-free observations.
penalty	(beta version) an optional list suitable for Penalized Maximum Likelihood Estimation. The list must contain the item fun indicating the penalty function, and the item value equal to the value of the penalty parameter. At this stage the only available fun is "SCAD", and covtype must be "gauss". Default is NULL, corresponding to (un-penalized) Maximum Likelihood Estimation.
optim.method	an optional character string indicating which optimization method is chosen for the likelihood maximization. "BFGS" is the optim quasi-Newton procedure of package stats, with the method "L-BFGS-B". "gen" is the genoud genetic algorithm (using derivatives) from package rgenoud (>= 5.3.3).

lower,	(see below)
upper	optional vectors containing the bounds of the correlation parameters for optimization. The default values are given by covParametersBounds.
parinit	an optional vector containing the initial values for the variables to be optimized over. If no vector is given, an initial point is generated as follows. For method "gen", the initial point is generated uniformly inside the hyper-rectangle domain defined by lower and upper. For method "BFGS", some points (see control below) are generated uniformly in the domain. Then the best point with respect to the likelihood (or penalized likelihood, see penalty) criterion is chosen.
multistart	an optional integer indicating the number of initial points from which running the BFGS optimizer. These points will be selected as the best multistart one(s) among those evaluated (see above parinit). The multiple optimizations will be performed in parallel provided that a parallel backend is registered (see package foreach).
control	an optional list of control parameters for optimization. See details below.
gr	an optional boolean indicating whether the analytical gradient should be used. Default is TRUE.
iso	an optional boolean that can be used to force a tensor-product covariance structure (see covTensorProduct-class) to have a range parameter common to all dimensions. Default is FALSE. Not used (at this stage) for the power-exponential type.
scaling	an optional boolean indicating whether a scaling on the covariance structure should be used.
knots	an optional list of knots for scaling. The j-th element is a vector containing the knots for dimension j. If scaling=TRUE and knots are not specified, than knots are fixed to 0 and 1 in each dimension (which corresponds to affine scaling for the domain [0,1]^d).
kernel	an optional function containing a new covariance structure. At this stage, the parameters must be provided as well, and are not estimated. See an example below.

Details

The optimisers are tunable by the user by the argument control. Most of the control parameters proposed by BFGS and genoud can be passed to control except the ones that must be forced [for the purpose of optimization setting], as indicated in the table below. See optim and genoud to get more details about them.

BFGS	trace, parscale, ndeps, maxit, abstol, reltol, REPORT, lnm, factr, pgtol
genoud	all parameters EXCEPT: fn, nvars, max, starting.values, Domains, gr, gradient.check, boundary.er

Notice that the right places to specify the optional starting values and boundaries are in parinit and lower, upper, as explained above. Some additional possibilities and initial values are indicated in the table below:

trace	Turn it to FALSE to avoid printing during optimization progress.
pop.size	For method "BFGS", it is the number of candidate initial points generated before optimization starts (see
max.generations	Default is 5
wait.generations	Default is 2
BFGSburnin	Default is 0

Value

An object of class km (see km-class).

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

D. Ginsbourger (2009), *Multiples metamodeles pour l'approximation et l'optimisation de fonctions numeriques multivariables*, Ph.D. thesis, Ecole Nationale Superieure des Mines de Saint-Etienne, 2009.

D. Ginsbourger, D. Dupuy, A. Badea, O. Roustant, and L. Carraro (2009), A note on the choice and the estimation of kriging models for the analysis of deterministic computer experiments, *Applied Stochastic Models for Business and Industry*, **25** no. 2, 115-131.

A.G. Journel and M.E. Rossi (1989), When do we need a trend model in kriging ?, *Mathematical Geology*, **21** no. 7, 715-739.

D.G. Krige (1951), A statistical approach to some basic mine valuation problems on the witwatersrand, *J. of the Chem., Metal. and Mining Soc. of South Africa*, **52** no. 6, 119-139.

R. Li and A. Sudjianto (2005), Analysis of Computer Experiments Using Penalized Likelihood in Gaussian Kriging Models, *Technometrics*, **47** no. 2, 111-120.

K.V. Mardia and R.J. Marshall (1984), Maximum likelihood estimation of models for residual covariance in spatial regression, *Biometrika*, **71**, 135-146.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

G. Matheron (1969), Le krigeage universel, *Les Cahiers du Centre de Morphologie Mathematique de Fontainebleau*, **1**.

W.R. Jr. Mebane and J.S. Sekhon, in press (2009), Genetic optimization using derivatives: The rgenoud package for R, *Journal of Statistical Software*.

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

See Also

kmData for another interface with the data, show,km-method, predict,km-method, plot,km-method. Some programming details and initialization choices can be found in kmEstimate, kmNoNugget.init, km1Nugget.init and kmNuggets.init

Examples

```
# ----------------------------------
# A 2D example - Branin-Hoo function
# ----------------------------------

# a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(x1=seq(0,1,length=4), x2=seq(0,1,length=4))
y <- apply(design.fact, 1, branin)

# kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
m1 <- km(design=design.fact, response=y)

# kriging model 2 : matern5_2 covariance structure,
#                   linear trend + interactions, no nugget effect
m2 <- km(~.^2, design=design.fact, response=y)

# graphics
n.grid <- 50
x.grid <- y.grid <- seq(0,1,length=n.grid)
design.grid <- expand.grid(x1=x.grid, x2=y.grid)
response.grid <- apply(design.grid, 1, branin)
predicted.values.model1 <- predict(m1, design.grid, "UK")$mean
predicted.values.model2 <- predict(m2, design.grid, "UK")$mean
par(mfrow=c(3,1))
contour(x.grid, y.grid, matrix(response.grid, n.grid, n.grid), 50, main="Branin")
points(design.fact[,1], design.fact[,2], pch=17, cex=1.5, col="blue")
contour(x.grid, y.grid, matrix(predicted.values.model1, n.grid, n.grid), 50,
        main="Ordinary Kriging")
points(design.fact[,1], design.fact[,2], pch=17, cex=1.5, col="blue")
contour(x.grid, y.grid, matrix(predicted.values.model2, n.grid, n.grid), 50,
        main="Universal Kriging")
points(design.fact[,1], design.fact[,2], pch=17, cex=1.5, col="blue")
par(mfrow=c(1,1))


# (same example) how to use the multistart argument
# -------------------------------------------------
require(foreach)

# below an example for a computer with 2 cores, but also work with 1 core

nCores <- 2
require(doParallel)
cl <-  makeCluster(nCores)
```

```
registerDoParallel(cl)

# kriging model 1, with 4 starting points
m1_4 <- km(design=design.fact, response=y, multistart=4)

stopCluster(cl)

# ------------------------------
# A 1D example with penalized MLE
# ------------------------------

# from Fang K.-T., Li R. and Sudjianto A. (2006), "Design and Modeling for
# Computer Experiments", Chapman & Hall, pages 145-152

n <- 6; d <- 1
x <- seq(from=0, to=10, length=n)
y <- sin(x)
t <- seq(0,10, length=100)

# one should add a small nugget effect, to avoid numerical problems
epsilon <- 1e-3
model <- km(formula<- ~1, design=data.frame(x=x), response=data.frame(y=y),
            covtype="gauss", penalty=list(fun="SCAD", value=3), nugget=epsilon)

p <- predict(model, data.frame(x=t), "UK")

plot(t, p$mean, type="l", xlab="x", ylab="y",
                     main="Prediction via Penalized Kriging")
points(x, y, col="red", pch=19)
lines(t, sin(t), lty=2, col="blue")
legend(0, -0.5, legend=c("Sine Curve", "Sample", "Fitted Curve"),
       pch=c(-1,19,-1), lty=c(2,-1,1), col=c("blue","red","black"))


# ------------------------------------------------------------------------
# A 1D example with known trend and known or unknown covariance parameters
# ------------------------------------------------------------------------

x <- c(0, 0.4, 0.6, 0.8, 1);
y <- c(-0.3, 0, -0.8, 0.5, 0.9)

theta <- 0.01; sigma <- 3; trend <- c(-1,2)

model <- km(~x, design=data.frame(x=x), response=data.frame(y=y),
            covtype="matern5_2", coef.trend=trend, coef.cov=theta,
            coef.var=sigma^2)

# below: if you want to specify trend only, and estimate both theta and sigma:
# model <- km(~x, design=data.frame(x=x), response=data.frame(y=y),
#             covtype="matern5_2", coef.trend=trend, lower=0.2)
# Remark: a lower bound or penalty function is useful here,
#         due to the very small number of design points...
```

```
# kriging with gaussian covariance C(x,y)=sigma^2 * exp(-[(x-y)/theta]^2),
#         and linear trend t(x) = -1 + 2x

t <- seq(from=0, to=1, by=0.005)
p <- predict(model, newdata=data.frame(x=t), type="SK")
# beware that type = "SK" for known parameters (default is "UK")

plot(t, p$mean, type="l", ylim=c(-7,7), xlab="x", ylab="y")
lines(t, p$lower95, col="black", lty=2)
lines(t, p$upper95, col="black", lty=2)
points(x, y, col="red", pch=19)
abline(h=0)


# ------------------------------------------------------------
# Kriging with noisy observations (heterogeneous noise variance)
# ------------------------------------------------------------

fundet <- function(x){
return((sin(10*x)/(1+x)+2*cos(5*x)*x^3+0.841)/1.6)
}

level <- 0.5; epsilon <- 0.1
theta <- 1/sqrt(30); p <- 2; n <- 10
x <- seq(0,1, length=n)

# Heteregeneous noise variances: number of Monte Carlo evaluation among
#                                a total budget of 1000 stochastic simulations
MC_numbers <- c(10,50,50,290,25,75,300,10,40,150)
noise.var <- 3/MC_numbers

# Making noisy observations from 'fundet' function (defined above)
y <- fundet(x) + noise.var*rnorm(length(x))

# kriging model definition (no estimation here)
model <- km(y~1, design=data.frame(x=x), response=data.frame(y=y),
            covtype="gauss", coef.trend=0, coef.cov=theta, coef.var=1,
            noise.var=noise.var)

# prediction
t <- seq(0, 1, by=0.01)
p <- predict.km(model, newdata=data.frame(x=t), type="SK")
lower <- p$lower95; upper <- p$upper95

# graphics
par(mfrow=c(1,1))
plot(t, p$mean, type="l", ylim=c(1.1*min(c(lower,y)) , 1.1*max(c(upper,y))),
                xlab="x", ylab="y",col="blue", lwd=1.5)
polygon(c(t,rev(t)), c(lower, rev(upper)), col=gray(0.9), border = gray(0.9))
lines(t, p$mean, type="l", ylim=c(min(lower) ,max(upper)), xlab="x", ylab="y",
                 col="blue", lwd=1)
lines(t, lower, col="blue", lty=4, lwd=1.7)
lines(t, upper, col="blue", lty=4, lwd=1.7)
```

```
lines(t, fundet(t), col="black", lwd=2)
points(x, y, pch=8,col="blue")
text(x, y, labels=MC_numbers, pos=3)


# ------------------------------
# Checking parameter estimation
# ------------------------------

d <- 3        # problem dimension
n <- 40 # size of the experimental design
design <- matrix(runif(n*d), n, d)

covtype <- "matern5_2"
theta <- c(0.3, 0.5, 1) # the parameters to be found by estimation
sigma <- 2
nugget <- NULL  # choose a numeric value if you want to estimate nugget
nugget.estim <- FALSE # choose TRUE if you want to estimate it

n.simu <- 30 # number of simulations
sigma2.estimate <- nugget.estimate <- mu.estimate <- matrix(0, n.simu, 1)
coef.estimate <- matrix(0, n.simu, length(theta))

model <- km(~1, design=data.frame(design), response=rep(0,n), covtype=covtype,
            coef.trend=0, coef.cov=theta, coef.var=sigma^2, nugget=nugget)
y <- simulate(model, nsim=n.simu)

for (i in 1:n.simu) {
# parameter estimation: tune the optimizer by changing optim.method, control
model.estimate <- km(~1, design=data.frame(design), response=data.frame(y=y[i,]),
covtype=covtype, optim.method="BFGS", control=list(pop.size=50, trace=FALSE),
        nugget.estim=nugget.estim)

# store results
coef.estimate[i,] <- covparam2vect(model.estimate@covariance)
sigma2.estimate[i] <- model.estimate@covariance@sd2
mu.estimate[i] <- model.estimate@trend.coef
if (nugget.estim) nugget.estimate[i] <- model.estimate@covariance@nugget
}

# comparison true values / estimation
cat("\nResults with ", n, "design points,
    obtained with ", n.simu, "simulations\n\n",
    "Median of covar. coef. estimates: ", apply(coef.estimate, 2, median), "\n",
    "Median of trend  coef. estimates: ", median(mu.estimate), "\n",
    "Mean of the var. coef. estimates: ", mean(sigma2.estimate))
if (nugget.estim) cat("\nMean of the nugget effect estimates: ",
                      mean(nugget.estimate))

# one figure for this specific example - to be adapted
split.screen(c(2,1))        # split display into two screens
split.screen(c(1,2), screen = 2) # now split the bottom half into 3
```

```
screen(1)
boxplot(coef.estimate[,1], coef.estimate[,2], coef.estimate[,3],
        names=c("theta1", "theta2", "theta3"))
abline(h=theta, col="red")
fig.title <- paste("Empirical law of the parameter estimates
                    (n=", n , ", n.simu=", n.simu, ")", sep="")
title(fig.title)

screen(3)
boxplot(mu.estimate, xlab="mu")
abline(h=0, col="red")

screen(4)
boxplot(sigma2.estimate, xlab="sigma2")
abline(h=sigma^2, col="red")

close.screen(all = TRUE)

# ------------------------------------------------------------
# Kriging with non-linear scaling on Xiong et al.'s function
# ------------------------------------------------------------

f11_xiong <- function(x){
return( sin(30*(x - 0.9)^4)*cos(2*(x - 0.9)) + (x - 0.9)/2)
}

t <- seq(0,1,,300)
f <- f11_xiong(t)

plot(t,f,type="l", ylim=c(-1,0.6), lwd=2)

doe <- data.frame(x=seq(0,1,,20))
resp <- f11_xiong(doe)

knots <- list(  c(0,0.5,1) )
eta <- list(c(15, 2, 0.5))
m <- km(design=doe, response=resp, scaling=TRUE, gr=TRUE,
knots=knots, covtype="matern5_2",  coef.var=1, coef.trend=0)

p <- predict(m, data.frame(x=t), "UK")

plot(t, f, type="l", ylim=c(-1,0.6), lwd=2)

lines(t, p$mean, col="blue", lty=2, lwd=2)
lines(t, p$mean + 2*p$sd, col="blue")
lines(t, p$mean - 2*p$sd,col="blue")

abline(v=knots[[1]], lty=2, col="green")


# ------------------------------------------------------
# Kriging with a symmetric kernel: example with covUser
# ------------------------------------------------------
```

```
x <- c(0, 0.15, 0.3, 0.4, 0.5)
y <- c(0.3, -0.2, 0, 0.5, 0.2)

k <- function(x,y) {
  theta <- 0.15
  0.5*exp(-((x-y)/theta)^2) + 0.5*exp(-((1-x-y)/theta)^2)
}

muser <- km(design=data.frame(x=x), response=data.frame(y=y),
            coef.trend=0, kernel=k)

u <- seq(from=0, to=1, by=0.01)
puser <- predict(muser, newdata=data.frame(x=u), type="SK")

set.seed(0)
nsim <- 5
zuser <- simulate(muser, nsim=nsim, newdata=data.frame(x=u), cond=TRUE, nugget.sim=1e-8)
par(mfrow=c(1,1))
matplot(u, t(zuser), type="l", lty=rep("solid", nsim), col=1:5, lwd=1)
polygon(c(u, rev(u)), c(puser$upper, rev(puser$lower)), col="lightgrey", border=NA)
lines(u, puser$mean, lwd=5, col="blue", lty="dotted")
matlines(u, t(zuser), type="l", lty=rep("solid", nsim), col=1:5, lwd=1)
points(x, y, pch=19, cex=1.5)
```

`km-class` *Kriging models class*

Description

S4 class for kriging models.

Objects from the Class

To create a `km` object, use `km`. See also this function for more details.

Slots

`d`: Object of class `"integer"`. The spatial dimension.

`n`: Object of class `"integer"`. The number of observations.

`X`: Object of class `"matrix"`. The design of experiments.

`y`: Object of class `"matrix"`. The vector of response values at design points.

`p`: Object of class `"integer"`. The number of basis functions of the linear trend.

`F`: Object of class `"matrix"`. The experimental matrix corresponding to the evaluation of the linear trend basis functions at the design of experiments.

`trend.formula`: Object of class `"formula"`. A formula specifying the trend as a linear model (no response needed).

`trend.coef`: Object of class `"numeric"`. Trend coefficients.

`covariance`: Object of class `"covTensorProduct"`. See `covTensorProduct-class`.

`noise.flag`: Object of class `"logical"`. Are the observations noisy?

`noise.var`: Object of class `"numeric"`. If the observations are noisy, the vector of noise variances.

`known.param`: Object of class `"character"`. Internal use. One of: `"None"`, `"All"`, `"CovAndVar"` or `"Trend"`.

`case`: Object of class `"character"`. Indicates the likelihood to use in estimation (Internal use). One of: `"LLconcentration_beta"`, `"LLconcentration_beta_sigma2"`, `"LLconcentration_beta_v_alpha"`.

`param.estim`: Object of class `"logical"`. `TRUE` if at least one parameter is estimated, `FALSE` otherwise.

`method`: Object of class `"character"`. `"MLE"` or `"PMLE"` depending on `penalty`.

`penalty`: Object of class `"list"`. For penalized ML estimation.

`optim.method`: Object of class `"character"`. To be chosen between `"BFGS"` and `"gen"`.

`lower`: Object of class `"numeric"`. Lower bounds for covariance parameters estimation.

`upper`: Object of class `"numeric"`. Upper bounds for covariance parameters estimation.

`control`: Object of class `"list"`. Additional control parameters for covariance parameters estimation.

`gr`: Object of class `"logical"`. Do you want analytical gradient to be used ?

`call`: Object of class `"language"`. User call reminder.

`parinit`: Object of class `"numeric"`. Initial values for covariance parameters estimation.

`logLik`: Object of class `"numeric"`. Value of the concentrated log-Likelihood at its optimum.

`T`: Object of class `"matrix"`. Triangular matrix delivered by the Choleski decomposition of the covariance matrix.

`z`: Object of class `"numeric"`. Auxiliary variable: see `computeAuxVariables`.

`M`: Object of class `"matrix"`. Auxiliary variable: see `computeAuxVariables`.

Methods

coef `signature(x = "km")` Get the coefficients of the `km` object.

plot `signature(x = "km")`: see `plot,km-method`.

predict `signature(object = "km")`: see `predict,km-method`.

show `signature(object = "km")`: see `show,km-method`.

simulate `signature(object = "km")`: see `simulate,km-method`.

Author(s)

O. Roustant, D. Ginsbourger

See Also

`km` for more details about slots and to create a `km` object, `covStruct.create` to construct a covariance structure, and `covTensorProduct-class` for the S4 covariance class defined in this package.

kmData *Fit and/or create kriging models*

Description

kmData is equivalent to km, except for the interface with the data. In kmData, the user must supply both the design and the response within a single data.frame data. To supply them separately, use km.

Usage

```
kmData(formula, data, inputnames = NULL, ...)
```

Arguments

formula	an object of class "formula" specifying the linear trend of the kriging model (see lm). At this stage, transformations of the response are not taken into account.
data	a data.frame containing both the design (input variables) and the response (1-dimensional output given by the objective function at the design points).
inputnames	an optional vector of character containing the names of variables in data to be considered as input variables. By default, all variables but the response are input variables.
...	other arguments for creating or fitting Kriging models, to be taken among the arguments of km function apart from design and response.

Value

An object of class km (see km-class).

Author(s)

O. Roustant

See Also

km

Examples

```
# a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
design.fact <- expand.grid(x1=seq(0,1,length=4), x2=seq(0,1,length=4))
y <- apply(design.fact, 1, branin)
data <- cbind(design.fact, y=y)

# kriging model 1 : matern5_2 covariance structure, no trend, no nugget effect
m1 <- kmData(y~1, data=data)
# this is equivalent to: m1 <- km(design=design.fact, response=y)
```

```
# now, add a second response to data:
data2 <- cbind(data, y2=-y)
# the previous model is now obtained with:
m1_2 <- kmData(y~1, data=data2, inputnames=c("x1", "x2"))
```

`leaveOneOut.km` *Leave-one-out for a km object*

Description

Cross validation by leave-one-out for a `km` object without noisy observations.

Usage

```
leaveOneOut.km(model, type, trend.reestim=FALSE)
```

Arguments

`model`	an object of class "km" without noisy observations.
`type`	a character string corresponding to the kriging family, to be chosen between simple kriging ("SK"), or universal kriging ("UK").
`trend.reestim`	should the trend be reestimated when removing an observation? Default to FALSE.

Details

Leave-one-out (LOO) consists of computing the prediction at a design point when the corresponding observation is removed from the learning set (and this, for all design points). A quick version of LOO based on Dubrule formula is also implemented; It is limited to 2 cases: `type=="SK" & (!trend.reestim)` and `type=="UK" & trend.reestim`. Leave-one-out is not implemented yet for noisy observations.

Value

A list composed of

`mean`	a vector of length n. The ith coordinate is equal to the kriging mean (including the trend) at the ith observation number when removing it from the learning set,
`sd`	a vector of length n. The ith coordinate is equal to the kriging standard deviation at the ith observation number when removing it from the learning set,

where n is the total number of observations.

Warning

Kriging parameters are not re-estimated when removing one observation. With few points, the re-estimated values can be far from those obtained with the entire learning set. One option is to reestimate the trend coefficients, by setting `trend.reestim=TRUE`.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

F. Bachoc (2013), Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics and Data Analysis*, **66**, 55-69. http://www.lpma.math.upmc.fr/pageperso/bachoc/publications.html

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

O. Dubrule (1983), Cross validation of Kriging in a unique neighborhood. *Mathematical Geology*, **15**, 687-699.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

predict,km-method, plot,km-method

---

leaveOneOutFun *Leave-one-out least square criterion of a km object*

Description

Returns the mean of the squared leave-one-out errors, computed with Dubrule's formula.

Usage

```
leaveOneOutFun(param, model, envir = NULL)
```

Arguments

param	a vector containing the optimization variables.
model	an object of class km.
envir	an optional environment specifying where to assign intermediate values for future gradient calculations. Default is NULL.

Value

The mean of the squared leave-one-out errors.

Note

At this stage, only the standard case has been implemented: no nugget effect, no observation noise.

Author(s)

O. Roustant, Ecole des Mines de St-Etienne

References

F. Bachoc (2013), Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics and Data Analysis*, **66**, 55-69. http://www.lpma.math.upmc.fr/pageperso/bachoc/publications.html

O. Dubrule (1983), Cross validation of Kriging in a unique neighborhood. *Mathematical Geology*, **15**, 687-699.

See Also

leaveOneOut.km, leaveOneOutGrad

---

leaveOneOutGrad *Leave-one-out least square criterion - Analytical gradient*

Description

Returns the analytical gradient of leaveOneOutFun.

Usage

```
leaveOneOutGrad(param, model, envir)
```

Arguments

param	a vector containing the optimization variables.
model	an object of class km.
envir	an environment specifying where to get intermediate values calculated in leaveOneOutFun.

Value

the gradient of leaveOneOutFun at param.

Author(s)

O. Roustant, Ecole des Mines de St-Etienne

References

F. Bachoc (2013), Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification. *Computational Statistics and Data Analysis*, **66**, 55-69. http://www.lpma.math.upmc.fr/pageperso/bachoc/publications.html

O. Dubrule (1983), Cross validation of Kriging in a unique neighborhood. *Mathematical Geology*, **15**, 687-699.

See Also

leaveOneOutFun

---

logLik *log-likelihood of a km object*

Description

Returns the log-likelihood value of a km object.

Usage

```
## S4 method for signature 'km'
logLik(object, ...)
```

Arguments

object	an object of class km containing the trend and covariance structures.
...	no other argument for this method.

Value

The log likelihood value.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

D. Ginsbourger, D. Dupuy, A. Badea, O. Roustant, and L. Carraro (2009), A note on the choice and the estimation of kriging models for the analysis of deterministic computer experiments, *Applied Stochastic Models for Business and Industry*, **25** no. 2, 115-131.

R. Li and A. Sudjianto (2005), Analysis of Computer Experiments Using Penalized Likelihood in Gaussian Kriging Models, *Technometrics*, **47** no. 2, 111-120.

K.V. Mardia and R.J. Marshall (1984), Maximum likelihood estimation of models for residual covariance in spatial regression, *Biometrika*, **71**, 135-146.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn (1989), Design and analysis of computer experiments, *Statistical Science*, **4**, 409-435.

M.L. Stein (1999), *Interpolation of spatial data, some theory for kriging*, Springer.

See Also

km, logLikFun

`logLikFun` *Concentrated log-likelihood of a km object*

Description

Returns the concentrated log-likelihood, obtained from the likelihood by plugging in the estimators of the parameters that can be expressed in function of the other ones.

Usage

```
logLikFun(param, model, envir=NULL)
```

Arguments

`param`	a vector containing the optimization variables.
`model`	an object of class `km`.
`envir`	an optional environment specifying where to assign intermediate values for future gradient calculations. Default is NULL.

Details

When there is no nugget effect nor observation noise, the concentrated log-likelihood is obtained by plugging in the variance and the trend MLE. Maximizing the likelihood is then equivalent to maximizing the concentrated log-likelihood with respect to the covariance parameters. In the other cases, the maximization of the concentrated log-likelihood also involves other parameters (the variance explained by the stationary part of the process for noisy observations, and this variance divided by the total variance if there is an unknown homogeneous nugget effect).

Value

The concentrated log-likelihood value.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne

References

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

See Also

logLik,km-method, km, logLikGrad

---

ninput *Get the spatial dimension*

Description

Get the spatial dimension (number of input variables).

Usage

```
ninput(x)
```

Arguments

x	an object containing the covariance structure.

Value

An integer equal to the spatial dimension.

---

nuggetflag *Get the nugget flag*

Description

Get a boolean indicating whether there is a nugget effect.

Usage

```
nuggetflag(x)
```

Arguments

x	an object containing the covariance structure.

Value

A boolean.

nuggetvalue *Get or set the nugget value*

Description

Get or set the nugget value.

Usage

```
nuggetvalue(x)
nuggetvalue(x) <- value
```

Arguments

x	an object containing the covariance structure.
value	an optional variance value standing for the homogeneous nugget effect.

plot *Diagnostic plot for the validation of a km object*

Description

Three plots are currently available, based on the `leaveOneOut.km` results: one plot of fitted values against response values, one plot of standardized residuals, and one qqplot of standardized residuals.

Usage

```
## S4 method for signature 'km'
plot(x, y, kriging.type = "UK", trend.reestim = FALSE, ...)
```

Arguments

x	an object of class "km" without noisy observations.
y	not used.
kriging.type	an optional character string corresponding to the kriging family, to be chosen between simple kriging ("SK") or universal kriging ("UK").
trend.reestim	should the trend be reestimated when removing an observation? Default to FALSE.
...	no other argument for this method.

Details

The diagnostic plot has not been implemented yet for noisy observations. The standardized residuals are defined by `( y(xi) - yhat_{-i}(xi) ) / sigmahat_{-i}(xi)`, where `y(xi)` is the response at the point `xi`, `yhat_{-i}(xi)` is the fitted value when removing the observation `xi` (see `leaveOneOut.km`), and `sigmahat_{-i}(xi)` is the corresponding kriging standard deviation.

Value

A list composed of:

mean	a vector of length *n*. The ith coordinate is equal to the kriging mean (including the trend) at the ith observation number when removing it from the learning set,
sd	a vector of length *n*. The ith coordinate is equal to the kriging standard deviation at the ith observation number when removing it from the learning set,

where *n* is the total number of observations.

Warning

Kriging parameters are not re-estimated when removing one observation. With few points, the re-estimated values can be far from those obtained with the entire learning set. One option is to reestimate the trend coefficients, by setting `trend.reestim=TRUE`.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

M. Schonlau (1997), *Computer experiments and global optimization*, Ph.D. thesis, University of Waterloo.

See Also

predict,km-method, leaveOneOut.km

Examples

```
# A 2D example - Branin-Hoo function

# a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
fact.design <- expand.grid(seq(0,1,length=4), seq(0,1,length=4))
fact.design <- data.frame(fact.design); names(fact.design)<-c("x1", "x2")
branin.resp <- data.frame(branin(fact.design)); names(branin.resp) <- "y"

# kriging model 1 : gaussian covariance structure, no trend,
#                   no nugget effect
m1 <- km(~.^2, design=fact.design, response=branin.resp, covtype="gauss")
plot(m1)  # LOO without parameter reestimation
plot(m1, trend.reestim=TRUE)  # LOO with trend parameters reestimation
                              # (gives nearly the same result here)
```

predict *Predict values and confidence intervals at newdata for a km object*

Description

Predicted values and (marginal of joint) conditional variances based on a km model. 95 % confidence intervals are given, based on strong assumptions: Gaussian process assumption, specific prior distribution on the trend parameters, known covariance parameters. This might be abusive in particular in the case where estimated covariance parameters are plugged in.

Usage

```
## S4 method for signature 'km'
predict(object, newdata, type, se.compute = TRUE,
         cov.compute = FALSE, light.return = FALSE,
         bias.correct = FALSE, checkNames = TRUE, ...)
```

Arguments

object	an object of class km.
newdata	a vector, matrix or data frame containing the points where to perform predictions.
type	a character string corresponding to the kriging family, to be chosen between simple kriging ("SK"), or universal kriging ("UK").
se.compute	an optional boolean. If FALSE, only the kriging mean is computed. If TRUE, the kriging variance (actually, the corresponding standard deviation) and confidence intervals are computed too.
cov.compute	an optional boolean. If TRUE, the conditional covariance matrix is computed.
light.return	an optional boolean. If TRUE, c and Tinv.c are not returned. This should be reserved to expert users who want to save memory and know that they will not miss these values.
bias.correct	an optional boolean to correct bias in the UK variance and covariances. Default is FALSE. See Section Warning below.
checkNames	an optional boolean. If TRUE (default), a consistency test is performed between the names of newdata and the names of the experimental design (contained in object@X), see Section Warning below.
...	no other argument for this method.

Value

mean	kriging mean (including the trend) computed at newdata.
sd	kriging standard deviation computed at newdata. Not computed if se.compute=FALSE.
trend	the trend computed at newdata.

cov	kriging conditional covariance matrix. Not computed if cov.compute=FALSE (default).
lower95, upper95	bounds of the 95 % confidence interval computed at newdata (to be interpreted with special care when parameters are estimated, see description above). Not computed if se.compute=FALSE.
c	an auxiliary matrix, containing all the covariances between newdata and the initial design points. Not returned if light.return=TRUE.
Tinv.c	an auxiliary vector, equal to T^(-1)*c. Not returned if light.return=TRUE.

Warning

1. Contrarily to DiceKriging<=1.3.2, the estimated (UK) variance and covariances are NOT multiplied by n/(n-p) by default (n and p denoting the number of rows and columns of the design matrix F). Recall that this correction would contribute to limit bias: it would totally remove it if the correlation parameters were known (which is not the case here). However, this correction is often useless in the context of computer experiments, especially in adaptive strategies. It can be activated by turning bias.correct to TRUE, when type="UK".

2. The columns of newdata should correspond to the input variables, and only the input variables (nor the response is not admitted, neither external variables). If newdata contains variable names, and if checkNames is TRUE (default), then checkNames performs a complete consistency test with the names of the experimental design. Otherwise, it is assumed that its columns correspond to the same variables than the experimental design and in the same order.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

A.G. Journel and C.J. Huijbregts (1978), *Mining Geostatistics*, Academic Press, London.

D.G. Krige (1951), A statistical approach to some basic mine valuation problems on the witwatersrand, *J. of the Chem., Metal. and Mining Soc. of South Africa*, **52** no. 6, 119-139.

J.D. Martin and T.W. Simpson (2005), Use of kriging models to approximate deterministic computer models, *AIAA Journal*, **43** no. 4, 853-863.

G. Matheron (1963), Principles of geostatistics, *Economic Geology*, **58**, 1246-1266.

G. Matheron (1969), Le krigeage universel, *Les Cahiers du Centre de Morphologie Mathematique de Fontainebleau*, **1**.

J.-S. Park and J. Baek (2001), Efficient computation of maximum likelihood estimators in a spatial linear model with power exponential covariogram, *Computer Geosciences*, **27** no. 1, 1-7.

C.E. Rasmussen and C.K.I. Williams (2006), *Gaussian Processes for Machine Learning*, the MIT Press, http://www.GaussianProcess.org/gpml

J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn (1989), Design and analysis of computer experiments, *Statistical Science*, **4**, 409-435.

See Also

km, plot,km-method

Examples

```
# ------------
# a 1D example
# ------------

x <- c(0, 0.4, 0.6, 0.8, 1)
y <- c(-0.3, 0, 0, 0.5, 0.9)

formula <- y~x   # try also   y~1  and  y~x+I(x^2)

model <- km(formula=formula, design=data.frame(x=x), response=data.frame(y=y),
            covtype="matern5_2")

tmin <- -0.5; tmax <- 2.5
t <- seq(from=tmin, to=tmax, by=0.005)
color <- list(SK="black", UK="blue")

# Results with Universal Kriging formulae (mean and and 95% intervals)
p.UK <- predict(model, newdata=data.frame(x=t), type="UK")
plot(t, p.UK$mean, type="l", ylim=c(min(p.UK$lower95),max(p.UK$upper95)),
                xlab="x", ylab="y")
lines(t, p.UK$trend, col="violet", lty=2)
lines(t, p.UK$lower95, col=color$UK, lty=2)
lines(t, p.UK$upper95, col=color$UK, lty=2)
points(x, y, col="red", pch=19)
abline(h=0)

# Results with Simple Kriging (SK) formula. The difference between the width of
# SK and UK intervals are due to the estimation error of the trend parameters
# (but not to the range parameters, not taken into account in the UK formulae).
p.SK <- predict(model, newdata=data.frame(x=t), type="SK")
lines(t, p.SK$mean, type="l", ylim=c(-7,7), xlab="x", ylab="y")
lines(t, p.SK$lower95, col=color$SK, lty=2)
lines(t, p.SK$upper95, col=color$SK, lty=2)
points(x, y, col="red", pch=19)
abline(h=0)

legend.text <- c("Universal Kriging (UK)", "Simple Kriging (SK)")
legend(x=tmin, y=max(p.UK$upper), legend=legend.text,
       text.col=c(color$UK, color$SK), col=c(color$UK, color$SK),
       lty=3, bg="white")


# ------------------------------------------------------------------------------
# a 1D example (following)- COMPARISON with the PREDICTION INTERVALS for REGRESSION
# ------------------------------------------------------------------------------
# There are two interesting cases:
# *  When the range parameter is near 0 ; Then the intervals should be nearly
```

```
#    the same for universal kriging (when bias.correct=TRUE, see above) as for regression.
#    This is because the uncertainty around the range parameter is not taken into account
#    in the Universal Kriging formula.
# *  Where the predicted sites are "far" (relatively to the spatial correlation)
#    from the design points ; in this case, the kriging intervals are not equal
#    but nearly proportional to the regression ones, since the variance estimate
#    for regression is not the same than for kriging (that depends on the
#    range estimate)

x <- c(0, 0.4, 0.6, 0.8, 1)
y <- c(-0.3, 0, 0, 0.5, 0.9)

formula <- y~x   # try also   y~1  and  y~x+I(x^2)
upper <- 0.05    # this is to get something near to the regression case.
                 # Try also upper=1 (or larger) to get usual results.

model <- km(formula=formula, design=data.frame(x=x), response=data.frame(y=y),
               covtype="matern5_2", upper=upper)

tmin <- -0.5; tmax <- 2.5
t <- seq(from=tmin, to=tmax, by=0.005)
color <- list(SK="black", UK="blue", REG="red")

# Results with Universal Kriging formulae (mean and and 95% intervals)
p.UK <- predict(model, newdata=data.frame(x=t), type="UK", bias.correct=TRUE)
plot(t, p.UK$mean, type="l", ylim=c(min(p.UK$lower95),max(p.UK$upper95)),
                   xlab="x", ylab="y")
lines(t, p.UK$trend, col="violet", lty=2)
lines(t, p.UK$lower95, col=color$UK, lty=2)
lines(t, p.UK$upper95, col=color$UK, lty=2)
points(x, y, col="red", pch=19)
abline(h=0)

# Results with Simple Kriging (SK) formula. The difference between the width of
# SK and UK intervals are due to the estimation error of the trend parameters
# (but not to the range parameters, not taken into account in the UK formulae).
p.SK <- predict(model, newdata=data.frame(x=t), type="SK")
lines(t, p.SK$mean, type="l", ylim=c(-7,7), xlab="x", ylab="y")
lines(t, p.SK$lower95, col=color$SK, lty=2)
lines(t, p.SK$upper95, col=color$SK, lty=2)
points(x, y, col="red", pch=19)
abline(h=0)

# results with regression given by lm (package stats)
m.REG <- lm(formula)
p.REG <- predict(m.REG, newdata=data.frame(x=t), interval="prediction")
lines(t, p.REG[,1], col=color$REG)
lines(t, p.REG[,2], col=color$REG, lty=2)
lines(t, p.REG[,3], col=color$REG, lty=2)

legend.text <- c("UK with bias.correct=TRUE", "SK", "Regression")
legend(x=tmin, y=max(p.UK$upper), legend=legend.text,
       text.col=c(color$UK, color$SK, color$REG),
```

```
       col=c(color$UK, color$SK, color$REG), lty=3, bg="white")


# ----------------------------------
# A 2D example - Branin-Hoo function
# ----------------------------------

# a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
fact.design <- expand.grid(x1=seq(0,1,length=4), x2=seq(0,1,length=4))
branin.resp <- apply(fact.design, 1, branin)

# kriging model 1 : gaussian covariance structure, no trend,
#                   no nugget effect
m1 <- km(~1, design=fact.design, response=branin.resp, covtype="gauss")

# predicting at testdata points
testdata <- expand.grid(x1=s <- seq(0,1, length=15), x2=s)
predicted.values.model1 <- predict(m1, testdata, "UK")
```

SCAD *Penalty function*

Description

Smoothly Clipped Absolute Deviation function.

Usage

```
SCAD(x, lambda)
```

Arguments

x	a vector where the function is to be evaluated.
lambda	a number representing a tuning parameter.

Details

SCAD is an even continuous function equal to 0 at `x=0`, and defined piecewise with derivative `lambda` in `[0, lambda]`, `(a*lambda - x)/(a-1)` in `[lambda, a*lambda]`, and `0` for `x` larger than `a*lambda`. As suggested by (Li, Sudjianto, 2005), we set `a=3.7`.

Value

A vector containing the SCAD values at `x`.

Note

In MLE problems, the penalty value `lambda` should tend to 0 when the sample size tends to infinity to insure that the asymptotic properties of Penalized-MLE and MLE are the same (see Li, Sudjianto, 2005).

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

R. Li and A. Sudjianto (2005), Analysis of Computer Experiments Using Penalized Likelihood in Gaussian Kriging Models, *Technometrics*, **47** no. 2, 111-120.

See Also

`SCAD.derivative` and `km` for a famous example

Examples

```
x <- seq(-8,8, length=200)
a <- 3.7

lambda <- 1.5
y <- SCAD(x, lambda)
plot(x, y, type="l", ylim=c(0,6))
x.knots <- c(-a*lambda, -lambda, 0, lambda, a*lambda)
points(x.knots, SCAD(x.knots, lambda), pch=19, cex=0.5)
text(6, SCAD(6, lambda)+0.3, paste("lambda =", lambda))

for (i in 1:2) {
   lambda <- lambda - 0.5
   y <- SCAD(x, lambda)
   lines(x, y, type="l")
   x.knots <- c(-a*lambda, -lambda, 0, lambda, a*lambda)
   points(x.knots, SCAD(x.knots, lambda), pch=19, cex=0.5)
   text(6, SCAD(6, lambda)+0.3, paste("lambda =", lambda))
}

abline(v=0, h=0, lty="dotted")
title("SCAD function")
```

---

scalingFun *Scaling function*

Description

Parametric transformation of the input space variables. The transformation is obtained coordinate-wise by integrating piecewise affine marginal "densities" parametrized by a vector of knots and a matrix of density values at the knots. See references for more detail.

Usage

```
scalingFun(X, knots, eta, plot=FALSE)
```

Arguments

X	an n*d matrix standing for a design of n experiments in d-dimensional space
knots	a list of knots parametrizing the transformation.
eta	a list of coefficients parametrizing the d marginal transformations. Each element stands for a set of marginal density values at the knots defined above.
plot	if TRUE plots the image of the columns of X according to the corresponding marginal transformations.

Value

The image of X by a scaling transformation of parameters knots and eta

References

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

See Also

scalingGrad

Examples

```
## 1D Transform of Xiong et al.
knots <- c(0, 0.3, 0.8, 1); eta <- c(2, 0.4, 1.4, 1.1)
nk <- length(knots)
t <- seq(from = 0, to = 1, length = 200)
f <- scalingFun(X = matrix(t), knots = list(knots), eta = list(eta))

## for text positions only
itext <- round(length(t) * 0.7)
xtext <- t[itext]; ftext <- f[itext] / 2; etamax <- max(eta)

## plot the transform function
opar <- par(mfrow = c(2, 1))
par(mar = c(0, 4, 5, 4))
plot(x = t, y = f, type = "l", lwd = 2, col = "orangered",
     main = "scaling transform f(x) and density g(x)",
     xlab = "", ylab = "", xaxt = "n", yaxt = "n")
axis(side = 4)
```

```
abline(v = knots, lty = "dotted"); abline(h = 0)
text(x = xtext, y = ftext, cex = 1.4,
     labels = expression(f(x) == integral(g(t)*dt, 0, x)))

## plot the density function, which is piecewise linear
scalingDens1d <- approxfun(x = knots, y = eta)
g <- scalingDens1d(t)
gtext <- 0.5 * g[itext] + 0.6 * etamax
par(mar = c(5, 4, 0, 4))
plot(t, g, type = "l", lwd = 2, ylim = c(0, etamax * 1.2),
     col = "SpringGreen4", xlab = expression(x), ylab ="")
abline(v = knots, lty = "dotted")
lines(x = knots, y = eta, lty = 1, lwd = 2, type = "h", col = "SpringGreen4")
abline(h = 0)
text(x = 0.7, y = gtext, cex = 1.4, labels = expression(g(x)))

## show knots with math symbols eta, zeta
for (i in 1:nk) {
  text(x = knots[i], y = eta[i] + 0.12 * etamax, cex = 1.4,
       labels = substitute(eta[i], list(i = i)))
  mtext(side = 1, cex = 1.4, at = knots[i], line = 2.4,
        text = substitute(zeta[i], list(i = i)))
}
polygon(x = c(knots, knots[nk], knots[1]),  y = c(eta, 0, 0),
        density = 15, angle = 45, col = "SpringGreen", border = NA)
par(opar)
```

`scalingGrad` *Gradient of the dimensional Scaling function*

Description

Gradient of the Scaling function (marginal in dimension k) of Xiong et al. with respect to eta

Usage

```
scalingGrad(X, knots, k)
```

Arguments

`X`	an n*d matrix standing for a design of n experiments in d-dimensional space.
`knots`	a list of knots parametrizing the transformation.
`k`	dimension of the input variables for which the gradient is calculated.

Value

Gradient of the Scaling function of Xiong et al. with respect to eta

References

Y. Xiong, W. Chen, D. Apley, and X. Ding (2007), *Int. J. Numer. Meth. Engng*, A non-stationary covariance-based Kriging method for metamodelling in engineering design.

See Also

scalingFun

---

show *Print values of a km object*

Description

Show method for km object. Printing the main features of a kriging model.

Usage

```
   ## S4 method for signature 'km'
show(object)
```

Arguments

object	an object of class km.

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

See Also

km

Examples

```
# A 2D example - Branin-Hoo function

# a 16-points factorial design, and the corresponding response
d <- 2; n <- 16
fact.design <- expand.grid(seq(0,1,length=4), seq(0,1,length=4))
fact.design <- data.frame(fact.design); names(fact.design)<-c("x1", "x2")
branin.resp <- data.frame(branin(fact.design)); names(branin.resp) <- "y"

# kriging model 1 : power-exponential covariance structure, no trend,
#                   no nugget effect
m1 <- km(y~1, design=fact.design, response=branin.resp, covtype="powexp")
m1    # equivalently : show(m1)
```

simulate *Simulate GP values at any given set of points for a km object*

Description

simulate is used to simulate Gaussian process values at any given set of points for a specified km object.

Usage

```
## S4 method for signature 'km'
simulate(object, nsim=1, seed=NULL, newdata=NULL,
                            cond=FALSE, nugget.sim=0, checkNames=TRUE, ...)
```

Arguments

object	an object of class km.
nsim	an optional number specifying the number of response vectors to simulate. Default is 1.
seed	usual seed argument of method simulate. Not used yet in simulated.km.
newdata	an optional vector, matrix or data frame containing the points where to perform predictions. Default is NULL: simulation is performed at design points specified in object.
cond	an optional boolean indicating the type of simulations. If TRUE, the simulations are performed conditionally to the response vector defined by using km, and contained in model (slot y: model@y). If FALSE, the simulations are non conditional. Default is FALSE.
nugget.sim	an optional number corresponding to a numerical nugget effect, which may be useful in presence of numerical instabilities. If specified, it is added to the diagonal terms of the covariance matrix (that is: newdata if cond=TRUE, or of (newdata, model@y) either) to ensure that it is positive definite. In any case, this parameter does not modify model. It has no effect if newdata=NULL. Default is 0.
checkNames	an optional boolean. If TRUE (default), a consistency test is performed between the names of newdata and the names of the experimental design (contained in object@X), see section Warning below.
...	no other argument for this method.

Value

A matrix containing the simulated response vectors at the newdata points, with one sample in each row.

Warning

The columns of newdata should correspond to the input variables, and only the input variables (nor the response is not admitted, neither external variables). If newdata contains variable names, and if checkNames is TRUE (default), then checkNames performs a complete consistency test with the names of the experimental design. Otherwise, it is assumed that its columns correspond to the same variables than the experimental design and in the same order.

Note

1. When constructing a km object with known parameters, note that the argument y (the output) is required in km even if it w
2. Sometimes, a small nugget effect is necessary to avoid numerical instabilities (see the ex. below).

Author(s)

O. Roustant, D. Ginsbourger, Ecole des Mines de St-Etienne.

References

N.A.C. Cressie (1993), *Statistics for spatial data*, Wiley series in probability and mathematical statistics.

A.G. Journel and C.J. Huijbregts (1978), *Mining Geostatistics*, Academic Press, London.

B.D. Ripley (1987), *Stochastic Simulation*, Wiley.

See Also

km

Examples

```
# ----------------
# some simulations
# ----------------

n <- 200
x <- seq(from=0, to=1, length=n)

covtype <- "matern3_2"
coef.cov <- c(theta <- 0.3/sqrt(3))
sigma <- 1.5
trend <- c(intercept <- -1, beta1 <- 2, beta2 <- 3)
nugget <- 0   # may be sometimes a little more than zero in some cases,
              # due to numerical instabilities

formula <- ~x+I(x^2)    # quadratic trend (beware to the usual I operator)
```

```
ytrend <- intercept + beta1*x + beta2*x^2
plot(x, ytrend, type="l", col="black", ylab="y", lty="dashed",
     ylim=c(min(ytrend)-2*sigma, max(ytrend) + 2*sigma))

model <- km(formula, design=data.frame(x=x), response=rep(0,n),
            covtype=covtype, coef.trend=trend, coef.cov=coef.cov,
            coef.var=sigma^2, nugget=nugget)
y <- simulate(model, nsim=5, newdata=NULL)

for (i in 1:5) {
  lines(x, y[i,], col=i)
}


# ------------------------------------------------------------------
# conditional simulations and consistancy with Simple Kriging formulas
# ------------------------------------------------------------------

n <- 6
m <- 101
x <- seq(from=0, to=1, length=n)
response <- c(0.5, 0, 1.5, 2, 3, 2.5)

covtype <- "matern5_2"
coef.cov <- 0.1
sigma <- 1.5

trend <- c(intercept <- 5, beta <- -4)
model <- km(formula=~cos(x), design=data.frame(x=x), response=response,
            covtype=covtype, coef.trend=trend, coef.cov=coef.cov,
            coef.var=sigma^2)

t <- seq(from=0, to=1, length=m)
nsim <- 1000
y <- simulate(model, nsim=nsim, newdata=data.frame(x=t), cond=TRUE, nugget.sim=1e-5)

## graphics

plot(x, intercept + beta*cos(x), type="l", col="black",
     ylim=c(-4, 7), ylab="y", lty="dashed")
for (i in 1:nsim) {
lines(t, y[i,], col=i)
}

p <- predict(model, newdata=data.frame(x=t), type="SK")
lines(t, p$lower95, lwd=3)
lines(t, p$upper95, lwd=3)

points(x, response, pch=19, cex=1.5, col="red")

# compare theoretical kriging mean and sd with the mean and sd of
# simulated sample functions
```

```
mean.theoretical <- p$mean
sd.theoretical <- p$sd
mean.simulated <- apply(y, 2, mean)
sd.simulated <- apply(y, 2, sd)
par(mfrow=c(1,2))
plot(t, mean.theoretical, type="l")
lines(t, mean.simulated, col="blue", lty="dotted")
points(x, response, pch=19, col="red")
plot(t, sd.theoretical, type="l")
lines(t, sd.simulated, col="blue", lty="dotted")
points(x, rep(0, n), pch=19, col="red")
par(mfrow=c(1,1))

# estimate the confidence level at each point
level <- rep(0, m)
for (j in 1:m) {
level[j] <- sum((y[,j]>=p$lower95[j]) & (y[,j]<=p$upper95[j]))/nsim
}
level    # level computed this way may be completely wrong at interpolation
         # points, due to the numerical errors in the calculation of the
         # kriging mean


# ----------------------------------------------------------------------
# covariance kernel + simulations for "exp", "matern 3/2", "matern 5/2"
#                                  and "exp" covariances
# ----------------------------------------------------------------------

covtype <- c("exp", "matern3_2", "matern5_2", "gauss")

d <- 1
n <- 500
x <- seq(from=0, to=3, length=n)

par(mfrow=c(1,2))
plot(x, rep(0,n), type="l", ylim=c(0,1), xlab="distance", ylab="covariance")

param <- 1
sigma2 <- 1

for (i in 1:length(covtype)) {
covStruct <- covStruct.create(covtype=covtype[i], d=d, known.covparam="All",
                     var.names="x", coef.cov=param, coef.var=sigma2)
y <- covMat1Mat2(covStruct, X1=as.matrix(x), X2=as.matrix(0))
lines(x, y, col=i, lty=i)
}
legend(x=1.3, y=1, legend=covtype, col=1:length(covtype),
       lty=1:length(covtype), cex=0.8)

plot(x, rep(0,n), type="l", ylim=c(-2.2, 2.2), xlab="input, x",
     ylab="output, f(x)")
for (i in 1:length(covtype)) {
model <- km(~1, design=data.frame(x=x), response=rep(0,n), covtype=covtype[i],
```

```
    coef.trend=0, coef.cov=param, coef.var=sigma2, nugget=1e-4)
y <- simulate(model)
lines(x, y, col=i, lty=i)
}
par(mfrow=c(1,1))

# ------------------------------------------------------
# covariance kernel + simulations for "powexp" covariance
# ------------------------------------------------------

covtype <- "powexp"

d <- 1
n <- 500
x <- seq(from=0, to=3, length=n)

par(mfrow=c(1,2))
plot(x, rep(0,n), type="l", ylim=c(0,1), xlab="distance", ylab="covariance")

param <- c(1, 1.5, 2)
sigma2 <- 1

for (i in 1:length(param)) {
covStruct <- covStruct.create(covtype=covtype, d=d, known.covparam="All",
                     var.names="x", coef.cov=c(1, param[i]), coef.var=sigma2)
y <- covMat1Mat2(covStruct, X1=as.matrix(x), X2=as.matrix(0))
lines(x, y, col=i, lty=i)
}
legend(x=1.4, y=1, legend=paste("p=", param), col=1:3, lty=1:3)

plot(x, rep(0,n), type="l", ylim=c(-2.2, 2.2), xlab="input, x",
     ylab="output, f(x)")
for (i in 1:length(param)) {
model <- km(~1, design=data.frame(x=x), response=rep(0,n), covtype=covtype,
        coef.trend=0, coef.cov=c(1, param[i]), coef.var=sigma2, nugget=1e-4)
y <- simulate(model)
lines(x, y, col=i)
}
par(mfrow=c(1,1))
```

---

update *Update of a kriging model*

Description

Update a km object when one or many new observations are added. Many, but not all, fields of the km object need to be recalculated when new observations are added. It is also possible to modify the k last (existing) observations.

Usage

```
## S4 method for signature 'km'
update(object, newX, newy, newX.alreadyExist = FALSE,
          cov.reestim = TRUE, trend.reestim = TRUE, nugget.reestim = FALSE,
          newnoise.var = NULL, kmcontrol = NULL, newF = NULL,...)
```

Arguments

object	Kriging model of km class.
newX	Matrix with object@d columns and r rows corresponding to the r locations of the observations to be updated. These locations can be new locations or existing ones.
newy	Matrix with one column and r rows corresponding to the r responses at the r locations newX.
newX.alreadyExist	Boolean: indicate whether the locations newX are all news or not.
cov.reestim	Should the covariance parameters of the km object be re-estimated?
trend.reestim	Should the trend parameters be re-estimated?
nugget.reestim	Should the nugget effect be re-estimated?
newnoise.var	Vector containing the noise variance at each new observations.
kmcontrol	Optional list representing the control variables for the re-estimation of the kriging model once new points are sampled. The items are the same as in km
newF	Optional matrix containing the value of the trend at the new locations. Setting this argument avoids a call to an expensive function.
...	Further arguments

Value

Updated km object

Author(s)

Clement Chevalier (IMSV, Switzerland, and IRSN, France)

References

Bect J., Ginsbourger D., Li L., Picheny V., Vazquez E. (2010), *Sequential design of computer experiments for the estimation of a probability of failure*, Statistics and Computing, pp.1-21, 2011, http://arxiv.org/abs/1009.5177

Chevalier C., Bect J., Ginsbourger D., Vazquez E., Picheny V., Richet Y. (2011), *Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set*, http://hal.archives-ouvertes.fr/hal-00641108/

See Also

km

Examples

```
set.seed(8)
N <- 9 # number of observations
testfun <- branin

# a 9 points initial design
design <- expand.grid(x1=seq(0,1,length=3), x2=seq(0,1,length=3))
response <- testfun(design)

# km object with matern3_2 covariance
# params estimated by ML from the observations
model <- km(formula = ~., design = design,
response = response, covtype = "matern3_2")
model@covariance

newX <- matrix(c(0.4,0.5), ncol = 2) #the point that we are going to add in the km object
newy <- testfun(newX)
newmodel <- update(object = model, newX = newX, newy = newy, cov.reestim = TRUE)
newmodel@covariance
```

Index