Package ‘kergp’

December 23, 2015

Type Package
Title Gaussian Process Laboratory
Version 0.2.0
Date 2015-12-22
Author Yves Deville, David Ginsbourger, Olivier Roustant. Contributors: Nicolas Durrande.
Maintainer Yves Deville <deville.yves@alpestat.com>
Description Gaussian Process models with customised covariance kernels.
License GPL-3
Depends Rcpp (>= 0.10.5), methods, testthat
Suggests DiceKriging, DiceDesign, lhs, inline, foreach
Imports MASS, numDeriv, stats4, doParallel
LinkingTo Rcpp
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-12-23 09:15:28

R topics documented:

  kergp-package .................................................. 2
  checkX .......................................................... 6
  checkX-methods ................................................. 6
  coef-methods ..................................................... 7
  coef<- .......................................................... 8
  coefLower ........................................................ 9
  covAll-class ..................................................... 9
  covMan .......................................................... 10
  covMan-class .................................................... 13
  covMat .......................................................... 15
  covMat-methods ................................................ 15
  covTS ............................................................ 17
  covTS-class ..................................................... 19
kergp-package

Description

Laboratory Package for Gaussian Process interpolation, regression and simulation, with an emphasis on user-defined covariance kernels.

Details

<table>
<thead>
<tr>
<th>Package:</th>
<th>kergp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>0.2.0</td>
</tr>
<tr>
<td>Date:</td>
<td>2015-12-22</td>
</tr>
<tr>
<td>License:</td>
<td>GPL-3</td>
</tr>
</tbody>
</table>

Warning

As a lab, kergp may strongly evolve in its future life. Users interested in stable software for the Analysis of Computer Experiments are encouraged to use other packages such as DiceKriging instead.
Note

This package was developed within the frame of the ReDice Consortium, gathering industrial partners (CEA, EDF, IFPEN, IRSN, Renault) and academic partners (Mines Saint-Étienne, INRIA, and the University of Bern) around advanced methods for Computer Experiments.

Author(s)

Yves Deville (Alpestat), David Ginsbourger (University of Bern), Olivier Roustant (Mines Saint-Étienne), with contributions from Nicolas Durrande (Mines Saint-Étienne).

Maintainer: Olivier Roustant, <olivier.roustant@mines-stetienne.fr>

References


Olivier Roustant, David Ginsbourger, Yves Deville (2012). "DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodelling and Optimization". *Journal of Statistical Software*, 51(1), 1-55. [link](http://example.com)

Examples

```r
## Gaussian process modelling of function with invariance properties,
## by using an argumentwise invariant kernel

## -- define manually an argumentwise invariant kernel --

kernFun <- function(x1, x2, par) {
  h <- (abs(x1) - abs(x2)) / par[1]
  S <- sum(h^2)
  d2 <- exp(-S)
  d1 <- 2 * K * S / par[1]
  attr(K, "gradient") <- c(theta = d1, sigma2 = d2)
  return(K)
}
```
## quicken: with Rcpp; see also an example with package inline
## in "gp" doc. file
# Not run:

```r
cppFunction(
  NumericVector cppKernFun(NumericVector x1, NumericVector x2,
      NumericVector par){
    int n1 = x1.size();
    double S, d1, d2;
    NumericVector K(1), h(n1);
    h = (abs(x1) - abs(x2)) / par[0]; // sugar function "abs"
    S = sum(h * h); // sugar "+" and "sum"
    d2 = exp(-S);
    K[0] = par[1] * d2;
    d1 = 2 * K[0] * S / par[0];
    K.attr("gradient") = NumericVector::create(Named("theta", d1),
      Named("sigma2", d2));
    return K;
  }
)
```

## End(Not run)

## Below: with the R-based code for the kernel namely 'kernFun'.
## You can also replace 'kernFun' by 'cppKernFun' for speed.

covSymGauss <- covMan(kernel = kernFun,
  hasGrad = TRUE,
  label = "argumentwise invariant",
  d = 2,
  parLower = c(theta = 0.0, sigma2 = 0.0),
  parUpper = c(theta = Inf, sigma2 = Inf),
  parNames = c("theta", "sigma2"),
  par = c(theta = 0.5, sigma2 = 2))

covSymGauss

## -- simulate a path from the corresponding GP --

nGrid <- 24; n <- nGrid^2; d <- 2
xGrid <- seq(from = -1, to = 1, length.out = nGrid)
xGrid <- expand.grid(x1 = xGrid, x2 = xGrid)
Kmat <- covMat(object = covSymGauss, X = xGrid,
  compGrad = FALSE, index = 1L)

library(MASS)
set.seed(1)
kergp-package

```r
ygrid <- mvrnorm(mu = rep(0, n), Sigma = Kmat)

## -- extract a design and the corr. response from the grid --

nDesign <- 25
tab <- subset(cbind(Xgrid, ygrid), x1 > 0 & x2 > 0)
rowIndex <- seq(1, nrow(tab), length = nDesign)
X <- tab[rowIndex, 1:2]
y <- tab[rowIndex, 3]

opar <- par(mfrow = c(1, 3))
contour(x = xGrid, y = xGrid,
        z = matrix(ygrid, nrow = nGrid, ncol = nGrid),
        nlevels = 15)
abline(h = 0, v = 0, col = "SpringGreen3")
points(x2 ~ x1, data = X, type = "p", pch = 21,
       col = "orangered", bg = "yellow", cex = 0.8)
title("GRF Simulation")

## -- Fit the Gaussian process model (trend + covariance parameters) --

symgp <- gp(formula = y ~ 1, data = data.frame(y, X),
           inputs = names(X),
           cov = covSymGauss,
           parCovIni = c(0.1, 2),
           varNoiseLower = 1e-8, varNoiseUpper = 1e-8)

# mind that the noise is not a symmetric kernel
# so varNoiseUpper should be chosen as small as possible.
summary(symgp)

## -- predict and compare --
predSymgp <- predict(object = symgp, newdata = Xgrid, type = "UK")

contour(x = xGrid, y = xGrid,
        z = matrix(predSymgp$mean, nrow = nGrid, ncol = nGrid),
        nlevels = 15)
abline(h = 0, v = 0, col = "SpringGreen3")
points(x2 ~ x1, data = X, type = "p", pch = 21,
       col = "orangered", bg = "yellow", cex = 0.8)
title("Kriging mean")

contour(x = xGrid, y = xGrid,
        z = matrix(predSymgp$sd, nrow = nGrid, ncol = nGrid),
        nlevels = 15)
abline(h = 0, v = 0, col = "SpringGreen3")
points(x2 ~ x1, data = X, type = "p", pch = 21,
       col = "orangered", bg = "yellow", cex = 0.8)
title("Kriging s.d.")
```
checkX-methods

Generic function to check the compatibility of a design matrix with a given covariance object

Description

Generic function to check the compatibility of a design matrix with a covariance object.

Usage

checkX(object, X, ...)

Arguments

object A covariance kernel object.
X A design matrix.
... Other arguments for methods.

Value

A matrix with columns taken from X and with column names identical to inputNames(object).

See Also

The inputNames method.

checkX-methods Check the compatibility of a design with a given covariance object

Description

Check the compatibility of a design matrix with a covariance object.

Usage

## S4 method for signature 'covAll'
checkX(object, X, strict = FALSE)
Arguments

object A covariance kernel object.
X A design matrix.
strict Logical. If TRUE, the character vectors colnames(X) and inputNames(object) must be the same sets, and hence have the same length. If FALSE the vector inputNames(object) must be a subset of colnames(X) which then can have unused columns.

Details

The matrix X must have the number of columns expected from the covariance kernel object description, and it must have named columns conforming to the kernel inputnames as returned by the inputNames method. If the two sets of names are identical but the names are in a different order, the columns are permuted in order to be in the same order as the inputnames. If the names sets differ, an error occurs.

Value

A matrix with columns names identical to the inputnames attached with the kernel object, i.e. inputNames(object). The columns are copies of those found under the same names in X, but are put in the order of inputNames(object). When an input name does not exist in colnames(X) an error occurs.

See Also

The inputNames method.

Description

Extract some of or all the coefficients of a covariance kernel object as vector, list or matrix.

Usage

```r
## S4 method for signature 'covMan'
coef(object)
```

```r
## S4 method for signature 'covTS'
coef(object, type = "all", as = "vector")
```
Arguments

object An object representing a covariance kernel, the coefficient of which will be extracted.

type Character string or vector specifying which type(s) of coefficients in the structure will be extracted. Can be "all" (all coefficients are extracted) or any parameter name(s) of the corresponding kernel.

as Character string specifying the output structure to be used. The default is "vector", leading to a numeric vector. Using "list" one gets a list of numeric vectors, one by kernel parameter. Finally, using "matrix" one gets a matrix with one row by input (or dimension) and one column by (selected) kernel parameter.

Value

A numeric vector of coefficients or a structure as specified by as containing the coefficients selected by type.

See Also

The coef<- replacement method which takes a vector of replacement values.

Examples

d <- 3
myCov1 <- covTS(d = d, kernel = "k1exp", dep = c(range = "input"),
               value = c(range = 1.1))
myCov1
## versatile 'coef' method
coef(myCov1)
coef(myCov1, as = "matrix")
coef(myCov1, as = "list")
coef(myCov1, as = "matrix", type = "range")
coef(myCov1) <- c(0.2, 0.3, 0.4, 4, 16, 25)
coef(myCov1, as = "matrix")

---

\[ \text{coef<-} \quad \text{Generic function for the replacement of coefficient values} \]

Description

Generic function for the replacement of coefficient values

Usage

`\text{coef<-}\text{(object, ..., value)}`
Arguments

- **object**: Object having a numeric vector of coefficients, typically a covariance kernel object.
- **...**: Other arguments for methods.
- **value**: The value of the coefficients to be set.

Value

The modified object.

---

**coefLower**

*Extract or set lower/upper bounds on coefficients*

### Description

Extract or set lower/upper bounds on coefficients for covariance kernel objects.

### Usage

```r
coefLower(object, ...)  
coefUpper(object, ...)
```

Arguments

- **object**: A covariance kernel object.
- **...**: Other arguments for methods.

Value

The lower or upper bounds on the covariance kernel parameters.

---

**covAll-class**

*Virtual class "covAll"*

### Description

Virtual class “covAll”, union of classes including "covTS", "covMan”.

### Methods

- **checkX** signature(object = "covAll", X = "matrix"): checks the compatibility of a design with a given covariance object.
- **inputNames** signature(object = "covAll"): returns the character vector of input names.
- **simulPar** signature(object = "covTS"): simulates random values for the parameters.
covMan

Creator function for covMan objects

Description

Creator function for covMan objects representing a covariance kernel entered manually.

Usage

```r
covMan(kernel, hasGrad = FALSE, acceptMatrix = FALSE, 
       inputs = paste("x", 1:d, sep = ""), 
       d = length(inputs), parNames, par = NULL, 
       label = "covMan", ...)
```

Arguments

- `kernel`: A (semi-)positive definite function. This must be an object of class "function" with formal arguments named "x1", "x2" and "par". The first two formal arguments are locations vectors or matrices. The third formal is for the vector \( \theta \) of all covariance parameters. An analytical gradient can be computed and returned as an attribute of the result with name "gradient". See Details.
- `hasGrad`: Logical indicating whether the `kernel` function returns the gradient w.r.t. the vector of parameters as a "gradient" attribute of the result. See Details.
- `acceptMatrix`: Logical indicating whether `kernel` admits matrices as arguments. Default is FALSE. See Examples below.
- `inputs`: Character vector giving the names of the inputs used as arguments of `kernel`. Optional if `d` is given.
- `d`: Integer specifying the spatial dimension (equal to the number of inputs). Optional if `inputs` is given.
- `parNames`: Vector of character strings containing the parameter names.
- `par`: Optional numeric vector containing the parameter values.
- `label`: Optional character string describing the kernel.
- `...`: Not used at this stage.

Details

The formals and the returned value of the `kernel` function must be in accordance with the value of `acceptMatrix`. 

Examples

```r
showClass("covAll")
```
• When `acceptMatrix` is `FALSE`, the formal arguments `x1` and `xR` of `kernel` are numeric vectors with length `d`. The returned result is a numeric vector of length 1. The attribute named "gradient" of the returned value (if provided in accordance with the value of `hasGrad`) must then be a numeric vector with length equal to the number of covariance parameters. It must contain the derivative of the kernel value \( K(x_1, x_2; \theta) \) with respect to the parameter vector \( \theta \).

• When `acceptMatrix` is `TRUE`, the formals `x1` and `x2` are matrices with `d` columns and with `n_1` and `n_2` rows. The result is then a covariance matrix with `n_1` rows and `n_2` columns. The gradient attribute (if provided in accordance with the value of `hasGrad`) must be a list with length equal to the number of covariance parameters. The list element \( \ell \) must contain a numeric matrix with dimension \((n_1, n_2)\) which is the derivative of the covariance matrix w.r.t. the covariance parameter \( \theta_\ell \).

**Note**

The kernel function must be symmetric with respect to its first two arguments, and it must be positive definite, which is not checked. If the function returns an object with a "gradient" attribute but `hasGrad` was set to `FALSE`, the gradient will not be used in optimization.

The name of the class was motivated by earlier stages in the development.

**Examples**

```r
myCovMan <- covMan(
  kernel = function(x1, x2, par) {
    htilde <- (x1 - x2) / par[1]
    SS2 <- sum(htilde^2)
    d2 <- exp(-SS2)
    kern <- par[2] * d2
    d1 <- 2 * kern * SS2 / par[1]
    attr(kern, "gradient") <- c(theta = d1, sigma2 = d2)
    return(kern)
  },
  hasGrad = TRUE,
  d = 1,
  label = "myGauss",
  parLower = c(theta = 0.0, sigma2 = 0.0),
  parUpper = c(theta = Inf, sigma2 = Inf),
  parNames = c("theta", "sigma2"),
  par = c(NA, NA)
)

# Let us now code the same kernel in C
kernCode <- "
SEXP kern, dkern;
int nprotect = 0, d;
double SS2 = 0.0, d2, z, *rkern, *rdkern;

d = LENGTH(x1);
PROTECT(kern = allocVector(REALSP, 1)); nprotect++;
PROTECT(dkern = allocVector(REALSP, 2)); nprotect++;
```

r kern = REAL(k kern);
rd kern = REAL(d kern);

for (int i = 0; i < d; i++) {
    z = (REAL(x1)[i] - REAL(x2)[i]) / REAL(par)[0];
    SS2 += z * z;
}

d2 = exp(-SS2);
r kern[0] = REAL(par)[1] * d2;
rd kern[1] = d2;
r kern[0] = 2 * r kern[0] * SS2 / REAL(par)[0];

SET_ATTR(kern, install("gradient"), dkern);
UNPROTECT(nprotect);
return kern;

myCovMan

## "inline" the C function into an R function: much more efficient!

## Not run:
require.inline)
kernC <- cfunction(sig = signature(x1 = "numeric", x2 = "numeric",
par = "numeric"),
body = kernCode)
myCovMan <- covMan(kernel = kernC, hasGrad = TRUE, d = 1,
parNames = c("theta", "sigma2"))

myCovMan

## End(Not run)

## A kernel admitting matricial arguments
myCov <- covMan(

kernel = function(X1, X2, par) {
# X1 : matrix of size n1xd
# X2 : matrix of size n2xd

d <- ncol(X1)

SS2 <- 0
for (j in 1:d){
    Aj <- outer(X1[, j], X2[, j], "+")
    Aj2 <- Aj^2
    SS2 <- SS2 + Aj2 / par[j]^2
}
D2 <- exp(-SS2)
kern <- par[d + 1] * D2
},
acceptMatrix = TRUE,
d = 2,
covMan-class

Description

S4 class representing a covariance kernel defined manually by a (semi-)positive definite function.

Objects from the Class

Objects can be created by calling `new("covMan", ...)` or by using the `covMan` function.

Slots

- `kernel`: object of class "function" defining the kernel (supposed to be (semi-)positive definite).
- `hasGrad`: logical indicating whether kernel returns the gradient (w.r.t. the vector of parameters) as "gradient" attribute of the result.
- `acceptMatrix`: logical indicating whether kernel admits matrix arguments. Default is FALSE.
- `label`: object of class character, typically one or two words, used to describe the kernel.
- `d`: object of class "integer", the spatial dimension or number of inputs of the covariance.
- `inputNames`: object of class "character", vector of input names. Length d.
- `parLower`:,
parUpper: object of class "numeric", vector of (possibly infinite) lower/upper bounds on parameters.
par: object of class "numeric", numeric vector of parameter values.
parN: object of class "integer", total number of parameters.
kernParNames: object of class "character", name of the kernel parameters.

Methods

coef<- signature(object = "covMan"): replace the whole vector of coefficients, as required during ML estimation.
coefLower<- signature(object = "covMan"): replacement method for lower bounds on covMan coefficients.
coefLower signature(object = "covMan"): extracts the numeric values of the lower bounds.
coef signature(object = "covMan"): extracts the numeric values of the covariance parameters.
coefUpper<- signature(object = "covMan"): replacement method for upper bounds on covMan coefficients.
coefUpper signature(object = "covMan"): ...
covMat signature(object = "covMan"): builds the covariance matrix or the cross covariance matrix between two sets of locations for a covMan object.
scores signature(object = "covMan"): computes the scores (derivatives of the log-likelihood w.r.t. the covariance parameters.
show signature(object = "covMan"): prints in a custom format

Note

While the coef<- replacement method is typically intended for internal use during likelihood maximization, the coefLower<- and coefUpper<- replacement methods can be used when some information is available on the possible values of the parameters.

Author(s)

Y. Deville, O. Roustant, D. Ginsbourger and N. Durrande.

See Also

The covMan function providing a creator.

Examples

showClass("covMan")
**covMat**

*Generic function returning a covariance or a cross-covariance matrix between two sets of locations*

**Description**
Generic function returning a covariance or a cross-covariance matrix between two sets of locations.

**Usage**

covMat(object, X, Xnew, ...)

**Arguments**
- `object` Covariance kernel object.
- `X` A matrix with $d$ columns, where $d$ is the number of inputs of the covariance kernel. The $n_1$ rows define a first set of sites or locations, typically used for learning.
- `Xnew` A matrix with $d$ columns, where $d$ is the number of inputs of the covariance kernel. The $n_2$ rows define a second set of sites or locations, typically used for testing or prediction. If `Xnew = NULL` the same locations are used: `Xnew = X`.
- `...` Other arguments for methods.

**Value**
A rectangular matrix with `nrow(X)` rows and `nrow(Xnew)` columns containing the covariances $K(x_1, x_2)$ for all the couples of sites $x_1$ and $x_2$.

---

**Description**
Covariance matrix for a covariance kernel object.

**Usage**

```r
# S4 method for signature 'covMan'
covMat(object, X, Xnew, compGrad = FALSE,
       checkNames = NULL, index = -1L, ...)
```

```r
# S4 method for signature 'covTS'
covMat(object, X, Xnew, compGrad = FALSE,
       checkNames = TRUE, index = -1L, ...)
```
Arguments

- **object**: An object with S4 class corresponding to a covariance kernel.
- **X**: The usual matrix of spatial design points, with \( n \) rows and \( d \) cols where \( n \) is the number of spatial points and \( d \) is the 'spatial' dimension.
- **Xnew**: An optional new matrix of spatial design points. If missing, the same matrix is used: \( X_{\text{new}} = X \).
- **compGrad**: Logical. If TRUE a derivative with respect to a parameter will be computed and returned as an attribute of the result. For the `covMan` class, this is possible only when the gradient of the kernel is computed and returned as a "gradient" attribute of the result.
- **checkNames**: Logical. If TRUE (default), check the compatibility of \( X \) with object. see `checkX`.
- **index**: Integer giving the index of the derivation parameter in the official order.
- **...**: not used yet.

Details

The covariance matrix is computed in a C program using the .Call interface. The R kernel function is evaluated within the C code using `eval`.

Value

A \( n_1 \times n_2 \) matrix with general element \( C_{ij} := K(x_{1,i}, x_{2,j}; \theta) \) where \( K(x_1, x_2; \theta) \) is the covariance kernel function.

Note

The value of the parameter \( \theta \) can be extracted from the object with the `coef` method.

Author(s)


See Also

- `coef` method

Examples

```r
myCov <- covTS(inputs = c("Temp", "Humid", "Press"),
               kernel = "k1powExp",
               dep = c(range = "cst", shape = "cst"),
               value = c(shape = 1.8, range = 1.1))
n <- 100; X <- matrix(runif(n*3), nrow = n, ncol = 3)
try(C1 <- covMat(myCov, X)) ## bad colnames
colnames(X) <- inputNames(myCov)
C2 <- covMat(myCov, X)
Xnew <- matrix(runif(n * 3), nrow = n, ncol = 3)
```
covTS

Creator function for covTS objects

Description

Creator function for covTS objects representing a Tensor Sum covariance kernel.

Usage

covTS(inputs = paste("x", 1:d, sep = ""),
      d = length(inputs), kernel = "k1matern5_2",
      dep = NULL, value = NULL, var = 1, ...)

Arguments

- **inputs**: Character vector giving the names of the inputs used as arguments of `kernel`. Optional if `d` is given.
- **d**: Integer specifying the spatial dimension (equal to the number of inputs). Optional if `inputs` is given.
- **kernel**: Character, name of the one-dimensional kernel.
- **dep**: Character vector with elements "cst" or "input" usually built using the concatenation `c`. The names must correspond to parameters of the kernel specified with `kernel`. When an element is "cst", the corresponding parameter of the 1d kernel will be the same for all inputs. When the element is "input", the corresponding parameter of the 1d kernel gives birth to `d` parameters in the covTS object, one by input.
- **value**: Named numeric vector. The names must correspond to the 1d kernel parameters.
- **var**: Numeric vector giving the variances $\sigma_i^2$ that weight the `d` components.
- **...**: Not used at this stage.

Details

A covTS object represents a `d`-dimensional kernel object $K$ of the form

$$K(x, x'; \theta) = \sum_{i=1}^{d} k(x_i, x'_i; \theta_{s_i})$$

where $k$ is the covariance kernel for a Gaussian Process $Y_x$ indexed by a scalar $x$. The `d` numbers $x_i$ stand for the components of the `d`-dimensional location vector $x$. The length $p$ of all the vectors...
$s_i$ is the number of parameters of the one-dimensional kernel $k$, i.e. 2 or 3 for classical covariance kernels.

The package comes with the following covariance kernels which can be given as kernel argument.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>$p$</th>
<th>par. names</th>
</tr>
</thead>
<tbody>
<tr>
<td>k1exp</td>
<td>exponential</td>
<td>2</td>
<td>range, var</td>
</tr>
<tr>
<td>k1matern3_2</td>
<td>Matérn $\nu = 3/2$</td>
<td>2</td>
<td>range, var</td>
</tr>
<tr>
<td>k1matern5_2</td>
<td>Matérn $\nu = 5/2$</td>
<td>2</td>
<td>range, var</td>
</tr>
<tr>
<td>k1powexp</td>
<td>power exponential</td>
<td>3</td>
<td>range, shape, var</td>
</tr>
<tr>
<td>k1gauss</td>
<td>gaussian or &quot;square exponential&quot;</td>
<td>2</td>
<td>range, var</td>
</tr>
</tbody>
</table>

Note that the exponential kernel of k1exp is identical to the Matérn kernel for $\nu = 1/2$, and that the three Matérns kernels provided here for $\nu = 1/2$, $\nu = 3/2$ and $\nu = 5/2$ are special cases of Continuous AutoRegressive (CAR) process covariances, with respective order 1, 2 and 3.

Value

An object with S4 class "covTS".

Caution

The $1d$ kernel $k$ as given in kernel is always assumed to have a variance parameter with name var. This assumption may be relaxed in future versions.

Note

Most arguments receive default values or are recycled if necessary.

Author(s)

Y. Deville, O. Roustant D. Ginsbourger

References


Examples

```r
myCov1 <- covTS(kernel = "k1exp", inputs = c("v1", "v2", "v3"),
                dep = c(range = "input"))
coef(myCov1) <- c(range = c(0.3, 0.7, 0.9), sigma2 = c(2, 2, 8))

myCov1
c coef(myCov1)
coef(myCov1, as = "matrix")
coef(myCov1, as = "list")
coef(myCov1, as = "matrix", type = "range")
```
# with a common range parameter
myCov2 <- covTS(kernel = "k1exp", inputs = c("v1", "v2", "v3"),
  dep = c(range = "cst"), value = c(range = 0.7),
  var = c(2, 2, 8))
myCov2

myCov3 <- covTS(d = 3, kernel = "k1powExp",
  dep = c(range = "cst", shape = "cst"),
  value = c(shape = 1.8, range = 1.1),
  var = c(2, 2, 8))
myCov3

covTS-class

Class "covTS"

Description

S4 class representing a Tensor Sum (TS) covariance kernel.

Objects from the Class

Objects can be created by call of the form new("covTS", ...) or by using the covTS function.

Slots

d: Object of class "integer", the spatial dimension or number of inputs of the covariance.
inputNames: Object of class "character", vector of input names. Length d.
kernel: Object of class "covMan" representing a 1d kernel.
kernParNames: Object of class "character", name of the kernel (among the allowed ones).
kernParCodes: Object of class "integer", an integer code stating the dependence of the parameter to the input.
par: Object of class "numeric", numeric vector of parameter values.
parN: Object of class "integer", total number of parameters.
parInput: Object of class "integer", the number of the inputs for each parameter. Same length as par, values between 1 and d.
parLower: .
parUpper: Object of class "numeric" numeric, vector of (possibly infinite) lower/upper bounds on parameters.
parBlock: Object of class "integer"
Methods

- **coef** signature(object = "covTS"): extracts the numeric values of the covariance parameters.
- **coef<-** signature(object = "covTS"): replaces the whole vector of coefficients, as required during ML estimation.
- **coefLower** signature(object = "covTS"): extracts the numeric values of the lower bounds.
- **coefLower<-** signature(object = "covTS"): replacement method for lower bounds on covTS coefficients.
- **coefUpper** signature(object = "covTS"): ...
- **coefUpper<-** signature(object = "covTS"): replacement method for upper bounds on covTS coefficients.
- **covMat** signature(object = "covTS"): builds the covariance matrix, or the cross covariance matrix between two sets of locations for a covTS object.
- **kernelName** signature(object = "covTS"): return the character value of the kernel name.
- **parMap** signature(object = "covTS"): an integer matrix used to map the covTS parameters on the inputs and kernel parameters during the computations.
- **scores** signature(object = "covTS"): computes the scores.
- **show** signature(object = "covTS"): prints in a custom format.
- **simulPar** signature(object = "covTS"): simulates random values for the covariance parameters.

Note

The names of the methods strive to respect a camelCase naming convention.

While the coef<- replacement method is typically intended for internal use during likelihood maximization, the coefLower<- and coefUpper<- replacement methods can be used when some rough information exists on the possible values of the parameters.

Author(s)

Y. Deville, O. Roustant, D. Ginsbourger.

See Also

The covTS function providing a creator.

Examples

showClass("covTS")
**gls**

*Generic function computing a Generalized Least Squares estimation with a given covariance kernel*

---

**Description**

Generic function computing a Generalized Least Squares estimation with a given covariance kernel.

**Usage**

\[ \text{gls}(\text{object}, \ldots) \]

**Arguments**

- **object**: An object representing a covariance kernel.
- **\ldots**: Other arguments for methods.

**Value**

A list with several elements corresponding to the estimation results.

---

**gls-methods**

*Generalized Least Squares estimation with a given covariance kernel*

---

**Description**

Generalized Least Squares (GLS) estimation for a linear model with a covariance given by the covariance kernel object. The method gives auxiliary variables as needed in many algebraic computations.

**Usage**

```r
## S4 method for signature 'covAll'
gls(object,
     y, X, F = NULL, varNoise = NULL,
     beta = NULL, checkNames = TRUE,
     \ldots)
```
Arguments

- **object**: An object with "covAll" class.
- **y**: The response vector with length \( n \).
- **X**: The input (or spatial design) matrix with \( n \) rows and \( d \) columns. This matrix must be compatible with the given covariance object, see `checkX`, `covAll`, `matrix-method`.
- **F**: A trend design matrix with \( n \) rows and \( p \) columns. When \( F \) is NULL no trend is used and the response \( y \) is simply a realization of a centered Gaussian Process with covariance kernel given by \( \text{object} \).
- **varNoise**: A known noise variance. When provided, must be a positive numeric value.
- **beta**: A known vector of trend parameters. Default is NULL indicating that the trend parameters must be estimated.
- **checkNames**: Logical. If TRUE (default), check the compatibility of \( X \) with \( \text{object} \), see `checkX`.
- **...**: not used yet.

Details

There are two options: for unknown trend, this is the usual GLS estimation with given covariance kernel; for a known trend, it returns the corresponding auxiliary variables (see `value` below).

Value

A list with several elements.

- **betaHat**: Vector \( \hat{\beta} \) of length \( p \) containing the estimated coefficients if \( \text{beta} = \text{NULL} \), or the known coefficients \( \beta \) either.
- **L**: The (lower) Cholesky root matrix \( L \) of the covariance matrix \( C \). This matrix has \( n \) rows and \( n \) columns and \( C = LL^T \).
- **eStar**: Vector of length \( n \): \( e^* = L^{-1}(y - X\hat{\beta}) \).
- **FStar**: Matrix \( n \times p \): \( F^* := L^{-1}F \).
- **sseStar**: Sum of squared errors: \( e^* \top e^* \).
- **RStar**: The 'R' upper triangular \( p \times p \) matrix in the QR decomposition of \( FStar \): \( F^* = QR^* \).

All objects having length \( p \) or having one of their dimension equal to \( p \) will be NULL when \( F \) is NULL, meaning that \( p = 0 \).

Author(s)

Y. Deville, O. Roustant

References

Kenneth Lange (2010), *Numerical Analysis for Statisticians* 2nd ed. pp. 102-103. Springer-Verlag,
## Examples

```r
## a possible 'covTS'
myCov <- covTS(inputs = c("Temp", "Humid"),
    kernel = "k1matern5_2",
    dep = c(range = "input"),
    value = c(range = 0.4))
d <- myCov@d; n <- 100;
X <- matrix(runif(n*d), nrow = n, ncol = d)
colnames(X) <- inputNames(myCov)
## generate the 'GP part'
C <- covMat(myCov, X = X)
L <- t(chol(C))
zeta <- L %*% rnorm(n)
## trend matrix 'F' for Ordinary Kriging
F <- matrix(1, nrow = n, ncol = 1)
varNoise <- 0.5
epsilon <- rnorm(n, sd = sqrt(varNoise))
beta <- 10
y <- F %*% beta + zeta + epsilon
fit <- gls(myCov, X = X, y = y, F = F, varNoise = varNoise)
```

## gp

### Gaussian Process model

#### Description

Gaussian Process model.

#### Usage

```r
gp(formula, data, inputs, cov, estim = TRUE, ...)
```

#### Arguments

- **formula**: A formula with a left-hand side specifying the response name, and the right-hand side the trend covariates (see examples below). Factors are not allowed neither as response nor as covariates.
- **data**: A data frame containing the response, the inputs specified in `inputs`, and all the trend variables required in `formula`.
- **inputs**: A character vector giving the names of the inputs.
- **cov**: A covariance kernel object or call.
- **estim**: Logical. If `TRUE`, the model parameters are estimated by Maximum Likelihood. The initial values can then be specified using the `parCovIni` and `varNoiseIni`
arguments of \texttt{mle, covAll-method} passed through dots. If \texttt{FALSE}, a simple Generalised Least Squares estimation will be used, see \texttt{gls, covAll-method}. Then the value of \texttt{varNoise} must be given and passed through dots in case noise is \texttt{TRUE}.

... Other arguments passed to the estimation method. This will be the \texttt{mle, covAll-method} if \texttt{estim} is \texttt{TRUE} or \texttt{gls, covAll-method} if \texttt{estim} is \texttt{FALSE}. In the first case, the arguments will typically include \texttt{varNoiseIni}. In the second case, they will typically include \texttt{varNoise}. Note that a logical \texttt{noise} can be used in the "mle" case. In both cases, the arguments \texttt{y}, \texttt{X}, \texttt{F} can not be used since they are automatically passed.

\textbf{Value}

A list object which is given the S3 class "gp". The list content is very likely to change, and should be used through methods.

\textbf{Note}

When \texttt{estim} is \texttt{TRUE}, the covariance object in \texttt{cov} is expected to provide a gradient when used to compute a covariance matrix, since the default value of \texttt{compGrad mle}.

\textbf{Author(s)}

Y. Deville, D. Ginsbourger, O. Roustant

\textbf{See Also}

\texttt{mle} for a detailed example.

\textbf{Examples}

```r
## Example 1. Data sampled from a GP model with a known covTS object

set.seed(1234)
myCov <- covTS(inputs = c("Temp", "Humid"),
               kernel = "k1matern5.2",
               dep = c(range = "input"),
               value = c(range = 0.4))

## change coefficients (variances)
coef(myCov) <- c(0.5, 0.8, 2, 16)
d <- myCov@d; n <- 20

## design matrix
X <- matrix(runif(n*d), nrow = n, ncol = d)
colnames(X) <- inputNames(myCov)

## generate the GP realization
C <- covMat(myCov, X = X)
L <- t(chol(C))
zeta <- L %*% rnorm(n)

## GP modelling with constant mean
F <- matrix(1, nrow = n, ncol = 1)
```
```r
varNoise <- 0.05
epsilon <- rnorm(n, sd = sqrt(varNoise))
beta <- 10
y <- F %*% beta + zeta + epsilon
## Towards prediction
fit <- gls(myCov, X = X, y = y, F = F, varNoise = varNoise)

## parIni: add noise to true parameters
parCovIni <- coef(myCov)
parCovIni[] <- 0.9 * parCovIni[] + 0.1 * runif(length(parCovIni))
coefLower(myCov) <- rep(1e-2, 4)
coefUpper(myCov) <- c(5, 5, 20, 20)
est <- gp(y ~ 1, data = data.frame(y = y, X),
  inputs = colnames(X), cov = myCov,
  noise = TRUE, varNoiseLower = 1e-2,
  parCovIni = parCovIni)
summary(est)
coef(est)
```

---

### Example 2. Predicting an additive function with an additive GP model

---

```r
## Not run:
addfun6d <- function(x){
  res <- x[1]^3 + cos(pi * x[2]) + abs(x[3]) * sin(x[3]^2) +
        3 * x[4]^3 + 3 * cos(pi * x[5]) + 3 * abs(x[6]) * sin(x[6]^2)
}

## 'Fit' is for the learning set, 'Val' for the validation set
set.seed(123)
nFit <- 50
nVal <- 200
d <- 6
inputs <- paste("x", 1L:d, sep = "")

## create design matrices with DiceDesign package
require(DiceDesign)
require(DiceKriging)
set.seed(0)
dataFitIni <- DiceDesign::lhsDesign(nFit, d)$design
dataValIni <- DiceDesign::lhsDesign(nVal, d)$design
dataFit <- DiceDesign::maximinSA_LHS(dataFitIni)$design
dataVal <- DiceDesign::maximinSA_LHS(dataValIni)$design

colnames(dataFit) <- colnames(dataVal) <- inputs
testfun <- addfun6d
dataFit <- data.frame(dataFit, y = apply(dataFit, 1, testfun))
dataVal <- data.frame(dataVal, y = apply(dataVal, 1, testfun))

## Creation of "CovTS" object with one range by input
myCov <- covTS(inputs = inputs, d = d, kernel = "k1matern3_2",
```
dep = c(range = "input")

## Creation of a gp object
fitgp <- gp(formula = y ~ 1, data = dataFit, inputs = inputs,
            cov = myCov, noise = TRUE,
            parCovIni = rep(1, 2*d),
            parCovLower = c(rep(1e-4, 2*d)),
            parCovUpper = c(rep(5, d), rep(10, d)),
            control = list(maxit = 300, REPORT = 10))

predTS <- predict(fitgp, newdata = as.matrix(dataVal[, inputs]), type = "UK")$mean

## Classical tensor product kernel as a reference for comparison
fitRef <- DiceKriging::km(formula = ~1,
                          design = dataFit[, inputs],
                          response = dataFit$y, covtype="matern3_2")

predRef <- predict(fitRef,
                   newdata = as.matrix(dataVal[, inputs]),
                   type = "UK")$mean

## Compare TS and Ref
RMSE <- data.frame(TS = sqrt(mean((dataVal$y - predTS)^2)),
                   Ref = sqrt(mean((dataVal$y - predRef)^2)),
                   row.names = "RMSE")
print(RMSE)

Comp <- data.frame(y = dataVal$y, predTS, predRef)
plot(predRef ~ y, data = Comp, col = "black", pch = 4,
     xlab = "True", ylab = "Predicted",
     main = paste("Prediction on a validation set (nFit = ",
                   nFit, ", nVal = ", nVal, ").", sep = ""))
points(predTS ~ y, data = Comp, col = "red", pch = 20)
abline(a = 0, b = 1, col = "blue", lty = "dotted")
legend("bottomright", pch = c(4, 20), col = c("black", "red"),
        legend = c("Ref", "Tensor Sum"))

## End(Not run)

## Example 3: a 'covMan' kernel with 3 implementations

### Define a 4-dimensional covariance structure with a kernel in R

myGaussFunR <- function(x1, x2, par) {
  h <- (x1 - x2) / par[1]
  SS2 <- sum(h^2)
  d2 <- exp(-SS2)
  kern <- par[2] * d2
  d1 <- 2 * kern * SS2 / par[1]
  attr(kern, "gradient") <- c(theta = d1, sigma2 = d2)
  return(kern)
myGaussR <- covMan(kernel = myGaussFunR,
    hasGrad = TRUE,
    d = d,
    parLower = c(theta = 0.0, sigma2 = 0.0),
    parUpper = c(theta = Inf, sigma2 = Inf),
    parNames = c("theta", "sigma2"),
    label = "Gaussian kernel: R implementation")

## -- The same, still in R, but with a kernel admitting matrices as arguments

myGaussFunRVec <- function(x1, x2, par) {
  # x1, x2 : matrices with same number of columns 'd' (dimension)
  n <- nrow(x1)
  d <- ncol(x1)
  SS2 <- 0
  for (j in 1:d){
    Aj <- outer(x1[, j], x2[, j], "+")
    Hj2 <- (Aj / par[1])^2
    SS2 <- SS2 + Hj2
  }
  D2 <- exp(-SS2)
  kern <- par[2] * D2
  D1 <- 2 * kern * SS2 / par[1]
  attr(kern, "gradient") <- list(theta = D1, sigma2 = D2)

  return(kern)
}

myGaussRVec <- covMan(
    kernel = myGaussFunRVec,
    hasGrad = TRUE,
    acceptMatrix = TRUE,
    d = d,
    parLower = c(theta = 0.0, sigma2 = 0.0),
    parUpper = c(theta = Inf, sigma2 = Inf),
    parNames = c("theta", "sigma2"),
    label = "Gaussian kernel: vectorised R implementation"
)

## -- The same, with inlined C code
## (see also another example with Rcpp by typing: ?kergp).

if (require(inline)) {
  kernCode <- "
    SEXP kern, dkern;
    int nprotect = 0, d;
    double SS2 = 0.0, d2, z, *rkern, *rdkern;

    d = LENGTH(x1);
    PROTECT(kern = allocVector(REALSXP, d)); nprotect++;
    
    ...
PROTECT(dkern = allocVector(REALSXP, 2)); nprotect++;  
rkern = REAL(kern);  
rdkern = REAL(dkern);  
for (int i = 0; i < d; i++) {  
  z = ( REAL(x1)[i] - REAL(x2)[i] ) / REAL(par)[0];  
  SS2 += z * z;  
}  
d2 = exp(-SS2);  
rkern[0] = REAL(par)[1] * d2;  
rdkern[1] = d2;  
rdkern[0] = 2 * rkern[0] * SS2 / REAL(par)[0];  
SET_ATTRIB(kern, install("gradient"), dkern);  
UNPROTECT(nprotect);  
return kern;  

text("myGaussFunC <- cfunction(sig = signature(x1 = "numeric", x2 = "numeric",  
par = "numeric"),  
body = kernCode)
myGaussC <- covMan(kernel = myGaussFunC,  
hasGrad = TRUE,  
d = d,  
parLower = c(theta = 0.0, sigma2 = 0.0),  
parUpper = c(theta = Inf, sigma2 = Inf),  
parNames = c("theta", "sigma2"),  
label = "Gaussian kernel: C/inline implementation")
}

## == Simulate data for covMan and trend ==

n <- 100; p <- d + 1  
X <- matrix(runif(n * d), nrow = n)  
colnames(X) <- inputNames(myGaussRVec)  
design <- data.frame(X)  
coef(myGaussRVec) <- myPar <- c(theta = 0.5, sigma2 = 2)  
C <- covMat(object = myGaussRVec, X = X,  
compGrad = FALSE, index = 1L, checkNames = FALSE)

L <- t(chol(C))  
y <- L %*% rnorm(n)  
F <- matrix(runif(n * p), nrow = n, ncol = p)  
beta <- (1:p) / p  
y <- tcrossprod(F, t(beta)) + y

## == ML estimation. ==

tRVec <- system.time(  
  resRVec <- gpr(formula = y ~ ., data = data.frame(y = y, design),  
  inputs = names(design), cov = myGaussRVec,  
  compGrad = TRUE,  
  t = t(1), ...)
influence.gp

Diagnostics for a Gaussian Process model, based on Leave-One-Out

Description

Cross Validation by leave-one-out for a gp object.

Usage

## S3 method for class 'gp'
influence(model, type = "UK", trend.reestim = TRUE, ...)

Arguments

model An object of class "gp".

type Character string corresponding to the GP "kriging" family, to be chosen between simple kriging ("SK"), or universal kriging ("UK").

trend.reestim Should the trend be re-estimated when removing an observation? Default to TRUE.

Details

Leave-one-out (LOO) consists in 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’s formula is also implemented; It is limited to 2 cases:

- (type == "SK") & !trend.reestim and
- (type == "UK") & trend.reestim.

Value

A list composed of the following elements, where \( n \) is the total number of observations.

mean Vector of length \( n \). The \( i \)-th element is the kriging mean (including the trend) at the \( i \)th observation number when removing it from the learning set.

sd Vector of length \( n \). The \( i \)-th element is the kriging standard deviation at the \( i \)-th observation number when removing it from the learning set.

Warning

Only trend parameters are re-estimated when removing one observation. When the number \( n \) of observations is small, the re-estimated values can be far away from those obtained with the entire learning set.

Author(s)

O. Roustant, D. Ginsbourger.

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 link


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


See Also

`predict_gp`, `plot_gp`

---

**inputNames**  Generic function returning the names of the inputs of a covariance kernel

**Description**

Generic function returning the names of the inputs attached with a covariance kernel.

**Usage**

`inputNames(object, ...)`

**Arguments**

- `object`  A covariance kernel object.
- `...`  Other arguments for methods.

**Value**

A character vector with distinct input names that are used e.g. in prediction.

**Note**

The input names are usually checked to control that they match the colnames of a spatial design matrix.

**See Also**

`checkX`
kernelName | Name of the 1d kernel in a composite kernel object

**Description**

Name of the 1d kernel in a composite kernel object.

**Usage**

`kernelName(object, ...)`

**Arguments**

- `object`: A covariance kernel.
- `...`: Arguments for methods.

**Value**

A character string giving the kernel name.

---

mle | Generic function for Maximum Likelihood estimation of a Gaussian Process model

**Description**

Generic function for the Maximum Likelihood estimation of a Gaussian Process model.

**Usage**

`mle(object, ...)`

**Arguments**

- `object`: An object representing a covariance kernel.
- `...`: Other arguments for methods, typically including a response, a design, ...

**Value**

An estimated model, typically a list.

**See Also**

`mle-methods` for examples.
Maximum Likelihood estimation of Gaussian Process model parameters

Description

Maximum Likelihood estimation of Gaussian Process models which covariance structure is described in a covariance kernel object.

Usage

```r
### S4 method for signature 'covAll'
mle(object,
    y, X, F = NULL, beta = NULL,
    parCovIni = coef(object),
    parCovLower = coefLower(object),
    parCovUpper = coefUpper(object),
    noise = TRUE, varNoiseIni = var(y)/10,
    varNoiseLower = 0, varNoiseUpper = Inf,
    parFixed = NULL,
    compGrad = TRUE,
    doOptim = TRUE,
    method = "L-BFGS-B",
    control = list(fnscale = -1, trace = 3, REPORT = 1),
    parTrack = FALSE, trace = 0, checkNames = TRUE)
```

Arguments

- `object` An object representing a covariance kernel.
- `y` Response vector.
- `X` Spatial (or input) design matrix.
- `F` Trend matrix.
- `beta` Vector of trend coefficients if known/fixed.
- `parCovIni` Vector with named elements giving the initial values for the parameters. See Details.
- `parCovLower` Lower bounds for the parameters.
- `parCovUpper` Upper bounds for the parameters.
- `noise` Logical. Should a noise be added to the error term?
- `varNoiseIni` Initial value for the noise variance.
- `varNoiseLower` Lower bound for the noise variance.
- `varNoiseUpper` Upper bound for the noise variance.
- `parFixed` Vector with named elements giving the fixed parameters (if any). Not implemented yet.
Logical: compute and use the analytical gradient in optim? This is only possible when object provides the analytical gradient.

dooptim Logical. If FALSE no optimization is done.

method To be passed to optim.

control To be passed to optim. The element fnscale = -1 states that a maximization is required and must not be changed.

parTrack If TRUE, the parameter vectors used during the optimization are tracked and returned as a matrix.

trace If TRUE, some elements are printed.

checkNames if TRUE (default), check the compatibility of X with object, see checkX.

Details

The vector with element with kernel parameters; parIni, parFixed, parLower, parUpper must have named elements, and have name corresponding to those in the covariance given in object. The control tasks are not yet completed.

Value

A list with elements hopefully having understandable names.

opt Optimization results if it was successful, or an error object otherwise.

coeff.kernel The vector of 'kernel' coefficients. This will include one or several variance parameters.

coeff.trend Estimate of the vector $\beta$ of the trend coefficients.

parTracked A matrix with rows giving the successive iterates during optimization if the parTrack argument was set to TRUE.

Note

Most checks concerning the parameter names, dimensions of provided objects, … are not implemented yet.

Author(s)

Y. Deville, O. Roustant

See Also

gp for various examples.
Examples

### Example. A 4-dimensional "covMan" kernel

```r
# Example: A 4-dimensional "covMan" kernel

d <- 4
myCovMan <-
covMan(
  kernel = function(x1, x2, par) {
    htilde <- (x1 - x2) / par[1]
    SS2 <- sum(htilde^2)
    d2 <- exp(-SS2)
    kern <- par[2] * d2
    d1 <- 2 * kern * SS2 / par[1]
    attr(kern, "gradient") <- c(theta = d1, sigma2 = d2)
    return(kern)
  },
  label = "myGauss",
  hasGrad = TRUE,
  d = 4,
  parLower = c(theta = 0.0, sigma2 = 0.0),
  parUpper = c(theta = +Inf, sigma2 = +Inf),
  parNames = c("theta", "sigma2"),
  par = c(NA, NA)
)

kernCode <- "
SEXP kern, dkern;
int nprotect = 0, d;
double SS2 = 0.0, d2, z, *rkern, *rdkern;

d = LENGTH(x1);
PROTECT(kern = allocVector(REALSXP, 1)); nprotect++;
PROTECT(dkern = allocVector(REALSXP, 2)); nprotect++;
rkern = REAL(kern);
rdkern = REAL(dkern);

for (int i = 0; i < d; i++) {
  z = (REAL(x1)[i] - REAL(x2)[i]) / REAL(par)[0];
  SS2 += z * z;
}

d2 = exp(-SS2);
rkern[0] = REAL(par)[1] * d2;
rdkern[1] = d2;
rkern[0] = 2 * rkern[0] * SS2 / REAL(par)[0];

SET_ATTRIB(kern, install("gradient"), dkern);
UNPROTECT(nprotect);
return kern;
"

### inline the C function into an R function: MUCH MORE EFFICIENT!!!

## Not run:
```
require(inline)
kernC <- cfunction(sig = signature(x1 = "numeric", x2 = "numeric",
    par = "numeric"),
    body = kernCode)
myCovMan <- covMan(kernel = kernC, hasGrad = TRUE, label = "myGauss", d = 4,
    parNames = c("theta", "sigma2"))

## End(Not run)

## Example (continued). Simulate data for covMan and trend
##
## Example (continued). ML estimation. Note the 'partrack' argument
##
est <- mle(object = myCovMan,
    parCovIni = parCovIni,
    y = y, X = X, F = F,
    parCovLower = c(0.05, 0.05), parCovUpper = c(10, 100),
    parTrack = TRUE, noise = FALSE, checkNames = FALSE)

## Example (continued). Grid for graphical analysis
##
## Not run:
theta.grid <- seq(from = 0.1, to = 0.7, by = 0.2)
sigma2.grid <- seq(from = 0.3, to = 6, by = 0.4)
par.grid <- expand.grid(theta = theta.grid, sigma2 = sigma2.grid)
ll <- apply(as.matrix(par.grid), 1, est$logLikFun)
llmat <- matrix(ll, nrow = length(theta.grid),
    ncol = length(sigma2.grid))

## End(Not run)
npar

Generic function returning the number of free parameters in a covariance kernel

Description

Generic function returning the number of free parameters in a covariance kernel.
Usage

npar(object, ...)

Arguments

object A covariance kernel object.
...
Other arguments for methods.

Value

The number of parameters.

---

npar-methods  

Number of parameters for a covariance kernel object

Description

Number of parameters for a covariance kernel object.

Usage

## S4 method for signature 'covMan'
npar(object, ...)

## S4 method for signature 'covTS'
npar(object, ...)

Arguments

object An object with S4 class corresponding to a covariance kernel.
...
Not used yet.

Value

The number of parameters.

See Also

coeff method
Description

Map the parameter of a composite covariance kernel on the inputs and the parameters of the 1d kernel.

Usage

parMap(object, ...)

Arguments

object A composite covariance kernel.
... Arguments for methods.

Value

A matrix with one row by input and one column for each of the parameters of the 1d kernel.

Description

Map the parameters of a structure on the inputs and kernel parameters.

Usage

## S4 method for signature 'covTS'
parMap(object, ...)

Arguments

object An object with class "covTS".
... Not used yet.

Value

A matrix with integer values. The rows correspond to the inputs of the object and the columns to the 1d kernel parameters. The matrix element is the number of the corresponding official coefficient. The same parameter of the structure can be used for several inputs but not (yet) for several kernel parameters. So each row must have different integer elements, while the same element can be repeated within a column.
Note

This function is for internal use only.

Examples

```r
myCov <- covTS(d = 3, kernel = "k1gauss",
               dep = c(range = "input"),
               value = c(range = 1.1))
parMap(myCov)
```

---

**plot.gp**  
*Diagnostic plot for the validation of a gp object*

**Description**

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

**Usage**

```r
## S3 method for class 'gp'
plot(x, y, kriging.type = "UK",
     trend.reestim = TRUE, which = 1:3, ...)
```

**Arguments**

- `x`: An object with S3 class "gp".
- `y`: Not used.
- `kriging.type`: Optional character string corresponding to the GP "kriging" family, to be chosen between simple kriging ("SK") or universal kriging ("UK").
- `trend.reestim`: Should the trend be re-estimated when removing an observation? Default to TRUE.
- `which`: A subset of 1, 2, 3 indicating which figures to plot (see Description above). Default is 1:3 (all figures).
- `...`: No other argument for this method.

**Details**

The standardized residuals are defined by \( \frac{y(x_i) - \hat{y}_{-i}(x_i)}{\hat{\sigma}_{-i}(x_i)} \), where \( y(x_i) \) is the response at the location \( x_i \), \( \hat{y}_{-i}(x_i) \) is the fitted value when the \( i \)-th observation is omitted (see `influence.gp`), and \( \hat{\sigma}_{-i}(x_i) \) is the corresponding kriging standard deviation.
Value

A list composed of the following elements where \( n \) is the total number of observations.

- **mean**: A vector of length \( n \). The \( i \)-th element is the kriging mean (including the trend) at the \( i \)-th observation number when removing it from the learning set.

- **sd**: A vector of length \( n \). The \( i \)-th element is the kriging standard deviation at the \( i \)-th observation number when removing it from the learning set.

Warning

Only trend parameters are re-estimated when removing one observation. When the number \( n \) of observations is small, re-estimated values can substantially differ from those obtained with the whole learning set.

References


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


See Also

- `predict.gp` and `influence.gp`, the predict and influence methods for "gp".

---

**Description**

Prediction method for the "gp" S3 class.
Usage

```r
## S3 method for class 'gp'
predict(object, newdata, type,
     seCompute = TRUE, covCompute = FALSE,
     lightReturn = FALSE, biasCorrect = FALSE,
     forceInterp,
     ...)
```

Arguments

- **object**: An object with S3 class "gp".
- **newdata**: A data frame containing all the variables required for prediction: inputs and trend variables, if applicable.
- **type**: A character string corresponding to the GP "kriging" family, to be chosen between simple kriging ("SK"), or universal kriging ("UK").
- **seCompute**: Optional logical. If FALSE, only the kriging mean is computed. If TRUE, the kriging variance (actually, the corresponding standard deviation) and prediction intervals are computed too.
- **covCompute**: Logical. If TRUE the covariance matrix is computed.
- **lightReturn**: Optional logical. If TRUE, c and cStar are not returned. This should be reserved to expert users who want to save memory and know that they will not miss these values.
- **biasCorrect**: Optional logical to correct bias in the UK variance and covariances. Default is FALSE. See Details below.
- **forceInterp**: Logical used to force a nugget-type prediction. If TRUE, the noise will be interpreted as a nugget effect. This argument is likely to be removed in the future.
- **...**: Not used yet.

Details

The estimated (UK) variance and covariances are NOT multiplied by \( \frac{n}{n-p} \) by default (\( n \) and \( p \) denoting the number of rows and columns of the trend 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 ignored in the context of computer experiments, especially in adaptive strategies. It can be activated by turning biasCorrect to TRUE, when type = "UK"

Value

A list with the following elements.

- **mean**: GP mean ("kriging") predictor (including the trend) computed at newdata.
- **sd**: GP prediction ("kriging") standard deviation computed at newdata. Not computed if seCompute is FALSE.
scores

sdSK Part of the above standard deviation corresponding to simple kriging (coincides with sd when type = "SK"). Not computed if seCompute is FALSE.
trend The computed trend function, evaluated at newdata.
cov GP prediction ("kriging") conditional covariance matrix. Not computed if covCompute is FALSE (default).
lower95, upper95 Bounds of the 95 % GP prediction interval computed at newdata (to be interpreted with special care when parameters are estimated, see description above). Not computed if seCompute is FALSE.
c An auxiliary matrix c, containing all the covariances between newdata and the initial design points. Not returned if lightReturn is TRUE.
cStar An auxiliary vector, equal to \(L^{-1}c\) where \(L\) is the Cholesky root of the covariance matrix \(C\) used in the estimation. Not returned if lightReturn is TRUE.

Author(s)
O. Roustant, D. Ginsbourger, Y. Deville

See Also
gp for the creation/estimation of a model. See gls-methods for the signification of the auxiliary variables.

Generic function returning the scores for a covariance kernel object

Description
Generic function returning the scores for a covariance kernel object.

Usage
scores(object, ...)

Arguments

object A covariance object.
... Other arguments passed to methods.

Details
Compute the derivatives \(\partial_{\theta_k} \ell\) for the (possibly concentrated) log-likelihood \(\ell := \log L\) of a covariance object with parameter vector \(\theta\). The score for \(\theta_k\) is obtained as a matrix scalar product
\[
\partial_{\theta_k} \ell = \text{trace}(WD)
\]
where \(D := \partial_{\theta_k} C\) and where \(W\) is the matrix \(W := ee^T - C^{-1}\). The vector \(e\) is the vector of residuals and the matrix \(C\) is the covariance computed for the design \(X\).
Value

A numeric vector of length npar(object) containing the scores.

Note

The scores can be efficiently computed when the matrix $W$ has already been pre-computed.
simulPar

Generic function to draw random values for the parameters of a covariance kernel

Description

Generic function to draw random values for the parameters of a covariance kernel object.

Usage

simulPar(object, nsim = 1L, seed = NULL, ...)

Arguments

- object: A covariance kernel.
- nsim: Number of drawings.
- seed: Seed for the random generator.
- ...: Other arguments for methods.

Details

Draw random values for the parameters of a covariance kernel object using the informations coefLower and coefUpper.

Value

A matrix with nsim rows and npar(object) columns.
Index

*Topic classes
  covAll-class, 9
  covMan-class, 13
  covTS-class, 19

*Topic methods
  plot.gp, 40

*Topic models
  influence.gp, 29
  plot.gp, 40

c. 17
checkX, 6, 16, 22, 31, 34
checkX, covAll, matrix-method
  (covAll-class), 9
checkX, covAll-method (checkX-methods), 6
checkX-methods, 6
c1f, 16, 38
c1f, covMan-method (c1f-methods), 7
c1f, covTS-method (c1f-methods), 7
c1f-methods, 7
c1f<-, 8
c1f<-, covMan, numeric-method
  (covMan-class), 13
c1f<-, covTS, numeric-method
  (covTS-class), 19
c1fLower, 9
c1fLower, covMan-method (covMan-class), 13
c1fLower, covTS-method (covTS-class), 19
c1fLower<-, (c1fLower), 9
c1fLower<-, covMan-method
  (covMan-class), 13
c1fLower<-, covTS-method (covTS-class), 19
c1fUpper (c1fLower), 9
c1fUpper, covMan-method (covMan-class), 13
c1fUpper, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
c1fUpper<-, covMan-method
  (covMan-class), 13
c1fUpper<-, covTS-method (covTS-class), 19
c1fUpper<- (c1fLower), 9
c1fUpper<- (c1fUpper), 9
INDEX

optim, 34
parMap, 39
parMap, covTS-method (parMap-methods), 39
parMap-methods, 39
plot.gp, 31, 40
predict.gp, 31, 41, 41
scores, 43
scores, covMan-method (covMan-class), 13
scores, covTS-method (covTS-class), 19
shapeSlot, 44
show, covMan-method (covMan-class), 13
show, covTS-method (covTS-class), 19
simulPar, 45
simulPar, covAll-method (covAll-class), 9