Package ‘kergp’

March 18, 2021

Type Package

Title Gaussian Process Laboratory

Version 0.5.5

Date 2021-03-17

Author Yves Deville, David Ginsbourger, Olivier Roustant. Contributors: Nicolas Durrande.

Maintainer Olivier Roustant `<roustant@insa-toulouse.fr>`

Description Gaussian process regression with an emphasis on kernels.
Quantitative and qualitative inputs are accepted. Some pre-defined
kernels are available, such as radial or tensor-sum for
quantitative inputs, and compound symmetry, low rank, group kernel
for qualitative inputs. The user can define new kernels and
composite kernels through a formula mechanism. Useful methods
include parameter estimation by maximum likelihood, simulation,
prediction and leave-one-out validation.

License GPL-3

Depends Rcpp (>= 0.10.5), methods, testthat, nloptr, lattice

Suggests DiceKriging, DiceDesign, inline, foreach, knitr, ggplot2,
reshape2, corrplot

Imports MASS, numDeriv, stats4, doParallel, doFuture, utils

LinkingTo Rcpp

RoxygenNote 6.0.1

Collate 'CovFormulas.R' 'allGenerics.R' 'checkGrad.R' 'covComp.R'
'covMan.R' 'covQual.R' 'q1CompSymm.R' 'q1Symm.R' 'q1LowRank.R'
'covQualNested.R' 'covQualOrd.R' 'covRadial.R' 'covTS.R'
'covTP.R' 'covANOVA.R' 'covZZAll.R' 'gp.R' 'kFuns.R'
'kernelNorm.R' 'kernels1d_Call.R' 'logLikFuns.R' 'methodGLS.R'
'methodMLE.R' 'miscUtils.R' 'prinKrige.R' 'q1Diag.R'
'simulate_gp.R' 'warpFuns.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-03-18 09:50:03 UTC

R topics documented:

kergp-package . . . 3
as.list, covTP-method . . . 7
checkGrad . . . 8
checkPar . . . 10
checkX . . . 11
checkX-methods . . . 11
coef-methods . . . 12
coef<- . . . 13
coefLower . . . 14
contr.helmod . . . 14
corLevCompSymm . . . 15
corLevDiag . . . 17
corLevLowRank . . . 18
corLevSymm . . . 19
covAll-class . . . 22
covANOVA . . . 23
covANOVA-class . . . 25
covComp . . . 27
covComp-class . . . 28
covMan . . . 30
covMan-class . . . 33
covMat . . . 35
covMat-methods . . . 35
covOrd . . . 37
covOrd-class . . . 39
covQual-class . . . 41
covQualNested . . . 43
covQualNested-class . . . 46
covRadial . . . 47
covRadial-class . . . 50
covTP . . . 52
covTP-class . . . 54
covTS . . . 55
covTS-class . . . 57
gls . . . 59
gls-methods . . . 60
gp . . . 61
hasGrad . . . 68
influence.gp . . . 68
inputNames . . . 70
k1Exp . . . 71
k1Matern3_2 . . . 72
kernelName . . . 73
kGauss . . . 74
kMatern . . . 75
mle . . . 76

mle-methods 76
npar 82
npar-methods 83
optimMethods 83
parMap 84
parMap-methods 85
parNamesSymm 86
parseCovFormula 86
plot 88
plot.gp 89
plot.simulate.gp 90
predict.gp 91
prinKrige 93
q1CompSymm 94
q1Diag 96
q1LowRank 97
q1Symm 98
scores 99
shapeSlot 100
simulate, covAll-method 101
simulate.gp 102
simulPar 105
simulPar,covAll-method 106
symIndices 106
translude 107
varVec 108
varVec-methods 108
warpNorm 110

Index **111**

`kergp-package` *Gaussian Process Laboratory*

Description

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

Details

Package:	kergp
Type:	Package
Title:	Gaussian Process Laboratory
Version:	0.5.5
Date:	2021-03-17
Author:	Yves Deville, David Ginsbourger, Olivier Roustant. Contributors: Nicolas Durrande.

Maintainer:	Olivier Roustant <roustant@insa-toulouse.fr>
Description:	Gaussian process regression with an emphasis on kernels. Quantitative and qualitative inputs are accepted. S
License:	GPL-3
Depends:	Rcpp (>= 0.10.5), methods, testthat, nloptr, lattice
Suggests:	DiceKriging, DiceDesign, inline, foreach, knitr, ggplot2, reshape2, corrplot
Imports:	MASS, numDeriv, stats4, doParallel, doFuture, utils
LinkingTo:	Rcpp
RoxygenNote:	6.0.1
Collate:	'CovFormulas.R' 'allGenerics.R' 'checkGrad.R' 'covComp.R' 'covMan.R' 'covQual.R' 'q1CompSymm.R'

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

Nicolas Durrande, David Ginsbourger, Olivier Roustant (2012). "Additive covariance kernels for high-dimensional gaussian process modeling". *Annales de la Faculté des Sciences de Toulouse*, 21 (3): 481-499. link

Nicolas Durrande, David Ginsbourger, Olivier Roustant, Laurent Carraro (2013). "ANOVA kernels and RKHS of zero mean functions for model-based sensitivity analysis". *Journal of Multivariate Analysis*, 115, 57-67. link

David Ginsbourger, Xavier Bay, Olivier Roustant, Laurent Carraro (2012). "Argumentwise invariant kernels for the approximation of invariant functions". *Annales de la Faculté des Sciences de Toulouse*, 21 (3): 501-527. link

David Ginsbourger, Nicolas Durrande, Olivier Roustant (2013). "Kernels and designs for modelling invariant functions: From group invariance to additivity". *mODa 10 - Advances in Model-Oriented Design and Analysis. Contributions to Statistics*, 107-115. link

Olivier Roustant, David Ginsbourger, 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. link

Examples

```
## ------------------------------------------------------------------
## 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)
  K <- par[2] * d2
  d1 <- 2 * K * S / par[1]
  attr(K, "gradient") <- c(theta = d1,  sigma2 = d2)
  return(K)
}

## ---------------------------------------------------------------
## quicker: with Rcpp; see also an example  with package inline
## in "gp" doc. file. Note that the Rcpp "sugar" fucntions are
## vectorized, so no for loops is required.
## ---------------------------------------------------------------

## Not run:

    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)
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) --
covSymGauss
symgp <- gp(formula = y ~ 1, data = data.frame(y, X),
            inputs = names(X),
            cov = covSymGauss,
            parCovIni = c(0.1, 2),
            varNoiseIni = 1.0e-8,
            varNoiseLower = 0.9e-8, varNoiseUpper = 1.1e-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.")

par(opar)
```

`as.list, covTP-method` *Coerce a* `covTP` *Object into a List*

Description

Coerce a `covTP` object representing a Tensor-Product covariance kernel on the d-dimensional Euclidean space into a list containing d one-dimensional kernels.

Usage

```
## S4 method for signature 'covTP'
as.list(x)
```

Arguments

`x`	A `covTP` object representing a Tensor-Product covariance kernel.

Value

A list with length `d` or `d + 1` where `d` is the "dimension" slot `x@d` of the object `x`. The first `d` elements of the list are one-dimensional *correlation* kernel objects with class `"covTP"`. When `x` is a *covariance* kernel (as opposed to a *correlation* kernel), the list contains one more element which gives the variance.

Caution

When x is not a correlation kernel the (d + 1)-th element of the returned list may be different in future versions: it may be a constant covariance kernel.

See Also

covTP and covTP-class.

Examples

```
set.seed(123)
d <- 6
myCov1 <- covTP(d = d, cov = "corr")
coef(myCov1) <- as.vector(simulPar(myCov1, nsim = 1))
as.list(myCov1)

## more examples and check the value of a 'covMat'
L <- list()
myCov <- list()

myCov[[1]] <- covTP(d = d, cov = "corr")
coef(myCov[[1]]) <- as.vector(simulPar(myCov[[1]], nsim = 1))
L[[1]] <- as.list(myCov[[1]])

myCov[[2]] <- covTP(k1Fun1 = k1Fun1PowExp, d = d, cov = "corr")
coef(myCov[[2]]) <- as.vector(simulPar(myCov[[2]], nsim = 1))
L[[2]] <- as.list(myCov[[2]])

myCov[[3]] <- covTP(k1Fun1 = k1Fun1PowExp, d = d, iso1 = 0L, cov = "corr")
coef(myCov[[3]]) <- as.vector(simulPar(myCov[[3]], nsim = 1))
L[[3]] <- as.list(myCov[[3]])

n <- 10
X <- matrix(runif(n * d), nrow = n,
            dimnames = list(NULL, paste("x", 1:d, sep = "")))
for (iTest in 1:3) {
   C <- covMat(L[[iTest]][[1]], X[ , 1, drop = FALSE])
   for (j in 2:d) {
      C <- C * covMat(L[[iTest]][[j]], X[ , j, drop = FALSE])
   }
   CTest <- covMat(myCov[[iTest]], X)
   print(max(abs(abs(C - CTest))))
}
```

---

checkGrad *Check the Gradient Provided in a* covMan *Object*

Description

Check the gradient provided in a covMan object.

Usage

```
checkGrad(object, sym = TRUE,
          x1 = NULL, n1 = 10,
          x2 = NULL, n2 = NULL,
          XLower = NULL, XUpper = NULL,
          plot = TRUE)
```

Arguments

object	A covMan object.
sym	Logical. If TRUE, the check is done assuming that x2 is identical to x1, so the provided values for x2 and n2 (if any) will be ignored.
x1	Matrix to be used as the first argument of the kernel.
n1	Number of rows for the matrix x1. Used only when x1 is not provided.
x2	Matrix to be used as the second argument of the kernel.
n2	Number of rows for the matrix x2. Used only when x2 is not provided.
XLower	Vector of lower bounds to draw x1 and x2 when needed.
XUpper	Vector of upper bounds to draw x1 and x2 when needed.
plot	Logical. If TRUE, a plot is shown comparing the two arrays of gradients.

Details

Each of the two matrices x1 and x2 with n1 and n2 rows can be given or instead be drawn at random. The matrix of kernel values with dimension c(n1,n2) is computed, together with its gradient with dimension c(n1,n2,npar) where npar is the number of parameters of the kernel. A numerical differentiation w.r.t. the kernel parameters is performed for the kernel value at x1 and x2, and the result is compared to that provided by the kernel function (the function described in the slot named "kernel" of object). Note that the value of the parameter vector is the value provided by coef(object) and it can be changed by using the replacement method `coef<-` if needed.

Value

A list of results related to the Jacobians

- test Max of the absolute difference between the gradient obtained by numeric differentiation and the gradient provided by the kernel object.
- Jnum, J Jacobians (arrays) computed with numDeriv::jacobian and provided by the kernel object.
- x1, x2, K The matrices used for the check, and the matrix of kernel values with dimension c(n1,n2). The element x2 can be NULL if the determination of the matrix x2 was not necessary.

Caution

For now the function only works when `object` has class `"covMan"`.

Note

As a rule of thumb, a gradient coded without error gives a value of `test` less than `1e-4`, and usually the value is much smaller than that.

Author(s)

Yves Deville

`checkPar` *Check Length and Names of a Vector of Values for Parameters or Bounds*

Description

Check length/names for a vector of values for parameters or bounds.

Usage

```
checkPar(value, parN, parNames, default)
```

Arguments

`value`	Numeric vector of values.
`parN`	Number of wanted values.
`parNames`	character. Names of the wanted values.
`default`	numeric. Default value.

Value

A numeric vector.

Examples

```
checkPar(value = c(1, 2), parN = 2L, parNames = c("theta", "sigma2"),
         default = 1.0)
checkPar(value = NULL, parN = 2L, parNames = c("theta", "sigma2"),
         default = 1.0)
checkPar(value = c("sigma2" = 100, "theta" = 1),
         parN = 2L, parNames = c("theta", "sigma2"),
         default = 1.0)
```

`checkX` *Generic function: 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 or data frame.
`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.
`...`	Not used yet.

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 input names 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 input names. If the names sets differ, an error occurs.

Value

A matrix with columns names identical to the input names 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.

`coef-methods` *Extract Coefficients of a Covariance Kernel Object as Vector, List or Matrix*

Description

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

Usage

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

## 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")
```

`coef<-` *Generic Function: Replacement of Coefficient Values*

Description

Generic function for the replacement of coefficient values.

Usage

```
`coef<-`(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

```
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.

`contr.helmod` *Modified Helmert Contrast Matrix*

Description

Modified Helmert contrast (or coding) matrix.

Usage

```
contr.helmod(n)
```

Arguments

`n`	Integer.

Details

The returned matrix is a scaled version of `contr.helmert(A)`.

Value

An orthogonal matrix with `n` rows and `n -1` columns. The columns form a basis of the subspace orthogonal to a vector of `n` ones.

Examples

```
A <- contr.helmod(6)
crossprod(A)
```

`corLevCompSymm` *Correlation Matrix for the Compound Symmetry Structure*

Description

Compute the correlation matrix for a the compound symmetry structure.

Usage

```
corLevCompSymm(par, nlevels, levels, lowerSQRT = FALSE, compGrad = TRUE,
  cov = FALSE, impl = c("C", "R"))
```

Arguments

`par`	Numeric vector of length `1` if `cov` is `TRUE` or with length `2` else. The first element is the correlation coefficient and the second one (when it exists) is the variance.
`nlevels`	Number of levels.
`levels`	Character representing the levels.
`lowerSQRT`	Logical. When `TRUE` the (lower) Cholesky root $\mathbf{L}$ of the correlation matrix $\mathbf{C}$ is returned instead of the correlation matrix.
`compGrad`	Logical. Should the gradient be computed?
`cov`	Logical. If `TRUE` the matrix is a covariance matrix (or its Cholesky root) rather than a correlation matrix and the last element in `par` is the variance.
`impl`	A character telling which of the C and R implementations should be chosen.

Value

A correlation matrix (or its Cholesky root) with the optional `gradient` attribute.

Note

When `lowerSQRT` is `FALSE`, the implementation used is always in R because no gain would then result from an implementation in C.

Author(s)

Yves Deville

Examples

```
checkGrad <- TRUE
lowerSQRT <- FALSE
nlevels <- 12
set.seed(1234)
par <- runif(1L, min = 0, max = pi)

##===========================================================================
## Compare R and C implementations for 'lowerSQRT = TRUE'
##===========================================================================
tR <- system.time(TR <- corLevCompSymm(nlevels = nlevels, par = par,
                                       lowerSQRT = lowerSQRT, impl = "R"))
tC <- system.time(T <- corLevCompSymm(nlevels = nlevels, par = par,
                                      lowerSQRT = lowerSQRT))
tC2 <- system.time(T2 <- corLevCompSymm(nlevels = nlevels, par = par,
                                        lowerSQRT = lowerSQRT, compGrad = FALSE))
## time
rbind(R = tR, C = tC, C2 = tC2)

## results
max(abs(T - TR))
max(abs(T2 - TR))

##==========================================================================
## Compare the gradients
##==========================================================================

if (checkGrad) {

    library(numDeriv)

    ##=======================
    ## lower SQRT case only
    ##========================
    JR <- jacobian(fun = corLevCompSymm, x = par, nlevels = nlevels,
                   lowerSQRT = lowerSQRT, impl = "R", method = "complex")
    J <- attr(T, "gradient")

    ## redim and compare.
    dim(JR) <- dim(J)
    max(abs(J - JR))
    nG <- length(JR)
    plot(1:nG, as.vector(JR), type = "p", pch = 21, col = "SpringGreen3",
         cex = 0.8, ylim = range(J, JR),
         main = paste("gradient check, lowerSQRT =", lowerSQRT))
    points(x = 1:nG, y = as.vector(J), pch = 16, cex = 0.6, col = "orangered")
}
```

---

corLevDiag *Correlation or Covariance Matrix for a Diagonal Structure*

Description

Compute the correlation or covariance matrix for a diagonal structure.

Usage

```
corLevDiag(par, nlevels, levels, lowerSQRT = FALSE, compGrad = TRUE,
  cov = 0)
```

Arguments

`par`	A numeric vector with length `npVar` where `npVar` is the number of variance parameters, namely `0`, `1` or `nlevels` corresponding to the values of `cov`: `0`, `1` and `2`.
`nlevels`	Number of levels.
`levels`	Character representing the levels.
`lowerSQRT`	Logical. When `TRUE` the (lower) Cholesky root $\mathbf{L}$ of the correlation or covariance matrix $\mathbf{C}$ is returned instead of the correlation matrix.
`compGrad`	Logical. Should the gradient be computed?
`cov`	Integer `0`, `1` or `2`. If `cov` is `0`, the matrix is a *correlation* matrix (or its Cholesky root) i.e. an identity matrix. If `cov` is `1` or `2`, the matrix is a *covariance* (or its square root) with constant variance vector for `code = 1` and with arbitrary variance vector for `code = 2`.

Value

A correlation matrix (or its Cholesky root) with the optional `gradient` attribute.

Examples

```
set.seed(123)
checkGrad <- TRUE
nlevels <- 12
sigma2 <- rexp(n = nlevels)
T0 <- corLevDiag(nlevels = nlevels, par = sigma2, cov = 2)
L0 <- corLevDiag(nlevels = nlevels, par = sigma2, cov = 2,
                 lowerSQRT = TRUE)
```

corLevLowRank *Correlation Matrix for a Low-Rank Structure*

Description

Compute the correlation matrix for a low-rank structure.

Usage

```
corLevLowRank(par, nlevels, rank, levels,
              lowerSQRT = FALSE, compGrad = TRUE,
              cov = 0, impl = c("C", "R"))
```

Arguments

`par`	A numeric vector with length `npCor + npVar` where `npCor = (rank -1) * (nlevels -rank / 2)` is the number of correlation parameters, and `npVar` is the number of variance parameters, which depends on the value of `cov`. The value of `npVar` is `0`, `1` or `nlevels` corresponding to the values of `cov`: `0`, `1` and `2`. The correlation parameters are assumed to be located at the head of `par` i.e. at indices `1` to `npCor`. The variance parameter(s) are assumed to be at the tail, i.e. at indices `npCor +1` to `npCor + npVar`.
`nlevels`	Number of levels m.
`rank`	The rank, which must be `>1` and `< nlevels`.
`levels`	Character representing the levels.
`lowerSQRT`	Logical. When `TRUE` a lower-triangular root **L** of the correlation or covariance matrix **C** is returned instead of the correlation matrix. Note that this matrix can have negative diagonal elements hence is not a (pivoted) Cholesky root.
`compGrad`	Logical. Should the gradient be computed? This is only possible for the C implementation.
`cov`	Integer `0`, `1` or `2`. If `cov` is `0`, the matrix is a *correlation* matrix (or its root). If `cov` is `1` or `2`, the matrix is a *covariance* (or its root) with constant variance vector for `code = 1` and with arbitrary variance for `code = 2`. The variance parameters `par` are located at the tail of the `par` vector, so at locations `npCor + 1` to `npCor + nlevels` when `code = 2` where `npCor` is the number of correlation parameters.
`impl`	A character telling which of the C and R implementations should be chosen. The R implementation is only for checks and should not be used.

Details

The correlation matrix with size m is the general symmetric correlation matrix with rank $\leq r$ where r is given, as described by Rapisarda et al. It depends on $(r-1) \times (m - r/2)/2$ parameters θ_{ij} where the indices i and j are such that $1 \leq j < i$ for $i \leq r$ or such that $1 \leq j < r$ for $r < i \leq n$. The parameters θ_{ij} are angles and are to be taken to be in $[0, 2\pi)$ if $j = 1$ and in $[0, \pi)$ otherwise.

Value

A correlation matrix (or its root) with the optional `gradient` attribute.

Note

This function is essentially for internal use and the corresponding correlation or covariance kernels are created as `covQual` objects by using the `q1LowRank` creator.

Here the parameters θ_{ij} are used *in row order* rather than in the column order. This order simplifies the computation of the gradient.

References

Francesco Rapisarda, Damanio Brigo, Fabio Mercurio (2007). "Parameterizing Correlations a Geometric Interpretation". *IMA Journal of Management Mathematics*, **18**(1): 55-73.

Igor Grubišić, Raoul Pietersz (2007). "Efficient Rank Reduction of Correlation Matrices". *Linear Algebra and its Applications*, **422**: 629-653.

See Also

The `q1LowRank` creator of a corresponding kernel object with class `"covQual"`, and the similar `corLevSymm` function for the full-rank case.

`corLevSymm` *Correlation Matrix for a General Symmetric Correlation Structure*

Description

Compute the correlation matrix for a general symmetric correlation structure.

Usage

```
corLevSymm(par, nlevels, levels, lowerSQRT = FALSE, compGrad = TRUE,
           cov = 0, impl = c("C", "R"))
```

Arguments

`par`	A numeric vector with length `npCor + npVar` where `npCor = nlevels * (nlevels -1) / 2` is the number of correlation parameters, and `npVar` is the number of variance parameters, which depends on the value of `cov`. The value of `npVar` is `0`, `1` or `nlevels` corresponding to the values of `cov`: `0`, `1` and `2`. The correlation parameters are assumed to be located at the head of `par` i.e. at indices `1` to `npCor`. The variance parameter(s) are assumed to be at the tail, i.e. at indices `npCor + 1` to `npCor + npVar`.
`nlevels`	Number of levels.

`levels`	Character representing the levels.
`lowerSQRT`	Logical. When `TRUE` the (lower) Cholesky root **L** of the correlation or covariance matrix **C** is returned instead of the correlation matrix.
`compGrad`	Logical. Should the gradient be computed? This is only possible for the C implementation.
`cov`	Integer `0`, `1` or `2`. If `cov` is `0`, the matrix is a *correlation* matrix (or its Cholesky root). If `cov` is `1` or `2`, the matrix is a *covariance* (or its Cholesky root) with constant variance vector for `code = 1` and with arbitrary variance for `code = 2`. The variance parameters `par` are located at the tail of the `par` vector, so at locations `npCor + 1` to `npCor + nlevels` when `code = 2` where `npCor` is the number of correlation parameters, i.e. `nlevels * (nlevels -1) / 2`.
`impl`	A character telling which of the C and R implementations should be chosen.

Details

The correlation matrix with dimension n is the *general symmetric correlation matrix* as described by Pinheiro and Bates and implemented in the **nlme** package. It depends on $n \times (n-1)/2$ parameters θ_{ij} where the indices i and j are such that $1 \leq j < i \leq n$. The parameters θ_{ij} are angles and are to be taken to be in $[0, \pi)$ for a one-to-one parameterisation.

Value

A correlation matrix (or its Cholesky root) with the optional `gradient` attribute.

Note

This function is essentially for internal use and the corresponding correlation or covariance kernels are created as `covQual` objects by using the `q1Symm` creator.

The parameters θ_{ij} are used *in row order* rather than in the column order as in the reference or in the **nlme** package. This order simplifies the computation of the gradients.

References

Jose C. Pinheiro and Douglas M. Bates (1996). "Unconstrained Parameterizations for Variance-Covariance matrices". *Statistics and Computing*, 6(3) pp. 289-296.

Jose C. Pinheiro and Douglas M. Bates (2000) *Mixed-Effects Models in S and S-PLUS*, Springer.

See Also

The `corSymm` correlation structure in the **nlme** package.

Examples

```
checkGrad <- TRUE
nlevels <- 12
npar <- nlevels * (nlevels - 1) / 2
par <- runif(npar, min = 0, max = pi)
##============================================================================
## Compare R and C implementations for 'lowerSQRT = TRUE'
##============================================================================
tR <- system.time(TR <- corLevSymm(nlevels = nlevels,
                                   par = par, lowerSQRT = TRUE, impl = "R"))
tC <- system.time(T <- corLevSymm(nlevels = nlevels, par = par,
                                  lowerSQRT = TRUE))
tC2 <- system.time(T2 <- corLevSymm(nlevels = nlevels, par = par,
                                    lowerSQRT = TRUE, compGrad = FALSE))
## time
rbind(R = tR, C = tC, C2 = tC2)

## results
max(abs(T - TR))
max(abs(T2 - TR))

##============================================================================
## Compare R and C implementations for 'lowerSQRT = FALSE'
##============================================================================
tR <- system.time(TRF <- corLevSymm(nlevels = nlevels, par = par,
                                    lowerSQRT = FALSE, impl = "R"))
tC <- system.time(TCF <- corLevSymm(nlevels = nlevels, par = par,
                                    compGrad = FALSE, lowerSQRT = FALSE))
tC2 <- system.time(TCF2 <- corLevSymm(nlevels = nlevels, par = par,
                                      compGrad = TRUE, lowerSQRT = FALSE))
rbind(R = tR, C = tC, C2 = tC2)
max(abs(TCF - TRF))
max(abs(TCF2 - TRF))

##===========================================================================
## Compare the gradients
##===========================================================================

if (checkGrad) {

    library(numDeriv)

    ##==================
    ## lower SQRT case
    ##==================
    JR <- jacobian(fun = corLevSymm, x = par, nlevels = nlevels,
                   lowerSQRT = TRUE, method = "complex", impl = "R")
    J <- attr(T, "gradient")

    ## redim and compare.
    dim(JR) <- dim(J)
    max(abs(J - JR))
    nG <- length(JR)
    plot(1:nG, as.vector(JR), type = "p", pch = 21, col = "SpringGreen3",
         cex = 0.8, ylim = range(J, JR),
         main = "gradient check, lowerSQRT = TRUE")
    points(x = 1:nG, y = as.vector(J), pch = 16, cex = 0.6, col = "orangered")

    ##==================
    ## Symmetric case
    ##==================
    JR <- jacobian(fun = corLevSymm, x = par, nlevels = nlevels,
                   lowerSQRT = FALSE, impl = "R", method = "complex")
    J <- attr(TCF2, "gradient")

    ## redim and compare.
    dim(JR) <- dim(J)
    max(abs(J - JR))
    nG <- length(JR)
    plot(1:nG, as.vector(JR), type = "p", pch = 21, col = "SpringGreen3",
         cex = 0.8,
         ylim = range(J, JR),
         main = "gradient check, lowerSQRT = FALSE")
    points(x = 1:nG, y = as.vector(J), pch = 16, cex = 0.6, col = "orangered")
}
```

---

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.

checkX signature(object = "covAll",X = "data.frame"): checks the compatibility of a design with a given covariance object.

inputNames signature(object = "covAll"): returns the character vector of input names.

hasGrad signature(object = "covAll"): returns the logical slot hasGrad.

simulPar signature(object = "covTS"): simulates random values for the parameters.

Examples

```
showClass("covAll")
```

covANOVA *Creator for the Class* "covANOVA"

Description

Creator for the class "covANOVA".

Usage

```
covANOVA(k1Fun1 = k1Fun1Gauss,
      cov = c("corr", "homo"),
      iso = 0, iso1 = 1L,
      hasGrad = TRUE,
      inputs = NULL,
      d = NULL,
      parNames,
      par = NULL, parLower = NULL, parUpper = NULL,
      label = "ANOVA kernel",
      ...)
```

Arguments

k1Fun1	A kernel function of a *scalar* numeric variable, and possibly of an extra "shape" parameter. This function can also return the first-order derivative or the two-first order derivatives as an attribute with name "der" and with a matrix content. When an extra shape parameter exists, the gradient can also be returned as an attribute with name "gradient", see **Examples** later. The name of the function can be given as a character string.
cov	A character string specifying the value of the variance parameter δ for the covariance kernel. Contrarily to other kernel classes, that parameter is not equal to the variance. Thus, mind that choosing ("corr") corresponds to $\delta = 1$ but *does not* correspond to a correlation kernel, see details below. Partial matching is allowed.
iso	Integer. The value 1L corresponds to an isotropic covariance, with all the inputs sharing the same range value.
iso1	Integer. This applies only when k1Fun1 contains one or more parameters that can be called 'shape' parameters. At now, only one such parameter can be found in k1Fun1 and consequently iso1 must be of length one. With iso1 = 0 the shape parameter in k1Fun1 will generate d parameters in the covANOVA object with their name suffixed by the dimension. When iso1 is 1 only one shape parameter will be created in the covANOVA object.
hasGrad	Integer or logical. Tells if the value returned by the function k1Fun1 has an attribute named "der" giving the derivative(s).
inputs	Character. Names of the inputs.
d	Integer. Number of inputs.

`parNames`	Names of the parameters. By default, ranges are prefixed `"theta_"` in the non-iso case and the range is named `"theta"` in the iso case.
`par`	Numeric values for the parameters. Can be `NA`.
`parLower`	Numeric values for the lower bounds on the parameters. Can be `-Inf`.
`parUpper`	Numeric values for the upper bounds on the parameters. Can be `Inf`.
`label`	A short description of the kernel object.
`...`	Other arguments passed to the method `new`.

Details

A ANOVA kernel on the d-dimensional Euclidean space takes the form

$$K(\mathbf{x}, \mathbf{x}') = \delta^2 \prod_{\ell=1}^{d} (1 + \tau_\ell^2 \kappa(r_\ell))$$

where $\kappa(r)$ is a suitable correlation kernel for a one-dimensional input, and r_ℓ is given by $r_\ell := [x_\ell - x'_\ell]/\theta_\ell$ for $\ell = 1$ to d.

In this default form, the ANOVA kernel depends on $2d + 1$ parameters: the *ranges* $\theta_\ell > 0$, the *variance ratios* τ_ℓ^2, and the variance parameter δ^2.

An *isotropic* form uses the same range θ for all inputs, i.e. sets $\theta_\ell = \theta$ for all ℓ. This is obtained by using `iso = TRUE`.

A *correlation* version uses $\delta^2 = 1$. This is obtained by using `cov = "corr"`. Mind that it does not correspond to a correlation kernel. Indeed, in general, the variance is equal to

$$K(\mathbf{x}, \mathbf{x}) = \delta^2 \prod_{\ell=1}^{d} (1 + \tau_\ell^2).$$

Finally, the correlation kernel $\kappa(r)$ can depend on a "shape" parameter, e.g. have the form $\kappa(r;\, \alpha)$. The extra shape parameter α will be considered then as a parameter of the resulting ANOVA kernel, making it possible to estimate it by ML along with the range(s) and the variance.

Value

An object with class `"covANOVA"`.

Examples

```
## Not run:
if (require(DiceKriging)) {
    ## a 16-points factorial design and the corresponding response
    d <- 2; n <- 16; x <- seq(from = 0.0, to = 1.0, length.out = 4)
    X <- expand.grid(x1 = x, x2 = x)
    y <- apply(X, 1, DiceKriging::branin)

    ## kriging model with matern5_2 covariance structure, constant
    ## trend. A crucial point is to set the upper bounds!
    mycov <- covANOVA(k1Fun1 = k1Fun1Matern5_2, d = 2, cov = "homo")
    coefUpper(mycov) <- c(2.0, 2.0, 5.0, 5.0, 1e10)
    mygp <- gp(y ~ 1, data = data.frame(X, y),
               cov = mycov, multistart = 100, noise = TRUE)

    nGrid <- 50; xGrid <- seq(from = 0, to = 1, length.out = nGrid)
    XGrid <- expand.grid(x1 = xGrid, x2 = xGrid)
    yGrid <- apply(XGrid, 1, DiceKriging::branin)
    pgp <- predict(mygp, XGrid)$mean

    mykm <- km(design = X, response = y)
    pkm <- predict(mykm, XGrid, "UK")$mean
    c("km" = sqrt(mean((yGrid - pkm)^2)),
      "gp" = sqrt(mean((yGrid - pgp)^2)))

}

## End(Not run)
```

covANOVA-class *Class* "covANOVA"

Description

S4 class representing a Tensor Product (ANOVA) covariance kernel.

Objects from the Class

Objects can be created by calls of the form `new("covANOVA",...)` or by using the `covANOVA` function.

Slots

`k1Fun1`**:** Object of class `"function"` A function of a scalar numeric variable.

`k1Fun1Char`**:** Object of class `"character"` describing the function in the slot `k1Fun1`.

`hasGrad`**:** Object of class `"logical"`. Tells if the value returned by the function `kern1Fun` has an attribute named `"der"` giving the derivative(s).

`cov`**:** Object of class `"integer"`. The value `1L` corresponds to a general covariance kernel. The value of `0L` sets the variance parameter to `1`, which does *not* correspond to a correlation kernel. See Section 'details' of `covANOVA`.

`iso`**:** Object of class `"integer"`. The value `1L` corresponds to an isotropic covariance, with all the inputs sharing the same range value.

`iso1`**:** Object of class `"integer"` used only when the function in the slot `k1Fun1` depends on parameters i.e. has more than one formal argument. NOT IMPLEMENTED YET.

`label`**:** Object of class `"character"`. Short description of the object.

`d`**:** Object of class `"integer"`. Dimension, i.e. number of inputs.

`inputNames`**:** Object of class `"optCharacter"`. Names of the inputs.

`parLower`**:** Object of class `"numeric"`. Numeric values for the lower bounds on the parameters. Can be `-Inf`.

`parUpper`**:** Object of class `"numeric"`. Numeric values for the upper bounds on the parameters. Can be `Inf`.

`par`**:** Object of class `"numeric"`. Numeric values for the parameters. Can be `NA`.

`kern1ParN1`**:** Object of class `"integer"`. The number of parameters in `k1Fun1` (such as a shape).

`parN1`**:** Object of class `"integer"`. Number of parameters of the function `kern1Fun` (such as a shape).

`parN`**:** Object of class `"integer"`. Number of parameters for the object. The include: *direct* parameters in the function `kern1Fun`, ranges, and variance.

`kern1ParNames`**:** Object of class `"character"`. Names of the *direct* parameters.

`kernParNames`**:** Object of class `"character"`. Names of the parameters.

Extends

Class `"covAll"`, directly.

Methods

coef `signature(object = "covANOVA")`: Get the vector of values for the parameters.

coef<- `signature(object = "covANOVA",value = "numeric")`: Set the vector of values for the parameters.

coefLower `signature(object = "covANOVA")`: Get the vector of lower bounds on the parameters.

coefLower<- `signature(object = "covANOVA")`: Set the vector of lower bounds on the parameters.

coefUpper `signature(object = "covANOVA")`: Get the vector of upper bounds on the parameters.

coefUpper<- `signature(object = "covANOVA")`: Set the vector of upper bounds on the parameters.

covMat `signature(object = "covANOVA")`: Compute the covariance matrix for given sites.

npar `signature(object = "covANOVA")`: Get the number of parameters.

scores `signature(object = "covANOVA")`: Compute the scores i.e. the derivatives w.r.t. the parameters of the contribution of the covariance in the log-likelihood of a gp.

show `signature(object = "covANOVA")`: Print or show the object.

varVec `signature(object = "covANOVA")`: Compute the variance vector for given sites.

See Also

`covTP`.

Examples

```
showClass("covANOVA")
```

`covComp` *Creator for the Class* "`covComp`" *for Composite Covariance Kernels*

Description

Creator for the class "covComp" for Composite Covariance kernels.

Usage

```
covComp(formula, where = .GlobalEnv, topParLower = NULL,
  topParUpper = NULL, trace = 0, ...)
```

Arguments

`formula`	A formula. See **Examples**.
`where`	An environment where the covariance kernels objects and top parameters will be looked for.
`topParLower`	A numeric vector of lower bounds for the "top" parameters.
`topParUpper`	A numeric vector of upper bounds for the "top" parameters.
`trace`	Integer level of verbosity.
`...`	Not used yet. For passing other slot values.

Details

A covariance object is built using `formula` which involves kernel objects inheriting from the class "`covAll`" and possibly of other scalar numeric parameters called *top* parameters. The formula can be thought of as involving the covariance matrices rather than the kernel objects, each kernel object say `obj` being replaced by `covMat(obj,X)` for some design matrix or data frame `X`. Indeed, the sum or the product of two kernel objects lead to a covariance which is simply the sum or product of the kernel covariances. The top parameters are considered as parameters of the covariance structure, as well as the parameters of the covariance objects used in the formula. Their value at the creation time will be used and thus will serve as initial value in estimation.

Value

An object with S4 class "`covComp`".

Caution

The class definition and its creator are to regarded as a DRAFT, many changes being necessary until a stable implementation will be reached. The functions relating to this class are not for final users of GP models, but rather to those interested in the conception and specification in view of a future release of the **kergp** package.

Examples

```
## ==========================================================================
## build some kernels (with their inputNames) in the global environment
## ==========================================================================

myCovExp3 <- kMatern(d = 3, nu = "1/2")
inputNames(myCovExp3) <- c("x", "y", "z")

myCovGauss2 <- kGauss(d = 2)
inputNames(myCovGauss2) <- c("temp1", "temp2")

k <- kMatern(d = 1)
inputNames(k) <- "x"

ell <- kMatern(d = 1)
inputNames(ell) <- "y"

tau2 <- 100
sigma2 <- 4

myCovComp <- covComp(formula = ~ tau2 * myCovGauss2() * myCovExp3() + sigma2 * k())

myCovComp1 <- covComp(formula = ~ myCovGauss2() * myCovExp3() + k())

inputNames(myCovComp)
coef(myCovComp)

n <- 5
set.seed(1234)
X <- data.frame(x = runif(n), y = runif(n), z = runif(n),
                temp1 = runif(n), temp2 = runif(n))

C <- covMat(myCovComp, X = X)

Cg <- covMat(myCovComp, X = X, compGrad = TRUE)

## Simulation: purely formal example, not meaningful.

Y <- simulate(myCovComp, X = X, nsim = 100)
```

---

covComp-class *Class* "covComp"

Description

Class "covComp" representing a composite kernel combining several kernels objects inheriting from the class "covAll".

Objects from the Class

Objects can be created by calls of the form `new("covComp",...)` or by using `covComp`.

Slots

`def`: Object of class `"expression"` defining the This is a parsed and cleaned version of the value of the `formula` formal in `covComp`.

`covAlls`: Object of class `"list"` containing the kernel objects used by the formula. The coefficients of these kernels can be changed.

`hasGrad`: Object of class `"logical"`: can we differentiate the kernel w.r.t. all its parameters?

`label`: Object of class `"character"` A label attached to the kernel to describe it.

`d`: Object of class `"integer"`: dimension (or number of inputs).

`parN`: Object of class `"integer"`: number of parameters.

`parNames`: Object of class `"character"`: vector of parameter names. Its length is in slot `parN`.

`inputNames`: Object of class `"character"`: names of the inputs used by the kernel.

`topParN`: Object of class `"integer"`: number of *top* parameters.

`topParNames`: Object of class `"character"`. Names of the top parameters.

`topPar`: Object of class `"numeric"`. Values of the top parameters.

`topParLower`: Object of class `"numeric"`. Lower bounds for the top parameters.

`topParUpper`: Object of class `"numeric"`. Upper bounds for the top parameters.

`parsedFormula`: Object of class `"list"`. Ugly draft for some slots to be added in the next versions.

Extends

Class `"covAll"`, directly.

Methods

as.list `signature(object = "covComp")`: coerce `object` into a list of covariance kernels, each inheriting from the virual class `"covAll"`. This is useful e.g., to extract the coefficients or to plot a covariance component.

checkX `signature(object = "covComp",X = "data.frame")`: check that the inputs exist with suitable column names and suitable *factor* content. The levels should match the prescribed levels. Returns a matrix with the input columns in the order prescribed by `object`.

coef, coef<- `signature(object = "covComp")`: extract or replace the vector of coefficients.

coefLower, coefUpper `signature(object = "covComp")`: extract the vector of Lower or Upper bounds on the coefficients.

scores `signature(object = "covComp")`: return the vector of scores, i.e. the derivative of the log-likelihood w.r.t. the parameter vector at the current parameter values.

See Also

The `covComp` creator.

Examples

```
showClass("covComp")
```

---

covMan *Creator Function for* `covMan` *Objects*

Description

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

Usage

```
    covMan(kernel, hasGrad = FALSE, acceptMatrix = FALSE,
           inputs = paste("x", 1:d, sep = ""),
           d = length(inputs), parNames,
           par = NULL, parLower = NULL, parUpper = 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 $\boldsymbol{\theta}$ 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, parLower, parUpper`	Optional numeric vectors containing the parameter values, lower bounds and upper bounds.
`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`.

- When `acceptMatrix` is `FALSE`, the formal arguments `x1` and `x2` 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 $K(\mathbf{x}_1,\, \mathbf{x}_2;\, \boldsymbol{\theta})$ with respect to the parameter vector $\boldsymbol{\theta}$.
- 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 ℓ 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 θ_ℓ.

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

```
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(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;
       rdkern[0] =  2 * rkern[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,
    label = "myGauss",
    parLower = c(theta1 = 0.0, theta2 = 0.0, sigma2 = 0.0),
    parUpper = c(theta1 = Inf, theta2 = Inf, sigma2 = Inf),
    parNames = c("theta1", "theta2", "sigma2"),
    par = c(NA, NA, NA)

)

coef(myCov) <- c(0.5, 1, 4)
show(myCov)

## computing the covariance kernel between two points
X <- matrix(c(0, 0), ncol = 2)
Xnew <- matrix(c(0.5, 1), ncol = 2)
colnames(X) <- colnames(Xnew) <- inputNames(myCov)
covMat(myCov, X)            ## same points
covMat(myCov, X, Xnew)      ## two different points

# computing covariances between sets of given locations
X <- matrix(c(0, 0.5, 0.7, 0, 0.5, 1), ncol = 2)
t <- seq(0, 1, length.out = 3)
Xnew <- as.matrix(expand.grid(t, t))
colnames(X) <- colnames(Xnew) <- inputNames(myCov)
covMat(myCov, X)         ## covariance matrix
covMat(myCov, X, Xnew)   ## covariances between design and new data
```

covMan-class *Class* "covMan"

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: Covariance or 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 $K(\mathbf{x}_1, \mathbf{x}_2)$ for all the couples of sites $\mathbf{x}_1$ and $\mathbf{x}_2$.

covMat-methods — *Covariance Matrix for a Covariance Kernel Object*

Description

Covariance matrix for a covariance kernel object.

Usage

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

## 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 matrix (or data.frame) of 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: `Xnew = 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. Ignored if `compGrad = FALSE`.
`...`	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 $C_{ij} := K(\mathbf{x}_{1,i},\, \mathbf{x}_{2,j};\, \boldsymbol{\theta})$ where $K(\mathbf{x}_1,\, \mathbf{x}_2;\, \boldsymbol{\theta})$ is the covariance kernel function.

Note

The value of the parameter $\boldsymbol{\theta}$ can be extracted from the object with the `coef` method.

Author(s)

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

See Also

`coef` method

Examples

```
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)
colnames(Xnew) <- inputNames(myCov)
C2 <- covMat(myCov, X, Xnew)

## check with the same matrix in 'X' and 'Xnew'
CMM <- covMat(myCov, X, X)
CM <- covMat(myCov, X)
max(abs(CM - CMM))
```

---

covOrd *Warping-Based Covariance for an Ordinal Input*

Description

Creator function for the class `covOrd-class`

Usage

```
covOrd(ordered,
       k1Fun1 = k1Fun1Matern5_2,
       warpFun = c("norm", "unorm", "power", "spline1", "spline2"),
       cov = c("corr", "homo"),
       hasGrad = TRUE, inputs = "u",
       par = NULL, parLower = NULL, parUpper = NULL,
       warpKnots = NULL, nWarpKnots = NULL,
       label = "Ordinal kernel",
       intAsChar = TRUE,
       ...)
```

Arguments

`ordered`	An object coerced to `ordered` representing an ordinal input. Only the levels and their order will be used.
`k1Fun1`	A function representing a 1-dimensional stationary kernel function, with no or fixed parameters.
`warpFun`	Character corresponding to an increasing warping function. See `warpFun`.
`cov`	Character indicating whether a correlation or homoscedastic kernel is used.
`hasGrad`	Object of class `"logical"`. If `TRUE`, both `k1Fun1` and `warpFun` must return the gradient as an attribute of the result.
`inputs`	Character: name of the ordinal input.
`par, parLower, parUpper`	Numeric vectors containing covariance parameter values/bounds in the following order: warping, range and variance if required (`cov == "homo"`).
`warpKnots`	Numeric vector containing the knots used when a spline warping is chosen. The knots must be in [0, 1], and 0 and 1 are automatically added if not provided. The number of knots cannot be greater than the number of levels.

`nWarpKnots`	Number of knots when a spline warping is used. Ignored if `warpKnots` is given. `nWarpKnots` cannot be greater than the number of levels.
`label`	Character giving a brief description of the kernel.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.
`...`	Not used at this stage.

Details

Covariance kernel for qualitative ordered inputs obtained by warping.

Let u be an ordered factor with levels $u_1, \dots, u_L$. Let k_1 be a 1-dimensional stationary kernel (with no or fixed parameters), F a warping function i.e. an increasing function on the interval $[0, 1]$ and θ a scale parameter. Then k is defined by:

$$k(u_i, u_j) = k_1([F(z_i) - F(z_j)]/\theta)$$

where $z_1, \dots, z_L$ form a regular sequence from 0 to 1 (included). At this stage, the possible choices are:

- A distribution function (cdf) truncated to $[0, 1]$, among the Power and Normal cdfs.
- For the Normal distribution, an unnormalized version, corresponding to the restriction of the cdf on $[0, 1]$, is also implemented (`warp = "unorm"`).
- An increasing spline of degree 1 (piecewise linear function) or 2. In this case, F is unnormalized. For degree 2, the implementation depends on scaling functions from DiceKriging package, which must be loaded here.

Notice that for unnormalized `F`, we set θ to 1, in order to avoid overparameterization.

Value

An object of class `covOrd-class`, inheriting from `covQual-class`.

See Also

`covOrd-class`, `warpFun`

Examples

```
u <- ordered(1:6, labels = letters[1:6])

myCov <- covOrd(ordered = u, cov = "homo", intAsChar = FALSE)
myCov
coef(myCov) <- c(mean = 0.5, sd = 1, theta = 3, sigma2 = 2)
myCov

checkX(myCov, X = data.frame(u = c(1L, 3L)))
covMat(myCov, X = data.frame(u = c(1L, 3L)))

myCov2 <- covOrd(ordered = u, k1Fun1 = k1Fun1Cos, warpFun = "power")
coef(myCov2) <- c(pow = 1, theta = 1)
```

```
myCov2

plot(myCov2, type = "cor", method = "ellipse")
plot(myCov2, type = "warp", col = "blue", lwd = 2)

myCov3 <- covOrd(ordered = u, k1Fun1 = k1Fun1Cos, warpFun = "spline1")
coef(myCov3) <- c(rep(0.5, 2), 2, rep(0.5, 2))
myCov3

plot(myCov3, type = "cor", method = "ellipse")
plot(myCov3, type = "warp", col = "blue", lwd = 2)


str(warpPower)  # details on the list describing the Power cdf
str(warpNorm)   # details on the list describing the Normal cdf
```

`covOrd-class` *Class* `"covOrd"`

Description

Covariance kernel for qualitative ordered inputs obtained by warping.

Let u be an ordered factor with levels $u_1, \dots, u_L$. Let k_1 be a 1-dimensional stationary kernel (with no or fixed parameters), F a warping function i.e. an increasing function on the interval $[0, 1]$ and θ a scale parameter. Then k is defined by:

$$k(u_i, u_j) = k_1([F(z_i) - F(z_j)]/\theta)$$

where $z_1, \dots, z_L$ form a regular sequence from 0 to 1 (included). Notice that an example of warping is a distribution function (cdf) restricted to $[0, 1]$.

Objects from the Class

Objects can be created by calls of the form `new("covOrd",...)`.

Slots

`covLevels`: Same as for `covQual-class`.

`covLevMat`: Same as for `covQual-class`.

`hasGrad`: Same as for `covQual-class`.

`acceptLowerSQRT`: Same as for `covQual-class`.

`label`: Same as for `covQual-class`.

`d`: Same as for `covQual-class`. Here equal to 1.

`inputNames`: Same as for `covQual-class`.

`nlevels`: Same as for `covQual-class`.

`levels`: Same as for `covQual-class`.

`parLower`**:** Same as for `covQual-class`.

`parUpper`**:** Same as for `covQual-class`.

`par`**:** Same as for `covQual-class`.

`parN`**:** Same as for `covQual-class`.

`kernParNames`**:** Same as for `covQual-class`.

`k1Fun1`**:** A function representing a 1-dimensional stationary kernel function, with no or fixed parameters.

`warpFun`**:** A cumulative density function representing a warping.

`cov`**:** Object of class `"integer"`. The value `0L` corresponds to a correlation kernel while `1L` is for a covariance kernel.

`parNk1`**:** Object of class `"integer"`. Number of parameters of `k1Fun1`. Equal to `0` at this stage.

`parNwarp`**:** Object of class `"integer"`. Number of parameters of `warpFun`.

`k1ParNames`**:** Object of class `"character"`. Parameter names of `k1Fun1`.

`warpParNames`**:** Object of class `"character"`. Parameter names of `warpFun`.

`warpKnots`**:** Object of class `"numeric"`. Parameters of `warpFun`.

`ordered`**:** Object of class `"logical"`. `TRUE` for an ordinal input.

`intAsChar`**:** Object of class `"logical"`. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Methods

checkX `signature(object = "covOrd", X = "data.frame")`: check that the inputs exist with suitable column names and suitable *factor* content. The levels should match the prescribed levels. Returns a matrix with the input columns in the order prescribed by `object`.

`signature(object = "covOrd", X = "matrix")`: check that the inputs exist with suitable column names and suitable *numeric* content for coercion into a factor with the prescribed levels. Returns a data frame with the input columns in the order prescribed by `object`.

coef<- `signature(object = "covOrd")`: replace the whole vector of coefficients, as required during ML estimation.

coefLower<- `signature(object = "covOrd")`: replacement method for lower bounds on covOrd coefficients.

coefLower `signature(object = "covOrd")`: extracts the numeric values of the lower bounds.

coef `signature(object = "covOrd")`: extracts the numeric values of the covariance parameters.

coefUpper<- `signature(object = "covOrd")`: replacement method for upper bounds on `covOrd` coefficients.

coefUpper `signature(object = "covOrd")`: ...

covMat `signature(object = "covOrd")`: build the covariance matrix or the cross covariance matrix between two sets of locations for a `covOrd` object.

npar `signature(object = "covOrd")`: returns the number of parameters.

scores `signature(object = "covOrd")`: return the vector of scores, i.e. the derivative of the log-likelihood w.r.t. the parameter vector at the current parameter values.

simulate `signature(object = "covOrd")`: simulate `nsim` paths from a Gaussian Process having the covariance structure. The paths are indexed by the finite set of levels of factor inputs, and they are returned as columns of a matrix.

varVec `signature(object = "covOrd")`: build the variance vector corresponding to a set locations for a `covOrd` object.

Note

This class is to be regarded as experimental. The slot names or list may be changed in the future. The methods `npar`, `inputNames` or `` `inputNames<-` `` should provide a more robust access to some slot values.

See Also

See `covMan` for a comparable structure dedicated to kernels with continuous inputs.

Examples

```
showClass("covOrd")
```

`covQual-class` *Class* `"covQual"`

Description

Covariance kernel for qualitative inputs.

Objects from the Class

Objects can be created by calls of the form `new("covQual",...)`.

Slots

`covLevels`**:** Object of class `"function"`. This function has arguments `'par'` and optional arguments `lowerSQRT` and `compGrad`. It returns the covariance matrix for an input corresponding to all the levels.

`covLevMat`**:** Object of class `"matrix"`. This is the result returned by the function `covLevels` (former slot) with `lowerSQRT = FALSE` and `gradient = FALSE`.

`hasGrad`**:** Object of class `"logical"`. When `TRUE`, the covariance matrix returned by the function in slot `covLevels` must compute the gradients. The returned covariance matrix must have a `"gradient"` attribute; this must be an array with dimension `c(m,m,np)` where `m` stands for the number of levels and np is the number of parameters.

`acceptLowerSQRT`**:** Object of class `"logical"`. When `TRUE`, the function in slot `covLevels` must have a formal `lowerSQRT` which can receive a logical value. When the value is `TRUE` the Cholesky (lower) root of the covariance is returned instead of the covariance.

`label`**:** Object of class `"character"`. A description of the kernel which will remained attached with it.

`d`**:** Object of class `"integer"`. The dimension or number of (qualitative) inputs of the kernel.

`inputNames`**:** Object of class `"character"`. The names of the (qualitative) inputs. These will be matched against the columns of a data frame when the kernel will be evaluated.

`nlevels`**:** Object of class `"integer"`. A vector with length `d` giving the number of levels for each of the `d` inputs.

`levels`**:** Object of class `"list"`. A list of length `d` containing the `d` character vectors of levels for the `d` (qualitative) inputs.

`parLower`**:** Object of class `"numeric"`. Vector of `parN` lower values for the parameters of the structure. The value `-Inf` can be used when needed.

`parUpper`**:** Object of class `"numeric"`. Vector of `parN` upper values for the parameters of the structure. The value `Inf` can be used when needed.

`par`**:** Object of class `"numeric"`. Vector of `parN` current values for the structure.

`parN`**:** Object of class `"integer"`. Number of parameters for the structure, as returned by the `npar` method.

`kernParNames`**:** Object of class `"character"`. Vector of length `parN` giving the names of the parameters. E.g. `"range"`, `"var"`, `"sigma2"` are popular names.

`ordered`**:** Vector of class `"logical"` indicating whether the factors are ordered or not.

`intAsChar`**:** Object of class `"logical"` indicating how to cope with an integer input. When `intAsChar` is `TRUE` the input is coerced into a character; the values taken by this character vector should then match the levels in the `covQual` object as given by `levels(object)[[1]]`. If instead `intAsChar` is `FALSE`, the integer values are assumed to correspond to the levels of the `covQual` object in the same order.

Methods

checkX `signature(object = "covQual",X = "data.frame")`: check that the inputs exist with suitable column names and suitable *factor* content. The levels should match the prescribed levels. Returns a matrix with the input columns in the order prescribed by `object`.

`signature(object = "covQual",X = "matrix")`: check that the inputs exist with suitable column names and suitable *numeric* content for coercion into a factor with the prescribed levels. Returns a data frame with the input columns in the order prescribed by `object`.

coef<- `signature(object = "covQual")`: replace the whole vector of coefficients, as required during ML estimation.

coefLower<- `signature(object = "covQual")`: replacement method for lower bounds on covQual coefficients.

coefLower `signature(object = "covQual")`: extracts the numeric values of the lower bounds.

coef `signature(object = "covQual")`: extracts the numeric values of the covariance parameters.

coefUpper<- `signature(object = "covQual")`: replacement method for upper bounds on `covQual` coefficients.

coefUpper `signature(object = "covQual")`: ...

covMat `signature(object = "covQual")`: build the covariance matrix or the cross covariance matrix between two sets of locations for a `covQual` object.

npar `signature(object = "covQual")`: returns the number of parameters.

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

scores `signature(object = "covQual")`: return the vector of scores, i.e. the derivative of the log-likelihood w.r.t. the parameter vector at the current parameter values.

simulate `signature(object = "covQual")`: simulate `nsim` paths from a Gaussian Process having the covariance structure. The paths are indexed by the finite set of levels of factor inputs, and they are returned as columns of a matrix.

varVec `signature(object = "covQual")`: build the variance vector corresponding to a set locations for a `covQual` object.

Note

This class is to be regarded as experimental. The slot names or list may be changed in the future. The methods `npar`, `inputNames` or `` `inputNames<-` `` should provide a more robust access to some slot values.

See Also

See `covMan` for a comparable structure dedicated to kernels with continuous inputs.

Examples

```
showClass("covQual")
```

`covQualNested` *Nested Qualitative Covariance*

Description

Nested Qualitative Covariance

Usage

```
covQualNested(input = "x",
              groupList = NULL,
              group = NULL,
              nestedLevels = NULL,
              between = "Symm",
              within = "Diag",
              covBet = c("corr", "homo", "hete"),
              covWith = c("corr", "homo", "hete"),
              compGrad = TRUE,
              contrasts = contr.helmod,
              intAsChar = TRUE)
```

Arguments

`input`	Name of the input, i.e. name of the column in the data frame when the covariance kernel is evaluated with the `covMat,covQual-method` method.
`groupList`	A list giving the groups, see **Examples**. Groups of size 1 are accepted. Note that the group values should be given in some order, with no gap between repeated values, see **Examples**.
`group`	Inactive if `groupList` is used. A factor or vector giving the groups, see **Examples**. Groups of size 1 are accepted. Note that the group values should be given in some order, with no gap between repeated values, see **Examples**.
`nestedLevels`	Inactive if `groupList` is used. A factor or a vector giving the (nested) levels within the group for each level of `group`. If this is missing, each element of `group` is assumed to correspond to one nested level within the group and the levels within the group are taken as integers in the order of `group` elements.
`between`	Character giving the type of structure to use for the *between* part. For now this can be one of the three choices `"Diag"`, the diagonal structure of `q1Diag`, `"Symm"` for the general covariance of `q1Symm`, or `"CompSymm"` for the Compound Symmetry covariance of `q1CompSymm`. Default is `Symm`, corresponding to a specific correlation value for each pair of groups. On the other hand, `Diag` corresponds to a common correlation value for all pairs of groups.
`within`	Character vector giving the type of structure to use for the *within* part. The choices are the same as for `between`. The character vector is recycled to have length G so the *within* covariances can differ across groups. Default is `"Diag"`, corresponding to a compound symmetry matrix.
`covBet`	
`covWith`	Character vector indicating the type of covariance matrix to be used for the generator between- or within- matrices, as in `q1Diag`, `q1Symm` or `q1CompSymm`: correlation ("corr"), homoscedastic ("homo") or heteroscedastic ("hete"). Partial matching is allowed. This is different from the form of the resulting covariance matrix, see section **Caution**.
`compGrad`	Logical.
`contrasts`	Object of class `"function"`. This function is similar to the `contr.helmert` or `contr.treatment` functions, but it must return an *orthogonal* matrix. For a given integer `n`, it returns a matrix with `n` rows and `n -1` columns forming a basis for the supplementary of a vector of ones in the n-dimensional Euclidean space. The `contr.helmod` can be used to obtain an orthogonal matrix hence defining an orthonormal basis.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Value

An object with class `"covQualNested"`.

Caution

When covBet and covWith are zero, the resulting matrix *is not a correlation matrix*, due to the mode of construction. The "between" covariance matrix is a correlation but diagonal blocks are added to the extended matrix obtained by re-sizing the "between" covariance into a $n \times n$ matrix.

Note

For now the replacement method such as 'coef<-' are inherited from the class covQual1. Consequently when these methods are used they do not update the covariance structure in the between slot nor those in the within (list) slot.

This covariance kernel involves two categorical (i.e. factor) inputs, but these are nested. It could be aliased in the future as q1Nested or q2Nested.

Examples

```
### Ex 1. See the vignette "groupKernel" for an example
###       inspired from computer experiments.

### Ex 2. Below an example in data analysis.

country <- c("B", "B", "B", "F", "F" ,"F", "D", "D", "D")
cities <- c("AntWerp", "Ghent" , "Charleroi", "Paris", "Marseille",
            "Lyon", "Berlin", "Hamburg", "Munchen")
myGroupList <- list(B = cities[1:3],
                    F = cities[4:6],
                    D = cities[7:9])

## create a nested covariance.
# first way, with argument 'groupList':

nest1 <- covQualNested(input = "ccities",
                       groupList = myGroupList,
                       between = "Symm", within = "Diag",
                       compGrad = TRUE,
                       covBet = "corr", covWith = "corr")

# second way, with arguments 'group' and 'nestedLevels'
nest2 <- covQualNested(input = "ccities",
                       group = country, nestedLevels = cities,
                       between = "Symm", within = "Diag",
                       compGrad = TRUE,
                       covBet = "corr", covWith = "corr")


## 'show' and 'plot' method as automatically invocated
nest2
plot(nest2, type = "cor")

## check that the covariance matrices match for nest1 and nest2
max(abs(covMat(nest1) - covMat(nest2)))
```

```
## When the groups are not given in order, an error occurs!

countryBad <- c("B", "B", "F", "F", "F", "D", "D", "D", "B")
cities <- c("AntWerp", "Ghent", "Paris", "Marseille", "Lyon",
            "Berlin", "Hamburg", "Munchen", "Charleroi")

nestBad <- try(covQualNested(input = "ccities",
                             group = countryBad, nestedLevels = cities,
                             between = "Symm", within = "Diag",
                             compGrad = TRUE,
                             covBet = "corr", covWith = "corr"))
```

covQualNested-class *Class* "covQualNested"

Description

Correlation or covariance structure for qualitative inputs (i.e. factors) obtained by nesting.

Objects from the Class

Objects can be created by calls of the form `new("covQualNested",...)`.

Slots

`covLevels`: Object of class `"function"` computing the covariance matrix for the set of all levels.

`covLevMat`: Object of class `"matrix"`. The matrix returned by the function in slot `covLevels`. Since this matrix is often needed, it can be stored rather than recomputed.

`hasGrad`: Object of class `"logical"`. If `TRUE`, the analytical gradient can be computed.

`acceptLowerSQRT`: Object of class `"logical"`. If `TRUE`, the lower square root of the matrix can be returned

`label`: Object of class `"character"`. A label to describe the kernel.

`d`: Object of class `"integer"`. The number of inputs.

`inputNames`: Object of class `"character"` Names of the inputs.

`nlevels`: Object of class `"integer"` with length `d` give the number of levels for the factors.

`levels`: Object of class `"list"` with length `d`. Gives the levels for the inputs.

`parLower`: Object of class `"numeric"`. Lower bounds on the (hyper) parameters.

`parUpper`: Object of class `"numeric"`. Upper bounds on the (hyper) parameters.

`par`: Object of class `"numeric"`. Value of the (hyper) parameters.

`parN`: Object of class `"integer"`. Number of (hyper) parameters.

`kernParNames`**:** Object of class `"character"`. Name of the parameters.

`group`**:** Object of class `"integer"`. Group numbers: one for each final level.

`groupLevels`**:** Object of class `"character"`. Vector of labels for the groups.

`between`**:** Object of class `"covQual"`. A covariance or correlation structure that can be used between groups.

`within`**:** Object of class `"list"`. A list of covariance or correlation structures that are used within the groups. Each item has class `"covQual"`.

`parNCum`**:** Object of class `"integer"`. Cumulated number of parameters. Used for technical computations.

`contrasts`**:** Object of class `"function"`. A contrast function like `contr.helmod`. This function must return a contrast matrix with columns having unit norm.

Extends

Class `"covQual"`, directly. Class `"covAll"`, by class "covQual", distance 2.

Methods

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

Examples

```
showClass("covQualNested")
```

`covRadial` *Creator for the Class* `"covRadial"`

Description

Creator for the class `"covRadial"`, which describes *radial kernels*.

Usage

```
    covRadial(k1Fun1 = k1Fun1Gauss,
              cov = c("corr", "homo"),
              iso = 0, hasGrad = TRUE,
              inputs = NULL, d = NULL,
              parNames, par = NULL,
              parLower = NULL, parUpper = NULL,
              label = "Radial kernel",
              ...)
```

Arguments

`k1Fun1`	A function of a *scalar* numeric variable, and possibly of an extra "shape" parameter. This function should return the first-order derivative or the two-first order derivatives as an attribute with name `"der"` and with a matrix content. When an extra shape parameter exists, the gradient should also be returned as an attribute with name `"gradient"`, see **Examples** later. The name of the function can be given as a character string.
`cov`	A character string specifying the kind of covariance kernel: correlation kernel (`"corr"`) or kernel of a homoscedastic GP (`"homo"`). Partial matching is allowed.
`iso`	Integer. The value `1L` corresponds to an isotropic covariance, with all the inputs sharing the same range value.
`hasGrad`	Integer or logical. Tells if the value returned by the function `k1Fun1` has an attribute named `"der"` giving the derivative(s).
`inputs`	Character. Names of the inputs.
`d`	Integer. Number of inputs.
`par, parLower, parUpper`	Optional numeric values for the lower bounds on the parameters. Can be `NA` for `par`, can be `-Inf` for `parLower` and `Inf` for `parUpper`.
`parNames`	Names of the parameters. By default, ranges are prefixed `"theta_"` in the non-iso case and the range is named `"theta"` in the iso case.
`label`	A short description of the kernel object.
`...`	Other arguments passed to the method `new`.

Details

A radial kernel on the d-dimensional Euclidean space takes the form

$$K(\mathbf{x}, \mathbf{x}') = \sigma^2 k_1(r)$$

where $k_1(r)$ is a suitable correlation kernel for a one-dimensional input, and r is given by

$$r = \left\{ \sum_{\ell=1}^{d} [x_\ell - x'_\ell]^2 / \theta_\ell^2 \right\}^{1/2}.$$

In this default form, the radial kernel depends on $d + 1$ parameters: the *ranges* $\theta_\ell > 0$ and the variance σ^2.

An *isotropic* form uses the same range θ for all inputs, i.e. sets $\theta_\ell = \theta$ for all ℓ. This is obtained by using `iso = TRUE`.

A *correlation* version uses $\sigma^2 = 1$. This is obtained by using `cov = "corr"`.

Finally, the correlation kernel $k_1(r)$ can depend on a "shape" parameter, e.g. have the form $k_1(r;\, \alpha)$. The extra shape parameter α will be considered then as a parameter of the resulting radial kernel, making it possible to estimate it by ML along with the range(s) and the variance.

Value

An object with class "covRadial".

Note

When k1Fun1 has more than one formal argument, its arguments with position > 1 are assumed to be "shape" parameters of the model. Examples are functions with formals function(x,shape = 1.0) or function(x,alpha = 2.0,beta = 3.0), corresponding to vector of parameter names c("shape") and c("alpha","beta"). Using more than one shape parameter has not been tested yet.

Remind that using a one-dimensional correlation kernel $k_1(r)$ here *does not* warrant that a positive semi-definite kernel will result for *any* dimension d. This question relates to Schoenberg's theorem and the concept of completely monotone functions.

References

Gregory Fassauher and Michael McCourt (2016) *Kernel-based Approximation Methods using MATLAB*. World Scientific.

See Also

k1Fun1Exp, k1Fun1Matern3_2, k1Fun1Matern5_2 or k1Fun1Gauss for examples of functions that can be used as values for the k1Fun1 formal.

Examples

```
set.seed(123)
d <- 2; ng <- 20
xg <- seq(from = 0, to = 1, length.out = ng)
X <- as.matrix(expand.grid(x1 = xg, x2 = xg))

## ============================================================================
## A radial kernel using the power-exponential one-dimensional
## function
## ============================================================================

d <- 2
myCovRadial <- covRadial(k1Fun1 = k1Fun1PowExp, d = 2, cov = "homo", iso = 1)
coef(myCovRadial)
inputNames(myCovRadial) <- colnames(X)
coef(myCovRadial) <- c(alpha = 1.8, theta = 2.0, sigma2 = 4.0)
y <- simulate(myCovRadial, X = X, nsim = 1)
persp(x = xg, y = xg, z = matrix(y, nrow = ng))

## ============================================================================
## Define the inverse multiquadric kernel function. We return the first two
## derivatives and the gradient as attributes of the result.
## ============================================================================

myk1Fun <- function(x, beta = 2) {
    prov <- 1 + x * x
    res <- prov^(-beta)
    der <- matrix(NA, nrow = length(x), ncol = 2)
    der[ , 1] <- - beta * 2 * x * res / prov
    der[ , 2] <- -2 * beta * (1 - (1 + 2 * beta) * x * x) * res / prov / prov
    grad <- -log(prov) * res
    attr(res, "gradient") <- grad
    attr(res, "der") <- der
    res
}

myCovRadial1 <- covRadial(k1Fun1 = myk1Fun, d = 2, cov = "homo", iso = 1)
coef(myCovRadial1)
inputNames(myCovRadial1) <- colnames(X)
coef(myCovRadial1) <- c(beta = 0.2, theta = 0.4, sigma2 = 4.0)
y1 <- simulate(myCovRadial1, X = X, nsim = 1)
persp(x = xg, y = xg, z = matrix(y1, nrow = ng))
```

covRadial-class *Class* "covRadial"

Description

Class of radial covariance kernels.

Objects from the Class

Objects can be created by calls of the form `covRadial(...)` of `new("covRadial",...)`.

Slots

- `k1Fun1`: Object of class `"function"` A function of a scalar numeric variable. Note that using a one-dimensional kernel here *does not* warrant that a positive semi-definite kernel results for any dimension d.
- `k1Fun1Char`: Object of class `"character"` describing the function in the slot `k1Fun1`.
- `hasGrad`: Object of class `"logical"`. Tells if the value returned by the function `kern1Fun` has an attribute named `"der"` giving the derivative(s).
- `cov`: Object of class `"integer"`. The value `0L` corresponds to a correlation kernel while `1L` is for a covariance kernel.
- `iso`: Object of class `"integer"`. The value `1L` corresponds to an isotropic covariance, with all the inputs sharing the same range value.
- `label`: Object of class `"character"`. Short description of the object.
- `d`: Object of class `"integer"`. Dimension, i.e. number of inputs.
- `inputNames`: Object of class `"optCharacter"`. Names of the inputs.
- `parLower`: Object of class `"numeric"`. Numeric values for the lower bounds on the parameters. Can be `-Inf`.

- `parUpper`**:** Object of class `"numeric"`. Numeric values for the upper bounds on the parameters. Can be `Inf`.
- `par`**:** Object of class `"numeric"`. Numeric values for the parameters. Can be `NA`.
- `parN1`**:** Object of class `"integer"`. Number of parameters of the function `kern1Fun` (such as a shape).
- `parN`**:** Object of class `"integer"`. Number of parameters for the object. The include: *direct* parameters in the function `kern1Fun`, ranges, and variance.
- `kern1ParNames`**:** Object of class `"character"`. Names of the *direct* parameters.
- `kernParNames`**:** Object of class `"character"`. Names of the parameters.

Extends

Class `"covAll"`, directly.

Methods

- **coef<-** `signature(object = "covRadial", value = "numeric")`: Set the vector of values for the parameters.
- **coefLower<-** `signature(object = "covRadial")`: Set the vector of lower bounds on the parameters.
- **coefLower** `signature(object = "covRadial")`: Get the vector of lower bounds on the parameters.
- **coef** `signature(object = "covRadial")`: Get the vector of values for the parameters.
- **coefUpper<-** `signature(object = "covRadial")`: Set the vector of upper bounds on the parameters.
- **coefUpper** `signature(object = "covRadial")`: Get the vector of upper bounds on the parameters.
- **covMat** `signature(object = "covRadial")`: Compute the covariance matrix for given sites.
- **npar** `signature(object = "covRadial")`: Get the number of parameters.
- **scores** `signature(object = "covRadial")`: Compute the scores i.e. the derivatives w.r.t. the parameters of the contribution of the covariance in the log-likelihood of a gp.
- **show** `signature(object = "covRadial")`: Print or show the object.
- **varVec** `signature(object = "covRadial")`: Compute the variance vector for given sites.

See Also

The creator function `covRadial`, where some details are given on the form of kernel. `covMan` and `covMan` for a comparable but more general class.

Examples

```
showClass("covRadial")
```

covTP *Creator for the Class* "covTP"

Description

Creator for the class "covTP".

Usage

```
covTP(k1Fun1 = k1Fun1Gauss,
      cov = c("corr", "homo"),
      iso = 0, iso1 = 1L,
      hasGrad = TRUE,
      inputs = NULL,
      d = NULL,
      parNames,
      par = NULL, parLower = NULL, parUpper = NULL,
      label = "Tensor product kernel",
      ...)
```

Arguments

k1Fun1	A kernel function of a *scalar* numeric variable, and possibly of an extra "shape" parameter. This function can also return the first-order derivative or the two-first order derivatives as an attribute with name "der" and with a matrix content. When an extra shape parameter exists, the gradient can also be returned as an attribute with name "gradient", see **Examples** later. The name of the function can be given as a character string.
cov	A character string specifying the kind of covariance kernel: correlation kernel ("corr") or kernel of a homoscedastic GP ("homo"). Partial matching is allowed.
iso	Integer. The value 1L corresponds to an isotropic covariance, with all the inputs sharing the same range value.
iso1	Integer. This applies only when k1Fun1 contains one or more parameters that can be called 'shape' parameters. At now, only one such parameter can be found in k1Fun1 and consequently iso1 must be of length one. With iso1 = 0 the shape parameter in k1Fun1 will generate d parameters in the covTP object with their name suffixed by the dimension. When iso1 is 1 only one shape parameter will be created in the covTP object.
hasGrad	Integer or logical. Tells if the value returned by the function k1Fun1 has an attribute named "der" giving the derivative(s).
inputs	Character. Names of the inputs.
d	Integer. Number of inputs.
parNames	Names of the parameters. By default, ranges are prefixed "theta_" in the non-iso case and the range is named "theta" in the iso case.

`par`	Numeric values for the parameters. Can be `NA`.
`parLower`	Numeric values for the lower bounds on the parameters. Can be `-Inf`.
`parUpper`	Numeric values for the upper bounds on the parameters. Can be `Inf`.
`label`	A short description of the kernel object.
`...`	Other arguments passed to the method `new`.

Details

A tensor-product kernel on the d-dimensional Euclidean space takes the form

$$K(\mathbf{x}, \mathbf{x}') = \sigma^2 \prod_{\ell=1}^{d} \kappa(r_\ell)$$

where $\kappa(r)$ is a suitable correlation kernel for a one-dimensional input, and r_ℓ is given by $r_\ell := [x_\ell - x'_\ell]/\theta_\ell$ for $\ell = 1$ to d.

In this default form, the tensor-product kernel depends on $d + 1$ parameters: the *ranges* $\theta_\ell > 0$ and the variance σ^2.

An *isotropic* form uses the same range θ for all inputs, i.e. sets $\theta_\ell = \theta$ for all ℓ. This is obtained by using `iso = TRUE`.

A *correlation* version uses $\sigma^2 = 1$. This is obtained by using `cov = "corr"`.

Finally, the correlation kernel $\kappa(r)$ can depend on a "shape" parameter, e.g. have the form $\kappa(r; \alpha)$. The extra shape parameter α will be considered then as a parameter of the resulting tensor-product kernel, making it possible to estimate it by ML along with the range(s) and the variance.

Value

An object with class `"covTP"`.

Examples

```
## Not run:
if (require(DiceKriging)) {
    ## a 16-points factorial design and the corresponding response
    d <- 2; n <- 16; x <- seq(from = 0.0, to = 1.0, length.out = 4)
    X <- expand.grid(x1 = x, x2 = x)
    y <- apply(X, 1, DiceKriging::branin)

    ## kriging model with matern5_2 covariance structure, constant
    ## trend. A crucial point is to set the upper bounds!
    mycov <- covTP(k1Fun1 = k1Fun1Matern5_2, d = 2, cov = "homo")
    coefUpper(mycov) <- c(2.0, 2.0, 1e10)
    mygp <- gp(y ~ 1, data = data.frame(X, y),
               cov = mycov, multistart = 100, noise = FALSE)

    nGrid <- 50; xGrid <- seq(from = 0, to = 1, length.out = nGrid)
    XGrid <- expand.grid(x1 = xGrid, x2 = xGrid)
    yGrid <- apply(XGrid, 1, DiceKriging::branin)
    pgp <- predict(mygp, XGrid)$mean
```

```
        mykm <- km(design = X, response = y)
        pkm <- predict(mykm, XGrid, "UK")$mean
        c("km" = sqrt(mean((yGrid - pkm)^2)),
          "gp" = sqrt(mean((yGrid - pgp)^2)))

    }

    ## End(Not run)
```

`covTP-class` *Class* `"covTP"`

Description

S4 class representing a Tensor Product (TP) covariance kernel.

Objects from the Class

Objects can be created by calls of the form `new("covTP",...)` or by using the `covTP` function.

Slots

`k1Fun1`**:** Object of class `"function"` A function of a scalar numeric variable.

`k1Fun1Char`**:** Object of class `"character"` describing the function in the slot `k1Fun1`.

`hasGrad`**:** Object of class `"logical"`. Tells if the value returned by the function `kern1Fun` has an attribute named `"der"` giving the derivative(s).

`cov`**:** Object of class `"integer"`. The value `0L` corresponds to a correlation kernel while `1L` is for a covariance kernel.

`iso`**:** Object of class `"integer"`. The value `1L` corresponds to an isotropic covariance, with all the inputs sharing the same range value.

`iso1`**:** Object of class `"integer"` used only when the function in the slot `k1Fun1` depends on parameters i.e. has more than one formal argument. NOT IMPLEMENTED YET.

`label`**:** Object of class `"character"`. Short description of the object.

`d`**:** Object of class `"integer"`. Dimension, i.e. number of inputs.

`inputNames`**:** Object of class `"optCharacter"`. Names of the inputs.

`parLower`**:** Object of class `"numeric"`. Numeric values for the lower bounds on the parameters. Can be `-Inf`.

`parUpper`**:** Object of class `"numeric"`. Numeric values for the upper bounds on the parameters. Can be `Inf`.

`par`**:** Object of class `"numeric"`. Numeric values for the parameters. Can be `NA`.

`kern1ParN1`**:** Object of class `"integer"`. The number of parameters in `k1Fun1` (such as a shape).

`parN1`**:** Object of class `"integer"`. Number of parameters of the function `kern1Fun` (such as a shape).

`parN`**:** Object of class `"integer"`. Number of parameters for the object. The include: *direct* parameters in the function `kern1Fun`, ranges, and variance.

`kern1ParNames`**:** Object of class `"character"`. Names of the *direct* parameters.

`kernParNames`**:** Object of class `"character"`. Names of the parameters.

Extends

Class `"covAll"`, directly.

Methods

coef `signature(object = "covTP")`: Get the vector of values for the parameters.

coef<- `signature(object = "covTP",value = "numeric")`: Set the vector of values for the parameters.

coefLower `signature(object = "covTP")`: Get the vector of lower bounds on the parameters.

coefLower<- `signature(object = "covTP")`: Set the vector of lower bounds on the parameters.

coefUpper `signature(object = "covTP")`: Get the vector of upper bounds on the parameters.

coefUpper<- `signature(object = "covTP")`: Set the vector of upper bounds on the parameters.

covMat `signature(object = "covTP")`: Compute the covariance matrix for given sites.

npar `signature(object = "covTP")`: Get the number of parameters.

scores `signature(object = "covTP")`: Compute the scores i.e. the derivatives w.r.t. the parameters of the contribution of the covariance in the log-likelihood of a gp.

show `signature(object = "covTP")`: Print or show the object.

varVec `signature(object = "covTP")`: Compute the variance vector for given sites.

See Also

`covRadial` which is a similar covariance class and `covTP` which is intended to be the standard creator function for this class.

Examples

```
showClass("covTP")
```

`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 σ_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

$$K(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \sum_{i=1}^{d} k(x_i, x_i'; \boldsymbol{\theta}_{\mathbf{s}_i})$$

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 $\mathbf{x}$. The length p of all the vectors $\mathbf{s}_i$ 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.

name	*description*	*p*	*par. names*
`k1Exp`	exponential	2	`range`, `var`
`k1Matern3_2`	Matérn $\nu = 3/2$	2	`range`, `var`
`k1Matern5_2`	Matérn $\nu = 5/2$	2	`range`, `var`
`k1PowExp`	power exponential	3	`range`, `shape`, `var`
`k1Gauss`	gaussian or "square exponential"	2	`range`, `var`

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

N. Durrande, D. Ginsbourger, and O. Roustant (2012) Additive "Covariance kernels for high-dimensional Gaussian Process modeling", *Annales de la Faculté des Sciences de Toulouse* 21(3), pp. 481–499.

Examples

```
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
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: Generalized Least Squares Estimation with a Given Covariance Kernel*

Description

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

Usage

```
gls(object, ...)
```

Arguments

`object`	An object representing a covariance kernel.
`...`	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

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

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 `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 `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 $\widehat{\boldsymbol{\beta}}$ of length p containing the estimated coefficients if `beta = NULL`, or the known coefficients $\boldsymbol{\beta}$ either.

`L`	The (lower) Cholesky root matrix $\mathbf{L}$ of the covariance matrix $\mathbf{C}$. This matrix has n rows and n columns and $\mathbf{C} = \mathbf{L}\mathbf{L}^\top$.
`eStar`	Vector of length n: $\mathbf{e}^\star = \mathbf{L}^{-1}(\mathbf{y} - \mathbf{X}\widehat{\boldsymbol{\beta}})$.
`Fstar`	Matrix $n \times p$: $\mathbf{F}^\star := \mathbf{L}^{-1}\mathbf{F}$.
`sseStar`	Sum of squared errors: $\mathbf{e}^{\star\top}\mathbf{e}^\star$.
`RStar`	The 'R' upper triangular $p \times p$ matrix in the QR decomposition of `FStar`: $\mathbf{F}^\star = \mathbf{Q}\mathbf{R}^\star$.

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

```
## 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

```
gp(formula, data, inputs = inputNames(cov), 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 `mle,covAll-method` passed though `dots`. If `FALSE`, a simple Generalized Least Squares estimation will be used, see `gls,covAll-method`. Then the value of `varNoise` must be given and passed through `dots` in case `noise` is `TRUE`.
`...`	Other arguments passed to the estimation method. This will be the `mle,covAll-method` if `estim` is `TRUE` or `gls,covAll-method` if `estim` is `FALSE`. In the first case, the arguments will typically include `varNoiseIni`. In the second case, they will typically include `varNoise`. Note that a logical `noise` can be used in the `"mle"` case. In both cases, the arguments `y`, `X`, `F` can not be used since they are automatically passed.

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.

Note

When `estim` is `TRUE`, the covariance object in `cov` is expected to provide a gradient when used to compute a covariance matrix, since the default value of `compGrad` it `TRUE`, see `mle,covAll-method`.

Author(s)

Y. Deville, D. Ginsbourger, O. Roustant

See Also

`mle,covAll-method` for a detailed example of maximum-likelihood estimation.

Examples

```
## ===================================================================
## 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
myGp <- gp(formula = y ~ 1, data = data.frame(y = rep(0, n), X),
            cov = myCov, estim = FALSE,
            beta = 10, varNoise = 0.05)
y <- simulate(myGp, cond = FALSE)$sim

## 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),
          cov = myCov,
          noise = TRUE,
          varNoiseLower = 1e-2,
          varNoiseIni = 1.0,
          parCovIni = parCovIni)
summary(est)
coef(est)

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

## 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,
                cov = myCov, noise = TRUE,
                parCovIni = rep(1, 2*d),
                parCovLower = c(rep(1e-4, 2*d)),
                parCovUpper = c(rep(5, d), rep(10,d)))

    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
##=======================================================================

d <- 4

## -- 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, 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;
       rdkern[0] =  2 * rkern[0] * SS2 / REAL(par)[0];

       SET_ATTR(kern, install(\"gradient\"), dkern);
       UNPROTECT(nprotect);
       return kern;
   "
    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)
myGp <- gp(formula = y ~ 1, data = data.frame(y = rep(0, n), design),
            cov = myGaussRVec, estim = FALSE,
            beta = 0, varNoise = 1e-8)
y <- simulate(myGp, cond = FALSE)$sim
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 <- gp(formula = y ~ ., data = data.frame(y = y, design),
                  cov = myGaussRVec,
                  compGrad = TRUE,
                  parCovIni = c(0.5, 0.5), varNoiseLower = 1e-4,
                  parCovLower = c(1e-5, 1e-5), parCovUpper = c(Inf, Inf))
)

summary(resRVec)
coef(resRVec)
pRVec <- predict(resRVec, newdata = design, type = "UK")
tAll <- tRVec
coefAll <- coef(resRVec)
## compare time required by the 3 implementations
## Not run:
    tR <- system.time(
        resR <- gp(formula = y ~ ., data = data.frame(y = y, design),
                   cov = myGaussR,
                   compGrad = TRUE,
                   parCovIni = c(0.5, 0.5), varNoiseLower = 1e-4,
                   parCovLower = c(1e-5, 1e-5), parCovUpper = c(Inf, Inf))
    )
    tAll <- rbind(tRVec = tAll, tR)
    coefAll <- rbind(coefAll, coef(resR))
    if (require(inline)) {
        tC <- system.time(
            resC <- gp(formula = y ~ ., data = data.frame(y = y, design),
                       cov = myGaussC,
                       compGrad = TRUE,
                       parCovIni = c(0.5, 0.5), varNoiseLower = 1e-4,
                       parCovLower = c(1e-5, 1e-5), parCovUpper = c(Inf, Inf))
        )
        tAll <- rbind(tAll, tC)
        coefAll <- rbind(coefAll, coef(resC))
    }

## End(Not run)
tAll

## rows must be identical
coefAll
```

`hasGrad` *Generic Function: Extract slot hasGrad of a Covariance Kernel*

Description

Generic function returning the slot hasGrad of a Covariance Kernel.

Usage

```
hasGrad(object, ...)

## S4 method for signature 'covAll'
hasGrad(object, ...)
```

Arguments

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

Value

A logical indicating whether the gradient is supplied in `object` (as indicated in the slot 'hasGrad').

`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`.
`...`	Not used.

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 ith 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

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.gp`, `plot.gp`

`inputNames` *Generic Function: Names of the Inputs of a Covariance Kernel*

Description

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

Usage

```
inputNames(object, ...)

## S4 replacement method for signature 'covAll'
inputNames(object, ...) <- value
```

Arguments

`object`	A covariance kernel object.
`value`	A suitable character vector.
`...`	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. They play an important role since in general the inputs are found among other columns of a data frame, and their order is not fixed. For instance in a data frame used as `newdata` formal in the `predict` method, the inputs are generally found at positions which differ from those in the data frame used at the creation of the object.

See Also

`checkX`

k1Exp *Predefined covMan Objects for 1D Kernels*

Description

Predefined kernel Objects as covMan objects.

Usage

```
k1Exp
k1Matern3_2
k1Matern5_2
k1Gauss
```

Format

Objects with class "covMan".

Details

These objects are provided mainly as examples. They are used covTS.

Examples

```
set.seed(1234)
x <- sort(runif(40))
X <- cbind(x = x)
yExp <- simulate(k1Exp, nsim = 20, X = X)
matplot(X, yExp, type = "l", col = "SpringGreen", ylab = "")
yGauss <- simulate(k1Gauss, nsim = 20, X = X)
matlines(X, yGauss, col = "orangered")
title("Simulated paths from 'k1Exp' (green) and 'k1Gauss' (orange)")

## ==========================================================================
## You can build a similar object using a creator of
## 'covMan'. Although the objects 'k1Gauss' and 'myk1Gauss' differ,
## they describe the same mathematical object.
## ==========================================================================

myk1Gauss <- kGauss(d = 1)
```

k1Matern3_2 *One-Dimensional Classical Covariance Kernel Functions*

Description

One-dimensional classical covariance kernel Functions.

Usage

```
k1FunExp(x1, x2, par)
k1FunGauss(x1, x2, par)
k1FunPowExp(x1, x2, par)
k1FunMatern3_2(x1, x2, par)
k1FunMatern5_2(x1, x2, par)

k1Fun1Cos(x)
k1Fun1Exp(x)
k1Fun1Gauss(x)
k1Fun1PowExp(x, alpha = 1.5)
k1Fun1Matern3_2(x)
k1Fun1Matern5_2(x)
```

Arguments

x1	First location vector.
x2	Second location vector. Must have the same length as x1.
x	For stationary covariance functions, the vector containing difference of positions: x = x1 -x2.
alpha	Regularity parameter in $(0, 2]$ for Power Exponential covariance function.
par	Vector of parameters. The length and the meaning of the elements in this vector depend on the chosen kernel. The first parameter is the range parameter (if there is one), the last is the variance. So the shape parameter of k1FunPowExp is the second one out of the three parameters.

Details

These kernel functions are described in the Roustant et al (2012), table 1 p. 8. More details are given in chap. 4 of Rasmussen et al (2006).

Value

A matrix with a "gradient" attribute. This matrix has n_1 rows and n_2 columns where n_1 and n_2 are the length of x1 and x2. If x1 and x2 have length 1, the attribute is a vector of the same length p as par and gives the derivative of the kernel with respect to the parameters in the same order. If x1 or x2 have length > 1, the attribute is an array with dimension (n_1, n_2, p).

Note

The kernel functions are coded in C through the `.Call` interface and are mainly intended for internal use. They are used by the `covTS` class.

Be aware that very few checks are done (length of objects, order of the parameters, ...).

Author(s)

Oivier Roustant, David Ginsbourger, Yves Deville

References

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

O. Roustant, D. Ginsbourger, Y. 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. https://www.jstatsoft.org/v51/i01/

Examples

```
## show the functions
n <- 300
x0 <- 0
x <- seq(from = 0, to = 3, length.out = n)
kExpVal <- k1FunExp(x0, x, par = c(range = 1, var = 2))
kGaussVal <- k1FunGauss(x0, x, par = c(range = 1, var = 2))
kPowExpVal <- k1FunPowExp(x0, x, par = c(range = 1, shape = 1.5, var = 2))
kMatern3_2Val <- k1FunMatern3_2(x0, x, par = c(range = 1, var = 2))
kMatern5_2Val <- k1FunMatern5_2(x0, x, par = c(range = 1, var = 2))
kerns <- cbind(as.vector(kExpVal), as.vector(kGaussVal), as.vector(kPowExpVal),
               as.vector(kMatern3_2Val), as.vector(kMatern5_2Val))
matplot(x, kerns, type = "l", main = "five 'kergp' 1d-kernels", lwd = 2)

## extract gradient
head(attr(kPowExpVal, "gradient"))
```

`kernelName` *Name of the One-Dimensional 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.

`kGauss` *Gauss (Squared-Exponential) Kernel*

Description

Gauss (or squared exponential) covariance function.

Usage

```
kGauss(d)
```

Arguments

`d`	Dimension.

Value

An object of class `"covMan"` with default parameters: 1 for ranges and variance values.

References

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

Examples

```
kGauss()  # default: d = 1, nu = 5/2
myGauss <- kGauss(d = 2)
coef(myGauss) <- c(range = c(2, 5), sigma2 = 0.1)
myGauss
```

`kMatern` *Matérn Kernels*

Description

Matérn kernels, obtained by plugging the Euclidian norm into a 1-dimensional Matérn function.

Usage

```
kMatern(d, nu = "5/2")
```

Arguments

`d`	Dimension.
`nu`	Character corresponding to the smoothness parameter ν of Matérn kernels. At this stage, the possible values are "1/2" (exponential kernel), "3/2" or "5/2".

Value

An object of class `"covMan"` with default parameters: 1 for ranges and variance values.

Note

Notice that these kernels are NOT obtained by tensor product.

References

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

Examples

```
kMatern()  # default: d = 1, nu = 5/2
kMatern(d = 2)
myMatern <- kMatern(d = 5, nu = "3/2")
coef(myMatern) <- c(range = 1:5, sigma2 = 0.1)
myMatern
try(kMatern(nu = 2))  # error
```

mle — *Generic Function: 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.

mle-methods — *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

```
## 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,
    compGrad = hasGrad(object),
    doOptim = TRUE,
    optimFun = c("nloptr::nloptr", "stats::optim"),
    optimMethod = ifelse(compGrad, "NLOPT_LD_LBFGS", "NLOPT_LN_COBYLA"),
    optimCode = NULL,
    multistart = 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 or matrix giving the initial values for the parameters. See **Examples**. When this argument is omitted, the vector of covariance parameters given in `object` is used if `multistart` == 1; If `multistart` > 1, a matrix of parameters is simulated by using `simulPar`. Remind that you can use the `coef` and `coef<-` methods to get and set this slot of the covariance object.
`parCovLower`	Lower bounds for the parameters. When this argument is omitted, the vector of parameters lower bounds in the covariance given in `object` is used. You can use `coefLower` and `coefLower<-` methods to get and set this slot of the covariance object.
`parCovUpper`	Upper bounds for the parameters. When this argument is omitted, the vector of parameters lower bounds in the covariance given in `object` is used. You can use `coefUpper` and `coefUpper<-` methods to get and set this slot of the covariance object.
`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. Should be <= `varNoiseIni`.
`varNoiseUpper`	Upper bound for the noise variance. Should be >= `varNoiseIni`.
`compGrad`	Logical: compute and use the analytical gradient in optimization? This is only possible when `object` provides the analytical gradient.
`doOptim`	Logical. If `FALSE` no optimization is done.
`optimFun`	Function used for optimization. The two pre-defined choices are `nloptr::nloptr` (default) and `stats::optim`, both in combination with a few specific optimization methods. Ignored if `optimCode` is provided.
`optimMethod`	Name of the optimization method or algorithm. This is passed as the `"algorithm"` element of the `opts` argument when `nloptr::nloptr` is used (default), or to the `method` argument when `stats::optim` is used. When another value of `optimFun` is given, the value of `optimMethod` is ignored. Ignored if `optimCode` is provided. Use `optimMethods` to obtain the list of usable values.

`optimCode`	An object with class `"expression"` or `"character"` representing a user-written R code to be parsed and performing the log-likelihood maximization. Notice that this argument will bypass `optimFun` and `optimMethod`. The expression must define an object named `"opt"`, which is either a list containing optimization results, either an object inheriting from `"try-error"` to cope with the case where a problem occurred during the optimization.
`multistart`	Number of optimizations to perform from different starting points (see `parCovIni`). Parallel backend is encouraged.
`parTrack`	If `TRUE`, the parameter vectors used during the optimization are tracked and returned as a matrix.
`trace`	Integer level of verbosity.
`checkNames`	if `TRUE` (default), check the compatibility of `X` with `object`, see `checkX`.
`...`	Further arguments to be passed to the optimization function, `nloptr` or `optim`.

Details

The choice of optimization method is as follows.

- When `optimFun` is `nloptr:nloptr`, it is assumed that we are minimizing the negative log-likelihood $-\log L$. Note that both predefined methods `"NLOPT_LD_LBFGS"` and `"NLOPT_LN_COBYLA"` can cope with a non-finite value of the objective, except for the initial value of the parameter. Non-finite values of $-\log L$ are often met in practice during optimization steps. The method `"NLOPT_LD_LBFGS"` used when `compGrad` is `TRUE` requires that the gradient is provided by/with the covariance object. You can try other values of `optimMethod` corresponding to the possible choice of the `"algorithm"` element in the `opts` argument of `nloptr:nloptr`. It may be useful to give other options in order to control the optimization and its stopping rule.
- When `optimFun` is `"stats:optim"`, it is assumed that we are maximizing the log-likelihood $\log L$. We suggest to use one of the methods `"L-BFGS-B"` or `"BFGS"`. Notice that `control` can be provided in `...`, but `control$fnscale` is forced to be `-1`, corresponding to maximization. Note that `"L-BFGS-B"` uses box constraints, but the optimization stops as soon as the log-likelihood is non-finite or `NA`. The method `"BFGS"` does not use constraints but allows the log-likelihood to be non-finite or `NA`. Both methods can be used without gradient or with gradient if `object` allows this.

The vectors `parCovIni`, `parCovLower`, `parCovUpper` must have elements corresponding to those of the vector of kernel parameters given by `coef(object)`. These vectors should have suitably named elements.

Value

A list with elements hopefully having understandable names.

`opt`	List of optimization results if it was successful, or an error object otherwise.
`coef.kernel`	The vector of 'kernel' coefficients. This will include one or several variance parameters.
`coef.trend`	Estimate of the vector $\boldsymbol{\beta}$ of the trend coefficients.
`parTracked`	A matrix with rows giving the successive iterates during optimization if the `parTrack` argument was set to `TRUE`.

Note

The checks concerning the parameter names, dimensions of provided objects, . . . are not fully implemented yet.

Using the `optimCode` possibility requires a bit of programming effort, although a typical code only contains a few lines.

Author(s)

Y. Deville, O. Roustant

See Also

`gp` for various examples, `optimMethods` to see the possible values of the argument `optimMethod`.

Examples

```
set.seed(29770)

##=======================================================================
## 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;
       rdkern[0] =  2 * rkern[0] * SS2 / REAL(par)[0];

       SET_ATTR(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"),
                   parLower = c("theta" = 0.0, "sigma2" = 0.0),
                   parUpper = c("theta" = Inf, "sigma2" = Inf))

## End(Not run)

##=======================================================================
## Example (continued). Simulate data for covMan and trend
##=======================================================================
n <- 100;
X <- matrix(runif(n * d), nrow = n)
colnames(X) <- inputNames(myCovMan)

coef(myCovMan) <- myPar <- c(theta = 0.5, sigma2 = 2)
C <- covMat(object = myCovMan, X = X,
            compGrad = FALSE,  index = 1L)

library(MASS)
set.seed(456)
y <- mvrnorm(mu = rep(0, n), Sigma = C)
p <- rpois(1, lambda = 4)
if (p > 0) {
  F <- matrix(runif(n * p), nrow = n, ncol = p)
  beta <- rnorm(p)
  y <- F %*% beta + y
} else F <- NULL
par <- parCovIni <- c("theta" = 0.6, "sigma2" = 4)

##=======================================================================
## 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)
est$opt$value

## change the (constrained) optimization  method
## Not run:
est1 <- mle(object = myCovMan,
            parCovIni = parCovIni,
            optimFun = "stats::optim",
            optimMethod = "L-BFGS-B",
            y = y, X = X, F = F,
            parCovLower = c(0.05, 0.05), parCovUpper = c(10, 100),
            parTrack = TRUE, noise = FALSE, checkNames = FALSE)
est1$opt$value

## End(Not run)

##=======================================================================
## 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)

##=======================================================================
## Example (continued). Explore the surface ?
##=======================================================================
## Not run:
   require(rgl)
   persp3d(x = theta.grid, y = sigma2.grid, z = ll,
           xlab = "theta", ylab = "sigma2", zlab = "logLik",
           col = "SpringGreen3", alpha = 0.6)

## End(Not run)

##=======================================================================
## Example (continued). Draw a contour plot for the log-lik
##                        and show iterates
##=======================================================================
## Not run:
    contour(x = theta.grid, y = sigma2.grid, z = llmat,
            col = "SpringGreen3", xlab = "theta", ylab = "sigma2",
            main = "log-likelihood contours and iterates",
            xlim = range(theta.grid, est$parTracked[ , 1], na.rm = TRUE),
            ylim = range(sigma2.grid, est$parTracked[ , 2], na.rm = TRUE))
    abline(v = est$coef.kernel[1], h = est$coef.kernel[2], lty = "dotted")
    niter <- nrow(est$parTracked)
    points(est$parTracked[1:niter-1, ],
           col = "orangered", bg = "yellow", pch = 21, lwd = 2, type = "o")
    points(est$parTracked[niter, , drop = FALSE],
           col = "blue", bg = "blue", pch = 21, lwd = 2, type = "o", cex = 1.5)
    ann <- seq(from = 1, to = niter, by = 5)
    text(x = est$parTracked[ann, 1], y = est$parTracked[ann, 2],
         labels = ann - 1L, pos = 4, cex = 0.8, col = "orangered")
    points(x = myPar["theta"], y = myPar["sigma2"],
           bg = "Chartreuse3", col = "ForestGreen",
           pch = 22, lwd = 2, cex = 1.4)

    legend("topright", legend = c("optim", "optim (last)", "true"),
           pch = c(21, 21, 22), lwd = c(2, 2, 2), lty = c(1, 1, NA),
           col = c("orangered", "blue", "ForestGreen"),
           pt.bg = c("yellow", "blue", "Chartreuse3"))


## End(Not run)
```

npar *Generic function: 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

`coef` method

`optimMethods` *Optimization Methods (or Algorithms) for the* `mle` *Method*

Description

Optimization methods (or algorithms) for the `mle` method.

Usage

```
optimMethods(optimMethod = NULL,
             optimFun = c("both", "nloptr::nloptr", "stats::optim"))
```

Arguments

`optimMethod`	A character string used to find a method in a possible approximated fashion, see **Examples**.
`optimFun`	Value of the corresponding formal argument of the `mle` method, or `"both"`. In the later case the full list of algorithms will be obtained.

Value

A data frame with four character columns: `optimFun`, `optimMethod`, `globLoc` and `derNo`. The column `globLoc` indicate whether the method is global (`"G"`) or local (`"L"`). The column `derNo` indicates whether the method uses derivatives (`D`) or not (`"N"`) or *possibly* uses it (`"P"`). Only methods corresponding the `optimFun` = `"stats::optim"` can have the value `"P"` for `derNo`. The data frame can be zero-row if `optimMethod` is given and no method match.

Caution

The optimization method given in the argument `optimMethod` of the `mle` method should be compliant with the `compGrad` argument. Only a small number of possibilities have been tested, including the default values.

References

See The NLopt website.

See Also

`mle-methods`, `optim`, `nloptr`.

Examples

```
optimMethods()
optimMethods(optimMethod = "cobyla")
optimMethods(optimMethod = "nelder")
optimMethods(optimMethod = "BFGS")
optimMethods("CMAES")
```

`parMap` *Generic Function: Map the Parameters of a Composite Covariance Kernel*

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.

`parMap-methods` *Map the Parameters of a Structure on the Inputs and Kernel Parameters*

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

```
myCov <- covTS(d = 3, kernel = "k1Gauss",
               dep = c(range = "input"), value = c(range = 1.1))
parMap(myCov)
```

parNamesSymm *Vector of Names for the General 'Symm' Parameterisation*

Description

Vector of names for the general 'Symm' parameterisation.

Usage

```
parNamesSymm(nlev)
```

Arguments

`nlev`	Number of levels.

Value

Character vector of names.

Examples

```
parNamesSymm(nlev = 4)
```

parseCovFormula *Parse a Formula or Expression Describing a Composite Covariance Kernel*

Description

Parse a formula (or expression) describing a composite covariance kernel.

Usage

```
parseCovFormula(formula, where = .GlobalEnv, trace = 0)
```

Arguments

`formula`	A formula or expression describing a covariance kernel. See **Examples**.
`where`	An environment where kernel objects and top parameters are searched for.
`trace`	Integer level of verbosity.

Details

The formula involves existing covariance kernel objects and must define a valid kernel composition rule. For instance the sum and the product of kernels, the convex combination of kernels are classically used. The kernels objects are used in the formula with parentheses as is they where functions calls with no formal arguments e.g. `obj( )`. Non-kernel objects used in the formula must be numeric scalar parameters and are called *top* parameters. The covariance objects must exist in the environment defined by `where` because their slots will be used to identify the inputs and the parameters of the composite kernel defined by the formula.

Value

A list with the results of parsing. Although the results content is easy to understand, the function is not intended to be used by the final user, and the results may change in future versions.

Caution

Only relatively simple formulas are correctly parsed. So use only formulas having a structure similar to one of those given in the examples. In case of problems, error messages are likely to be difficult to understand.

Note

The parsing separates covariance objects from top parameters. It retrieves information about the kernel inputs and parameters from the slots. Obviously, any change in the covariances objects after the parsing (e.g. change in the parameters names or values) will not be reported in the results of the parsing, so kernel any needed customization must be done prior to the parsing.

Author(s)

Yves Deville

Examples

```
## ==========================================================================
## build some kernels (with their inputNames) in the global environment
## ==========================================================================

myCovExp3 <- kMatern(d = 3, nu = "1/2")
inputNames(myCovExp3) <- c("x", "y", "z")

myCovGauss2 <- kGauss(d = 2)
inputNames(myCovGauss2) <- c("temp1", "temp2")

k <- kMatern(d = 1)
inputNames(k) <- "x"

ell <- kMatern(d = 1)
inputNames(ell) <- "y"

## ==========================================================================
## Parse a formula. This formula is stupid because 'myCovGauss2'
## and 'myCovExp3' should be CORRELATION kernels and not
## covariance kernels to produce an identifiable model.
## ===========================================================================

cov <- ~ tau2 * myCovGauss2() * myCovExp3() + sigma2 * k()
pf <- parseCovFormula(cov, trace = 1)

## ===========================================================================
## Parse a formula with ANOVA composition
## ===========================================================================

cov1 <- ~ tau2 * myCovGauss2() * myCovExp3() + sigma2 * (1 + k()) * (1 + ell())
pf1 <- parseCovFormula(cov1, trace = 1)
```

plot *Plot for a qualitative input*

Description

Plots of the covariance matrix or the correlation matrix of a qualitative input. For an ordinal factor, the warping function can also be plotted.

Usage

```
## S4 method for signature 'covQual'
plot(x, y, type = c("cov", "cor", "warping"), ...)
```

Arguments

x	An object of class covQual-class.
y	Not used.
type	A character indicating the desired type of plot. Type warping only works for an ordinal input.
...	Other arguments passed to corrplot::corrplot or plot.

Details

Covariance / correlation plots are done with package corrplot if loaded, or lattice else.

See Also

covOrd.

Examples

```
u <- ordered(1:6, levels = letters[1:6])

myCov2 <- covOrd(ordered = u, k1Fun1 = k1Fun1Cos, warpFun = "norm")
coef(myCov2) <- c(mean = 0.5, sd = 0.05, theta = 0.1)

plot(myCov2, type = "cor", method = "ellipse")
plot(myCov2, type = "warp", col = "blue", lwd = 2)
```

`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

```
## 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 $[y(\mathbf{x}_i) - \widehat{y}_{-i}(\mathbf{x}_i)]/\widehat{\sigma}_{-i}(\mathbf{x}_i)$, where $y(\mathbf{x}_i)$ is the response at the location $\mathbf{x}_i$, $\widehat{y}_{-i}(\mathbf{x}_i)$ is the fitted value when the i-th observation is omitted (see `influence.gp`), and $\widehat{\sigma}_{-i}(\mathbf{x}_i)$ 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

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.

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

---

plot.simulate.gp *Plot Simulations from a* gp *Object*

Description

Function to plot simulations from a gp object.

Usage

```
## S3 method for class 'simulate.gp'
plot(x, y,
        col = list(sim = "SpringGreen3", trend = "orangered"),
        show = c(sim = TRUE, trend = TRUE, y = TRUE),
        ...)
```

Arguments

`x`	An object containing simulations, produced by 'simulate' with `output = "list"`.
`y`	Not used yet.
`col`	Named list of colors to be used, with elements `"sim"` and `"trend"`.
`show`	A logical vector telling which elements must be shown.
`...`	Further argument passed to `plot`.

Value

Nothing.

Note

For now, this function can be used only when the number of inputs is one.

See Also

`simulate.gp`.

`predict.gp` *Prediction Method for the* `"gp"` *S3 Class*

Description

Prediction method for the `"gp"` S3 class.

Usage

```
## S3 method for class 'gp'
predict(object, newdata,
        type = ifelse(object$trendKnown, "SK", "UK"),
        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 $n/(n-p)$ by default (n and p denoting the number of rows and columns of the trend matrix $\mathbf{F}$). 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`.
`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 $\mathbf{c}$, containing all the covariances between the points in `newdata` and those in the initial design. Not returned if `lightReturn` is `TRUE`.
`cStar`	An auxiliary vector, equal to $\mathbf{L}^{-1}\mathbf{c}$ where $\mathbf{L}$ is the Cholesky root of the covariance matrix $\mathbf{C}$ 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.

prinKrige *Principal Kriging Functions*

Description

Principal Kriging Functions.

Usage

```
prinKrige(object)
```

Arguments

`object`	An object with class `"gp"`.

Details

The Principal Kriging Functions (PKF) are the eigenvectors of a symmetric positive matrix $\mathbf{B}$ named the *Bending Energy Matrix* which is met when combining a linear trend and a covariance kernel as done in gp. This matrix has dimension $n \times n$ and rank $n - p$. The PKF are given in the *ascending* order of the eigenvalues e_i

$$e_1 = e_2 = \ldots = e_p = 0 < e_{p+1} \leq e_{p+2} \leq \ldots \leq e_n.$$

The p first PKF generate the same space as do the p columns of the trend matrix $\mathbf{F}$, say colspan$(\mathbf{F})$. The following $n - p$ PKFs generate a supplementary of the subspace colspan$(\mathbf{F})$, and they have a decreasing influence on the response. So the $p+1$-th PKF can give a hint on a possible deterministic trend functions that could be added to the p existing ones.

The matrix $\mathbf{B}$ is such that $\mathbf{BF} = \mathbf{0}$, so the columns of $\mathbf{F}$ can be thought of as the eigenvectors that are associated with the zero eigenvalues e_1, ..., e_p.

Value

A list

- `values` The eigenvalues of the energy bending matrix in *ascending* order. The first p values must be very close to zero, but will not be zero since they are provided by numerical linear algebra.
- `vectors` A matrix $\mathbf{U}$ with its columns $\mathbf{u}_i$ equal to the eigenvectors of the energy bending matrix, in correspondence with the eigenvalues e_i.
- `B` The Energy Bending Matrix $\mathbf{B}$. Remind that the eigenvectors are used here in the ascending order of the eigenvalues, which is the reverse of the usual order.

Note

When an eigenvalue e_i is such that $e_{i-1} < e_i < e_{i+1}$ (which can happen only for $i > p$), the corresponding PKF is unique up to a change of sign. However a run of $r > 1$ identical eigenvalues is associated with a r-dimensional eigenspace and the corresponding PKFs have no meaning when they are considered individually.

References

Sahu S.K. and Mardia K.V. (2003). A Bayesian kriged Kalman model for short-term forecasting of air pollution levels. *Appl. Statist.* 54 (1), pp. 223-244.

Examples

```
library(kergp)
set.seed(314159)
n <- 100
x <- sort(runif(n))
y <- 2 + 4 * x  + 2 * x^2 + 3 * sin(6 * pi * x ) + 1.0 * rnorm(n)
nNew <- 60; xNew <- sort(runif(nNew))
df <- data.frame(x = x, y = y)

##-------------------------------------------------------------------------
## use a Matern 3/2 covariance and a mispecified trend. We should guess
## that it lacks a mainily linear and slightly quadratic part.
##-------------------------------------------------------------------------

myKern <- k1Matern3_2
inputNames(myKern) <- "x"
mygp <- gp(formula = y ~ sin(6 * pi * x),
           data = df,
           parCovLower = c(0.01, 0.01), parCovUpper = c(10, 100),
           cov = myKern, estim = TRUE, noise = TRUE)
PK <- prinKrige(mygp)

## the third PKF suggests a possible linear trend term, and the
## fourth may suggest a possible quadratic linear trend

matplot(x, PK$vectors[ , 1:4], type = "l", lwd = 2)
```

---

q1CompSymm   *Qualitative Correlation or Covariance Kernel with one Input and Compound Symmetric Correlation*

Description

Qualitative correlation or covariance kernel with one input and compound symmetric correlation.

Usage

```
q1CompSymm(factor, input = "x", cov = c("corr", "homo"), intAsChar = TRUE)
```

Arguments

`factor`	A factor with the wanted levels for the covariance kernel object.
`input`	Name of (qualitative) input for the kernel.
`cov`	Character telling if the kernel is a correlation kernel or a homoscedastic covariance kernel.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Value

An object with class `"covQual"` with `d = 1` qualitative input.

Note

Correlation kernels are needed in tensor products because the tensor product of two covariance kernels each with unknown variance would not be identifiable.

See Also

The `corLevCompSymm` function used to compute the correlation matrix and its gradients w.r.t. the correlation parameters.

Examples

```
School <- factor(1L:3L, labels = c("Bad", "Mean" , "Good"))
myCor <- q1CompSymm(School, input = "School")
coef(myCor) <- 0.26
plot(myCor, type = "cor")

## Use a data.frame with a factor
set.seed(246)
newSchool <- factor(sample(1L:3L, size = 20, replace = TRUE),
                    labels = c("Bad", "Mean" , "Good"))
C1 <- covMat(myCor, X = data.frame(School = newSchool),
             compGrad = FALSE, lowerSQRT = FALSE)
```

q1Diag — *Qualitative Correlation or Covariance Kernel with one Input and Diagonal Structure*

Description

Qualitative correlation or covariance kernel with one input and diagonal structure.

Usage

```
q1Diag(factor, input = "x", cov = c("corr", "homo", "hete"), intAsChar = TRUE)
```

Arguments

`factor`	A factor with the wanted levels for the covariance kernel object.
`input`	Name of (qualitative) input for the kernel.
`cov`	Character telling if the result is a correlation kernel, an homoscedastic covariance kernel or an heteroscedastic covariance kernel with an arbitrary variance vector.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Value

An object with class `"covQual"` with `d = 1` qualitative input.

Note

The correlation version obtained with `cov = "corr"` has no parameters.

See Also

`q1Symm`, `q1CompSymm` are other covariance structures for one qualitative input.

Examples

```
School <- factor(1L:3L, labels = c("Bad", "Mean" , "Good"))

## correlation: no parameter!
myCor <- q1Diag(School, input = "School")

## covariance
myCov <- q1Diag(School, input = "School", cov = "hete")
coef(myCov) <- c(1.1, 2.2, 3.3)
```

q1LowRank *Qualitative Correlation or Covariance Kernel with one Input and Low-Rank Correlation*

Description

Qualitative correlation or covariance kernel with one input and low-rank correlation.

Usage

```
q1LowRank(factor, rank = 2L, input = "x",
          cov = c("corr", "homo", "hete"), intAsChar = TRUE)
```

Arguments

`factor`	A factor with the wanted levels for the covariance kernel object.
`rank`	The wanted rank, which must be ≥ 2 and $< m$ where m is the number of levels.
`input`	Name of (qualitative) input for the kernel.
`cov`	Character telling what variance structure will be chosen: *correlation* with no variance parameter, *homoscedastic* with one variance parameter or *heteroscedastic* with m variance parameters.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Details

The correlation structure involves $(r-1)(m-r/2)$ parameters. The parameterization of Rapisarda et al is used: the correlation parameters are angles $\theta_{i,j}$ corresponding to $1 < i \leq r$ and $1 \leq j < i$ or to $r < i \leq m$ and $1 \leq j < r$. The correlation matrix $\mathbf{C}$ for the levels, with size m, factors as $\mathbf{C} = \mathbf{L}\mathbf{L}^\top$ where $\mathbf{L}$ is a lower-triangular matrix with dimension $m \times r$ with all its rows having unit Euclidean norm. Note that the diagonal elements of $\mathbf{L}$ can be negative and correspondingly the angles $\theta_{i,1}$ are taken in the interval $[0, 2\pi)$ for $1 < i \leq r$. The matrix $\mathbf{L}$ is not unique. As explained in Grubišić and Pietersz, the parameterization is surjective: any correlation with rank $\leq r$ is obtained by choosing a suitable vector of parameters, but this vector is not unique.

Correlation kernels are needed in tensor products because the tensor product of two covariance kernels each with unknown variance would not be identifiable.

Value

An object with class `"covQual"` with `d = 1` qualitative input.

References

Francesco Rapisarda, Damanio Brigo, Fabio Mercurio (2007). "Parameterizing Correlations a Geometric Interpretation". *IMA Journal of Management Mathematics*, **18**(1): 55-73.

Igor Grubišić, Raoul Pietersz (2007). "Efficient Rank Reduction of Correlation Matrices". *Linear Algebra and its Applications*, **422**: 629-653.

See Also

The `q1Symm` function to create a kernel object for the full-rank case and `corLevLowRank` for the correlation function.

Examples

```
myFact <- factor(letters[1:8])
myCov <- q1LowRank(factor = myFact, rank = 3)
## corrplot
plot(myCov)
## find the rank using a pivoted Cholesky
chol(covMat(myCov), pivot = TRUE)
```

---

q1Symm — *Qualitative Correlation or Covariance Kernel with one Input and General Symmetric Correlation*

Description

Qualitative correlation or covariance kernel with one input and general symmetric correlation.

Usage

```
q1Symm(factor, input = "x", cov = c("corr", "homo", "hete"), intAsChar = TRUE)
```

Arguments

`factor`	A factor with the wanted levels for the covariance kernel object.
`input`	Name of (qualitative) input for the kernel.
`cov`	Character telling if the result is a correlation kernel, an homoscedastic covariance kernel or an heteroscedastic covariance kernel with an arbitrary variance vector.
`intAsChar`	Logical. If `TRUE` (default), an integer-valued input will be coerced into a character. Otherwise, it will be coerced into a factor.

Value

An object with class `"covQual"` with `d = 1` qualitative input.

Note

Correlation kernels are needed in tensor products because the tensor product of two covariance kernels each with unknown variance would not be identifiable.

See Also

The `corLevSymm` function used to compute the correlation matrix and its gradients w.r.t. the correlation parameters.

Examples

```
School <- factor(1L:3L, labels = c("Bad", "Mean" , "Good"))
myCor <- q1Symm(School, input = "School")
coef(myCor) <- c(theta_2_1 = pi / 3, theta_3_1 = pi / 4, theta_3_2 = pi / 8)
plot(myCor, type = "cor")

## Use a data.frame with a factor
set.seed(246)
newSchool <- factor(sample(1L:3L, size = 20, replace = TRUE),
                    labels = c("Bad", "Mean" , "Good"))
C1 <- covMat(myCor, X = data.frame(School = newSchool),
             compGrad = FALSE, lowerSQRT = FALSE)
```

`scores` *Generic Function: 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 $\boldsymbol{\theta}$. The score for θ_k is obtained as a matrix scalar product

$$\partial_{\theta_k}\ell = \text{trace}(\mathbf{WD})$$

where $\mathbf{D} := \partial_{\theta_k}\mathbf{C}$ and where $\mathbf{W}$ is the matrix $\mathbf{W} := \mathbf{e}\mathbf{e}^\top - \mathbf{C}^{-1}$. The vector $\mathbf{e}$ is the vector of residuals and the matrix $\mathbf{C}$ is the covariance computed for the design $\mathbf{X}$.

Value

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

Note

The scores can be efficiently computed when the matrix $\mathbf{W}$ has already been pre-computed.

`shapeSlot` *Extracts the Slots of a Structure*

Description

Extract the slot of a structure.

Usage

```
shapeSlot(object, slotName = "par", type = "all", as = "vector")
```

Arguments

`object`	An object to extract from, typically a covariance kernel.
`slotName`	Name of the slot to be extracted.
`type`	Type of slot to be extracted. Can be either a type of parameter, `"var"` or `"all"`.
`as`	Type of result wanted. Can be `"vector"`, `"list"` or `"matrix"`.

Value

A vector, list or matrix containing the extraction.

Note

This function is for internal use only.

`simulate, covAll-method`

Simulation of a `covAll` *Object*

Description

Simulation of a `covAll` object.

Usage

```
## S4 method for signature 'covAll'
simulate(object, nsim = 1, seed = NULL,
         X, mu = NULL, method = "mvrnorm", checkNames = TRUE,
         ...)
```

Arguments

`object`	A covariance kernel object.
`nsim`	Number of simulated paths.
`seed`	Not used yet.
`X`	A matrix with the needed inputs as its columns.
`mu`	Optional vector with length `nrow(X)` giving the expectation $\mu(\mathbf{x})$ of the Gaussian Process at the simulation locations $\mathbf{x}$.
`method`	Character used to choose the simulation method. For now the only possible value is `"mvrnorm"` corresponding to the function with this name in the **MASS** package.
`checkNames`	Logical. It `TRUE` the colnames of `X` and the input names of `object` as given by `inputNames(object)` must be identical sets.
`...`	Other arguments for methods.

Value

A numeric matrix with `nrow(X)` rows and `nsim` columns. Each column is the vector of the simulated path at the simulation locations.

Note

The simulation is unconditional.

See Also

The `mvrnorm` function.

Examples

```
## -- as in example(kergp) define an argumentwise invariant kernel --

kernFun <- function(x1, x2, par) {
  h <- (abs(x1) - abs(x2)) / par[1]
  S <- sum(h^2)
  d2 <- exp(-S)
  K <- par[2] * d2
  d1 <- 2 * K * S / par[1]
  attr(K, "gradient") <- c(theta = d1,  sigma2 = d2)
  return(K)
}

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

## -- 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)

ySim <- simulate(covSymGauss, X = Xgrid)
contour(x = xGrid, y = xGrid,
        z = matrix(ySim, nrow = nGrid, ncol = nGrid),
        nlevels = 15)
```

`simulate.gp` *Simulation of Paths from a* `gp` *Object*

Description

Simulation of paths from a gp object.

Usage

```
## S3 method for class 'gp'
simulate(object, nsim = 1L, seed = NULL,
         newdata = NULL,
         cond = TRUE,
         trendKnown = FALSE,
         newVarNoise = NULL,
         nuggetSim = 1e-8,
         checkNames = TRUE,
         output = c("list", "matrix"),
         label = "y", unit = "",
         ...)
```

Arguments

`object`	An object with class `"gp"`.
`nsim`	Number of paths wanted.
`seed`	Not used yet.
`newdata`	A data frame containing the inputs values used for simulation as well as the required trend covariates, if any. This is similar to the `newdata` formal in `predict.gp`.
`cond`	Logical. Should the simulations be conditional on the observations used in the object or not?
`trendKnown`	Logical. If `TRUE` the vector of trend coefficients will be regarded as known so all simulated paths share the same trend. When `FALSE`, the trend must have been estimated so that its estimation covariance is known. Then each path will have a different trend.
`newVarNoise`	Variance of the noise for the "new" simulated observations. For the default `NULL`, the noise variance found in `object` is used. Note that if a very small positive value is used, each simulated path is the sum of the trend the smooth GP part and an interval containing say 95% of the simulated responses can be regarded as a confidence interval rather than a prediction interval.
`nuggetSim`	Small positive number ("nugget") added to the diagonal of conditional covariance matrices before computing a Cholesky decomposition, for numerical lack of positive-definiteness. This may happen when the covariance kernel is not (either theoretically or numerically) positive definite.
`checkNames`	Logical. It `TRUE` the colnames of `X` and the input names of the covariance in `object` as given by `inputNames(object)` must be identical sets.
`output`	The type of output wanted. A simple matrix as in standard simulation methods may be quite poor, since interesting intermediate results are then lost.
`label, unit`	A label and unit that will be copied into the output object when `output` is `"list"`.
`...`	Further arguments to be passed to the `simulate` method of the `"covAll"` class.

Value

A matrix with the simulated paths as its columns or a more complete list with more results. This list which is given the S3 class `"simulate.gp"` has the following elements.

- `X`, `F`, `y` Inputs, trend covariates and response.
- `XNew`, `FNew` New inputs, new trend covariates.
- `sim` Matrix of simulated paths.
- `trend` Matrix of simulated trends.
- `trendKnown`, `noise`, `newVarNoise` Values of the formals.
- `Call` The call.

Note

When betaKnown is FALSE, the *trend* and the *smooth GP* parts of a simulation are usually correlated, and their sum will show less dispersion than each of the two components. The covariance of the vector $\widehat{\boldsymbol{\beta}}$ can be regarded as the posterior distribution corresponding to a non-informative prior, the distribution from which a new path is drawn being the predictive distribution.

Author(s)

Yves Deville

Examples

```
set.seed(314159)
n <- 40
x <- sort(runif(n))
y <- 2 + 4 * x  + 2 * x^2 + 3 * sin(6 * pi * x ) + 1.0 * rnorm(n)
df <- data.frame(x = x, y = y)

##-------------------------------------------------------------------------
## use a Matern 3/2 covariance. With model #2, the trend is mispecified,
## so the smooth GP part of the model captures a part of the trend.
##-------------------------------------------------------------------------

myKern <- k1Matern3_2
inputNames(myKern) <- "x"
mygp <- list()
mygp[[1]] <- gp(formula = y ~ x + I(x^2) + sin(6 * pi * x), data = df,
                parCovLower = c(0.01, 0.01), parCovUpper = c(10, 100),
                cov = myKern, estim = TRUE, noise = TRUE)
mygp[[2]] <- gp(formula = y ~ sin(6 * pi * x), data = df,
                parCovLower = c(0.01, 0.01), parCovUpper = c(10, 100),
                cov = myKern, estim = TRUE, noise = TRUE)

##-------------------------------------------------------------------------
## New data
##-------------------------------------------------------------------------

nNew <- 150
xNew <- seq(from = -0.2, to= 1.2, length.out = nNew)
dfNew <- data.frame(x = xNew)

opar <- par(mfrow = c(2L, 2L))

nsim <- 40
for (i in 1:2) {

    ##---------------------------------------------------------------------
    ## beta known or not, conditional
    ##---------------------------------------------------------------------

    simTU <- simulate(object = mygp[[i]], newdata = dfNew,  nsim = nsim,
                      trendKnown = FALSE)
    plot(simTU, main = "trend unknown, conditional")

    simTK <- simulate(object = mygp[[i]], newdata = dfNew, nsim = nsim,
                      trendKnown = TRUE)
    plot(simTK, main = "trend known, conditional")

    ##-------------------------------------------------------------------
    ## The same but UNconditional
    ##-------------------------------------------------------------------

    simTU <- simulate(object = mygp[[i]], newdata = dfNew,  nsim = nsim,
                     trendKnown = FALSE, cond = FALSE)
    plot(simTU, main = "trend unknown, unconditional")
    simTK <- simulate(object = mygp[[i]], newdata = dfNew, nsim = nsim,
                      trendKnown = TRUE, cond = FALSE)
    plot(simTK, main = "trend known, unconditional")
}

par(opar)
```

`simulPar` *Generic function: 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.

simulPar,covAll-method *Draw Random Values for the Parameters of a Covariance Kernel*

Description

Draw random values for the parameters of a covariance kernel

object.

Usage

```
## S4 method for signature 'covAll'
simulPar(object, nsim = 1L, seed = NULL)
```

Arguments

`object`	A covariance kernel.
`nsim`	Number of drawings.
`seed`	Seed for the random generator.

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.

symIndices *Vector of Indices Useful for Symmetric or Anti-Symmetric Matrices.*

Description

Vector of indices useful for symmetric or anti-symmetric matrices

Usage

```
symIndices(n, diag = FALSE)
```

Arguments

n	Size of a square matrix.
diag	Logical. When FALSE the diagonal is omitted in the lower and upper triangles.

Details

This function is intended to provide computations which are faster than `lower.tri` and `upper.tri`.

Value

A list containing the following integer vectors, each with length $(n-1)n/2$.

i, j	Row and column indices for the lower triangle to be used in a two-index style.
kL	Indices for the lower triangle, to be used in single-index style. The elements are picked in column order. So if `X` is a square matrix with size `n`, then `X[kL]` is the vector containing the elements of the lower triangle of `X` taken in column order.
kU	Indices for the upper triangle, to be used in a single-index style. The elements are picked in row order. So if `X` is a square matrix with size `n`, then `X[kU]` is the vector containing the elements of the upper triangle of `X` taken in row order.

Examples

```
n <- rpois(1, lambda = 10)
L <- symIndices(n)
X <- matrix(1L:(n * n), nrow = n)
max(abs(X[lower.tri(X, diag = FALSE)] - L$kL))
max(abs(t(X)[lower.tri(X, diag = FALSE)] - L$kU))
cbind(row = L$i, col = L$j)
```

---

translude *Make Translucent colors*

Description

Make translucent colors.

Usage

```
translude(colors, alpha = 0.6)
```

Arguments

colors	A vector of colors in a format that can be understood by `col2rgb`.
alpha	Level of opacity ("0" means fully transparent and "max" means opaque). After recycling to reach the required length, this value or vector is used as `alpha` in `rgb`.

Value

A vector of translucent (or semi-transparent) colors.

---

varVec *Generic Function: Variance of Gaussian Process at Specific Locations*

Description

Generic function returning a variance vector

Usage

```
varVec(object, X, ...)
```

Arguments

`object`	Covariance kernel object.
`X`	A matrix with d columns, where d is the number of inputs of the covariance kernel. The n rows define a set of sites or locations.
`...`	Other arguments for methods.

Value

A numeric vector with length `nrow(X)` containing the variances $K(\mathbf{x}, \mathbf{x})$ for all the sites $\mathbf{x}$.

---

varVec-methods *Covariance Matrix for a Covariance Kernel Object*

Description

Covariance matrix for a covariance kernel object.

Usage

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

## S4 method for signature 'covTS'
varVec(object, X, 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.
`compGrad`	Logical. If `TRUE` a derivative with respect to the vector of parameters 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 variance vector is computed in a C program using the `.Call` interface. The R kernel function is evaluated within the C code using `eval`.

Value

A vector of length `nrow(X)` with general element $V_i := K(\mathbf{x}_i, \mathbf{x}_i; \boldsymbol{\theta})$ where $K(\mathbf{x}_1, \mathbf{x}_2; \boldsymbol{\theta})$ is the covariance kernel function.

Note

The value of the parameter $\boldsymbol{\theta}$ can be extracted from the object with the `coef` method.

See Also

`coef` method

Examples

```
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(V1 <- varVec(myCov, X = X)) ## bad colnames
colnames(X) <- inputNames(myCov)
V2 <- varVec(myCov, X = X)

Xnew <- matrix(runif(n * 3), nrow = n, ncol = 3)
colnames(Xnew) <- inputNames(myCov)
V2 <- varVec(myCov, X = X)
```

`warpNorm` *Warpings for Ordinal Inputs*

Description

Given warpings for ordinal inputs.

Usage

```
warpNorm
warpUnorm
warpPower
warpSpline1
warpSpline2
```

Format

The format is a list of 6:

$ fun : the warping function. The second argument is the vector of parameters. The function returns a numeric vector with an attribute `"gradient"` giving the derivative w.r.t. the parameters.

$ parNames : names of warping parameters (character vector).

$ parDefault: default values of warping parameters (numeric vector).

$ parLower : lower bounds of warping parameters (numeric vector).

$ parUpper : upper bounds of warping parameters (numeric vector).

$ hasGrad : a boolean equal to `TRUE` if `gradient` is supplied as an attribute of `fun`.

Details

See `covOrd` for the definition of a warping in this context. At this stage, two warpings corresponding to cumulative density functions (cdf) are implemented:

- Normal distribution, truncated to $[0, 1]$:

  $$F(x) = [N(x) - N(0)]/[N(1) - N(0)]$$

  where $N(x) = \Phi([x - \mu]/\sigma)$ is the cdf of the normal distribution with mean μ and standard deviation σ.
- Power distribution on $[0, 1]$: $F(x) = x^{pow}$.

Furthermore, a warping corresponding to unnormalized Normal cdf is implemented, as well as spline warpings of degree 1 and 2. Splines are defined by a sequence of `k` knots between 0 and 1. The first knot is 0, and the last is 1. A spline warping of degree 1 is a continuous piecewise linear function. It is parameterized by a positive vector of length `k-1`, representing the increments at knots. A spline warping of degree 2 is a non-decreasing quadratic spline. It is obtained by integrating a spline of degree 1. Its parameters form a positive vector of length `k`, representing the derivatives at knots. The implementation relies on the function `scalingFun1d` of `DiceKriging` package.

Index

∗ **classes**
  covAll-class, 22
  covANOVA-class, 25
  covComp-class, 28
  covMan-class, 33
  covOrd-class, 39
  covQual-class, 41
  covQualNested-class, 46
  covRadial-class, 50
  covTP-class, 54
  covTS-class, 57
∗ **datasets**
  k1Exp, 71
∗ **methods**
  plot.gp, 89
∗ **models**
  influence.gp, 68
  plot.gp, 89

as.list, covTP-method, 7
as.list,covComp-method *(*covComp-class*)*, 28
as.list,covTP-method *(*as.list, covTP-method*)*, 7

c, *56*
checkGrad, 8
checkPar, 10
checkX, 11, *36*, *60*, *70*, *78*, *109*
checkX,covAll,data.frame-method *(*covAll-class*)*, 22
checkX,covAll,matrix-method *(*covAll-class*)*, 22
checkX,covAll-method *(*checkX-methods*)*, 11
checkX,covComp,data.frame-method *(*covComp-class*)*, 28
checkX,covOrd,data.frame-method *(*covOrd-class*)*, 39
checkX,covOrd,matrix-method *(*covOrd-class*)*, 39
checkX,covQual,data.frame-method *(*covQual-class*)*, 41
checkX,covQual,matrix-method *(*covQual-class*)*, 41
checkX-methods, 11
coef, *36*, *83*, *109*
coef,covANOVA-method *(*covANOVA-class*)*, 25
coef,covComp-method *(*covComp-class*)*, 28
coef,covMan-method *(*coef-methods*)*, 12
coef,covOrd-method *(*covOrd-class*)*, 39
coef,covQual-method *(*covQual-class*)*, 41
coef,covRadial-method *(*covRadial-class*)*, 50
coef,covTP-method *(*covTP-class*)*, 54
coef,covTS-method *(*coef-methods*)*, 12
coef,methods *(*coef-methods*)*, 12
coef-methods, 12
coef<-, 13
coef<-,covANOVA,numeric-method *(*covANOVA-class*)*, 25
coef<-,covComp,numeric-method *(*covComp-class*)*, 28
coef<-,covMan,numeric-method *(*covMan-class*)*, 33
coef<-,covOrd,numeric-method *(*covOrd-class*)*, 39
coef<-,covQual,numeric-method *(*covQual-class*)*, 41
coef<-,covRadial,numeric-method *(*covRadial-class*)*, 50
coef<-,covTP,numeric-method *(*covTP-class*)*, 54
coef<-,covTS,numeric-method *(*covTS-class*)*, 57
coefLower, 14
coefLower,covANOVA-method

*(*covANOVA-class*)*, 25
coefLower,covComp-method
*(*covComp-class*)*, 28
coefLower,covMan-method *(*covMan-class*)*,
33
coefLower,covOrd-method *(*covOrd-class*)*,
39
coefLower,covQual-method
*(*covQual-class*)*, 41
coefLower,covRadial-method
*(*covRadial-class*)*, 50
coefLower,covTP-method *(*covTP-class*)*, 54
coefLower,covTS-method *(*covTS-class*)*, 57
coefLower<- *(*coefLower*)*, 14
coefLower<-,covANOVA-method
*(*covANOVA-class*)*, 25
coefLower<-,covMan-method
*(*covMan-class*)*, 33
coefLower<-,covOrd-method
*(*covOrd-class*)*, 39
coefLower<-,covQual-method
*(*covQual-class*)*, 41
coefLower<-,covRadial-method
*(*covRadial-class*)*, 50
coefLower<-,covTP-method *(*covTP-class*)*,
54
coefLower<-,covTS-method *(*covTS-class*)*,
57
coefUpper *(*coefLower*)*, 14
coefUpper,covANOVA-method
*(*covANOVA-class*)*, 25
coefUpper,covComp-method
*(*covComp-class*)*, 28
coefUpper,covMan-method *(*covMan-class*)*,
33
coefUpper,covOrd-method *(*covOrd-class*)*,
39
coefUpper,covQual-method
*(*covQual-class*)*, 41
coefUpper,covRadial-method
*(*covRadial-class*)*, 50
coefUpper,covTP-method *(*covTP-class*)*, 54
coefUpper,covTS-method *(*covTS-class*)*, 57
coefUpper<- *(*coefLower*)*, 14
coefUpper<-,covANOVA-method
*(*covANOVA-class*)*, 25
coefUpper<-,covMan-method
*(*covMan-class*)*, 33
coefUpper<-,covOrd-method
*(*covOrd-class*)*, 39
coefUpper<-,covQual-method
*(*covQual-class*)*, 41
coefUpper<-,covRadial-method
*(*covRadial-class*)*, 50
coefUpper<-,covTP-method *(*covTP-class*)*,
54
coefUpper<-,covTS-method *(*covTS-class*)*,
57
coerce,covMan,function-method
*(*covMan-class*)*, 33
coerce,covOrd,function-method
*(*covOrd-class*)*, 39
coerce,covQual,function-method
*(*covQual-class*)*, 41
col2rgb, *107*
contr.helmert, *44*
contr.helmod, 14, *44*
contr.treatment, *44*
corLevCompSymm, 15, *95*
corLevDiag, 17
corLevLowRank, 18, *98*
corLevSymm, *19*, 19, *99*
covAll, *26*, *29*, *47*, *51*, *55*
covAll-class, 22
covANOVA, 23, *25*
covANOVA-class, 25
covComp, 27, *29*
covComp-class, 28
covMan, 30, *33*, *34*, *41*, *43*, *51*
covMan-class, 33
covMat, 35
covMat,covANOVA-method
*(*covANOVA-class*)*, 25
covMat,covComp-method *(*covComp-class*)*,
28
covMat,covMan-method *(*covMat-methods*)*,
35
covMat,covOrd-method *(*covOrd-class*)*, 39
covMat,covQual-method *(*covQual-class*)*,
41
covMat,covRadial-method
*(*covRadial-class*)*, 50
covMat,covTP-method *(*covTP-class*)*, 54
covMat,covTS-method *(*covMat-methods*)*, 35
covMat-methods, 35
covOrd, 37, *88*, *110*

covOrd-class, 39
covQual, *47*
covQual-class, 41
covQualNested, 43
covQualNested-class, 46
covRadial, 47, *51*, *55*
covRadial-class, 50
covTP, *8*, *26*, 52, *54*, *55*
covTP-class, 54
covTS, 55, *58*, *59*, *71*
covTS-class, 57

gls, 59
gls,covAll-method *(*gls-methods*)*, 60
gls-methods, 60
gp, 61, *79*, *92*, *93*

hasGrad, 68
hasGrad,covAll-method *(*hasGrad*)*, 68

influence.gp, 68, *89*, *90*
inputNames, *11*, *12*, 70
inputNames,covAll-method *(*inputNames*)*, 70
inputNames<- *(*inputNames*)*, 70
inputNames<-,covAll,character-method *(*inputNames*)*, 70
inputNames<-,covAll-method *(*inputNames*)*, 70
inputNames<-,covComp,character-method *(*covComp-class*)*, 28

k1Exp, 71
k1Fun1Cos *(*k1Matern3_2*)*, 72
k1Fun1Exp, *49*
k1Fun1Exp *(*k1Matern3_2*)*, 72
k1Fun1Gauss, *49*
k1Fun1Gauss *(*k1Matern3_2*)*, 72
k1Fun1Matern3_2, *49*
k1Fun1Matern3_2 *(*k1Matern3_2*)*, 72
k1Fun1Matern5_2, *49*
k1Fun1Matern5_2 *(*k1Matern3_2*)*, 72
k1Fun1PowExp *(*k1Matern3_2*)*, 72
k1FunExp *(*k1Matern3_2*)*, 72
k1FunGauss *(*k1Matern3_2*)*, 72
k1FunMatern3_2 *(*k1Matern3_2*)*, 72
k1FunMatern5_2 *(*k1Matern3_2*)*, 72
k1FunPowExp *(*k1Matern3_2*)*, 72
k1Gauss *(*k1Exp*)*, 71
k1Matern3_2, 72
k1Matern3_2 *(*k1Exp*)*, 71
k1Matern5_2 *(*k1Exp*)*, 71
k1PowExp *(*k1Exp*)*, 71
kergp *(*kergp-package*)*, 3
kergp-package, 3
kernelName, 73
kernelName,covTS-method *(*covTS-class*)*, 57
kExp *(*kMatern*)*, 75
kGauss, 74
kMatern, 75
kSE *(*kGauss*)*, 74

mle, 76
mle,covAll-method *(*mle-methods*)*, 76
mle-methods, 76
mvrnorm, *101*

nloptr, *78*, *84*
npar, 82
npar,covANOVA-method *(*covANOVA-class*)*, 25
npar,covMan-method *(*npar-methods*)*, 83
npar,covOrd-method *(*covOrd-class*)*, 39
npar,covQual-method *(*covQual-class*)*, 41
npar,covRadial-method *(*covRadial-class*)*, 50
npar,covTP-method *(*covTP-class*)*, 54
npar,covTS-method *(*npar-methods*)*, 83
npar-methods, 83

optim, *78*, *84*
optimMethods, *77*, *79*, 83
ordered, *37*

parMap, 84
parMap,covTS-method *(*parMap-methods*)*, 85
parMap-methods, 85
parNamesSymm, 86
parseCovFormula, 86
plot, 88
plot,covQual,ANY-method *(*plot*)*, 88
plot,covQual-method *(*plot*)*, 88
plot.covQual *(*plot*)*, 88
plot.gp, *69*, 89
plot.simulate.gp, 90
predict.gp, *69*, *90*, 91, *103*
prinKrige, 93

q1CompSymm, *44*, 94, *96*
q1Diag, *44*, 96
q1LowRank, *19*, 97
q1Symm, *20*, *44*, *96*, *98*, 98

rgb, *107*

scores, 99
scores,covANOVA-method (covANOVA-class), 25
scores,covComp-method (covComp-class), 28
scores,covMan-method (covMan-class), 33
scores,covOrd-method (covOrd-class), 39
scores,covQual-method (covQual-class), 41
scores,covRadial-method (covRadial-class), 50
scores,covTP-method (covTP-class), 54
scores,covTS-method (covTS-class), 57
shapeSlot, 100
show,covANOVA-method (covANOVA-class), 25
show,covComp-method (covComp-class), 28
show,covMan-method (covMan-class), 33
show,covOrd-method (covOrd-class), 39
show,covQual-method (covQual-class), 41
show,covRadial-method (covRadial-class), 50
show,covTP-method (covTP-class), 54
show,covTS-method (covTS-class), 57
simulate, covAll-method, 101
simulate,covAll-method (simulate, covAll-method), 101
simulate,covOrd-method (covOrd-class), 39
simulate,covQual-method (covQual-class), 41
simulate.gp, *91*, 102
simulPar, *77*, 105
simulPar,covAll-method, 106
symIndices, 106

translude, 107

varVec, 108
varVec,covANOVA-method (covANOVA-class), 25
varVec,covComp-method (covComp-class), 28
varVec,covMan-method (varVec-methods), 108
varVec,covOrd-method (covOrd-class), 39
varVec,covQual-method (covQual-class), 41
varVec,covRadial-method (covRadial-class), 50
varVec,covTP-method (covTP-class), 54
varVec,covTS-method (varVec-methods), 108
varVec-methods, 108

warpFun, *37, 38*
warpFun (warpNorm), 110
warpNorm, 110
warpPower (warpNorm), 110
warpSpline1 (warpNorm), 110
warpSpline2 (warpNorm), 110
warpUnorm (warpNorm), 110