Package 'lineqGPR'

July 31, 2018

Type Package

Title Gaussian Process Regression Models with Linear Inequality Constraints

Version 0.0.3

Date 2018-07-11

Author Andres Felipe LOPEZ-LOPERA

Maintainer Andres Felipe LOPEZ-LOPERA <andres-felipe.lopez@emse.fr>

Description Gaussian processes regression models with linear inequality constraints.

Note internal package of the Chair OQUAIDO.

License GPL-3

Depends stats, nloptr, broom, tmg, mvtnorm

Imports MASS, quadprog, Matrix, restrictedMVN, TruncatedNormal, graphics, grDevices, ggplot2

Suggests Rcpp (>= 0.10.5), testthat, DiceDesign, doMC, viridis, tikzDevice

RoxygenNote 6.0.1

NeedsCompilation no

R topics documented:

`lineqGPR-package` *Gaussian Processes with Linear Inequality Constraints*

Description

Package for Gaussian process interpolation, regression and simulation under linear inequality constraints based on (López-Lopera et al., 2017). The constrained models are given as objects with "lineqGP" S3 class. Implementations according to (Maatouk and Bay, 2017) are also provided as objects with "lineqDGP" S3 class.

Details

Package:	lineqGPR
Type:	Package
Title:	Gaussian Process Regression Models with Linear Inequality Constraints
Version:	0.0.3
Date:	2018-07-11
Author:	Andres Felipe LOPEZ-LOPERA
Maintainer:	Andres Felipe LOPEZ-LOPERA <andres-felipe.lopez@emse.fr>
Description:	Gaussian processes regression models with linear inequality constraints.
Note:	internal package of the Chair OQUAIDO.
License:	GPL-3
Depends:	stats, nloptr, broom, tmg, mvtnorm
Imports:	MASS, quadprog, Matrix, restrictedMVN, TruncatedNormal, graphics, grDevices, ggplot2

Suggests:	Rcpp (>= 0.10.5), testthat, DiceDesign, doMC, viridis, tikzDevice
RoxygenNote:	6.0.1
NeedsCompilation:	no

Warning

lineqGPR may strongly evolve in the future in order to incorporate other packages for Gaussian process regression modelling (see, e.g., **kergp**, **DiceKriging**, **DiceDesign**). It could be also scaled to higher dimensions and for a large number of observations.

Note

This package was developed within the frame of the Chair in Applied Mathematics OQUAIDO, gathering partners in technological research (BRGM, CEA, IFPEN, IRSN, Safran, Storengy) and academia (CNRS, Ecole Centrale de Lyon, Mines Saint-Etienne, University of Grenoble, University of Nice, University of Toulouse) around advanced methods for Computer Experiments.

	Important functions or methods
`create`	Creation function of GP models under linear inequality constraints.
`augment`	Augmentation of GP models according to local and covariance parameters.
`predict`	Prediction of the objective function at new points using a Kriging model under linear inequality constraints.
`simulate`	Simulation of kriging models under linear inequality constraints.
`plot`	Plot for a constrained Kriging model.
`ggplot`	GGPlot for a constrained Kriging model.

Author(s)

Andrés Felipe López-Lopera (Mines Saint-Étienne) with contributions from Olivier Roustant (Mines Saint-Étienne) and Yves Deville (Alpestat).

Maintainer: Andrés Felipe López-Lopera, <andres-felipe.lopez@emse.fr>

References

López-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

Bachoc, F., Lagnoux, A., and Lopez-Lopera, A. F. (2018), "Maximum likelihood estimation for Gaussian processes under inequality constraints". *ArXiv e-prints* [link]

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5): 557-582. [link]

Roustant, O., Ginsbourger, D., and Deville, Y. (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 regression modelling under boundedness constraint
## ------------------------------------------------------------------
library(lineqGPR)

#### generating the synthetic data ####
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, 0.001)
y <- sigfun(x)
DoE <- splitDoE(x, y, DoE.idx = c(201, 501, 801))

#### GP with inactive boundedness constraints ####
# creating the "lineqGP" model
model <- create(class = "lineqGP", DoE$xdesign, DoE$ydesign,
                constrType = c("boundedness"))
model$localParam$m <- 100
model$bounds <- c(-10,10)
model <- augment(model)

# sampling from the model
sim.model <- simulate(model, nsim = 1e3, seed = 1, xtest = DoE$xtest)
plot(sim.model, xlab = "x", ylab = "y(x)", ylim = range(y),
     main = "Unconstrained GP model")
lines(x, y, lty = 2)
legend("topleft", c("ytrain","ytest","mean","confidence"),
       lty = c(NaN,2,1,NaN), pch = c(20,NaN,NaN,15),
       col = c("black","black","darkgreen","gray80"))

#### GP with active boundedness constraints ####
# creating the "lineqGP" model
model <- create(class = "lineqGP", DoE$xdesign, DoE$ydesign,
                constrType = c("boundedness"))
model$localParam$m <- 100
model$bounds <- c(0,1)
model <- augment(model)

# sampling from the model
sim.model <- simulate(model, nsim = 1e3, seed = 1, xtest = DoE$xtest)
plot(sim.model, bounds = model$bounds,
     xlab = "x", ylab = "y(x)", ylim = range(y),
     main = "Constrained GP model under boundedness conditions")
lines(x, y, lty = 2)
legend("topleft", c("ytrain","ytest","mean","confidence"),
       lty = c(NaN,2,1,NaN), pch = c(20,NaN,NaN,15),
       col = c("black","black","darkgreen","gray80"))


## ------------------------------------------------------------------
## Gaussian process regression modelling under multiple constraints
## ------------------------------------------------------------------
library(lineqGPR)

#### generating the synthetic data ####
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, 0.001)
```

```
y <- sigfun(x)
DoE <- splitDoE(x, y, DoE.idx = c(201, 501, 801))

#### GP with boundedness and monotonicity constraints ####
# creating the "lineqGP" model
model <- create(class = "lineqGP", DoE$xdesign, DoE$ydesign,
                constrType = c("boundedness","monotonicity"))
model$localParam$m <- 50
model$bounds[1, ] <- c(0,1)
model <- augment(model)

# sampling from the model
sim.model <- simulate(model, nsim = 1e2, seed = 1, xtest = DoE$xtest)
plot(sim.model, bounds = model$bounds,
     xlab = "x", ylab = "y(x)", ylim = range(y),
     main = "Constrained GP model under boundedness & monotonicity conditions")
lines(x, y, lty = 2)
legend("topleft", c("ytrain","ytest","mean","confidence"),
       lty = c(NaN,2,1,NaN), pch = c(20,NaN,NaN,15),
       col = c("black","black","darkgreen","gray80"))


## ---------------------------------------------------------------------
## Gaussian process regression modelling under linear constraints
## ---------------------------------------------------------------------
library(lineqGPR)
library(Matrix)

#### generating the synthetic data ####
targetFun <- function(x){
  y <- rep(1, length(x))
  y[x <= 0.4] <- 2.5*x[x <= 0.4]
  return(y)
}
x <- seq(0, 1, by = 0.001)
y <- targetFun(x)
DoE <- splitDoE(x, y, DoE.idx = c(101, 301, 501, 701))

#### GP with predefined linear inequality constraints ####
# creating the "lineqGP" model
model <- create(class = "lineqGP", DoE$xdesign, DoE$ydesign,
                constrType = c("linear"))
m <- model$localParam$m <- 100

# building the predefined linear constraints
bounds1 <- c(0,Inf)
LambdaB1 <- diag(2*m/5)
LambdaM <- diag(2*m/5)
LambdaB2 <- diag(3*m/5)
lsys <- lineqGPSys(m = 2*m/5, constrType = "monotonicity",
                   l = bounds1[1], u = bounds1[2], lineqSysType = "oneside")
LambdaM[-seq(1),] <- lsys$M
model$Lambda <- as.matrix(bdiag(rbind(LambdaM,LambdaB1),LambdaB2))
model$lb <- c(-Inf, rep(0, 2*m/5-1), rep(0, 2*m/5), rep(0.85, 3*m/5))
model$ub <- c(rep(0.1, 2*m/5), rep(1.1, 2*m/5), rep(1.1, 3*m/5))
model <- augment(model)
```

```
# sampling from the model
sim.model <- simulate(model, nsim = 1e3, seed = 1, xtest = DoE$xtest)
plot(sim.model, bounds = c(0,1.1),
     xlab = "x", ylab = "y(x)", ylim = c(0,1.1),
     main = "Constrained GP model under linear conditions")
lines(x, y, lty = 2)
abline(v = 0.4, lty = 2)
lines(c(0.4, 1), rep(0.85, 2), lty = 2)
legend("bottomright", c("ytrain","ytest","mean","confidence"),
       lty = c(NaN,2,1,NaN), pch = c(20,NaN,NaN,15),
       col = c("black","black","darkgreen","gray80"))


## ------------------------------------------------------------------
## Note:
## 1. More examples are given as demos (run: demo(package="lineqGPR")).
## 2. See also the examples from inner functions of the package
## (run: help("simulate.lineqGP")).
## ------------------------------------------------------------------
```

`augment.lineqDGP` *Augmenting Method for the* `"lineqDGP"` *S3 Class*

Description

Augmenting method for the `"lineqDGP"` S3 class.

Usage

```
## S3 method for class 'lineqDGP'
augment(x, ...)
```

Arguments

`x`	an object with class `lineqDGP`.
`...`	further arguments passed to or from other methods.

Value

An expanded `"lineqDGP"` object with the following additional elements.

`Phi`	a matrix corresponding to the hat basis functions. The basis functions are indexed by rows.
`Gamma`	the covariance matrix of the Gassian vector $\boldsymbol{\xi}$.
`(Lambda,lb,ub)`	the linear system of inequalities.
`...`	further parameters passed to or from other methods.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`create.lineqDGP`, `predict.lineqDGP`, `simulate.lineqDGP`

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqDGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model2 <- augment(model)
image(model2$Gamma, main = "covariance matrix")
```

`augment.lineqGP` *Augmenting Method for the* `"lineqGP"` *S3 Class*

Description

Augmenting method for the `"lineqGP"` S3 class.

Usage

```
## S3 method for class 'lineqGP'
augment(x, ...)
```

Arguments

`x`	an object with class `lineqGP`.
`...`	further arguments passed to or from other methods.

Details

Some paramaters of the finite-dimensional GP with linear inequality constraints are computed. Here, $\boldsymbol{\xi}$ is a centred Gaussian vector with covariance $\boldsymbol{\Gamma}$, s.t. $\boldsymbol{\Phi}\boldsymbol{\xi} = \boldsymbol{y}$ (interpolation constraints) and $\boldsymbol{l} \leq \boldsymbol{\Lambda}\boldsymbol{\xi} \leq \boldsymbol{u}$ (inequality constraints).

Value

An expanded "lineqGP" object with the following additional elements.

Phi	a matrix corresponding to the hat basis functions. The basis functions are indexed by rows.
Gamma	the covariance matrix of the Gassian vector ξ.
(Lambda,lb,ub)	the linear system of inequalities.
...	further parameters passed to or from other methods.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

See Also

create.lineqGP, predict.lineqGP, simulate.lineqGP

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model2 <- augment(model)
image(model2$Gamma, main = "covariance matrix")
```

---

basisCompute.lineqDGP *Basis Functions for "lineqDGP" Models*

Description

Evaluate the basis functions for "lineqDGP" models.

Usage

```
basisCompute.lineqDGP(x, m, d = 1, constrType = c("boundedness",
  "monotonicity", "convexity"))
```

Arguments

`x`	a vector with the input data.
`m`	the number of basis functions used in the approximation.
`d`	a number corresponding to the dimension of the input space.
`constrType`	a character string corresponding to the type of the inequality constraint. Options: "boundedness", "monotonicity", "convexity".

Value

A matrix with the basis functions. The basis functions are indexed by rows.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

Examples

```
x <- seq(0, 1, 0.01)
Phib <- basisCompute.lineqDGP(x, m = 5, constrType = "boundedness")
matplot(Phib, type = "l", lty = 2,
        main = "Basis functions for boundedness constraints")
Phim <- basisCompute.lineqDGP(x, m = 5, constrType = "monotonicity")
matplot(Phim, type = "l", lty = 2,
        main = "Basis functions for monotonicity constraints")
Phic <- basisCompute.lineqDGP(x, m = 5, constrType = "convexity")
matplot(Phic, type = "l", lty = 2,
        main = "Basis functions for convexity constraints")
```

`basisCompute.lineqGP` *Hat Basis Functions for* `"lineqGP"` *Models*

Description

Evaluate the hat basis functions for `"lineqGP"` models.

Usage

```
basisCompute.lineqGP(x, u, d = 1)
```

Arguments

`x`	a vector (or matrix) with the input data.
`u`	a vector (or matrix) with the locations of the knots.
`d`	a number corresponding to the dimension of the input space.

Value

A matrix with the hat basis functions. The basis functions are indexed by rows.

Comments

This function was tested mainly for 1D or 2D input spaces. It could change in future versions for higher dimensions.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5): 557-582. [link]

Examples

```
x <- seq(0, 1, 0.01)
m <- 5
u <- seq(0, 1, 1/(m-1))
Phi <- basisCompute.lineqGP(x, u, d = 1)
matplot(Phi, type = "l", lty = 2, main = "Hat basis functions with m = 5")
```

---

bounds2lineqSys *Linear Systems of Inequalities*

Description

Build the linear system of inequalities given specific bounds.

Usage

```
bounds2lineqSys(d = nrow(A), l = 0, u = 1, A = diag(d),
  lineqSysType = "twosides", rmInf = TRUE)
```

Arguments

d	the number of linear inequality constraints.
l	the value (or vector) with the lower bound.
u	the value (or vector) with the upper bound.
A	a matrix containing the structure of the linear equations.

lineqSysType a character string corresponding to the type of the linear system. Options: `twosides`, `oneside`.
- `twosides` : Linear system given by

$$\boldsymbol{l} \leq \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{u}.$$

- `oneside` : Extended linear system given by

$$\boldsymbol{M}\boldsymbol{x} + \boldsymbol{g} \geq \boldsymbol{0} \quad \text{with} \quad \boldsymbol{M} = [\boldsymbol{A}, -\boldsymbol{A}]^\top \quad \text{and} \quad \boldsymbol{g} = [-\boldsymbol{l}, \boldsymbol{u}]^\top.$$

rmInf If `TRUE`, inactive constraints are removed (e.g. $-\infty \leq x \leq \infty$).

Value

A list with the linear system of inequalities: `list(A,l,u)` (`twosides`) or `list(M,g)` (`oneside`).

Author(s)

A. F. Lopez-Lopera.

Examples

```
n <- 5
A <- diag(n)
l <- rep(0, n)
u <- c(Inf, rep(1, n-1))
bounds2lineqSys(n, l, u, A, lineqSysType = "twosides")
bounds2lineqSys(n, l, u, A, lineqSysType = "oneside", rmInf = FALSE)
bounds2lineqSys(n, l, u, A, lineqSysType = "oneside", rmInf = TRUE)
```

---

constrlogLikFun *Log-Constrained-Likelihood of a Gaussian Process.*

Description

Compute the negative log-constrained-likelihood of a Gaussian Process conditionally to the inequality constraints (Lopez-Lopera et al., 2017).

Usage

```
constrlogLikFun(par = model$kernParam$par, model, parfixed = NULL,
  mcmc.opts = list(probe = c("Genz"), nb.mcmc = 1000),
  estim.varnoise = FALSE)
```

Arguments

par	the values of the covariance parameters.
model	an object with `"lineqGP"` S3 class.
parfixed	not used.
mcmc.opts	mcmc options. `mcmc.opts$probe` A character string corresponding to the estimator for the orthant multinormal probabilities. Options: `"Genz"` (Genz, 1992), `"ExpT"` (Botev, 2017). If `probe == "ExpT"`, `mcmc.opts$nb.mcmc` is the number of MCMC samples used for the estimation.
estim.varnoise	If `true`, a noise variance is estimated.

Details

Orthant multinormal probabilities are estimated according to (Genz, 1992; Botev, 2017). See (Lopez-Lopera et al., 2017).

Value

The value of the negative log-constrained-likelihood.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

Bachoc, F., Lagnoux, A., and Lopez-Lopera, A. F. (2018), "Maximum likelihood estimation for Gaussian processes under inequality constraints". *ArXiv e-prints* [link]

Genz, A. (1992), "Numerical computation of multivariate normal probabilities". *Journal of Computational and Graphical Statistics*, 1:141-150. [link]

Botev, Z. I. (2017), "The normal law under linear restrictions: simulation and estimation via minimax tilting". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(1):125-148. [link]

See Also

`constrlogLikGrad`, `logLikFun`, `logLikGrad`

`constrlogLikGrad` *Numerical Gradient of the Log-Constrained-Likelihood of a Gaussian Process.*

Description

Compute the gradient numerically of the negative log-constrained-likelihood of a Gaussian Process conditionally to the inequality constraints (Lopez-Lopera et al., 2017).

Usage

```
constrlogLikGrad(par = model$kernParam$par, model, parfixed = rep(FALSE,
  length(par)), mcmc.opts = list(probe = "Genz", nb.mcmc = 1000),
  estim.varnoise = FALSE)
```

Arguments

`par`	the values of the covariance parameters.
`model`	an object with class `lineqGP`.
`parfixed`	indices of fixed parameters to do not be optimised.

`mcmc.opts` mcmc options. `mcmc.opts$probe` A character string corresponding to the estimator for the orthant multinormal probabilities. Options: `"Genz"` (Genz, 1992), `"ExpT"` (Botev, 2017). If `probe == "ExpT"`, `mcmc.opts$nb.mcmc` is the number of MCMC samples used for the estimation.

`estim.varnoise` If `true`, a noise variance is estimated.

Details

Orthant multinormal probabilities are estimated via (Genz, 1992; Botev, 2017).

Value

The gradient of the negative log-constrained-likelihood.

Comments

As orthant multinormal probabilities don't have explicit expressions, the gradient is implemented numerically based on `nl.grad`.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

Bachoc, F., Lagnoux, A., and Lopez-Lopera, A. F. (2018), "Maximum likelihood estimation for Gaussian processes under inequality constraints". *ArXiv e-prints* [link]

Genz, A. (1992), "Numerical computation of multivariate normal probabilities". *Journal of Computational and Graphical Statistics*, 1:141-150. [link]

Botev, Z. I. (2017), "The normal law under linear restrictions: simulation and estimation via minimax tilting". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(1):125-148. [link]

See Also

`constrlogLikFun`, `logLikFun`, `logLikGrad`

`create` *Model Creations*

Description

Wrapper function for creations of model functions. The function invokes particular methods which depend on the `class` of the first argument.

Usage

```
create(class, ...)
```

Arguments

`class`	a character string corresponding to the desired class.
`...`	further arguments passed to or from other methods. (see, e.g., `create.lineqGP`)

Value

A model object created according to its `class`.

Author(s)

A. F. Lopez-Lopera.

See Also

`augment`, `predict`, `simulate`

Examples

```
## Not run:
model <- list()
model2 <- create(class = "ClassName", model)
model2
## End(Not run)
```

---

create.lineqDGP *Creation Method for the "lineqDGP" S3 Class*

Description

Creation method for the `"lineqDGP"` S3 class.

Usage

```
## S3 method for class 'lineqDGP'
create(x, y, constrType = c("boundedness", "monotonicity",
  "convexity"))
```

Arguments

`x`	a vector or matrix with the input data. The dimensions should be indexed by columns.
`y`	a vector with the output data.
`constrType`	a character string (or list) corresponding to the type(s) of inequality constraint(s). Options: "boundedness", "monotonicity", "convexity", "linear".

Value

A list with the following elements.

`x,y,constrType`	see **Arguments**.
`d`	a number corresponding to the input dimension.
`localParam`	a list with specific parameters required for `"lineqGP"` models: `m` (number of basis functions), `sampler`, and `samplingParams`. See `simulate.lineqDGP`.
`kernParam`	a list with the kernel parameters: `par` (kernel parameters), `type`, `nugget`. See `kernCompute`
`bounds`	the limit values if `constrType = "boundedness"`.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`augment.lineqDGP`, `predict.lineqDGP`, `simulate.lineqDGP`

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqDGP", x, y, constrType = "monotonicity")
model
```

`create.lineqGP` *Creation Method for the* `"lineqGP"` *S3 Class*

Description

Creation method for the `"lineqGP"` S3 class.

Usage

```
## S3 method for class 'lineqGP'
create(x, y, constrType)
```

Arguments

`x`	a vector or matrix with the input data. The dimensions should be indexed by columns.
`y`	a vector with the output data.
`constrType`	a character string corresponding to the type of the inequality constraint. Options: "boundedness", "monotonicity", "convexity", "linear"; Multiple constraints can be also defined, e.g. `constrType = c("boundedness", "monotonicity")`.

Value

A list with the following elements.

`x,y,constrType`	see **Arguments**.
`d`	a number corresponding to the input dimension.
`constrIdx`	for d > 1, a logical vector with the indices of active constrained dimensions.
`localParam`	a list with specific parameters required for `"lineqGP"` models: `m` (number of basis functions), `sampler`, and `samplingParams`. See `simulate.lineqGP`.
`kernParam`	a list with the kernel parameters: `par` (kernel parameters), `type`, `nugget`. See `kernCompute`
`bounds`	the limit values if `constrType = "boundedness"`.
`(Lambda,lb,ub)`	the linear system of inequalities if `constrType = "linear"`.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

See Also

`augment.lineqGP`, `predict.lineqGP`, `simulate.lineqGP`

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqGP", x, y, constrType = "monotonicity")
model
```

errorMeasureRegress *Error Measures for GP Models.*

Description

Compute error measures for GP models: mean absulte error ("mae"), mean squared error ("mse"), standardised mse ("smse"), mean standardised log loss ("msll"), Q2 ("q2"), predictive variance adequation ("pva"), confidence interval accuracy ("cia").

Usage

```
errorMeasureRegress(y, ytest, mu, varsigma, type = "all",
  control = list(nsigma = 1.96))
```

Arguments

y	a vector with the output observations used for training.
ytest	a vector with the output observations used for testing.
mu	a vector with the posterior mean.
varsigma	a vector with the posterior variances.
type	a character string corresponding to the type of the measure.
control	an optional list with parameters to be passed (e.g. cia: "nsigma").

Value

The values of the error measures.

Author(s)

A. F. Lopez-Lopera.

References

Rasmussen, C. E. and Williams, C. K. I. (2005), "Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)". *The MIT Press*. [link]

Bachoc, F. (2013), "Cross validation and maximum likelihood estimations of hyper-parameters of Gaussian processes with model misspecification". *Computational Statistics & Data Analysis*, 66:55-69. [link]

See Also

errorMeasureRegressMC

Examples

```
# generating the toy example
n <- 100
w <- 4*pi
x <- seq(0, 1, length = n)
y <- sin(w*x)

# results with high-level noises generating the toy example
nbsamples <- 100
set.seed(1)
ynoise <- y + matrix(rnorm(n*nbsamples, 0, 10), ncol = nbsamples)
mu <- apply(ynoise, 1, mean)
sigma <- apply(ynoise, 1, sd)
matplot(x, ynoise, type = "l", col = "gray70")
lines(x, y, lty = 2, col = "red")
lines(x, mu, col = "blue")
lines(x, mu+1.98*sigma, lty = 2)
lines(x, mu-1.98*sigma, lty = 2)
legend("topright", c("target", "mean", "confidence", "samples"),
       lty = c(2,1,2,1), col = c("red", "blue", "black", "gray70"))
t(errorMeasureRegress(y, y, mu, sigma^2))

# results with low-level noises generating the toy example
set.seed(1)
ynoise <- y + matrix(rnorm(n*nbsamples, 0, 0.05), ncol = nbsamples)
mu <- apply(ynoise, 1, mean)
sigma <- apply(ynoise, 1, sd)
matplot(x, ynoise, type = "l", col = "gray70")
lines(x, y, lty = 2, col = "red")
lines(x, mu, col = "blue")
lines(x, mu+1.98*sigma, lty = 2)
lines(x, mu-1.98*sigma, lty = 2)
legend("topright", c("target", "mean", "confidence", "samples"),
       lty = c(2,1,2,1), col = c("red", "blue", "black", "gray70"))
t(errorMeasureRegress(y, y, mu, sigma^2))
```

---

errorMeasureRegressMC *Error Measures for GP Models using Monte Carlo Samples.*

Description

Compute error measures for GP models using Monte Carlo samples: mean absulte error ("mae"), mean squared error ("mse"), standardised mse ("smse"), Q2 ("q2"), predictive variance adequation ("pva"), confidence interval accuracy ("cia").

Usage

```
errorMeasureRegressMC(y, ytest, ysamples, type = "all", control = list(probs
  = c(0.05, 0.95)))
```

Arguments

y	a vector with the output observations used for training.
ytest	a vector with the output observations used for testing.
ysamples	a matrix with posterior sample paths. Samples are indexed by columns.
type	a character string corresponding to the type of the measure.
control	an optional list with parameters to be passed (cia: "probs").

Value

The values of the error measures.

Author(s)

A. F. Lopez-Lopera.

References

Rasmussen, C. E. and Williams, C. K. I. (2005), "Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)". *The MIT Press*. [link]

Bachoc, F. (2013), "Cross validation and maximum likelihood estimations of hyper-parameters of Gaussian processes with model misspecification". *Computational Statistics & Data Analysis*, 66:55-69. [link]

See Also

errorMeasureRegress

Examples

```
# generating the toy example
n <- 100
w <- 4*pi
x <- seq(0, 1, length = n)
y <- sin(w*x)

# results with high-level noises generating the toy example
nbsamples <- 100
set.seed(1)
ynoise <- y + matrix(rnorm(n*nbsamples, 0, 10), ncol = nbsamples)
matplot(x, ynoise, type = "l", col = "gray70")
lines(x, y, lty = 2, col = "red")
legend("topright", c("target", "samples"), lty = c(2,1), col = c("red", "gray70"))
t(errorMeasureRegressMC(y, y, ynoise))

# results with low-level noises generating the toy example
set.seed(1)
ynoise <- y + matrix(rnorm(n*nbsamples, 0, 0.05), ncol = nbsamples)
matplot(x, ynoise, type = "l", col = "gray70")
lines(x, y, lty = 2, col = "red")
legend("topright", c("target", "samples"), lty = c(2,1), col = c("red", "gray70"))
t(errorMeasureRegressMC(y, y, ynoise))
```

ggplot.lineqDGP *GGPlot for the* "lineqDGP" *S3 Class*

Description

GGPlot for the "lineqDGP" S3 class. See ggplot.lineqGP for more details.

Usage

```
## S3 method for class 'lineqDGP'
ggplot(data, mapping, ...)
```

Arguments

data	an object with lineqDGP S3 class.
mapping	not used.
...	further arguments passed to or from other methods.

Value

GGPlot with the "lineqDGP" model.

Author(s)

A. F. Lopez-Lopera.

See Also

ggplot.lineqGP, ggplot

ggplot.lineqGP *GGPlot for the* "lineqGP" *S3 Class*

Description

GGPlot for the "lineqGP" S3 class.

Usage

```
## S3 method for class 'lineqGP'
ggplot(data, mapping, ytest = NULL, probs = c(0.05, 0.95),
  bounds = NULL, addlines = TRUE, nblines = 5, fillbackground = TRUE,
  alpha.qtls = 0.4, xlab = "", ylab = "", main = "", xlim = NULL,
  ylim = NULL, lwd = 1, cex = 1.5, ...)
```

Arguments

`data`	an object with `"lineqGP"` S3 class.
`mapping`	not used.
`ytest`	the values of the test observations. `If !is.null(ytest)`, ytest is drawn.
`probs`	the values of the confidence intervals evaluated at probs.
`bounds`	the values of the bounds of a constrained model. If `!is.null(bounds)`, bounds are drawn.
`addlines`	an optional Logical. If `TRUE`, some samples are drawn.
`nblines`	if `addlines`. The number of samples to be drawn.
`fillbackground`	an optional logical. If `TRUE`, fill gray background.
`alpha.qtls`	a number indicating the transparency of the quantiles.
`xlab`	a character string corresponding to the title for the x axis.
`ylab`	a character string corresponding to the title for the y axis.
`main`	a character string corresponding to the overall title for the plot.
`xlim`	the limit values for the x axis.
`ylim`	the limit values for the y axis.
`lwd`	a number indicating the line width.
`cex`	a number indicating the amount by which plotting text and symbols should be scaled.
`...`	further arguments passed to or from other methods.

Value

GGPlot with the `"lineqGP"` model.

Author(s)

A. F. Lopez-Lopera.

See Also

`ggplot`, `plot.lineqGP`

`ineqConstrKernCompute` *Kernel Matrix for* `"lineqDGP"` *Models.*

Description

Compute the kernel matrix for `"lineqDGP"` models.

Usage

```
ineqConstrKernCompute(u, constrType = c("boundedness", "monotonicity",
  "convexity"), kernType, par, d = 1L)
```

Arguments

`u`	a discretization vector of the input locations.
`constrType`	a character string corresponding to the type of the inequality constraint. Options: "boundedness", "monotonicity", "convexity".
`kernType`	a character string corresponding to the type of the kernel. Options: "gaussian", "matern32", "matern52", "exponential".
`par`	the values of the kernel parameters (variance, lengthscale).
`d`	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(u, u)$

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`kernCompute`

Examples

```
x <- seq(0, 1, 0.01)
par <- c(1, 0.1)
Kb <- ineqConstrKernCompute(x, constrType = "boundedness", kernType = "gaussian", par)
image(Kb, main = "covariance matrix for boundedness constraints")
Km <- ineqConstrKernCompute(x, constrType = "monotonicity", kernType = "gaussian", par)
image(Km, main = "covariance matrix for monotonicity constraints")
Kc <- ineqConstrKernCompute(x, constrType = "convexity", kernType = "gaussian", par)
image(Kc, , main = "covariance matrix for convexity constraints")
```

`k1exponential` *1D Exponential Kernel Matrix for* `"lineqGP"` *Models.*

Description

Compute the 1D Exponential kernel for `"lineqGP"` models. attr: "gradient".

Usage

```
k1exponential(x1, x2, par, d = 1)
```

Arguments

`x1`	a vector with the first input locations.
`x2`	a vector with the second input locations.
`par`	the values of the kernel parameters (variance, lengthscale).
`d`	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
x <- seq(0, 1, 0.01)
K <- k1exponential(x, x, par =  c(1, 0.1))
image(K, main = "covariance matrix using a Exponential kernel")
```

`k1gaussian` *1D Gaussian Kernel Matrix for* `"lineqGP"` *Models.*

Description

Compute the 1D Gaussian kernel matrix for `"lineqGP"` models. attr: "gradient", "derivative".

Usage

```
k1gaussian(x1, x2, par, d = 1)
```

Arguments

`x1`	a vector with the first input locations.
`x2`	a vector with the second input locations.
`par`	the values of the kernel parameters (variance, lengthscale).
`d`	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
x <- seq(0, 1, 0.01)
K <- k1gaussian(x, x, par =  c(1, 0.1))
image(K, main = "covariance matrix using a Squared Exponential kernel")
```

k1matern32 *1D Matern 3/2 Kernel Matrix for "lineqGP" Models.*

Description

Compute the 1D Matern 3/2 kernel for "lineqGP" models. attr: "gradient", "derivative".

Usage

```
k1matern32(x1, x2, par, d = 1)
```

Arguments

x1	a vector with the first input locations.
x2	a vector with the second input locations.
par	the values of the kernel parameters (variance, lengthscale).
d	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
x <- seq(0, 1, 0.01)
K <- k1matern32(x, x, par =  c(1, 0.1))
image(K, main = "covariance matrix using a Matern 3/2 kernel")
```

k1matern52 *1D Matern 5/2 Kernel Matrix for "lineqGP" Models.*

Description

Compute the 1D Matern 5/2 kernel for "lineqGP" models. attr: "gradient", "derivative".

Usage

```
k1matern52(x1, x2, par, d = 1)
```

Arguments

x1	A vector with the first input locations.
x2	A vector with the second input locations.
par	Values of the kernel parameters (variance, lengthscale).
d	A number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
x <- seq(0, 1, 0.01)
K <- k1matern52(x, x, par =  c(1, 0.1))
image(K, main = "covariance matrix using a Matern 5/2 kernel")
```

---

k2gaussian — *2D Gaussian Kernel Matrix for* "lineqGP" *Models.*

Description

Compute the 2D Gaussian kernel matrix for "lineqGP" models. attr: "gradient".

Usage

```
k2gaussian(x1, x2, par, d = 2)
```

Arguments

x1	a matrix with the first couple of input locations.
x2	a matrix with the second couple of input locations.
par	the values of the kernel parameters (variance, lengthscales).
d	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
xgrid <- seq(0, 1, 0.1)
x <- as.matrix(expand.grid(xgrid, xgrid))
K <- k2gaussian(x, x, par =  c(1, 0.1))
image(K, main = "covariance matrix using a 2D Gaussian kernel")
```

kernCompute *Kernel Matrix for "lineqGP" Models.*

Description

Compute the kernel matrix for "lineqGP" models. attr: "gradient".

Usage

```
kernCompute(x1, x2 = NULL, type, par, d = 1L)
```

Arguments

x1	a vector with the first input locations.
x2	a vector with the second input locations.
type	a character string corresponding to the type of the kernel. Options: "gaussian", "matern32", "matern52", "exponential".
par	the values of the kernel parameters (variance, lengthscale).
d	a number corresponding to the dimension of the input space.

Value

Kernel matrix $K(x_1, x_2)$ (or $K(x_1, x_1)$ if x_2 is not defined).

Author(s)

A. F. Lopez-Lopera.

Examples

```
x <- seq(0, 1, 0.01)
K <- kernCompute(x, type = "gaussian", par =  c(1, 0.1))
image(K, main = "covariance matrix")
```

lineqDGPSys *Linear Systems of Inequalities for "lineqDGP" Models*

Description

Build the linear system of inequalities for "lineqDGP" models.

Usage

```
lineqDGPSys(d, constrType = c("boundedness", "monotonicity", "convexity"),
  l = -Inf, u = Inf, lineqSysType = "twosides", rmInf = TRUE)
```

Arguments

d	the number of linear inequality constraints.
constrType	a character string corresponding to the type of the inequality constraint. Options: "boundedness", "monotonicity", "convexity".
l	the value (or vector) with the lower bound.
u	the value (or vector) with the upper bound.
lineqSysType	a character string corresponding to the type of the linear system. Options: `twosides`, `oneside` (see `bounds2lineqSys` for more details).
rmInf	If `TRUE`, inactive constraints are removed (e.g. $-\infty \leq x \leq \infty$).

Value

A list with the linear system of inequalities: `list(A,l,u)` (`twosides`) or `list(M,g)` (`oneside`).

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`bounds2lineqSys`

Examples

```
linSys1 <- lineqDGPSys(d = 5, constrType = "boundedness", l = 0, u = 1, lineqSysType = "twosides")
linSys1
linSys2 <- lineqDGPSys(d = 5, constrType = "boundedness", l = 0, u = 1, lineqSysType = "oneside")
linSys2
```

`lineqGPOptim` *Gaussian Process Model Optimizations*

Description

Function for optimizations of `"lineqGP"` S3 class objects.

Usage

```
lineqGPOptim(model, x0 = model$kernParam$par, eval_f = "logLik",
  lb = rep(0.01, length(par)), ub = rep(Inf, length(par)),
  opts = list(algorithm = "NLOPT_LD_MMA", print_level = 0, ftol_abs = 0.001,
  maxeval = 50, check_derivatives = FALSE, parfixed = rep(FALSE, length(par)),
  estim.varnoise = FALSE, bounds.varnoise = c(0, Inf)), add.constr = FALSE,
  mcmc.opts = list(probe = "Genz", nb.mcmc = 1000), max.trials = 10, ...)
```

Arguments

`model`	a list with the structure of the constrained Kriging model.
`x0`	the initial values for the parameters to be optimized over.
`eval_f`	a function to be minimized, with first argument the vector of parameters over which minimization is to take place. It should return a scalar result.
`lb`	a vector with lower bounds of the params. The params are forced to be positive. See `nloptr`.
`ub`	a vector with upper bounds of the params. See `nloptr`.
`opts`	see `nl.opts`. Parameter `parfixed` indices of fixed parameters to do not be optimised. If `estim.varnoise` is true, the noise variance is estimated.
`add.constr`	an optional logical. If `TRUE`, the inequality constraints are taken into account in the optimisation.
`mcmc.opts`	if `add.constr`, mcmc options passed to methods.
`max.trials`	the value of the maximum number of trials when errors are produced by instabilities.
`...`	further arguments passed to or from other methods.

Value

An optimized `lineqGP` model.

Comments

This function has to be improved in the future for more stable procedures. Cros-validation (CV) methods could be implemented in future versions.

Author(s)

A. F. Lopez-Lopera.

See Also

`nloptr`

`lineqGPSys` *Linear Systems of Inequalities for "`lineqGP`" Models*

Description

Build the linear system of inequalities for "`lineqGP`" models.

Usage

```
lineqGPSys(m = nrow(A), constrType = c("boundedness", "monotonicity",
  "convexity", "linear"), l = -Inf, u = Inf, A = diag(m), d = length(m),
  lineqSysType = "twosides", constrIdx = seq(length(m)), rmInf = TRUE)
```

Arguments

`m`	the number of linear inequality constraints.
`constrType`	a character string corresponding to the type of the inequality constraint. Options: "boundedness", "monotonicity", "convexity", "linear"
`l`	the value (or vector) with the lower bound.
`u`	the value (or vector) with the upper bound.
`A`	a matrix containing the structure of the linear equations.
`d`	the value with the input dimension.
`lineqSysType`	a character string corresponding to the type of the linear system. Options: `twosides`, `oneside` (see `bounds2lineqSys` for more details).
`constrIdx`	for d > 1, a logical vector with the indices of active constrained dimensions.
`rmInf`	If `TRUE`, inactive constraints are removed (e.g. $-\infty \leq x \leq \infty$).

Value

A list with the linear system of inequalities: `list(A,l,u)` (`twosides`) or `list(M,g)` (`oneside`).

Comments

This function could change in future versions for more types of inequality constraints in higher dimensions.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

See Also

`bounds2lineqSys`

Examples

```
linSys1 <- lineqGPSys(m = 5, constrType = "boundedness", l = 0, u = 1, lineqSysType = "twosides")
linSys1
linSys2 <- lineqGPSys(m = 5, constrType = "boundedness", l = 0, u = 1, lineqSysType = "oneside")
linSys2
```

logLikFun — *Log-Likelihood of a Gaussian Process.*

Description

Compute the negative log-likelihood of a Gaussian Process.

Usage

```
logLikFun(par = model$kernParam$par, model, parfixed = NULL,
  mcmc.opts = NULL, estim.varnoise = FALSE)
```

Arguments

`par`	the values of the covariance parameters.
`model`	an object with `"lineqGP"` S3 class.
`parfixed`	not used.
`mcmc.opts`	not used.
`estim.varnoise`	If `true`, a noise variance is estimated.

Value

The value of the negative log-likelihood.

Author(s)

A. F. Lopez-Lopera.

References

Rasmussen, C. E. and Williams, C. K. I. (2005), "Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)". *The MIT Press*. [link]

See Also

`logLikGrad`, `constrlogLikFun`, `constrlogLikGrad`

logLikGrad — *Gradient of the Log-Likelihood of a Gaussian Process.*

Description

Compute the gradient numerically of the negative log-likelihood of a Gaussian Process.

Usage

```
logLikGrad(par = model$kernParam$par, model, parfixed = rep(FALSE,
  length(par)), mcmc.opts = NULL, estim.varnoise = FALSE)
```

Arguments

`par`	the values of the covariance parameters.
`model`	an object with `"lineqGP"` S3 class.
`parfixed`	indices of fixed parameters to do not be optimised.
`mcmc.opts`	not used.
`estim.varnoise`	If `true`, a noise variance is estimated.

Value

the gradient of the negative log-likelihood.

Author(s)

A. F. Lopez-Lopera.

References

Rasmussen, C. E. and Williams, C. K. I. (2005), "Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)". *The MIT Press*. [link]

See Also

`logLikFun`, `constrlogLikFun`, `constrlogLikGrad`

`plot.lineqDGP` *Plot for the* `"lineqDGP"` *S3 Class*

Description

Plot for the `"lineqDGP"` S3 class. See `plot.lineqGP` for more details.

Usage

```
## S3 method for class 'lineqDGP'
plot(x, y, ...)
```

Arguments

`x`	an object with `"lineqDGP"` S3 class.
`y`	not used.
`...`	further arguments passed to or from other methods.

Value

Plot with the `"lineqDGP"` model.

Author(s)

A. F. Lopez-Lopera.

See Also

plot.lineqGP, plot

---

plot.lineqGP *Plot for the "lineqGP" S3 Class*

Description

Plot for the "lineqGP" S3 class.

Usage

```
## S3 method for class 'lineqGP'
plot(x, y, ytest = NULL, probs = c(0.05, 0.95),
  bounds = NULL, addlines = TRUE, nblines = 5, ...)
```

Arguments

x	an object with "lineqGP" S3 class.
y	not used.
ytest	the values of the test observations. If !is.null(ytest), ytest is drawn.
probs	the values of the confidence intervals evaluated at probs.
bounds	the values of the bounds of a constrained model. If !is.null(bounds), bounds are drawn.
addlines	pptional Logical. If TRUE, some samples are drawn.
nblines	if addlines. The number of samples to be drawn.
...	further arguments passed to or from other methods.

Value

Plot with the "lineqGP" model.

Author(s)

A. F. Lopez-Lopera.

See Also

plot, ggplot.lineqGP

predict.lineqDGP *Prediction Method for the* "lineqDGP" *S3 Class*

Description

Prediction method for the "lineqDGP" S3 class.

Usage

```
## S3 method for class 'lineqDGP'
predict(object, xtest, ...)
```

Arguments

object	an object with class "lineqDGP".
xtest	a vector (or matrix) with the test input design.
...	further arguments passed to or from other methods.

Details

The posterior sample-path of the finite-dimensional GP with inequality constraints is computed according to (Maatouk and Bay, 2017).

Value

An object with the predictions of "lineqDGP" models.

lb	The lower bound vector of the inequalities constraints.
ub	The upper bound vector of the inequalities constraints.
Phi.test	A matrix corresponding to the hat basis functions evaluated at xtest. The basis functions are indexed by rows.
mu	The unconstrained GP mean predictor.
xi.map	The GP maximum a posteriori (MAP) predictor given the inequality constraints.
Sigma.xi	The unconstrained GP prediction conditional covariance matrix.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

create.lineqDGP, augment.lineqDGP, simulate.lineqDGP

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqDGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model <- augment(model)

# predictions from the model
xtest <- seq(0, 1, length = 100)
ytest <- sigfun(xtest)
pred <- predict(model, xtest)
plot(xtest, ytest, type = "l", lty = 2, main = "Kriging predictions")
lines(xtest, pred$Phi.test %*% pred$mu, type = "l", col = "blue")
lines(xtest, pred$Phi.test %*% pred$xi.map, type = "l", col = "red")
legend("right", c("ytest", "mean", "mode"), lty = c(2,1,1),
       col = c("black","blue","red"))
```

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

Description

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

Usage

```
## S3 method for class 'lineqGP'
predict(object, xtest, ...)
```

Arguments

`object`	an object with class `"lineqGP"`.
`xtest`	a vector (or matrix) with the test input design.
`...`	further arguments passed to or from other methods.

Details

The posterior paramaters of the finite-dimensional GP with linear inequality constraints are computed. Here, $\boldsymbol{\xi}$ is a centred Gaussian vector with covariance $\boldsymbol{\Gamma}$, s.t. $\boldsymbol{\Phi}\boldsymbol{\xi} = \boldsymbol{y}$ (interpolation constraints) and $\boldsymbol{l} \leq \boldsymbol{\Lambda}\boldsymbol{\xi} \leq \boldsymbol{u}$ (inequality constraints).

Value

A "lineqGP" object with the following elements.

Lambda	a matrix corresponding to the linear set of inequality constraints.
lb	the lower bound vector of the inequalities constraints.
ub	the upper bound vector of the inequalities constraints.
Phi.test	a matrix corresponding to the hat basis functions evaluated at xtest. The basis functions are indexed by rows.
mu	the unconstrained GP mean predictor.
Sigma	the unconstrained GP prediction conditional covariance matrix.
xi.map	the GP maximum a posteriori (MAP) predictor given the inequality constraints.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

See Also

create.lineqGP, augment.lineqGP, simulate.lineqGP

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model <- augment(model)

# predictions from the model
xtest <- seq(0, 1, length = 100)
ytest <- sigfun(xtest)
pred <- predict(model, xtest)
plot(xtest, ytest, type = "l", lty = 2, main = "Kriging predictions")
lines(xtest, pred$Phi.test %*% pred$mu, type = "l", col = "blue")
lines(xtest, pred$Phi.test %*% pred$xi.map, type = "l", col = "red")
legend("right", c("ytest", "mean", "mode"), lty = c(2,1,1),
       col = c("black", "blue", "red"))
```

`simulate.lineqDGP` *Simulation Method for the* `"lineqDGP"` *S3 Class*

Description

Simulation method for the `"lineqDGP"` S3 class.

Usage

```
## S3 method for class 'lineqDGP'
simulate(object, nsim = 1, seed = NULL, xtest, ...)
```

Arguments

`object`	an object with class `"lineqDGP"`.
`nsim`	the number of simulations.
`seed`	see `simulate`.
`xtest`	a vector (or matrix) with the test input design.
`...`	further arguments passed to or from other methods.

Details

The posterior sample-path of the finite-dimensional GP with inequality constraints is computed according to (Maatouk and Bay, 2017).

Value

An object with the simulations of `"lineqDGP"` models.

`x`	A vector (or matrix) with the training input design.
`y`	The training output vector at `x`.
`xtest`	A vector (or matrix) with the test input design.
`Phi.test`	A matrix corresponding to the hat basis functions evaluated at `xtest`. The basis functions are indexed by rows.
`xi.sim`	Posterior sample-path of the finite-dimensional Gaussian vector.
`ysim`	Posterior sample-path of the observed GP. Note: `ysim = Phi.test %*% xi.sim`.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`create.lineqDGP`, `augment.lineqDGP`, `predict.lineqDGP`

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqDGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model <- augment(model)

# sampling from the model
xtest <- seq(0, 1, length = 100)
ytest <- sigfun(xtest)
sim.model <- simulate(model, nsim = 50, seed = 1, xtest = xtest)
mu <- apply(sim.model$ysim, 1, mean)
qtls <- apply(sim.model$ysim, 1, quantile, probs = c(0.05, 0.95))
matplot(xtest, t(qtls), type = "l", lty = 1, col = "gray90",
        main = "Constrained Kriging model")
polygon(c(xtest, rev(xtest)), c(qtls[2,], rev(qtls[1,])), col = 'gray90', border = NA)
lines(xtest, ytest, lty = 2)
lines(xtest, mu, type = "l", col = "darkgreen")
points(x, y, pch = 20)
legend("right", c("ytrain","ytest","mean","confidence"), lty = c(NaN,2,1,NaN),
       pch = c(20,NaN,NaN,15), col = c("black","black","darkgreen","gray80"))
```

---

simulate.lineqGP — *Simulation Method for the* "lineqGP" *S3 Class*

Description

Simulation method for the "lineqGP" S3 class.

Usage

```
## S3 method for class 'lineqGP'
simulate(object, nsim = 1, seed = NULL, xtest, ...)
```

Arguments

object	an object with class "lineqGP".
nsim	the number of simulations.
seed	see simulate.
xtest	a vector (or matrix) with the test input design.
...	further arguments passed to or from other methods.

Details

The posterior sample-path of the finite-dimensional GP with linear inequality constraints are computed. Here, $\boldsymbol{\xi}$ is a centred Gaussian vector with covariance $\boldsymbol{\Gamma}$, s.t. $\boldsymbol{\Phi}\boldsymbol{\xi} = \boldsymbol{y}$ (interpolation constraints) and $\boldsymbol{l} \leq \boldsymbol{\Lambda}\boldsymbol{\xi} \leq \boldsymbol{u}$ (inequality constraints).

Value

A "lineqGP" object with the following elements.

x	a vector (or matrix) with the training input design.
y	the training output vector at x.
xtest	a vector (or matrix) with the test input design.
Phi.test	a matrix corresponding to the hat basis functions evaluated at xtest. The basis functions are indexed by rows.
xi.sim	the posterior sample-path of the finite-dimensional Gaussian vector.
ysim	the posterior sample-path of the observed GP. Note: ysim = Phi.test %*% xi.sim.

Author(s)

A. F. Lopez-Lopera.

References

Lopez-Lopera, A. F., Bachoc, F., Durrande, N., and Roustant, O. (2017), "Finite-dimensional Gaussian approximation with linear inequality constraints". *ArXiv e-prints* [link]

See Also

create.lineqGP, augment.lineqGP, predict.lineqGP

Examples

```
# creating the model
sigfun <- function(x) return(1/(1+exp(-7*(x-0.5))))
x <- seq(0, 1, length = 5)
y <- sigfun(x)
model <- create(class = "lineqGP", x, y, constrType = "monotonicity")

# updating and expanding the model
model$localParam$m <- 30
model$kernParam$par <- c(1, 0.2)
model <- augment(model)

# sampling from the model
xtest <- seq(0, 1, length = 100)
ytest <- sigfun(xtest)
sim.model <- simulate(model, nsim = 50, seed = 1, xtest = xtest)
mu <- apply(sim.model$ysim, 1, mean)
qtls <- apply(sim.model$ysim, 1, quantile, probs = c(0.05, 0.95))
matplot(xtest, t(qtls), type = "l", lty = 1, col = "gray90",
        main = "Constrained Kriging model")
polygon(c(xtest, rev(xtest)), c(qtls[2,], rev(qtls[1,])), col = "gray90", border = NA)
lines(xtest, ytest, lty = 2)
lines(xtest, mu, type = "l", col = "darkgreen")
points(x, y, pch = 20)
legend("right", c("ytrain", "ytest", "mean", "confidence"), lty = c(NaN,2,1,NaN),
       pch = c(20,NaN,NaN,15), col = c("black", "black", "darkgreen", "gray80"))
```

splitDoE — *Training/test data generator according to a given Design of Experiment (DoE)*

Description

Split the data in training/test sets according to a given DoE.

Usage

```
splitDoE(x, y, DoE.idx = NULL, DoE.type = c("rand", "regs"), ratio = 0.3,
  seed = NULL)
```

Arguments

`x`	a vector (or matrix) with the input locations.
`y`	a vector with the output observations.
`DoE.idx`	the numeric indices of the training data used in the design.
`DoE.type`	if `is.null(DoE.idx)`, a character string corresponding to the type of DoE. Options: `rand` (random desings), `regs` (regular-spaced desings).
`ratio`	if `is.null(DoE.idx)`, a number with the ratio nb_train/nb_total (by default, ratio = 0.3).
`seed`	an optional value corresponding to the seed for random methods.

Value

A list with the DoE: `list(xdesign, ydesign, xtest, ytest)`.

Comments

This function is in progress. Other types of DoEs will be considered using the `DiceDesign` package.

Author(s)

A. F. Lopez-Lopera.

Examples

```
# generating the toy example
x <- seq(0, 1, length = 100)
y <- sin(4*pi*x)

# regular DoE
DoE <- splitDoE(x, y, DoE.type = "regs", ratio = 0.3)
plot(x,y)
points(DoE$xdesign, DoE$ydesign, col = "red", pch = 20)
points(DoE$xtest, DoE$ytest, col = "blue", pch = 20)
legend("topright", c("training data", "test data"),
       pch = rep(20, 2), col = c("red", "blue"))

# random DoE
```

```
DoE <- splitDoE(x, y, DoE.type = "rand", ratio = 0.3, seed = 1)
plot(x,y)
points(DoE$xdesign, DoE$ydesign, col = "red", pch = 20)
points(DoE$xtest, DoE$ytest, col = "blue", pch = 20)
legend("topright", c("training data", "test data"),
       pch = rep(20, 2), col = c("red", "blue"))
```

---

tmvrnorm *Sampling Methods of Truncated Multivariate Normal Distributions*

Description

Wrapper function with a collection of Monte Carlo and Markov Chain Monte Carlo samplers for truncated multivariate normal distributions. The function invokes particular samplers which depend on the class of the first argument.

Usage

```
tmvrnorm(object, nsim, ...)
```

Arguments

object	an object with: mu (mean vector), Sigma (covariance matrix), lb (lower bound vector), ub (upper bound vector).
nsim	an integer corresponding to the number of simulations.
...	further arguments passed to or from other methods.

Value

A matrix with the sample path. Samples are indexed by columns.

Author(s)

A. F. Lopez-Lopera.

See Also

tmvrnorm.RSM, tmvrnorm.Gibbs, tmvrnorm.HMC, tmvrnorm.ExpT

tmvrnorm.ExpT	*"tmvrnorm" Sampler for "ExpT" (Exponential Tilting) S3 Class*

Description

Sampler for truncated multivariate normal distributions via exponential tilting using the package TruncatedNormal (Botev, 2017).

Usage

```
## S3 method for class 'ExpT'
tmvrnorm(object, nsim, control = NULL, ...)
```

Arguments

object	an object with "ExpT" S3 class containing: mu (mean vector), Sigma (covariance matrix), lb (lower bound vector), ub (upper bound vector).
nsim	an integer corresponding to the number of simulations.
control	extra parameters required for the MC/MCMC sampler.
...	further arguments passed to or from other methods.

Value

A matrix with the simulated samples. Samples are indexed by columns.

Author(s)

A. F. Lopez-Lopera.

References

Botev, Z. I. (2017), "The normal law under linear restrictions: simulation and estimation via minimax tilting". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(1):125-148. [link]

See Also

tmvrnorm.RSM, tmvrnorm.Gibbs, tmvrnorm.HMC

Examples

```
n <- 100
x <- seq(0, 1, length = n)
Sigma <- kernCompute(x1 = x, type = "gaussian", par = c(1,0.2))
tmgPar <- list(mu = rep(0,n), Sigma = Sigma + 1e-9*diag(n), lb = rep(-1,n), ub = rep(1,n))
class(tmgPar) <- "ExpT"
y <- tmvrnorm(tmgPar, nsim = 10)
matplot(x, y, type = 'l', ylim = c(-1,1),
        main = "Constrained samples using expontial tilting")
abline(h = c(-1,1), lty = 2)
```

tmvrnorm.Gibbs — "tmvrnorm" *Sampler for* "Gibbs" *(Gibbs Sampling) S3 Class*

Description

Sampler for truncated multivariate normal distributions via Gibbs sampling using the package `restrictedMVN` (Taylor and Benjamini, 2017).

Usage

```
## S3 method for class 'Gibbs'
tmvrnorm(object, nsim, control = list(thinning = 100, burn.in
  = 100), ...)
```

Arguments

`object`	an object with `"Gibbs"` S3 class containing: `mu` (mean vector), `Sigma` (covariance matrix), `lb` (lower bound vector), `ub` (upper bound vector).
`nsim`	an integer corresponding to the number of simulations.
`control`	extra parameters required for the MC/MCMC sampler.
`...`	further arguments passed to or from other methods.

Value

A matrix with the simulated samples. Samples are indexed by columns.

Author(s)

A. F. Lopez-Lopera.

References

Taylor, J. and Benjamini, Y. (2017), "RestrictedMVN: multivariate normal restricted by affine constraints".

See Also

`tmvrnorm.RSM`, `tmvrnorm.HMC`, `tmvrnorm.ExpT`

Examples

```
n <- 100
x <- seq(0, 1, length = n)
Sigma <- kernCompute(x1 = x, type = "gaussian", par = c(1,0.2))
tmgPar <- list(mu = rep(0,n), Sigma = Sigma + 1e-9*diag(n), lb = rep(-1,n), ub = rep(1,n))
class(tmgPar) <- "Gibbs"
y <- tmvrnorm(tmgPar, nsim = 10)
matplot(x, y, type = 'l', ylim = c(-1,1),
        main = "Constrained samples using Gibbs sampling")
abline(h = c(-1,1), lty = 2)
```

tmvrnorm.HMC	"tmvrnorm" *Sampler for* "HMC" *(Hamiltonian Monte Carlo) S3 Class*

Description

Sampler for truncated multivariate normal distributions via Hamiltonian Monte Carlo using the package tmg (Pakman and Paninski, 2014).

Usage

```
## S3 method for class 'HMC'
tmvrnorm(object, nsim, control = list(burn.in = 100), ...)
```

Arguments

object	an object with "HMC" S3 class containing: mu (mean vector), Sigma (covariance matrix), lb (lower bound vector), ub (upper bound vector).
nsim	an integer corresponding to the number of simulations.
control	extra parameters required for the MC/MCMC sampler.
...	further arguments passed to or from other methods.

Value

A matrix with the simulated samples. Samples are indexed by columns.

Author(s)

A. F. Lopez-Lopera.

References

Pakman, A. and Paninski, L. (2014), "Exact Hamiltonian Monte Carlo for truncated multivariate Gaussians". *Journal of Computational and Graphical Statistics*, 23(2):518-542. [link]

See Also

tmvrnorm.RSM, tmvrnorm.Gibbs, tmvrnorm.ExpT

Examples

```
n <- 100
x <- seq(0, 1, length = n)
Sigma <- kernCompute(x1 = x, type = "gaussian", par = c(1,0.2))
tmgPar <- list(mu = rep(0,n), Sigma = Sigma + 1e-9*diag(n), lb = rep(-1,n), ub = rep(1,n))
class(tmgPar) <- "HMC"
y <- tmvrnorm(tmgPar, nsim = 10)
matplot(x, y, type = 'l', ylim = c(-1,1),
        main = "Constrained samples using Hamiltonian MC")
abline(h = c(-1,1), lty = 2)
```

tmvrnorm.RSM "tmvrnorm" *Sampler for* "RSM" *(Rejection Sampling from the Mode) S3 Class*

Description

Sampler for truncated multivariate normal distributions via RSM according to (Maatouk and Bay, 2017).

Usage

```
## S3 method for class 'RSM'
tmvrnorm(object, nsim, control = NULL, ...)
```

Arguments

`object`	an object with "RSM" S3 class containing: `mu` (mean vector), `Sigma` (covariance matrix), `lb` (lower bound vector), `ub` (upper bound vector).
`nsim`	an integer corresponding to the number of simulations.
`control`	extra parameters required for the MC/MCMC sampler.
`...`	further arguments passed to or from other methods.

Value

A matrix with the simulated samples. Samples are indexed by columns.

Author(s)

A. F. Lopez-Lopera.

References

Maatouk, H. and Bay, X. (2017), "Gaussian process emulators for computer experiments with inequality constraints". *Mathematical Geosciences*, 49(5):557-582. [link]

See Also

`tmvrnorm.Gibbs`, `tmvrnorm.HMC`, `tmvrnorm.ExpT`

Examples

```
n <- 100
x <- seq(0, 1, length = n)
Sigma <- kernCompute(x1 = x, type = "gaussian", par = c(1,0.2))
tmgPar <- list(mu = rep(0,n), Sigma = Sigma + 1e-9*diag(n), lb = rep(-1,n), ub = rep(1,n))
class(tmgPar) <- "RSM"
y <- tmvrnorm(tmgPar, nsim = 10)
matplot(x, y, type = 'l', ylim = c(-1,1),
        main = "Constrained samples using RSM")
abline(h = c(-1,1), lty = 2)
```

Index