Package 'hetGP'

November 29, 2017

Type Package

Title Heteroskedastic Gaussian Process Modeling and Design under Replication

Version 1.0.1

Date 2017-11-28

Author Mickael Binois, Robert B. Gramacy

Maintainer Mickael Binois <mickael.binois@chicagobooth.edu>

Description Performs Gaussian process regression with heteroskedastic noise following Binois, M., Gramacy, R., Ludkovski, M. (2016) <arXiv:1611.05902>. The input dependent noise is modeled as another Gaussian process. Replicated observations are encouraged as they yield computational savings. Sequential design procedures based on the integrated mean square prediction error and lookahead heuristics are provided, and notably fast update functions when adding new observations.

License LGPL

LazyData TRUE

Imports Rcpp (>= 0.12.3), MASS, methods, DiceDesign

LinkingTo Rcpp

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-11-29 14:48:26 UTC

R topics documented:

`hetGP-package` *Package hetGP*

Description

Performs Gaussian process regression with heteroskedastic noise following Binois, M., Gramacy, R., Ludkovski, M. (2016) <arXiv:1611.05902>. The input dependent noise is modeled as another Gaussian process. Replicated observations are encouraged as they yield computational savings. Sequential design procedures based on the integrated mean square prediction error and lookahead heuristics are provided, and notably fast update functions when adding new observations.

Details

Important functions:
`mleHetGP` as the main function to build a model.
`mleHomGP` the equivalent for homoskedastic modeling.
`crit_IMSE` for adding a new design based on the Integrated Mean Square Prediction Error.
`IMSE_nsteps_ahead` for augmenting a design based on a lookahead heuristic to add replication, of which a simplified version is `IMSE.search`.

Note

The authors are grateful for support from National Science Foundation grant DMS-1521702 and DMS-1521743.

Author(s)

Mickael Binois, Robert B. Gramacy

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

M. Binois, J. Huang, R. B. Gramacy, M. Ludkovski (2017+), Replication or exploration? Sequential design for stochastic simulation experiments

Examples

```
##------------------------------------------------------------
## Example 1: Heteroskedastic GP modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
nvar <- 1
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")

## Model fitting
model <- mleHetGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar),
                  covtype = "Matern5_2")

## A quick view of the fit
summary(model)

## Create a prediction grid and obtain predictions
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
predictions <- predict(x = xgrid, object =  model)

## Display averaged observations
points(model$X0, model$Z0, pch = 20)

## Display mean predictive surface
lines(xgrid, predictions$mean, col = 'red', lwd = 2)
## Display 95% confidence intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
## Display 95% prediction intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)

## Comparison with homoskedastic fit
predictions2 <- predict(x = xgrid, object = model$modHom)
lines(xgrid, predictions2$mean, col = 4, lty = 2, lwd = 2)
lines(xgrid, qnorm(0.05, predictions2$mean, sqrt(predictions2$sd2)), col = 4, lty = 3)
lines(xgrid, qnorm(0.95, predictions2$mean, sqrt(predictions2$sd2)), col = 4, lty = 3)
```

```
##-------------------------------------------------------------
## Example 2: Sequential design
##-------------------------------------------------------------
## Not run:
library(DiceDesign)

## Design configuration / Parameter settings
N_tot <- 500 # total number of points
n_init <- 10 # number of unique designs

## HetGP options
nvar <- 1 # number of variables
lower <- rep(0.001, nvar)
upper <- rep(1, nvar)

### Problem definition

## Mean function
forrester <- function(x){
 return(((x*6-2)^2)*sin((x*6-2)*2))
}

## Noise field via standard deviation
noiseFun <- function(x, coef = 1.1, scale = 1){
 if(is.null(nrow(x)))
   x <- matrix(x, nrow = 1)
 return(scale*(coef + sin(x * 2 * pi)))
}

### Test function defined in [0,1]
ftest <- function(x){
 if(is.null(nrow(x)))
   x <- matrix(x, ncol = 1)
 return(forrester(x) + rnorm(nrow(x), mean = 0, sd = noiseFun(x)))
}

## Predictive grid
ngrid <- 51
xgrid <- seq(0,1, length.out = ngrid)
Xgrid <- matrix(xgrid, ncol = 1)

par(mar = c(3,3,2,3)+0.1)
plot(xgrid, forrester(xgrid), type = 'l', lwd = 1, col = "blue", lty = 3,
    xlab = '', ylab = '', ylim = c(-8,16))

set.seed(42)

# Initial design
X <- maximinSA_LHS(lhsDesign(n_init, nvar, seed = 42)$design)$design
Z <- apply(X, 1, ftest)

points(X, Z)
```

```
model <- model_init <- mleHetGP(X = X, Z = Z, lower = lower, upper = upper)

for(ii in 1:(N_tot - n_init)){
 ##Precalculations
 Wijs <- hetGP:::Wij(mu1 = model$X0, theta = model$theta, type = model$covtype)

 ## Adapt the horizon based on the training rmspe/score
   current_horizon <- update_horizon(model = model, Wijs = Wijs)

 if(current_horizon == -1){
   opt <- IMSE.search(model = model, Wijs = Wijs)
 }else{
   opt <- IMSE_nsteps_ahead(model = model, h = current_horizon, Wijs = Wijs)
 }

 Xnew <- opt$par
 Znew <- apply(Xnew, 1, ftest)
 X <- rbind(X, Xnew)
 Z <- c(Z, Znew)
 points(Xnew, Znew)

 ## Update of the model
 model <- update(object = model, Xnew = Xnew, Znew = Znew, lower = lower, upper = upper)
 if(ii %% 25 == 0 || ii == (N_tot - n_init)){
  model_test <- mleHetGP(X = list(X0 = model$X0, Z0 = model$Z0, mult = model$mult), Z = model$Z,
                         lower = lower, upper = upper, maxit = 1000)
   model <- compareGP(model, model_test)
 }
}
### Plot result
preds <- predict(x = Xgrid, model)
lines(Xgrid, preds$mean, col = 'red', lwd = 2)
lines(Xgrid, qnorm(0.05, preds$mean, sqrt(preds$sd2)), col = 2, lty = 2)
lines(Xgrid, qnorm(0.95, preds$mean, sqrt(preds$sd2)), col = 2, lty = 2)
lines(Xgrid, qnorm(0.05, preds$mean, sqrt(preds$sd2 + preds$nugs)), col = 3, lty = 2)
lines(Xgrid, qnorm(0.95, preds$mean, sqrt(preds$sd2 + preds$nugs)), col = 3, lty = 2)
par(new = TRUE)
plot(NA,NA, xlim = c(0, 1), ylim = c(0,max(model$mult)), axes = FALSE, ylab = "", xlab = "")
segments(x0 = model$X, x1 = model$X, y0 = rep(0, nrow(model$X)), y1 = model$mult, col = 'grey')
axis(side = 4)
mtext(side = 4, line = 2, expression(a[i]), cex = 0.8)
mtext(side = 2, line = 2, expression(f(x)), cex = 0.8)
mtext(side = 1, line = 2, 'x', cex = 0.8)

## End(Not run)
```

`allocate_mult` *Allocation of replicates on existing designs*

Description

Allocation of replicates on existing design locations, based on (29) from (Ankenman et al, 2010)

Usage

```
allocate_mult(model, N, Wijs = NULL, use.Ki = FALSE)
```

Arguments

model	hetGP model
N	total budget of replication to allocate
Wijs	optional previously computed matrix of Wijs, see Wij
use.Ki	should Ki from model be used? Using the inverse of C (covariance matrix only, without noise, using ginv) is also possible

Value

vector with approximated best number of replicates per design

References

B. Ankenman, B. Nelson, J. Staum (2010), Stochastic kriging for simulation metamodeling, Operations research, pp. 371–382, 58

Examples

```
##------------------------------------------------------------
## Example: Heteroskedastic GP modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
nvar <- 1

data_m <- find_reps(X, Z, rescale = TRUE)

plot(rep(data_m$X0, data_m$mult), data_m$Z, ylim = c(-160, 90),
     ylab = 'acceleration', xlab = "time")


## Model fitting
model <- mleHetGP(X = list(X0 = data_m$X0, Z0 = data_m$Z0, mult = data_m$mult),
                  Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar),
                  covtype = "Matern5_2")
## Compute best allocation
A <- allocate_mult(model, N = 1000)
```

```
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")

## Create a prediction grid and obtain predictions
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
predictions <- predict(x = xgrid, object =  model)
par(new = TRUE)
plot(NA,NA, xlim = c(0,1), ylim = c(0,max(A)), axes = FALSE, ylab = "", xlab = "")
segments(x0 = model$X0, x1 = model$X0,
y0 = rep(0, nrow(model$X)), y1 = A, col = 'grey')
axis(side = 4)
mtext(side = 4, line = 2, expression(a[i]), cex = 0.8)
```

`compareGP` *Likelihood-based comparison of models*

Description

Compare two models based on the log-likelihood for `hetGP` and `homGP` models

Usage

```
compareGP(model1, model2)
```

Arguments

`model1, model2` `hetGP` or `homGP` models

Value

Best model based on the likelihood, first one in case of a tie

Note

If comparing homoskedastic and heteroskedastic models, the un-penalised likelihood is used for the later, see e.g., (Binois et al. 2017+).

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

cov_gen — *Correlation function of selected type, supporting both isotropic and product forms*

Description

Correlation function of selected type, supporting both isotropic and product forms

Usage

```
cov_gen(X1, X2 = NULL, theta, type = c("Gaussian", "Matern5_2",
  "Matern3_2"))
```

Arguments

X1	matrix of design locations, one point per row
X2	matrix of design locations if correlation is calculated between X1 and X2 (otherwise calculated between X1 and itself)
theta	vector of lengthscale parameters (either of size one if isotropic or of size d if anisotropic)
type	one of "Gaussian", "Matern5_2", "Matern3_2"

Details

Definition of univariate correlation function and hyperparameters:

- "Gaussian": $c(x,y) = exp(-(x-y)^2/theta)$
- "Matern5_2": $c(x,y) = (1 + sqrt(5)/theta * abs(x-y) + 5/(3*theta^2)(x-y)^2) * exp(-sqrt(5)*abs(x-y)/theta)$
- "Matern3_2": $c(x,y) = (1+sqrt(3)/theta*abs(x-y))*exp(-sqrt(3)*abs(x-y)/theta)$

Multivariate correlations are product of univariate ones.

crit_IMSE — *Sequential IMSPE criterion*

Description

Compute the integrated mean square prediction error after adding a new design

Usage

```
crit_IMSE(x, model, id = NULL, Wijs = NULL)
```

Arguments

x	matrix for the new design (size 1 x d)
model	homGP or hetGP model, including inverse matrices
id	instead of providing x, one can provide the index of a considered existing design
Wijs	optional previously computed matrix of Wijs, to avoid recomputing it; see Wij

Details

The computations are scale free, i.e., values are not multiplied by nu2_hat from homGP or hetGP.

See Also

deriv_crit_IMSE for the derivative

Examples

```
## One-d toy example

set.seed(42)
ftest <- function(x, coef = 0.1) return(sin(2*pi*x) + rnorm(1, sd = coef))

n <- 9
designs <- matrix(seq(0.1, 0.9, length.out = n), ncol = 1)
X <- matrix(designs[rep(1:n, sample(1:10, n, replace = TRUE)),])
Z <- apply(X, 1, ftest)

prdata <- find_reps(X, Z, inputBounds = matrix(c(0,1), nrow = 2, ncol = 1))
Z <- prdata$Z
plot(prdata$X0[rep(1:n, times = prdata$mult),], prdata$Z, xlab = "x", ylab = "Y")

model <- mleHetGP(X = list(X0 = prdata$X0, Z0 = prdata$Z0, mult = prdata$mult),
                  Z = Z, lower = 0.1, upper = 5)

ngrid <- 501
xgrid <- matrix(seq(0,1, length.out = ngrid), ncol = 1)

## Precalculations
Wijs <- Wij(mu1 = model$X0, theta = model$theta, type = model$covtype)


t0 <- Sys.time()

IMSE_grid <- apply(xgrid, 1, crit_IMSE, Wijs = Wijs, model = model)

t1 <- Sys.time()
print(t1 - t0)

plot(xgrid, IMSE_grid/ngrid, xlab = "x", ylab = "crit_IMSE values")
abline(v = designs)

################################################################################
```

```
## Bi-variate case

nvar <- 2

set.seed(42)
ftest <- function(x, coef = 0.1) return(sin(2*pi*sum(x)) + rnorm(1, sd = coef))

n <- 16 # must be a square
xgrid0 <- seq(0.1, 0.9, length.out = sqrt(n))
designs <- as.matrix(expand.grid(xgrid0, xgrid0))
X <- designs[rep(1:n, sample(1:10, n, replace = TRUE)),]
Z <- apply(X, 1, ftest)

prdata <- find_reps(X, Z, inputBounds = matrix(c(0,1), nrow = 2, ncol = 1))
Z <- prdata$Z

model <- mleHetGP(X = list(X0 = prdata$X0, Z0 = prdata$Z0, mult = prdata$mult), Z = Z,
 lower = rep(0.1, nvar), upper = rep(1, nvar))
ngrid <- 51
xgrid <- seq(0,1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
## Precalculations
Wijs <- Wij(mu1 = model$X0, theta = model$theta, type = model$covtype)
t0 <- Sys.time()

IMSE_grid <- apply(Xgrid, 1, crit_IMSE, Wijs = Wijs, model = model)
filled.contour(x = xgrid, y = xgrid, matrix(IMSE_grid, ngrid),
               nlevels = 20, color.palette = terrain.colors,
               main = "Sequential IMSPE values")
```

`deriv_crit_IMSE` *Derivative of crit_IMSE*

Description

Derivative of crit_IMSE

Usage

```
deriv_crit_IMSE(x, model, Wijs = NULL)
```

Arguments

`x`	matrix for the new design (size 1 x d)
`model`	`homGP` or `hetGP` model
`Wijs`	optional previously computed matrix of Wijs, see `Wij`

Value

Derivative of the sequential IMSPE with respect to `x`

See Also

crit_IMSE for the criterion

---

find_reps *Data preprocessing*

Description

Prepare data for use with mleHetGP, in particular to find replicated observations

Usage

```
find_reps(X, Z, return.Zlist = TRUE, rescale = FALSE, normalize = FALSE,
  inputBounds = NULL)
```

Arguments

X	matrix of design locations, one point per row
Z	vector of observations at X
return.Zlist	to return Zlist, see below
rescale	if TRUE, the inputs are rescaled to the unit hypercube
normalize	if TRUE, the outputs are centered and normalized
inputBounds	optional matrix of known boundaries in original input space, of size 2 times ncol(X). If not provided, and rescale == TRUE, it is estimated from the data.

Details

Replicates are searched based on character representation, using unique.

Value

A list with the following elements that can be passed to the main fitting functions, e.g., mleHetGP and mleHomGP

- X0 matrix with unique designs locations, one point per row,
- Z0 vector of averaged observations at X0,
- mult number of replicates at X0,
- Z vector with all observations, sorted according to X0,
- Zlist optional list, each element corresponds to observations at a design in X0,
- inputBounds optional matrix, to rescale back to the original input space,
- outputStats optional vector, with mean and variance of the original outputs.

Examples

```
##------------------------------------------------------------
## Find replicates on the motorcycle data
##------------------------------------------------------------
## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel

data_m <- find_reps(X, Z)

# Initial data
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")
# Display mean values
points(data_m$X0, data_m$Z0, pch = 20)
```

`IMSE.search` *IMSE minimization*

Description

Search for best reduction in IMSE

Usage

```
IMSE.search(model, replicate = FALSE, Xcand = NULL,
  control = list(tol_dist = 1e-04, tol_diff = 0.001, multi.start = 5, maxit =
  100, maximin = TRUE, Xstart = NULL), Wijs = NULL, seed = NULL)
```

Arguments

`model`	homGP or hetGP model
`replicate`	if `TRUE`, search only on existing designs
`Xcand`	optional set of of candidates for discrete search
`control`	list in case `Xcand == NULL`, with elements `multi.start`, to perform a multi-start optimization based on `optim`, with `maxit` iterations each. Also, `tol_dist` defines the minimum distance to an existing design for a new point to be added, otherwise the closest existing design is chosen. In a similar fashion, `tol_dist` is the minimum relative change of IMSE for adding a new design.
`Wijs`	optional previously computed matrix of Wijs, see `Wij`
`seed`	optional seed for the generation of designs with `maximinSA_LHS`

Value

list with `par`, `value` elements, and additional slot `new` (boolean if it is or not a new design) and `id` giving the index of the duplicated design.

Examples

```
##############################################################################
## Bi-variate example
##############################################################################

nvar <- 2

set.seed(42)
ftest <- function(x, coef = 0.1) return(sin(2*pi*sum(x)) + rnorm(1, sd = coef))

n <- 25 # must be a square
xgrid0 <- seq(0.1, 0.9, length.out = sqrt(n))
designs <- as.matrix(expand.grid(xgrid0, xgrid0))
X <- designs[rep(1:n, sample(1:10, n, replace = TRUE)),]
Z <- apply(X, 1, ftest)

model <- mleHomGP(X, Z, lower = rep(0.1, nvar), upper = rep(1, nvar))

ngrid <- 51
xgrid <- seq(0,1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))

preds <- predict(x = Xgrid, object =  model)

## Initial plots
contour(x = xgrid,  y = xgrid, z = matrix(preds$mean, ngrid),
        main = "Predicted mean", nlevels = 20)
points(model$X0, col = 'blue', pch = 20)

IMSE_grid <- apply(Xgrid, 1, crit_IMSE, model = model)
filled.contour(x = xgrid, y = xgrid, matrix(IMSE_grid, ngrid),
               nlevels = 20, color.palette = terrain.colors,
               main = "Initial IMSPE criterion landscape",
plot.axes = {axis(1); axis(2); points(model$X0, pch = 20)})

## Sequential IMSPE search
nsteps <- 1 # Increase for better results

for(i in 1:nsteps){
  res <- IMSE.search(model, control = list(multi.start = 100, maxit = 50))
  newX <- res$par
  newZ <- ftest(newX)
  model <- update(object = model, Xnew = newX, Znew = newZ)
}

## Final plots
contour(x = xgrid,  y = xgrid, z = matrix(preds$mean, ngrid),
        main = "Predicted mean", nlevels = 20)
points(model$X0, col = 'blue', pch = 20)

IMSE_grid <- apply(Xgrid, 1, crit_IMSE, model = model)
filled.contour(x = xgrid, y = xgrid, matrix(IMSE_grid, ngrid),
```

```
                nlevels = 20, color.palette = terrain.colors,
                main = "Final IMSPE criterion landscape",
plot.axes = {axis(1); axis(2); points(model$X0, pch = 20)})
```

`IMSE_nsteps_ahead` *h-IMSE with replication*

Description

h-steps lookahead strategy to favor designs with replication

Usage

```
IMSE_nsteps_ahead(model, h = 2, Xcand = NULL, control = list(multi.start =
  10, maxit = 100), Wijs = NULL)
```

Arguments

`model`	`homGP` or `hetGP` model
`h`	horizon (multi-step ahead framework)
`Xcand`	optional discrete set of new candidates (otherwise a maximin LHS is used to initialise search)
`control`	list to be passed to `IMSE.search`
`Wijs`	optional previously computed matrix of Wijs, see `Wij`

Details

The domain needs to be [0, 1]^d for now. The decision is made between:

- sequential IMSE search starting by a new design (optimized first) then adding `h` replicates
- sequential IMSE searches starting by `1` to `h` replicates before adding a new point

Value

list with elements:

- `par`: best first design,
- `value`: IMSPE h-steps ahead starting from adding `par`,
- `path`: list of elements list(`par`, `value`, `new`) at each step `h`

References

M. Binois, J. Huang, R. Gramacy, M. Ludkovski (2017+), Replication or exploration? Sequential design for stochastic simulation experiments.

Examples

```
##############################################################################
## Bi-variate example
##############################################################################

nvar <- 2

set.seed(42)
ftest <- function(x, coef = 0.1) return(sin(2*pi*sum(x)) + rnorm(1, sd = coef))

n <- 25 # must be a square
xgrid0 <- seq(0.1, 0.9, length.out = sqrt(n))
designs <- as.matrix(expand.grid(xgrid0, xgrid0))
X <- designs[rep(1:n, sample(1:10, n, replace = TRUE)),]
Z <- apply(X, 1, ftest)

model <- mleHomGP(X, Z, lower = rep(0.1, nvar), upper = rep(1, nvar))

## Not run:
ngrid <- 51
xgrid <- seq(0,1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))

nsteps <- 5 # Increase for more steps

for(i in 1:nsteps){
  res <- IMSE_nsteps_ahead(model, h = 3, control = list(multi.start = 100, maxit = 50))

  # If a replicate is selected
  if(res$path[[1]]$new) print("Add replicate")

  newX <- res$par
  newZ <- ftest(newX)
  model <- update(object = model, Xnew = newX, Znew = newZ)

  ## Plots
  preds <- predict(x = Xgrid, object =  model)
  contour(x = xgrid,  y = xgrid, z = matrix(preds$mean, ngrid),
          main = "Predicted mean", nlevels = 20)
  points(model$X0, col = 'blue', pch = 20)
  points(newX, col = "red", pch = 20)

  ## Precalculations
  Wijs <- Wij(mu1 = model$X0, theta = model$theta, type = model$covtype)

  IMSE_grid <- apply(Xgrid, 1, crit_IMSE, Wijs = Wijs, model = model)
  filled.contour(x = xgrid, y = xgrid, matrix(IMSE_grid, ngrid),
                 nlevels = 20, color.palette = terrain.colors,
  plot.axes = {axis(1); axis(2); points(model$X0, pch = 20)})
}

## End(Not run)
```

IMSPE *Integrated Mean Square Prediction Error*

Description

IMSPE of a given design

Usage

```
IMSPE(X, theta = NULL, Lambda = NULL, mult = NULL, covtype = NULL,
  nu2 = NULL, eps = sqrt(.Machine$double.eps))
```

Arguments

X	hetGP or homGP model. Alternatively, one can provide a matrix of unique designs considered
theta	lengthscales
Lambda	diagonal matrix for the noise
mult	number of replicates at each design
covtype	either "Gaussian", "Matern3_2" or "Matern5_2"
nu2	variance parameter
eps	numerical nugget

Details

One can provide directly a model of class hetGP or homGP, or provide X and all other arguments

mleHetGP *Gaussian process modeling with heteroskedastic noise*

Description

Gaussian process regression under input dependent noise based on maximum likelihood estimation of the hyperparameters. A second GP is used to model latent (log-) variances. This function is enhanced to deal with replicated observations.

Usage

```
mleHetGP(X, Z, lower, upper, noiseControl = list(k_theta_g_bounds = c(1, 100),
  g_max = 100, g_bounds = c(1e-06, 1)), settings = list(linkThetas = "joint",
  logN = TRUE, initStrategy = "residuals", checkHom = TRUE, penalty = TRUE,
  trace = 0, return.matrices = TRUE), covtype = c("Gaussian", "Matern5_2",
  "Matern3_2"), maxit = 100, known = NULL, init = NULL,
  eps = sqrt(.Machine$double.eps))
```

Arguments

X	matrix of all designs, one per row, or list with elements: • X0 matrix of unique design locations, one point per row • Z0 vector of averaged observations, of length nrow(X0) • mult number of replicates at designs in X0, of length nrow(X0)
Z	vector of all observations. If using a list with X, Z has to be ordered with respect to X0, and of length sum(mult)
lower, upper	bounds for the theta parameter (see cov_gen for the exact parameterization). In the multivariate case, it is possible to give vectors for bounds (resp. scalars) for anisotropy (resp. isotropy)
noiseControl	list with elements related to optimization of the noise process parameters: • g_min, g_max minimal and maximal noise to signal ratio (of the mean process) • lowerDelta, upperDelta optional vectors (or scalars) of bounds on Delta, of length nrow(X0) (default to rep(eps, nrow(X0)) and rep(noiseControl$g_max, nrow(X0)) resp., or their log) • lowerTheta_g, upperTheta_g optional vectors of bounds for the lengthscales of the noise process if linkThetas == 'none'. Same as for theta if not provided. • k_theta_g_bounds if linkThetas == 'joint', vector with minimal and maximal values for k_theta_g (default to c(1, 100)). See Details. • g_bounds vector for minimal and maximal noise to signal ratios for the noise of the noise process, i.e., the smoothing parameter for the noise process. (default to c(1e-6, 1)).
settings	list for options about the general modeling procedure, with elements: • linkThetas defines the relation between lengthscales of the mean and noise processes. Either 'none', 'joint'(default) or 'constr', see Details. • logN, when TRUE (default), the log-noise process is modeled. • initStrategy one of 'simple', 'residuals' (default) and 'smoothed' to obtain starting values for Delta, see Details • penalty when TRUE, the penalized version of the likelihood is used (i.e., the sum of the log-likelihoods of the mean and variance processes, see References). • hardpenalty is TRUE, the log-likelihood from the noise GP is taken into account only if negative (default if maxit > 1000). • checkHom when TRUE, if the log-likelihood with a homoskedastic model is better, then return it. • trace optional scalar (default to 0). If positive, tracing information on the fitting process. If 1, information is given about the result of the heterogeneous model optimization. Level 2 gives more details. Level 3 additionaly displays all details about initialization of hyperparameters. • return.matrices boolean too include the inverse covariance matrix in the object for further use (e.g., prediction).

`covtype` covariance kernel type, either `'Gaussian'`, `'Matern5_2'` or `'Matern3_2'`, see `cov_gen`

`maxit` maximum number of iterations for `L-BFGS-B` of `optim` dedicated to maximum likelihood optimization

`init, known` optional lists of starting values for mle optimization or that should not be optimized over, respectively. Values in `known` are not modified, while it can happen to these of `init`, see Details. One can set one or several of the following:

- `theta` lengthscale parameter(s) for the mean process either one value (isotropic) or a vector (anistropic)
- `Delta` vector of nuggets corresponding to each design in `X0`, that are smoothed to give `Lambda` (as the global covariance matrix depend on `Delta` and `nu2_hat`, it is recommended to also pass values for `theta`)
- `beta0` constant trend of the mean process
- `k_theta_g` constant used for link mean and noise processes lengthscales, when `settings$linkThetas == 'joint'`
- `theta_g` either one value (isotropic) or a vector (anistropic) for lengthscale parameter(s) of the noise process, when `settings$linkThetas != 'joint'`
- `g` scalar nugget of the noise process
- `g_H` scalar homoskedastic nugget for the initialisation with a `mleHomGP`. See Details.

`eps` jitter used in the inversion of the covariance matrix for numerical stability

Details

The global covariance matrix of the model is parameterized as `K = nu2_hat * (C + Lambda * diag(1/mult))`, with `C` the correlation matrix between unique designs, depending on the family of kernel used (see `cov_gen` for available choices) and values of lengthscale parameters. `nu2_hat` is the plugin estimator of the variance of the process. `Lambda` is the prediction on the noise level given by a second (homoskedastic) GP:

$$\Lambda = C_g(C_g + \mathrm{diag}(g/\mathrm{mult}))^{-1}\Delta$$

with `C_g` the correlation matrix between unique designs for this second GP, with lengthscales hyperparameters `theta_g` and nugget `g` and `Delta` the variance level at `X0` that are estimated.

It is generally recommended to use `find_reps` to pre-process the data, to rescale the inputs to the unit cube and to normalize the outputs.

The noise process lengthscales can be set in several ways:

- using `k_theta_g` (`settings$linkThetas == 'joint'`), supposed to be greater than one by default. In this case lengthscales of the noise process are multiples of those of the mean process.
- if `settings$linkThetas == 'constr'`, then the lower bound on `theta_g` correspond to estimated values of an homoskedastic GP fit.
- else lengthscales between the mean and noise process are independent (both either anisotropic or not).

When no starting nor fixed parameter values are provided with `init` or `known`, the initialization process consists of fitting first an homoskedastic model of the data, called `modHom`. Unless provided with `init$theta`, initial lengthscales are taken at the middle of the range determined with `lower` and `upper`, while `init$g_H` may be use to pass an initial nugget value. The resulting lengthscales provide initial values for `theta` (or update them if given in `init`).

If necessary, a second homoskedastic model, `modNugs`, is fitted to the empirical residual variance between the prediction given by `modHom` at `X0` and `Z` (up to `modHom$nu2_hat`). Note that when specifying `settings$linkThetas == 'joint'`, then this second homoskedastic model has fixed lengthscale parameters. Starting values for `theta_g` and `g` are extracted from `modNugs`.

Finally, three initialization schemes for `Delta` are available with `settings$initStrategy`:

- for `settings$initStrategy == 'simple'`, `Delta` is simply initialized to the estimated `g` value of `modHom`. Note that this procedure may fail when `settings$penalty == TRUE`.
- for `settings$initStrategy == 'residuals'`, `Delta` is initialized to the estimated residual variance from the homoskedastic mean prediction.
- for `settings$initStrategy == 'smoothed'`, `Delta` takes the values predicted by `modNugs` at `X0`.

Notice that `lower` and `upper` bounds cannot be equal for `optim`.

Value

a list which is given the S3 class `"hetGP"`, with elements:

- `theta`: unless given, maximum likelihood estimate (mle) of the lengthscale parameter(s),
- `Delta`: unless given, mle of the nugget vector (non-smoothed),
- `Lambda`: predicted input noise variance at `X0`,
- `nu2_hat`: plugin estimator of the variance,
- `theta_g`: unless given, mle of the lengthscale(s) of the noise/log-noise process,
- `k_theta_g`: if `settings$linkThetas == 'joint'`, mle for the constant by which lengthscale parameters of `theta` are multiplied to get `theta_g`,
- `g`: unless given, mle of the nugget of the noise/log-noise process,
- `trendtype`: either `"SK"` if `beta0` is provided, else `"OK"`,
- `beta0` constant trend of the mean process, plugin-estimator unless given,
- `nmean`: plugin estimator for the constant noise/log-noise process mean,
- `ll`: log-likelihood value, (`ll_non_pen`) is the value without the penalty,
- `nit_opt`, `msg`: `counts` and `message` returned by `optim`
- `modHom`: homoskedastic GP model of class `homGP` used for initialization of the mean process,
- `modNugs`: homoskedastic GP model of class `homGP` used for initialization of the noise/log-noise process,
- `nu2_hat_var`: variance of the noise process,
- `used_args`: list with arguments provided in the call to the function, which is saved in `call`,
- `Ki`, `Kgi`: inverse of the covariance matrices of the mean and noise processes,
- `X0`, `Z0`, `Z`, `eps`, `logN`, `covtype`: values given in input

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

See Also

predict.hetGP for predictions, update.hetGP for updating an existing model. A summary function is available as well. mleHetTP provide a Student-t equivalent.

Examples

```
##------------------------------------------------------------
## Example 1: Heteroskedastic GP modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
nvar <- 1
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")


## Model fitting
model <- mleHetGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar),
                  covtype = "Matern5_2")

## Display averaged observations
points(model$X0, model$Z0, pch = 20)

## A quick view of the fit
summary(model)

## Create a prediction grid and obtain predictions
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
predictions <- predict(x = xgrid, object =  model)

## Display mean predictive surface
lines(xgrid, predictions$mean, col = 'red', lwd = 2)
## Display 95% confidence intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
## Display 95% prediction intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)

##------------------------------------------------------------
## Example 2: 2D Heteroskedastic GP modeling
##------------------------------------------------------------
```

```
set.seed(1)
nvar <- 2

## Branin redefined in [0,1]^2
branin <- function(x){
  if(is.null(nrow(x)))
    x <- matrix(x, nrow = 1)
    x1 <- x[,1] * 15 - 5
    x2 <- x[,2] * 15
    (x2 - 5/(4 * pi^2) * (x1^2) + 5/pi * x1 - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(x1) + 10
}

## Noise field via standard deviation
noiseFun <- function(x){
  if(is.null(nrow(x)))
    x <- matrix(x, nrow = 1)
  return(1/5*(3*(2 + 2*sin(x[,1]*pi)*cos(x[,2]*3*pi) + 5*rowSums(x^2))))
}

## data generating function combining mean and noise fields
ftest <- function(x){
  return(branin(x) + rnorm(nrow(x), mean = 0, sd = noiseFun(x)))
}

## Grid of predictive locations
ngrid <- 51
xgrid <- matrix(seq(0, 1, length.out = ngrid), ncol = 1)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))

## Unique (randomly chosen) design locations
n <- 50
Xu <- matrix(runif(n * 2), n)

## Select replication sites randomly
X <- Xu[sample(1:n, 20*n, replace = TRUE),]

## obtain training data response at design locations X
Z <- ftest(X)

## Formating of data for model creation (find replicated observations)
prdata <- find_reps(X, Z, rescale = FALSE, normalize = FALSE)

## Model fitting
model <- mleHetGP(X = list(X0 = prdata$X0, Z0 = prdata$Z0, mult = prdata$mult), Z = prdata$Z,
                  lower = rep(0.01, nvar), upper = rep(10, nvar),
                  covtype = "Matern5_2")

## a quick view into the data stored in the "hetGP"-class object
summary(model)

## prediction from the fit on the grid
predictions <- predict(x = Xgrid, object =  model)
```

```
## Visualization of the predictive surface
par(mfrow = c(2, 2))
contour(x = xgrid,  y = xgrid, z = matrix(branin(Xgrid), ngrid),
  main = "Branin function", nlevels = 20)
points(X, col = 'blue', pch = 20)
contour(x = xgrid,  y = xgrid, z = matrix(predictions$mean, ngrid),
  main = "Predicted mean", nlevels = 20)
points(Xu, col = 'blue', pch = 20)
contour(x = xgrid,  y = xgrid, z = matrix(noiseFun(Xgrid), ngrid),
  main = "Noise standard deviation function", nlevels = 20)
points(Xu, col = 'blue', pch = 20)
contour(x = xgrid,  y= xgrid, z = matrix(sqrt(predictions$nugs), ngrid),
  main = "Predicted noise values", nlevels = 20)
points(Xu, col = 'blue', pch = 20)
par(mfrow = c(1, 1))
```

`mleHetTP` *Student-t process modeling with heteroskedastic noise*

Description

Student-t process regression under input dependent noise based on maximum likelihood estimation of the hyperparameters. A GP is used to model latent (log-) variances. This function is enhanced to deal with replicated observations.

Usage

```
mleHetTP(X, Z, lower, upper, noiseControl = list(k_theta_g_bounds = c(1, 100),
  g_max = 10000, g_bounds = c(1e-06, 0.1), nu_bounds = c(2 + 0.001, 30),
  sigma2_bounds = c(sqrt(.Machine$double.eps), 10000)),
  settings = list(linkThetas = "joint", logN = TRUE, initStrategy =
  "residuals", checkHom = TRUE, penalty = TRUE, hardpenalty = FALSE, trace = 0,
  return.matrices = TRUE), covtype = c("Gaussian", "Matern5_2", "Matern3_2"),
  maxit = 100, known = list(beta0 = 0), init = list(nu = 3),
  eps = sqrt(.Machine$double.eps))
```

Arguments

`X`	matrix of all designs, one per row, or list with elements: • `X0` matrix of unique design locations, one point per row • `Z0` vector of averaged observations, of length `nrow(X0)` • `mult` number of replicates at designs in `X0`, of length `nrow(X0)`
`Z`	vector of all observations. If using a list with `X`, `Z` has to be ordered with respect to `X0`, and of length `sum(mult)`
`lower, upper`	bounds for the `theta` parameter (see `cov_gen` for the exact parameterization). In the multivariate case, it is possible to give vectors for bounds (resp. scalars) for anisotropy (resp. isotropy)

`noiseControl` list with elements related to optimization of the noise process parameters:

- `g_min`, `g_max` minimal and maximal noise to signal ratio (of the mean process)
- `lowerDelta`, `upperDelta` optional vectors (or scalars) of bounds on `Delta`, of length `nrow(X0)` (default to `rep(eps, nrow(X0))` and `rep(noiseControl$g_max, nrow(X0))` resp., or their `log`)
- `lowerTheta_g`, `upperTheta_g` optional vectors of bounds for the lengthscales of the noise process if `linkThetas == 'none'`. Same as for `theta` if not provided.
- `k_theta_g_bounds` if `linkThetas == 'joint'`, vector with minimal and maximal values for `k_theta_g` (default to `c(1, 100)`). See Details.
- `g_bounds` vector for minimal and maximal noise to signal ratios for the noise of the noise process, i.e., the smoothing parameter for the noise process. (default to `c(1e-6, 1)`).
- `sigma2_bounds`, vector providing minimal and maximal signal variance.
- `nu_bounds`, vector providing minimal and maximal values for the degrees of freedom.

`settings` list for options about the general modeling procedure, with elements:

- `linkThetas` defines the relation between lengthscales of the mean and noise processes. Either `'none'`, `'joint'`(default) or `'constr'`, see Details.
- `logN`, when `TRUE` (default), the log-noise process is modeled.
- `initStrategy` one of `'simple'`, `'residuals'` (default) and `'smoothed'` to obtain starting values for `Delta`, see Details
- `penalty` when `TRUE`, the penalized version of the likelihood is used (i.e., the sum of the log-likelihoods of the mean and variance processes, see References).
- `hardpenalty` is `TRUE`, the log-likelihood from the noise GP is taken into account only if negative.
- `checkHom` when `TRUE`, if the log-likelihood with a homoskedastic model is better, then return it.
- `trace` optional scalar (default to `0`). If positive, tracing information on the fitting process. If `1`, information is given about the result of the heterogeneous model optimization. Level `2` gives more details. Level 3 additionaly displays all details about initialization of hyperparameters.
- `return.matrices` boolean too include the inverse covariance matrix in the object for further use (e.g., prediction).

`covtype` covariance kernel type, either `'Gaussian'`, `'Matern5_2'` or `'Matern3_2'`, see `cov_gen`

`maxit` maximum number of iterations for `L-BFGS-B` of `optim` dedicated to maximum likelihood optimization

`init, known` optional lists of starting values for mle optimization or that should not be optimized over, respectively. Values in `known` are not modified, while it can happen to those of `init`, see Details. One can set one or several of the following:

- theta lengthscale parameter(s) for the mean process either one value (isotropic) or a vector (anistropic)
- Delta vector of nuggets corresponding to each design in X0, that are smoothed to give Lambda (as the global covariance matrix depend on Delta and nu2_hat, it is recommended to also pass values for theta)
- beta0 constant trend of the mean process
- k_theta_g constant used for link mean and noise processes lengthscales, when settings$linkThetas == 'joint'
- theta_g either one value (isotropic) or a vector (anistropic) for lengthscale parameter(s) of the noise process, when settings$linkThetas != 'joint'
- g scalar nugget of the noise process
- nu degree of freedom parameter
- sigma2 scale variance
- g_H scalar homoskedastic nugget for the initialisation with a mleHomGP. See Details.

eps jitter used in the inversion of the covariance matrix for numerical stability

Details

The global covariance matrix of the model is parameterized as K = sigma2 * C + Lambda * diag(1/mult), with C the correlation matrix between unique designs, depending on the family of kernel used (see cov_gen for available choices). Lambda is the prediction on the noise level given by a (homoskedastic) GP:

$$\Lambda = C_g(C_g + \mathrm{diag}(g/\mathrm{mult}))^{-1}\Delta$$

with C_g the correlation matrix between unique designs for this second GP, with lengthscales hyperparameters theta_g and nugget g and Delta the variance level at X0 that are estimated.

It is generally recommended to use find_reps to pre-process the data, to rescale the inputs to the unit cube and to normalize the outputs.

The noise process lengthscales can be set in several ways:

- using k_theta_g (settings$linkThetas == 'joint'), supposed to be greater than one by default. In this case lengthscales of the noise process are multiples of those of the mean process.
- if settings$linkThetas == 'constr', then the lower bound on theta_g correspond to estimated values of an homoskedastic GP fit.
- else lengthscales between the mean and noise process are independent (both either anisotropic or not).

When no starting nor fixed parameter values are provided with init or known, the initialization process consists of fitting first an homoskedastic model of the data, called modHom. Unless provided with init$theta, initial lengthscales are taken at the middle of the range determined with lower and upper, while init$g_H may be use to pass an initial nugget value. The resulting lengthscales provide initial values for theta (or update them if given in init).

If necessary, a second homoskedastic model, modNugs, is fitted to the empirical residual variance between the prediction given by modHom at X0 and Z (up to modHom$nu2_hat). Note that when specifying settings$linkThetas == 'joint', then this second homoskedastic model has fixed lengthscale parameters. Starting values for theta_g and g are extracted from modNugs.

Finally, three initialization schemes for Delta are available with settings$initStrategy:

- for settings$initStrategy == 'simple', Delta is simply initialized to the estimated g value of modHom. Note that this procedure may fail when settings$penalty == TRUE.
- for settings$initStrategy == 'residuals', Delta is initialized to the estimated residual variance from the homoskedastic mean prediction.
- for settings$initStrategy == 'smoothed', Delta takes the values predicted by modNugs at X0.

Notice that lower and upper bounds cannot be equal for optim.

Value

a list which is given the S3 class "hetTP", with elements:

- theta: unless given, maximum likelihood estimate (mle) of the lengthscale parameter(s),
- Delta: unless given, mle of the nugget vector (non-smoothed),
- Lambda: predicted input noise variance at X0,
- nu2_hat: plugin estimator of the variance,
- theta_g: unless given, mle of the lengthscale(s) of the noise/log-noise process,
- k_theta_g: if settings$linkThetas == 'joint', mle for the constant by which lengthscale parameters of theta are multiplied to get theta_g,
- g: unless given, mle of the nugget of the noise/log-noise process,
- trendtype: either "SK" if beta0 is provided, else "OK",
- beta0 constant trend of the mean process, plugin-estimator unless given,
- nmean: plugin estimator for the constant noise/log-noise process mean,
- ll: log-likelihood value, (ll_non_pen) is the value without the penalty,
- nit_opt, msg: counts and message returned by optim
- modHom: homoskedastic GP model of class homGP used for initialization of the mean process,
- modNugs: homoskedastic GP model of class homGP used for initialization of the noise/log-noise process,
- nu2_hat_var: variance of the noise process,
- used_args: list with arguments provided in the call to the function, which is saved in call,
- X0, Z0, Z, eps, logN, covtype: values given in input

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

A. Shah, A. Wilson, Z. Ghahramani (2014), Student-t processes as alternatives to Gaussian processes, Artificial Intelligence and Statistics, 877–885.

See Also

predict.hetTP for predictions. A summary function is available as well.

Examples

```
##------------------------------------------------------------
## Example 1: Heteroskedastic TP modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
nvar <- 1
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")


## Model fitting
model <- mleHetTP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar),
                  covtype = "Matern5_2")

## Display averaged observations
points(model$X0, model$Z0, pch = 20)

## A quick view of the fit
summary(model)

## Create a prediction grid and obtain predictions
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
preds <- predict(x = xgrid, object =  model)

## Display mean predictive surface
lines(xgrid, preds$mean, col = 'red', lwd = 2)
## Display 95% confidence intervals
lines(xgrid, preds$mean + sqrt(preds$sd2) * qt(0.05, df = model$nu + nrow(X)), col = 2, lty = 2)
lines(xgrid, preds$mean + sqrt(preds$sd2) * qt(0.95, df = model$nu + nrow(X)), col = 2, lty = 2)
## Display 95% prediction intervals
lines(xgrid, preds$mean + sqrt(preds$sd2 + preds$nugs) * qt(0.05, df = model$nu + nrow(X)),
  col = 3, lty = 2)
lines(xgrid, preds$mean + sqrt(preds$sd2 + preds$nugs) * qt(0.95, df = model$nu + nrow(X)),
  col = 3, lty = 2)
```

```
##------------------------------------------------------------
## Example 2: 2D Heteroskedastic TP modeling
##------------------------------------------------------------
set.seed(1)
nvar <- 2

## Branin redefined in [0,1]^2
branin <- function(x){
  if(is.null(nrow(x)))
    x <- matrix(x, nrow = 1)
    x1 <- x[,1] * 15 - 5
    x2 <- x[,2] * 15
    (x2 - 5/(4 * pi^2) * (x1^2) + 5/pi * x1 - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(x1) + 10
}

## Noise field via standard deviation
noiseFun <- function(x){
  if(is.null(nrow(x)))
    x <- matrix(x, nrow = 1)
  return(1/5*(3*(2 + 2*sin(x[,1]*pi)*cos(x[,2]*3*pi) + 5*rowSums(x^2))))
}

## data generating function combining mean and noise fields
ftest <- function(x){
  return(branin(x) + rnorm(nrow(x), mean = 0, sd = noiseFun(x)))
}

## Grid of predictive locations
ngrid <- 51
xgrid <- matrix(seq(0, 1, length.out = ngrid), ncol = 1)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))

## Unique (randomly chosen) design locations
n <- 50
Xu <- matrix(runif(n * 2), n)

## Select replication sites randomly
X <- Xu[sample(1:n, 20*n, replace = TRUE),]

## obtain training data response at design locations X
Z <- ftest(X)

## Formating of data for model creation (find replicated observations)
prdata <- find_reps(X, Z, rescale = FALSE, normalize = FALSE)

## Model fitting
model <- mleHetTP(X = list(X0 = prdata$X0, Z0 = prdata$Z0, mult = prdata$mult), Z = prdata$Z, ,
                  lower = rep(0.01, nvar), upper = rep(10, nvar),
                  covtype = "Matern5_2")

## a quick view into the data stored in the "hetTP"-class object
summary(model)
```

```
## prediction from the fit on the grid
preds <- predict(x = Xgrid, object =  model)

## Visualization of the predictive surface
par(mfrow = c(2, 2))
contour(x = xgrid,  y = xgrid, z = matrix(branin(Xgrid), ngrid),
  main = "Branin function", nlevels = 20)
points(X, col = 'blue', pch = 20)
contour(x = xgrid,  y = xgrid, z = matrix(preds$mean, ngrid),
  main = "Predicted mean", nlevels = 20)
points(X, col = 'blue', pch = 20)
contour(x = xgrid,  y = xgrid, z = matrix(noiseFun(Xgrid), ngrid),
  main = "Noise standard deviation function", nlevels = 20)
points(X, col = 'blue', pch = 20)
contour(x = xgrid,  y= xgrid, z = matrix(sqrt(preds$nugs), ngrid),
  main = "Predicted noise values", nlevels = 20)
points(X, col = 'blue', pch = 20)
par(mfrow = c(1, 1))
```

`mleHomGP` *Gaussian process modeling with homoskedastic noise*

Description

Gaussian process regression under homoskedastic noise based on maximum likelihood estimation of the hyperparameters. This function is enhanced to deal with replicated observations.

Usage

```
mleHomGP(X, Z, lower, upper, known = NULL, noiseControl = list(g_bounds =
  c(sqrt(.Machine$double.eps), 100)), init = NULL, covtype = c("Gaussian",
  "Matern5_2", "Matern3_2"), maxit = 100, eps = sqrt(.Machine$double.eps),
  settings = list(return.Ki = TRUE))
```

Arguments

`X`	matrix of all designs, one per row, or list with elements: • `X0` matrix of unique design locations, one point per row • `Z0` vector of averaged observations, of length `nrow(X0)` • `mult` number of replicates at designs in `X0`, of length `nrow(X0)`
`Z`	vector of all observations. If using a list with `X`, `Z` has to be ordered with respect to `X0`, and of length `sum(mult)`
`lower, upper`	bounds for the `theta` parameter (see `cov_gen` for the exact parameterization). In the multivariate case, it is possible to give vectors for bounds (resp. scalars) for anisotropy (resp. isotropy)
`known`	optional list of known parameters, e.g., `beta0`, `theta` or `g`

noiseControl	list with element , • g_bounds, vector providing minimal and maximal noise to signal ratio
init	optional list specifying starting values for MLE optimization, with elements: • theta_init initial value of the theta parameters to be optimized over (default to (upper-lower)/2) • g_init initial value of the nugget parameter to be optimized over (default to 0.5)
covtype	covariance kernel type, either 'Gaussian', 'Matern5_2' or 'Matern3_2', see cov_gen
maxit	maximum number of iteration for L-BFGS-B of optim
eps	jitter used in the inversion of the covariance matrix for numerical stability
settings	list with argument return.Ki, to include the inverse covariance matrix in the object for further use (e.g., prediction).

Details

The global covariance matrix of the model is parameterized as K = nu2_hat * (C + g * diag(1/mult)), with C the correlation matrix between unique designs, depending on the family of kernel used (see cov_gen for available choices) and values of lengthscale parameters. nu2_hat is the plugin estimator of the variance of the process.

It is generally recommended to use find_reps to pre-process the data, to rescale the inputs to the unit cube and to normalize the outputs.

Value

a list which is given the S3 class "homGP", with elements:

- theta: maximum likelihood estimate of the lengthscale parameter(s),
- g: maximum likelihood estimate of the nugget variance,
- trendtype: either "SK" if beta0 is given, else "OK"
- beta0: estimated trend unless given in input,
- nu2_hat: plugin estimator of the variance,
- ll: log-likelihood value,
- X0, Z0, Z, mult, eps, covtype: values given in input,
- call: user call of the function
- used_args: list with arguments provided in the call
- nit_opt, msg: counts and msg returned by optim
- Ki, inverse covariance matrix (if return.Ki is TRUE in settings)

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

See Also

predict.homGP for predictions, update.homGP for updating an existing model. A summary function is available as well. mleHomTP provide a Student-t equivalent.

Examples

```
##------------------------------------------------------------
## Example 1: Homoskedastic GP modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")


model <- mleHomGP(X = X, Z = Z, lower = 0.01, upper = 100)

## Display averaged observations
points(model$X0, model$Z0, pch = 20)
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
predictions <- predict(x = xgrid, object =  model)

## Display mean prediction
lines(xgrid, predictions$mean, col = 'red', lwd = 2)
## Display 95% confidence intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2)), col = 2, lty = 2)
## Display 95% prediction intervals
lines(xgrid, qnorm(0.05, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)
lines(xgrid, qnorm(0.95, predictions$mean, sqrt(predictions$sd2 + predictions$nugs)),
  col = 3, lty = 2)
```

---

mleHomTP *Student-T process modeling with homoskedastic noise*

Description

Student-t process regression under homoskedastic noise based on maximum likelihood estimation of the hyperparameters. This function is enhanced to deal with replicated observations.

Usage

```
mleHomTP(X, Z, lower, upper, known = list(beta0 = 0),
  noiseControl = list(g_bounds = c(sqrt(.Machine$double.eps), 100), nu_bounds
  = c(2 + 0.001, 30), sigma2_bounds = c(sqrt(.Machine$double.eps), 10000)),
```

```
init = list(nu = 3, sigma2 = 1), covtype = c("Gaussian", "Matern5_2",
"Matern3_2"), maxit = 100, eps = sqrt(.Machine$double.eps),
settings = list(return.Ki = TRUE))
```

Arguments

X	matrix of all designs, one per row, or list with elements: • X0 matrix of unique design locations, one point per row • Z0 vector of averaged observations, of length nrow(X0) • mult number of replicates at designs in X0, of length nrow(X0)
Z	vector of all observations. If using a list with X, Z has to be ordered with respect to X0, and of length sum(mult)
lower, upper	bounds for the theta parameter (see cov_gen for the exact parameterization). In the multivariate case, it is possible to give vectors for bounds (resp. scalars) for anisotropy (resp. isotropy)
known	optional list of known parameters, e.g., beta0 (default to 0), theta, g, sigma2 or nu
noiseControl	list with element, • g_bound, vector providing minimal and maximal noise variance • sigma2_bounds, vector providing minimal and maximal signal variance • nu_bounds, vector providing minimal and maximal values for the degrees of freedom. The mininal value has to be stricly greater than 2. If the mle optimization gives a large value, e.g., 30, considering a GP with mleHomGP may be better.
init	list specifying starting values for MLE optimization, with elements: • theta_init initial value of the theta parameters to be optimized over (default to (upper-lower)/2) • g_init initial value of the nugget parameter to be optimized over (default to 0.5) • sigma2 initial value of the variance paramter (default to 1) • nu initial value of the degrees of freedom parameter (default to 3)
covtype	covariance kernel type, either 'Gaussian', 'Matern5_2' or 'Matern3_2', see cov_gen
maxit	maximum number of iteration for L-BFGS-B of optim
eps	jitter used in the inversion of the covariance matrix for numerical stability
settings	list with argument return.Ki, to include the inverse covariance matrix in the object for further use (e.g., prediction).

Details

The global covariance matrix of the model is parameterized as K = sigma2 * C + g * diag(1/mult), with C the correlation matrix between unique designs, depending on the family of kernel used (see cov_gen for available choices).

It is generally recommended to use find_reps to pre-process the data, to rescale the inputs to the unit cube and to normalize the outputs.

Value

a list which is given the S3 class "homGP", with elements:

- theta: maximum likelihood estimate of the lengthscale parameter(s),
- g: maximum likelihood estimate of the nugget variance,
- trendtype: either "SK" if beta0 is given, else "OK"
- beta0: estimated trend unless given in input,
- sigma2: maximum likelihood estimate of the scale variance,
- nu2: maximum likelihood estimate of the degrees of freedom parameter,
- ll: log-likelihood value,
- X0, Z0, Z, mult, eps, covtype: values given in input,
- call: user call of the function
- used_args: list with arguments provided in the call
- nit_opt, msg: counts and msg returned by optim
- Ki, inverse covariance matrix (if return.Ki is TRUE in settings)

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2017+), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

A. Shah, A. Wilson, Z. Ghahramani (2014), Student-t processes as alternatives to Gaussian processes, Artificial Intelligence and Statistics, 877–885.

See Also

predict.homTP for predictions. A summary function is available as well.

Examples

```
##------------------------------------------------------------
## Example 1: Homoskedastic Student-t modeling on the motorcycle data
##------------------------------------------------------------
set.seed(32)

## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
plot(X, Z, ylim = c(-160, 90), ylab = 'acceleration', xlab = "time")

noiseControl = list(g_bounds = c(1e-3, 1e4))
model <- mleHomTP(X = X, Z = Z, lower = 0.01, upper = 100, noiseControl = noiseControl)
summary(model)

## Display averaged observations
```

```
points(model$X0, model$Z0, pch = 20)
xgrid <- matrix(seq(0, 60, length.out = 301), ncol = 1)
preds <- predict(x = xgrid, object =  model)

## Display mean prediction
lines(xgrid, preds$mean, col = 'red', lwd = 2)
## Display 95% confidence intervals
lines(xgrid, preds$mean + sqrt(preds$sd2) * qt(0.05, df = model$nu + nrow(X)), col = 2, lty = 2)
lines(xgrid, preds$mean + sqrt(preds$sd2) * qt(0.95, df = model$nu + nrow(X)), col = 2, lty = 2)
## Display 95% prediction intervals
lines(xgrid, preds$mean + sqrt(preds$sd2 + preds$nugs) * qt(0.05, df = model$nu + nrow(X)),
  col = 3, lty = 2)
lines(xgrid, preds$mean + sqrt(preds$sd2 + preds$nugs) * qt(0.95, df = model$nu + nrow(X)),
  col = 3, lty = 2)
```

`predict.hetGP` *Gaussian process predictions using a heterogeneous noise GP object (of class* `hetGP`*)*

Description

Gaussian process predictions using a heterogeneous noise GP object (of class `hetGP`)

Usage

```
## S3 method for class 'hetGP'
predict(object, x, noise.var = FALSE, xprime = NULL,
  nugs.only = FALSE, ...)
```

Arguments

`object`	an object of class `hetGP`; e.g., as returned by `mleHetGP`
`x`	matrix of designs locations to predict at (one point per row)
`noise.var`	should the variance of the latent variance process be returned?
`xprime`	optional second matrix of predictive locations to obtain the predictive covariance matrix between `x` and `xprime`
`nugs.only`	if `TRUE`, only return noise variance prediction
`...`	no other argument for this method.

Details

The full predictive variance corresponds to the sum of `sd2` and `nugs`. See `mleHetGP` for examples.

Value

list with elements

- mean: kriging mean;
- sd2: kriging variance (filtered, e.g. without the nugget values)
- nugs: noise variance
- sd2_var: (optional) kriging variance of the noise process
- cov: (optional) predictive covariance matrix between x and xprime

---

`predict.hetTP` *Student-t process predictions using a heterogeneous noise TP object (of class* `hetTP`*)*

Description

Student-t process predictions using a heterogeneous noise TP object (of class hetTP)

Usage

```
## S3 method for class 'hetTP'
predict(object, x, noise.var = FALSE, xprime = NULL,
  nugs.only = FALSE, ...)
```

Arguments

object	an object of class hetTP; e.g., as returned by mleHetTP
x	matrix of designs locations to predict at
noise.var	should the variance of the latent variance process be returned?
xprime	optional second matrix of predictive locations to obtain the predictive covariance matrix between x and xprime
nugs.only	if TRUE, only return noise variance prediction
...	no other argument for this method.

Details

The full predictive variance corresponds to the sum of sd2 and nugs.

Value

list with elements

- mean: kriging mean;
- sd2: kriging variance (filtered, e.g. without the nugget values)
- nugs: noise variance
- sd2_var: (optional) kriging variance of the noise process
- cov: (optional) predictive covariance matrix between x and xprime

`predict.homGP` *Gaussian process predictions using a homoskedastic noise GP object (of class* `homGP`*)*

Description

Gaussian process predictions using a homoskedastic noise GP object (of class `homGP`)

Usage

```
## S3 method for class 'homGP'
predict(object, x, xprime = NULL, ...)
```

Arguments

`object`	an object of class `homGP`; e.g., as returned by `mleHomGP`
`x`	matrix of designs locations to predict at (one point per row)
`xprime`	optional second matrix of predictive locations to obtain the predictive covariance matrix between `x` and `xprime`
`...`	no other argument for this method

Details

The full predictive variance corresponds to the sum of `sd2` and `nugs`. See `mleHomGP` for examples.

Value

list with elements

- `mean`: kriging mean;
- `sd2`: kriging variance (filtered, e.g. without the nugget value)
- `cov`: predictive covariance matrix between `x` and `xprime`
- `nugs`: nugget value at each prediction location, for consistency with `mleHomGP`.

predict.homTP *Student-t process predictions using a homoskedastic noise GP object (of class* homGP*)*

Description

Student-t process predictions using a homoskedastic noise GP object (of class homGP)

Usage

```
## S3 method for class 'homTP'
predict(object, x, xprime = NULL, ...)
```

Arguments

object	an object of class homGP; e.g., as returned by mleHomTP
x	matrix of designs locations to predict at
xprime	optional second matrix of predictive locations to obtain the predictive covariance matrix between x and xprime
...	no other argument for this method

Details

The full predictive variance corresponds to the sum of sd2 and nugs.

Value

list with elements

- mean: kriging mean;
- sd2: kriging variance (filtered, e.g. without the nugget value)
- cov: predictive covariance matrix between x and xprime
- nugs: nugget value at each prediction location

`sirEval` *SIR test problem*

Description

Epidemiology problem, initial and rescaled to [0,1]^2 versions.

Usage

```
sirEval(x)

sirSimulate(S0 = 1990, I0 = 10, M = S0 + I0, beta = 0.75, gamma = 0.5,
  imm = 0)
```

Arguments

`x`	vector of size two
`S0`	initial nunber of susceptibles
`I0`	initial number of infected
`M`	total population
`beta, gamma, imm`	control rates

References

R. Hu, M. Ludkovski (2017), Sequential Design for Ranking Response Surfaces, SIAM/ASA Journal on Uncertainty Quantification, 5(1), 212-239.

Examples

```
## SIR test problem illustration
ngrid <- 10 # increase
xgrid <- seq(0, 1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))

nrep <- 5 # increase
X <- Xgrid[rep(1:nrow(Xgrid), nrep),]
Y <- apply(X, 1, sirEval)
dataSIR <- find_reps(X, Y)
filled.contour(xgrid, xgrid, matrix(lapply(dataSIR$Zlist, sd), ngrid),
               xlab = "Susceptibles", ylab = "Infecteds", color.palette = terrain.colors)
```

update.hetGP *Update "hetGP"-class model fit with new observations*

Description

Fast update of existing hetGP model with new observations.

Usage

```
## S3 method for class 'hetGP'
update(object, Xnew, Znew, ginit = 0.01, lower = NULL,
  upper = NULL, noiseControl = NULL, settings = NULL, known = NULL,
  maxit = 100, method = "quick", ...)
```

Arguments

object	previously fit "hetGP"-class model
Xnew	matrix of new design locations; ncol(Xnew) must match the input dimension encoded in object
Znew	vector new observations at those design locations, of length nrow(X). NAs can be passed, see Details
ginit	minimal value of the smoothing parameter (i.e., nugget of the noise process) for optimization initialisation. It is compared to the g hyperparameter in the object.
lower, upper, noiseControl, settings, known	optional bounds for mle optimization, see mleHetGP. If not provided, they are extracted from the existing model
maxit	maximum number of iterations for the internal L-BFGS-B optimization method; see optim for more details
method	one of "quick", "mixed" see Details.
...	no other argument for this method.

Details

The update can be performed with or without re-estimating hyperparameter. In the first case, mleHetGP is called, based on previous values for initialization. The only missing values are the latent variables at the new points, that are initialized based on two possible update schemes in method:

- "quick" the new delta value is the predicted nugs value from the previous noise model;
- "mixed" new values are taken as the barycenter between prediction given by the noise process and empirical variance.

The subsequent number of MLE computations can be controlled with maxit.

In case hyperparameters need not be updated, maxit can be set to 0. In this case it is possible to pass NAs in Znew, then the model can still be used to provide updated variance predictions.

Examples

```
##------------------------------------------------------------
## Sequential update example
##------------------------------------------------------------
set.seed(42)

## Spatially varying noise function
noisefun <- function(x, coef = 1){
  return(coef * (0.05 + sqrt(abs(x)*20/(2*pi))/10))
}

## Initial data set
nvar <- 1
n <- 20
X <- matrix(seq(0, 2 * pi, length=n), ncol = 1)
mult <- sample(1:10, n, replace = TRUE)
X <- rep(X, mult)
Z <- sin(X) + rnorm(length(X), sd = noisefun(X))

## Initial fit
testpoints <- matrix(seq(0, 2*pi, length = 10*n), ncol = 1)
model <- model_init <- mleHetGP(X = X, Z = Z, lower = rep(0.1, nvar),
  upper = rep(50, nvar), maxit = 1000, settings = list(hardpenalty = TRUE))

## Visualizing initial predictive surface
preds <- predict(x = testpoints, model_init)
plot(X, Z)
lines(testpoints, preds$mean, col = "red")

## 10 fast update steps
nsteps <- 5
npersteps <- 10
for(i in 1:nsteps){
  newIds <- sort(sample(1:(10*n), npersteps))

  newX <- testpoints[newIds, drop = FALSE]
  newZ <- sin(newX) + rnorm(length(newX), sd = noisefun(newX))
  points(newX, newZ, col = "blue", pch = 20)
  model <- update(object = model, Xnew = newX, Znew = newZ)
  X <- c(X, newX)
  Z <- c(Z, newZ)
  plot(X, Z)
  print(model$nit_opt)
}

## Final predictions after 10 updates
preds_fin <- predict(x=testpoints, model)

## Visualizing the result by augmenting earlier plot
lines(testpoints, preds_fin$mean, col = "blue")
lines(testpoints, qnorm(0.05, preds_fin$mean, sqrt(preds_fin$sd2)), col = "blue", lty = 2)
lines(testpoints, qnorm(0.95, preds_fin$mean, sqrt(preds_fin$sd2)), col = "blue", lty = 2)
```

```
lines(testpoints, qnorm(0.05, preds_fin$mean, sqrt(preds_fin$sd2 + preds_fin$nugs)),
  col = "blue", lty = 3)
lines(testpoints, qnorm(0.95, preds_fin$mean, sqrt(preds_fin$sd2 + preds_fin$nugs)),
  col = "blue", lty = 3)

## Now compare to what you would get if you did a full batch fit instead
model_direct <-  mleHetGP(X = X, Z = Z, maxit = 1000, settings = list(hardpenalty = TRUE),
                          lower = rep(0.1, nvar), upper = rep(50, nvar),
                      init = list(theta = model_init$theta, k_theta_g = model_init$k_theta_g))
preds_direct <- predict(x = testpoints, model_direct)
print(model_direct$nit_opt)
lines(testpoints, preds_direct$mean, col = "green")
lines(testpoints, qnorm(0.05, preds_direct$mean, sqrt(preds_direct$sd2)), col = "green",
  lty = 2)
lines(testpoints, qnorm(0.95, preds_direct$mean, sqrt(preds_direct$sd2)), col = "green",
  lty = 2)
lines(testpoints, qnorm(0.05, preds_direct$mean, sqrt(preds_direct$sd2 + preds_direct$nugs)),
  col = "green", lty = 3)
lines(testpoints, qnorm(0.95, preds_direct$mean, sqrt(preds_direct$sd2 + preds_direct$nugs)),
  col = "green", lty = 3)
lines(testpoints, sin(testpoints), col = "red", lty = 2)

## Compare outputs
summary(model_init)
summary(model)
summary(model_direct)
```

---

update.homGP *Fast* homGP-*update*

Description

Update existing homGP model with new observations

Usage

```
## S3 method for class 'homGP'
update(object, Xnew, Znew = NULL, lower = NULL,
  upper = NULL, noiseControl = NULL, known = NULL, maxit = 100, ...)
```

Arguments

object	initial model of class homGP
Xnew	matrix of new design locations; ncol(Xnew) must match the input dimension encoded in object
Znew	vector new observations at those new design locations, of length nrow(X). NAs can be passed, see Details

`lower, upper, noiseControl, known`
: optional bounds for MLE optimization, see `mleHomGP`. If not provided, they are extracted from the existing model

`maxit`
: maximum number of iterations for the internal L-BFGS-B optimization method; see `optim` for more details

`...`
: no other argument for this method.

Details

In case hyperparameters need not be updated, `maxit` can be set to `0`. In this case it is possible to pass `NA`s in `Znew`, then the model can still be used to provide updated variance predictions.

Examples

```
## Not run:
##------------------------------------------------------------
## Example : Sequential Homoskedastic GP modeling
##------------------------------------------------------------
set.seed(42)

## Spatially varying noise function
noisefun <- function(x, coef = 1){
  return(coef * (0.05 + sqrt(abs(x)*20/(2*pi))/10))
}

nvar <- 1
n <- 10
X <- matrix(seq(0, 2 * pi, length=n), ncol = 1)
mult <- sample(1:10, n)
X <- rep(X, mult)
Z <- sin(X) + rnorm(length(X), sd = noisefun(X))

testpoints <- matrix(seq(0, 2*pi, length = 10*n), ncol = 1)
model <- model_init <- mleHomGP(X = X, Z = Z,
                                lower = rep(0.1, nvar), upper = rep(50, nvar))
preds <- predict(x = testpoints, object = model_init)
plot(X, Z)
lines(testpoints, preds$mean, col = "red")


nsteps <- 10
for(i in 1:nsteps){
  newIds <- sort(sample(1:(10*n), 10))

  newX <- testpoints[newIds, drop = FALSE]
  newZ <- sin(newX) + rnorm(length(newX), sd = noisefun(newX))
  points(newX, newZ, col = "blue", pch = 20)
  model <- update(object = model, newX, newZ)
  X <- c(X, newX)
  Z <- c(Z, newZ)
  plot(X, Z)
  print(model$nit_opt)
```

```
}
preds_fin <- predict(x = testpoints, object = model)
lines(testpoints, preds_fin$mean, col = "blue")
lines(testpoints, qnorm(0.05, preds_fin$mean, sqrt(preds_fin$sd2)), col = "blue", lty = 2)
lines(testpoints, qnorm(0.95, preds_fin$mean, sqrt(preds_fin$sd2)), col = "blue", lty = 2)
lines(testpoints, qnorm(0.05, preds_fin$mean, sqrt(preds_fin$sd2 + preds_fin$nugs)),
      col = "blue", lty = 3)
lines(testpoints, qnorm(0.95, preds_fin$mean, sqrt(preds_fin$sd2 + preds_fin$nugs)),
      col = "blue", lty = 3)

model_direct <-  mleHomGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar))
preds_direct <- predict(x = testpoints, object = model_direct)
print(model_direct$nit_opt)
lines(testpoints, preds_direct$mean, col = "green")
lines(testpoints, qnorm(0.05, preds_direct$mean, sqrt(preds_direct$sd2)), col = "green", lty = 2)
lines(testpoints, qnorm(0.95, preds_direct$mean, sqrt(preds_direct$sd2)), col = "green", lty = 2)
lines(testpoints, qnorm(0.05, preds_direct$mean, sqrt(preds_direct$sd2 + preds_direct$nugs)),
      col = "green", lty = 3)
lines(testpoints, qnorm(0.95, preds_direct$mean, sqrt(preds_direct$sd2 + preds_direct$nugs)),
      col = "green", lty = 3)

lines(testpoints, sin(testpoints), col = "red", lty = 2)

## Compare outputs
summary(model_init)
summary(model)
summary(model_direct)



## End(Not run)
```

`update_horizon` *Adapt horizon*

Description

Adapt the look-ahead horizon depending on the replicate allocation or a target ratio

Usage

```
update_horizon(model, current_horizon = NULL, previous_ratio = NULL,
  target = NULL, Wijs = NULL)
```

Arguments

`model`	`hetGP` or `homGP` model
`current_horizon`	horizon used for the previous iteration, see details

`previous_ratio`	ratio before adding the previous new design
`target`	scalar in]0,1] for desired n/N
`Wijs`	optional previously computed matrix of Wijs, see `Wij`

Details

If `target` is provided, along with `previous_ratio` and `current_horizon`:

- the horizon is increased by one if more replicates are needed but a new ppint has been added at the previous iteration,
- the horizon is decreased by one if new points are needed but a replicate has been added at the previous iteration,
- otherwise it is unchanged.

If no `target` is provided, `allocate_mult` is used to obtain the best allocation of the existing replicates, then the new horizon is sampled from the difference between the actual allocation and the best one, bounded below by 0. See (Binois et al. 2017).

Value

randomly selected horizon for next iteration (adpative) if no `target` is provided, otherwise returns the update horizon value.

References

M. Binois, J. Huang, R. Gramacy, M. Ludkovski (2017+), Replication or exploration? Sequential design for stochastic simulation experiments.

`Wij` *Compute double integral of the covariance kernel over a [0,1]^d domain*

Description

Compute double integral of the covariance kernel over a [0,1]^d domain

Usage

```
Wij(mu1, mu2 = NULL, theta, type)
```

Arguments

`mu1, mu2`	input locations considered
`theta`	lengthscale hyperparameter of the kernel
`type`	kernel type, one of "`Gaussian`", "`Matern5_2`" or "`Matern3_2`", see `cov_gen`

References

M. Binois, J. Huang, R. Gramacy, M. Ludkovski (2017+), Replication or exploration? Sequential design for stochastic simulation experiments.

Index