Package 'hetGP'

March 8, 2023

Type Package

Title Heteroskedastic Gaussian Process Modeling and Design under Replication

Version 1.1.5

Date 2023-03-06

Author Mickael Binois, Robert B. Gramacy

Maintainer Mickael Binois <mickael.binois@inria.fr>

Description
Performs Gaussian process regression with heteroskedastic noise following the model by Binois, M., Gramacy, R., Ludkovski, M. (2016) <arXiv:1611.05902>, with implementation details in Binois, M. & Gramacy, R. B. (2021) <doi:10.18637/jss.v098.i13>. 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 FALSE

Depends R (>= 2.10),

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

LinkingTo Rcpp

Suggests knitr, monomvn, lhs, colorspace

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation yes

Repository CRAN

Date/Publication 2023-03-08 09:40:06 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_IMSPE` for adding a new design based on the Integrated Mean Square Prediction Error.
`IMSPE_optim` for augmenting a design, possibly based on a lookahead heuristic to bias the search toward replication.
`crit_optim` is similar to `IMSPE_optim` but for the optimization or contour finding criterion available.

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 (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on arXiv:1611.05902.

M. Binois, J. Huang, R. B. Gramacy, M. Ludkovski (2019), Replication or exploration? Sequential design for stochastic simulation experiments, Technometrics, 61(1), 7–23.
Preprint available on arXiv:1710.03206.

M. Chung, M. Binois, R. B. Gramacy, DJ Moquin, AP Smith, AM Smith (2019). Parameter and Uncertainty Estimation for Dynamical Systems Using Surrogate Stochastic Processes. SIAM Journal on Scientific Computing, 41(4), 2212–2238.

Preprint available on arXiv:1802.00852.

M. Binois, R. B. Gramacy (2021). hetGP: Heteroskedastic Gaussian Process Modeling and Sequential Design in R. Journal of Statistical Software, 98(13), 1–44.

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
settings <- list(return.hom = TRUE) # To keep homoskedastic model used for training
model <- mleHetGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar),
                  covtype = "Matern5_2", settings = settings)

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

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

 if(current_horizon == -1){
   opt <- IMSPE_optim(model = model, h = 0, Wijs = Wijs)
 }else{
   opt <- IMSPE_optim(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(5, nvar),
                  covtype = "Matern5_2")
## Compute best allocation
A <- allocate_mult(model, N = 1000)
```

```
## Create a prediction grid and obtain predictions
xgrid <- matrix(seq(0, 1, 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)

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)
```

`ato` *Assemble To Order (ATO) Data and Fits*

Description

A batch design-evaluated ATO data set, random partition into training and testing, and fitted **hetGP** model; similarly a sequentially designed adaptive horizon data set, and associated fitted **hetGP** model

Usage

```
data(ato)
```

Format

Calling `data(ato)` causes the following objects to be loaded into the namespace.

`X` 2000x8 `matrix` of inputs coded from 1,...,20 to the unit 8-cube; original inputs can be recreated as `X*19 + 1`

`Z` 2000x10 `matrix` of normalized outputs obtained in ten replicates at each of the 2000 inputs `X`. Original outputs can be obtained as `Z*sqrt(Zv) + Zm`

`Zm` scalar mean used to normalize `Z`

`Zv` scalar variance used to normalize `Z`

`train` vector of 1000 rows of `X` and `Z` selected for training

`Xtrain` 1000x8 `matrix` obtained as a random partition of `X`

`Ztrain` length 1000 list of vectors containing the selected (replicated) observations at each row of `Xtrain`

`mult` the length of each entry of `Ztrain`; same as `unlist(lapply(Ztrain, length))`

`kill` a `logical` vector indicating which rows of `Xtrain` for which all replicates of `Z` are selected for `Ztrain`; same as `mult == 10`

`Xtrain.out` 897x8 `matrix` comprised of the subset of `X` where not all replicates are selected for training; i.e., those for which `kill == FALSE`

`Ztrain.out` `list` of length 897 containing the replicates of `Z` not selected for `Ztrain`

`nc` `noiseControl` argument for `mleHetGP` call

`out` `mleHetGP` model based on `Xtrain` and `Ztrain` using `noiseControl=nc`

`Xtest` 1000x8 `matrix` containing the other partition of `X` of locations not selected for training

`Ztest` 1000x10 `matrix` of responses from the partition of `Z` not selected for training

`ato.a` 2000x9 `matrix` of sequentially designed inputs (8) and outputs (1) obtained under an adaptive horizon scheme

`Xa` 2000x8 matrix of coded inputs from `ato.a` as `(ato.a[,1:8]-1)/19`

`Za` length 2000 vector of outputs from `ato.a` as `(ato.a[,9] - Zm)/sqrt(Zv)`

`out.a` `mleHetGP` model based on `Xa` and `Za` using `noiseControl=nc`

Details

The assemble to order (ATO) simulator (Hong, Nelson, 2006) is a queuing simulation targeting inventory management scenarios. The setup is as follows. A company manufactures m products. Products are built from base parts called items, some of which are "key" in that the product cannot be built without them. If a random request comes in for a product that is missing a key item, a replenishment order is executed, and is filled after a random period. Holding items in inventory is expensive, so there is a balance between inventory costs and revenue. Hong & Nelson built a `Matlab` simulator for this setup, which was subsequently reimplemented by Xie, et al., (2012).

Binois, et al (2018a) describe an out-of-sample experiment based on this latter implementation in its default (Hong & Nelson) setting, specifying item cost structure, product makeup (their items) and revenue, distribution of demand and replenishment time, under target stock vector inputs $b \in \{1, \ldots, 20\}^8$ for eight items. They worked with 2000 random uniform input locations (`X`), and ten replicate responses at each location (`Z`). The partition of 1000 training data points (`Xtrain` and `Ztrain`) and 1000 testing (`Xtest` and `Ztest`) sets provided here is an example of one that was used for the Monte Carlo experiment in that paper. The elements `Xtrain.out` and `Ztrain.out` comprise of replicates from the training inputs which were not used in training, so may be used for out-of-sample testing. For more details on how the partitions were build, see the code in the examples section below.

Binois, et al (2018b) describe an adaptive lookahead horizon scheme for building a sequential design (`Xa`, `Za`) of size 2000 whose predictive performance, via proper scores, is almost as good as the approximately 5000 training data sites in each of the Monte Carlo repetitions described above. The example code below demonstrates this via out-of-sample predictions on `Xtest` (measured against `Ztest`) when `Xtrain` and `Ztrain` are used compared to those from `Xa` and `Za`.

Note

The mleHetGP output objects were build with return.matrices=FALSE for more compact storage. Before these objects can be used for calculations, e.g., prediction or design, these covariance matrices need to be rebuilt with rebuild. The generic predict method will call rebuild automatically, however, some of the other methods will not, and it is often more efficient to call rebuild once at the outset, rather than for every subsequent predict call

Author(s)

Mickael Binois, <mbinois@mcs.anl.gov>, and Robert B. Gramacy, <rbg@vt.edu>

References

Hong L., Nelson B. (2006), Discrete optimization via simulation using COMPASS. Operations Research, 54(1), 115-129.

Xie J., Frazier P., Chick S. (2012). Assemble to Order Simulator. http://simopt.org/wiki/index.php?title=Assemble_to_Order&oldid=447.

M. Binois, J. Huang, R. Gramacy, M. Ludkovski (2018a), Replication or exploration? Sequential design for stochastic simulation experiments, arXiv preprint arXiv:1710.03206.

M. Binois, Robert B. Gramacy, M. Ludkovski (2018b), Practical heteroskedastic Gaussian process modeling for large simulation experiments, arXiv preprint arXiv:1611.05902.

See Also

bfs, sirEval, link{rebuild}, horizon, IMSPE_optim, mleHetGP, vignette("hetGP")

Examples

```
data(ato)

## Not run:
##
## the code below was used to create the random partition
##

## recover the data in its original form
X <- X*19+1
Z <- Z*sqrt(Zv) + Zm

## code the inputs and outputs; i.e., undo the transformation
## above
X <- (X-1)/19
Zm <- mean(Z)
Zv <- var(as.vector(Z))
Z <- (Z - Zm)/sqrt(Zv)

## random training and testing partition
train <- sample(1:nrow(X), 1000)
Xtrain <- X[train,]
Xtest <- X[-train,]
```

```
Ztest <- as.list(as.data.frame(t(Z[-train,])))
Ztrain <- Ztrain.out <- list()
mult <- rep(NA, nrow(Xtrain))
kill <- rep(FALSE, nrow(Xtrain))
for(i in 1:length(train)) {
  reps <- sample(1:ncol(Z), 1)
  w <- sample(1:ncol(Z), reps)
  Ztrain[[i]] <- Z[train[i],w]
  if(reps < 10) Ztrain.out[[i]] <- Z[train[i],-w]
  else kill[i] <- TRUE
  mult[i] <- reps
}

## calculate training locations and outputs for replicates not
## included in Ztrain
Xtrain.out <- Xtrain[!kill,]
Ztrain.out <- Ztrain[which(!kill)]

## fit hetGP model
out <- mleHetGP(X=list(X0=Xtrain, Z0=sapply(Ztrain, mean), mult=mult),
  Z=unlist(Ztrain), lower=rep(0.01, ncol(X)), upper=rep(30, ncol(X)),
  covtype="Matern5_2", noiseControl=nc, known=list(beta0=0),
  maxit=100000, settings=list(return.matrices=FALSE))

##
## the adaptive lookahead design is read in and fit as
## follows
##
Xa <- (ato.a[,1:8]-1)/19
Za <- ato.a[,9]
Za <- (Za - Zm)/sqrt(Zv)

## uses nc defined above
out.a <- mleHetGP(Xa, Za, lower=rep(0.01, ncol(X)),
  upper=rep(30, ncol(X)), covtype="Matern5_2", known=list(beta0=0),
  noiseControl=nc, maxit=100000, settings=list(return.matrices=FALSE))

## End(Not run)

##
## the following code duplicates a predictive comparison in
## the package vignette
##

## first using the model fit to the train partition (out)
out <- rebuild(out)

## predicting out-of-sample at the test sights
phet <- predict(out, Xtest)
phets2 <- phet$sd2 + phet$nugs
mhet <- as.numeric(t(matrix(rep(phet$mean, 10), ncol=10)))
s2het <- as.numeric(t(matrix(rep(phets2, 10), ncol=10)))
sehet <- (unlist(t(Ztest)) - mhet)^2
```

```
sc <- - sehet/s2het - log(s2het)
mean(sc)

## predicting at the held-out training replicates
phet.out <- predict(out, Xtrain.out)
phets2.out <- phet.out$sd2 + phet.out$nugs
s2het.out <- mhet.out <- Ztrain.out
for(i in 1:length(mhet.out)) {
  mhet.out[[i]] <- rep(phet.out$mean[i], length(mhet.out[[i]]))
  s2het.out[[i]] <- rep(phets2.out[i], length(s2het.out[[i]]))
}
mhet.out <- unlist(t(mhet.out))
s2het.out <- unlist(t(s2het.out))
sehet.out <- (unlist(t(Ztrain.out)) - mhet.out)^2
sc.out <- - sehet.out/s2het.out - log(s2het.out)
mean(sc.out)

## Not run:
## then using the model trained from the "adaptive"
## sequential design, with comparison from the "batch"
## one above, using the scores function
out.a <- rebuild(out.a)
sc.a <- scores(out.a, Xtest = Xtest, Ztest = Ztest)
c(batch=mean(sc), adaptive=sc.a)

## an example of one iteration of sequential design

  Wijs <- Wij(out.a$X0, theta=out.a$theta, type=out.a$covtype)
  h <- horizon(out.a, Wijs=Wijs)
  control = list(tol_dist=1e-4, tol_diff=1e-4, multi.start=30, maxit=100)
  opt <- IMSPE_optim(out.a, h, Wijs=Wijs, control=control)
  opt$par

## End(Not run)
```

---

bfs *Bayes Factor Data*

Description

Data from a Bayes factor MCMC-based simulation experiment comparing Student-t to Gaussian errors in an RJ-based Laplace prior Bayesian linear regession setting

Usage

```
data(ato)
```

Format

Calling `data(bfs)` causes the following objects to be loaded into the namespace.

`bfs.exp` 20x11 `data.frame` whose first column is θ, indicating the mean parameter of an exponential distribution encoding the prior of the Student-t degrees of freedom parameter ν. The remaining ten columns comprise of Bayes factor evaluations under that setting

`bfs.gamma` 80x7 `data.frame` whose first two columns are β and α, indicating the second and first parameters to a Gamma distribution encoding the prior of the Student-t degrees of freedom parameters ν. The remaining five columns comprise of Bayes factor evaluations under those settings

Details

Gramacy & Pantaleo (2010), Sections 3-3-3.4, describe an experiment involving Bayes factor (BF) calculations to determine if data are leptokurtic (Student-t errors) or not (simply Gaussian) as a function of the prior parameterization on the Student-t degrees of freedom parameter ν. Franck & Gramacy (2018) created a grid of hyperparameter values in θ describing the mean of an Exponential distribution, evenly spaced in $\log_{10}$ space from `10^(-3)` to `10^6` spanning "solidly Student-t" (even Cauchy) to "essentially Gaussian" in terms of the mean of the prior over ν. For each θ setting on the grid they ran the Reversible Jump (RJ) MCMC to approximate the BF of Student-t over Gaussian by feeding in sample likelihood evaluations provided by **monomvn**'s `blasso` to compute the BF. In order to understand the Monte Carlo variability in those calculations, ten replicates of the BFs under each hyperparameter setting were collected. These data are provided in `bfs.exp`.

A similar, larger experiment was provided with ν under a Gamma prior with parameters α and $\beta \equiv \theta$. In this higher dimensional space, a Latin hypercube sample of size eighty was created, and five replicates of BFs were recorded. These data are provided in `bfs.gamma`.

The examples below involve `mleHetTP` fits (Chung, et al., 2018) to these data and a visualization of the predictive surfaces thus obtained. The code here follows an example provided, with more detail, in `vignette("hetGP")`

Note

For code showing how these BFs were calculated, see supplementary material from Franck & Gramacy (2018)

Author(s)

Mickael Binois, <mbinois@mcs.anl.gov>, and Robert B. Gramacy, <rbg@vt.edu>

References

Franck CT, Gramacy RB (2018). Assessing Bayes factor surfaces using interactive visualization and computer surrogate modeling. Preprint available on arXiv:1809.05580.

Gramacy RB (2017). **monomvn**: Estimation for Multivariate Normal and Student-t Data with Monotone Missingness. R package version 1.9-7, https://CRAN.R-project.org/package=monomvn.

R.B. Gramacy and E. Pantaleo (2010). Shrinkage regression for multivariate inference with missing data, and an application to portfolio balancing. Bayesian Analysis 5(2), 237-262. Preprint available on arXiv:0907.2135

Chung M, Binois M, Gramacy RB, Moquin DJ, Smith AP, Smith AM (2018). Parameter and Uncertainty Estimation for Dynamical Systems Using Surrogate Stochastic Processes. SIAM Journal on Scientific Computing, 41(4), 2212-2238. Preprint available on arXiv:1802.00852.

See Also

ato, sirEval, mleHetTP, vignette("hetGP"), blasso

Examples

```
data(bfs)

##
## Exponential version first
##

thetas <- matrix(bfs.exp$theta, ncol=1)
bfs <- as.matrix(t(bfs.exp[,-1]))

## the data are heavy tailed, so t-errors help
bfs1 <- mleHetTP(X=list(X0=log10(thetas), Z0=colMeans(log(bfs)),
  mult=rep(nrow(bfs), ncol(bfs))), Z=log(as.numeric(bfs)), lower=10^(-4),
  upper=5, covtype="Matern5_2")

## predictions on a grid in 1d
dx <- seq(0,1,length=100)
dx <- 10^(dx*4 - 3)
p <- predict(bfs1, matrix(log10(dx), ncol=1))

## visualization
matplot(log10(thetas), t(log(bfs)), col=1, pch=21, ylab="log(bf)",
  main="Bayes factor surface")
lines(log10(dx), p$mean, lwd=2, col=2)
lines(log10(dx), p$mean + 2*sqrt(p$sd2 + p$nugs), col=2, lty=2, lwd=2)
lines(log10(dx), p$mean - 2*sqrt(p$sd2 + p$nugs), col=2, lty=2, lwd=2)
legend("topleft", c("hetTP mean", "hetTP interval"), lwd=2, lty=1:2, col=2)

##
## Now Gamma version
##

D <- as.matrix(bfs.gamma[,1:2])
bfs <- as.matrix(t(bfs.gamma[,-(1:2)]))

## fitting in 2fd
bfs2 <- mleHetTP(X=list(X0=log10(D), Z0=colMeans(log(bfs)),
  mult=rep(nrow(bfs), ncol(bfs))), Z = log(as.numeric(bfs)),
  lower = rep(10^(-4), 2), upper = rep(5, 2), covtype = "Matern5_2",
  maxit=100000)

## predictions on a grid in 2d
dx <- seq(0,1,length=100)
```

```
dx <- 10^(dx*4 - 3)
DD <- as.matrix(expand.grid(dx, dx))
p <- predict(bfs2, log10(DD))

## visualization via image-contour plots
par(mfrow=c(1,2))
mbfs <- colMeans(bfs)
image(log10(dx), log10(dx), t(matrix(p$mean, ncol=length(dx))),
  col=heat.colors(128), xlab="log10 alpha", ylab="log10 beta",
  main="mean log BF")
text(log10(D[,2]), log10(D[,1]), signif(log(mbfs), 2), cex=0.5)
contour(log10(dx), log10(dx),t(matrix(p$mean, ncol=length(dx))),
  levels=c(-5,-3,-1,0,1,3,5), add=TRUE, col=4)
image(log10(dx), log10(dx), t(matrix(sqrt(p$sd2 + p$nugs),
  ncol=length(dx))),  col=heat.colors(128), xlab="log10 alpha",
  ylab="log10 beta", main="sd log BF")
text(log10(D[,2]), log10(D[,1]), signif(apply(log(bfs), 2, sd), 2),
  cex=0.5)
```

`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 (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on 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_cSUR — *Contour Stepwise Uncertainty Reduction criterion*

Description

Computes cSUR infill criterion

Usage

```
crit_cSUR(x, model, thres = 0, preds = NULL)
```

Arguments

x	matrix of new designs, one point per row (size n x d)
model	homGP or hetGP model, including inverse matrices
thres	for contour finding
preds	optional predictions at x to avoid recomputing if already done (must contain cov)

References

Lyu, X., Binois, M. & Ludkovski, M. (2018+). Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation. arXiv:1807.06712.

Examples

```
## Infill criterion example
set.seed(42)
branin <- function(x){
  m <- 54.8104; s <- 51.9496
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  xx <- 15 * x[,1] - 5; y <- 15 * x[,2]
  f <- (y - 5.1 * xx^2/(4 * pi^2) + 5 * xx/pi - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(xx) + 10
  f <- (f - m)/s
  return(f)
}

ftest <- function(x, sd = 0.1){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  return(apply(x, 1, branin) + rnorm(nrow(x), sd = sd))
}

ngrid <- 101; xgrid <- seq(0, 1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
Zgrid <- ftest(Xgrid)

n <- 20
N <- 500
X <- Xgrid[sample(1:nrow(Xgrid), n),]
X <- X[sample(1:n, N, replace = TRUE),]
Z <- ftest(X)
model <- mleHetGP(X, Z, lower = rep(0.001,2), upper = rep(1,2))

critgrid <- apply(Xgrid, 1, crit_cSUR, model = model)

filled.contour(matrix(critgrid, ngrid), color.palette = terrain.colors, main = "cSUR criterion")
```

`crit_EI` *Expected Improvement criterion*

Description

Computes EI for minimization

Usage

```
crit_EI(x, model, cst = NULL, preds = NULL)
```

Arguments

`x`	matrix of new designs, one point per row (size n x d)
`model`	`homGP` or `hetGP` model, or their TP equivalents, including inverse matrices
`cst`	optional plugin value used in the EI, see details
`preds`	optional predictions at `x` to avoid recomputing if already done

Details

`cst` is classically the observed minimum in the deterministic case. In the noisy case, the min of the predictive mean works fine.

Note

This is a beta version at this point.

References

Mockus, J.; Tiesis, V. & Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum Towards Global Optimization, Amsterdam: Elsevier, 2, 2.

Vazquez E, Villemonteix J, Sidorkiewicz M, Walter E (2008). Global Optimization Based on Noisy Evaluations: An Empirical Study of Two Statistical Approaches, Journal of Physics: Conference Series, 135, IOP Publishing.

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

Examples

```
## Optimization example
set.seed(42)


## 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 + cos(x * 2 * pi)))
}

## Test function defined in [0,1]
ftest <- function(x){
if(is.null(nrow(x)))
x <- matrix(x, ncol = 1)
return(f1d(x) + rnorm(nrow(x), mean = 0, sd = noiseFun(x)))
}

n_init <- 10 # number of unique designs
N_init <- 100 # total number of points
X <- seq(0, 1, length.out = n_init)
X <- matrix(X[sample(1:n_init, N_init, replace = TRUE)], ncol = 1)
Z <- ftest(X)

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

model <- mleHetGP(X = X, Z = Z, lower = 0.001, upper = 1)

EIgrid <- crit_EI(Xgrid, model)
preds <- predict(x = Xgrid, model)

par(mar = c(3,3,2,3)+0.1)
plot(xgrid, f1d(xgrid), type = 'l', lwd = 1, col = "blue", lty = 3,
xlab = '', ylab = '', ylim = c(-8,16))
points(X, Z)
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(EIgrid)), axes = FALSE, ylab = "", xlab = "")
lines(xgrid, EIgrid, lwd = 2, col = 'cyan')
axis(side = 4)
mtext(side = 4, line = 2, expression(EI(x)), cex = 0.8)
mtext(side = 2, line = 2, expression(f(x)), cex = 0.8)
```

crit_ICU *Integrated Contour Uncertainty criterion*

Description

Computes ICU infill criterion

Usage

```
crit_ICU(x, model, thres = 0, Xref, w = NULL, preds = NULL, kxprime = NULL)
```

Arguments

x	matrix of new designs, one point per row (size n x d)
model	homGP or hetGP model, including inverse matrices
thres	for contour finding
Xref	matrix of input locations to approximate the integral by a sum
w	optional weights vector of weights for Xref locations
preds	optional predictions at Xref to avoid recomputing if already done
kxprime	optional covariance matrix between model$X0 and Xref to avoid its recomputation

References

Lyu, X., Binois, M. & Ludkovski, M. (2018+). Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation. arXiv:1807.06712.

Examples

```
## Infill criterion example
set.seed(42)
branin <- function(x){
  m <- 54.8104; s <- 51.9496
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  xx <- 15 * x[,1] - 5; y <- 15 * x[,2]
  f <- (y - 5.1 * xx^2/(4 * pi^2) + 5 * xx/pi - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(xx) + 10
  f <- (f - m)/s
  return(f)
}

ftest <- function(x, sd = 0.1){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  return(apply(x, 1, branin) + rnorm(nrow(x), sd = sd))
}

ngrid <- 51; xgrid <- seq(0, 1, length.out = ngrid)
```

```
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
Zgrid <- ftest(Xgrid)

n <- 20
N <- 500
X <- Xgrid[sample(1:nrow(Xgrid), n),]
X <- X[sample(1:n, N, replace = TRUE),]
Z <- ftest(X)
model <- mleHetGP(X, Z, lower = rep(0.001,2), upper = rep(1,2))

# Precalculations for speedup
preds <- predict(model, x = Xgrid)
kxprime <- cov_gen(X1 = model$X0, X2 = Xgrid, theta = model$theta, type = model$covtype)

critgrid <- apply(Xgrid, 1, crit_ICU, model = model, Xref = Xgrid,
                  preds = preds, kxprime = kxprime)

filled.contour(matrix(critgrid, ngrid), color.palette = terrain.colors, main = "ICU criterion")
```

`crit_IMSPE` *Sequential IMSPE criterion*

Description

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

Usage

```
crit_IMSPE(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 `nu_hat` from `homGP` or `hetGP`. Currently this function ignores the extra terms related to the estimation of the mean.

See Also

`deriv_crit_IMSPE` 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()

IMSPE_grid <- apply(xgrid, 1, crit_IMSPE, Wijs = Wijs, model = model)

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

plot(xgrid, IMSPE_grid * model$nu_hat, xlab = "x", ylab = "crit_IMSPE values")
abline(v = designs)

################################################################################
## Bi-variate case

nvar <- 2

set.seed(2)
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()

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

`crit_MCU` *Maximum Contour Uncertainty criterion*

Description

Computes MCU infill criterion

Usage

```
crit_MCU(x, model, thres = 0, gamma = 2, preds = NULL)
```

Arguments

`x`	matrix of new designs, one point per row (size n x d)
`model`	`homGP` or `hetGP` model, including inverse matrices
`thres`	for contour finding
`gamma`	optional weight in -|f(x) - thres| + gamma * s(x). Default to 2.
`preds`	optional predictions at `x` to avoid recomputing if already done

References

Srinivas, N., Krause, A., Kakade, S, & Seeger, M. (2012). Information-theoretic regret bounds for Gaussian process optimization in the bandit setting, IEEE Transactions on Information Theory, 58, pp. 3250-3265.

Bogunovic, J., Scarlett, J., Krause, A. & Cevher, V. (2016). Truncated variance reduction: A unified approach to Bayesian optimization and level-set estimation, in Advances in neural information processing systems, pp. 1507-1515.

Lyu, X., Binois, M. & Ludkovski, M. (2018+). Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation. arXiv:1807.06712.

Examples

```
## Infill criterion example
set.seed(42)
branin <- function(x){
  m <- 54.8104; s <- 51.9496
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  xx <- 15 * x[,1] - 5; y <- 15 * x[,2]
  f <- (y - 5.1 * xx^2/(4 * pi^2) + 5 * xx/pi - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(xx) + 10
  f <- (f - m)/s
  return(f)
}

ftest <- function(x, sd = 0.1){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  return(apply(x, 1, branin) + rnorm(nrow(x), sd = sd))
}

ngrid <- 101; xgrid <- seq(0, 1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
Zgrid <- ftest(Xgrid)

n <- 20
N <- 500
X <- Xgrid[sample(1:nrow(Xgrid), n),]
X <- X[sample(1:n, N, replace = TRUE),]
Z <- ftest(X)
model <- mleHetGP(X, Z, lower = rep(0.001,2), upper = rep(1,2))

critgrid <- apply(Xgrid, 1, crit_MCU, model = model)

filled.contour(matrix(critgrid, ngrid), color.palette = terrain.colors, main = "MEE criterion")
```

---

crit_MEE *Maximum Empirical Error criterion*

Description

Computes MEE infill criterion

Usage

```
crit_MEE(x, model, thres = 0, preds = NULL)
```

Arguments

x	matrix of new designs, one point per row (size n x d)
model	homGP or hetGP model, including inverse matrices
thres	for contour finding
preds	optional predictions at x to avoid recomputing if already done

References

Ranjan, P., Bingham, D. & Michailidis, G (2008). Sequential experiment design for contour estimation from complex computer codes, Technometrics, 50, pp. 527-541.

Bichon, B., Eldred, M., Swiler, L., Mahadevan, S. & McFarland, J. (2008). Efficient global reliability analysis for nonlinear implicit performance functions, AIAA Journal, 46, pp. 2459-2468.

Lyu, X., Binois, M. & Ludkovski, M. (2018+). Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation. arXiv:1807.06712.

Examples

```
## Infill criterion example
set.seed(42)
branin <- function(x){
  m <- 54.8104; s <- 51.9496
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  xx <- 15 * x[,1] - 5; y <- 15 * x[,2]
  f <- (y - 5.1 * xx^2/(4 * pi^2) + 5 * xx/pi - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(xx) + 10
  f <- (f - m)/s
  return(f)
}

ftest <- function(x, sd = 0.1){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  return(apply(x, 1, branin) + rnorm(nrow(x), sd = sd))
}

ngrid <- 101; xgrid <- seq(0, 1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
Zgrid <- ftest(Xgrid)

n <- 20
N <- 500
X <- Xgrid[sample(1:nrow(Xgrid), n),]
X <- X[sample(1:n, N, replace = TRUE),]
Z <- ftest(X)
model <- mleHetGP(X, Z, lower = rep(0.001,2), upper = rep(1,2))

critgrid <- apply(Xgrid, 1, crit_MEE, model = model)

filled.contour(matrix(critgrid, ngrid), color.palette = terrain.colors, main = "MEE criterion")
```

crit_optim *Criterion optimization*

Description

Search for the best value of available criterion, possibly using a h-steps lookahead strategy to favor designs with replication

Usage

```
crit_optim(
  model,
  crit,
  ...,
  h = 2,
  Xcand = NULL,
  control = list(multi.start = 10, maxit = 100),
  seed = NULL,
  ncores = 1
)
```

Arguments

model	homGP or hetGP model
crit	considered criterion, one of "crit_cSUR", "crit_EI", "crit_ICU", "crit_MCU" and "crit_tMSE". Note that crit_IMSPE has its dedicated method, see IMSPE_optim.
...	additional parameters of the criterion
h	horizon for multi-step ahead framework. The decision is made between: • sequential crit search starting by a new design (optimized first) then adding h replicates • sequential crit searches starting by 1 to h replicates before adding a new point Use h = 0 for the myopic criterion, i.e., not looking ahead.
Xcand	optional discrete set of candidates (otherwise a maximin LHS is used to initialize continuous 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 crit for adding a new design.
seed	optional seed for the generation of LHS designs with maximinSA_LHS
ncores	number of CPU available (> 1 mean parallel TRUE), see mclapply

Details

When looking ahead, the kriging believer heuristic is used, meaning that the non-observed value is replaced by the mean prediction in the update.

Value

list with elements:

- par: best first design,
- value: criterion 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. B. Gramacy, M. Ludkovski (2019), Replication or exploration? Sequential design for stochastic simulation experiments, Technometrics, 61(1), 7-23.
Preprint available on arXiv:1710.03206.

Examples

```
##############################################################################
## Bi-variate example (myopic version)
##############################################################################

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)

crit <- "crit_EI"
crit_grid <- apply(Xgrid, 1, crit, model = model)
filled.contour(x = xgrid, y = xgrid, matrix(crit_grid, ngrid),
```

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

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

for(i in 1:nsteps){
 res <- crit_optim(model, crit = crit, h = 0, control = list(multi.start = 50, maxit = 30))
  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)

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

##############################################################################
## Bi-variate example (look-ahead version)
##############################################################################
## Not run:
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))

nsteps <- 5 # Increase for more steps
crit <- "crit_EI"

# To use parallel computation (turn off on Windows)
library(parallel)
parallel <- FALSE #TRUE #
if(parallel) ncores <- detectCores() else ncores <- 1
```

```
for(i in 1:nsteps){
  res <- crit_optim(model, h = 3, crit = crit, ncores = ncores,
                    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)

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

## End(Not run)
```

`crit_qEI` *Parallel Expected improvement*

Description

Fast approximated batch-Expected Improvement criterion (for minimization)

Usage

```
crit_qEI(x, model, cst = NULL, preds = NULL)
```

Arguments

`x`	matrix of new designs representing the batch of q points, one point per row (size q x d)
`model`	`homGP` or `hetGP` model, including inverse matrices.
`cst`	optional plugin value used in the EI, see details
`preds`	optional predictions at `x` to avoid recomputing if already done (must include the predictive covariance, i.e., the `cov` slot)

Details

`cst` is classically the observed minimum in the deterministic case. In the noisy case, the min of the predictive mean works fine.

Note

This is a beta version at this point. It may work for for TP models as well.

References

M. Binois (2015), Uncertainty quantification on Pareto fronts and high-dimensional strategies in Bayesian optimization, with applications in multi-objective automotive design. Ecole Nationale Superieure des Mines de Saint-Etienne, PhD thesis.

Examples

```
## Optimization example (noiseless)
set.seed(42)

## Test function defined in [0,1]
ftest <- f1d

n_init <- 5 # number of unique designs
X <- seq(0, 1, length.out = n_init)
X <- matrix(X, ncol = 1)
Z <- ftest(X)

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

model <- mleHomGP(X = X, Z = Z, lower = 0.01, upper = 1, known = list(g = 2e-8))

# Regular EI function
cst <- min(model$Z0)
EIgrid <- crit_EI(Xgrid, model, cst = cst)
plot(xgrid, EIgrid, type = "l")
abline(v = X, lty = 2) # observations

# Create batch (based on regular EI peaks)
xbatch <- matrix(c(0.37, 0.17, 0.7), 3, 1)
abline(v = xbatch, col = "red")
fqEI <- crit_qEI(xbatch, model, cst = cst)

# Compare with Monte Carlo qEI
preds <- predict(model, xbatch, xprime = xbatch)
nsim <- 1e4
simus <- matrix(rnorm(3 * nsim), nsim) %*% chol(preds$cov)
simus <- simus + matrix(preds$mean, nrow = nsim, ncol = 3, byrow = TRUE)
MCqEI <- mean(apply(cst - simus, 1, function(x) max(c(x, 0))))
```

crit_tMSE *t-MSE criterion*

Description

Computes targeted mean squared error infill criterion

Usage

```
crit_tMSE(x, model, thres = 0, preds = NULL, seps = 0.05)
```

Arguments

x	matrix of new designs, one point per row (size n x d)
model	homGP or hetGP model, including inverse matrices
thres	for contour finding
preds	optional predictions at x to avoid recomputing if already done (must contain cov)
seps	parameter for the target window

References

Picheny, V., Ginsbourger, D., Roustant, O., Haftka, R., Kim, N. (2010). Adaptive designs of experiments for accurate approximation of a target region, Journal of Mechanical Design (132), p. 071008.

Lyu, X., Binois, M. & Ludkovski, M. (2018+). Evaluating Gaussian Process Metamodels and Sequential Designs for Noisy Level Set Estimation. arXiv:1807.06712.

Examples

```
## Infill criterion example
set.seed(42)
branin <- function(x){
  m <- 54.8104; s <- 51.9496
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  xx <- 15 * x[,1] - 5; y <- 15 * x[,2]
  f <- (y - 5.1 * xx^2/(4 * pi^2) + 5 * xx/pi - 6)^2 + 10 * (1 - 1/(8 * pi)) * cos(xx) + 10
  f <- (f - m)/s
  return(f)
}

ftest <- function(x, sd = 0.1){
  if(is.null(dim(x))) x <- matrix(x, nrow = 1)
  return(apply(x, 1, branin) + rnorm(nrow(x), sd = sd))
```

```
}

ngrid <- 101; xgrid <- seq(0, 1, length.out = ngrid)
Xgrid <- as.matrix(expand.grid(xgrid, xgrid))
Zgrid <- ftest(Xgrid)

n <- 20
N <- 500
X <- Xgrid[sample(1:nrow(Xgrid), n),]
X <- X[sample(1:n, N, replace = TRUE),]
Z <- ftest(X)
model <- mleHetGP(X, Z, lower = rep(0.001,2), upper = rep(1,2))

critgrid <- apply(Xgrid, 1, crit_tMSE, model = model)

filled.contour(matrix(critgrid, ngrid), color.palette = terrain.colors, main = "tMSE criterion")
```

`deriv_crit_EI` *Derivative of EI criterion for GP models*

Description

Derivative of EI criterion for GP models

Usage

```
deriv_crit_EI(x, model, cst = NULL, preds = NULL)
```

Arguments

`x`	matrix for the new design (size 1 x d)
`model`	`homGP` or `hetGP` model
`cst`	threshold for contour criteria
`preds`	pre-computed preds for contour criteria

References

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

Roustant, O., Ginsbourger, D., DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization, Journal of Statistical Software, 2012

See Also

`crit_EI` for the criterion

deriv_crit_IMSPE *Derivative of crit_IMSPE*

Description

Derivative of crit_IMSPE

Usage

```
deriv_crit_IMSPE(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_IMSPE for the criterion

f1d *1d test function (1)*

Description

1d test function (1)

Usage

```
f1d(x)
```

Arguments

x	scalar or matrix (size n x 1) in [0,1]

References

A. Forrester, A. Sobester, A. Keane (2008), Engineering design via surrogate modelling: a practical guide, John Wiley & Sons

Examples

```
plot(f1d)
```

`f1d2` *1d test function (2)*

Description

1d test function (2)

Usage

```
f1d2(x)
```

Arguments

`x`	scalar or matrix (size n x 1) in [0,1]

References

A. Boukouvalas, and D. Cornford (2009), Learning heteroscedastic Gaussian processes for complex datasets, Technical report.

M. Yuan, and G. Wahba (2004), Doubly penalized likelihood estimator in heteroscedastic regression, Statistics and Probability Letters 69, 11-20.

Examples

```
plot(f1d2)
```

`f1d2_n` *Noisy 1d test function (2) Add Gaussian noise with variance r(x) = scale * (exp(sin(2 pi x)))^2 to* `f1d2`

Description

Noisy 1d test function (2) Add Gaussian noise with variance r(x) = scale * (exp(sin(2 pi x)))^2 to `f1d2`

Usage

```
f1d2_n(x, scale = 1)
```

Arguments

`x`	scalar or matrix (size n x 1) in [0,1]
`scale`	scalar in [0, Inf] to control the signal to noise ratio

Examples

```
X <- matrix(seq(0, 1, length.out = 101), ncol = 1)
Xr <- X[sort(sample(x = 1:101, size = 500, replace = TRUE)),, drop = FALSE]
plot(Xr, f1d2_n(Xr))
lines(X, f1d2(X), col = "red", lwd = 2)
```

`f1d_n` *Noisy 1d test function (1) Add Gaussian noise with variance r(x) = scale * (1.1 + sin(2 pi x))^2 to* `f1d`

Description

Noisy 1d test function (1) Add Gaussian noise with variance r(x) = scale * (1.1 + sin(2 pi x))^2 to `f1d`

Usage

```
f1d_n(x, scale = 1)
```

Arguments

`x`	scalar or matrix (size n x 1) in [0,1]
`scale`	scalar in [0, Inf] to control the signal to noise ratio

Examples

```
X <- matrix(seq(0, 1, length.out = 101), ncol = 1)
Xr <- X[sort(sample(x = 1:101, size = 500, replace = TRUE)),, drop = FALSE]
plot(Xr, f1d_n(Xr))
lines(X, f1d(X), col = "red", lwd = 2)
```

`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)
```

horizon *Adapt horizon*

Description

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

Usage

```
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. B. Gramacy, M. Ludkovski (2019), Replication or exploration? Sequential design for stochastic simulation experiments, Technometrics, 61(1), 7-23.
Preprint available on arXiv:1710.03206.

`IMSPE` *Integrated Mean Square Prediction Error*

Description

IMSPE of a given design

Usage

```
IMSPE(
  X,
  theta = NULL,
  Lambda = NULL,
  mult = NULL,
  covtype = NULL,
  nu = 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"
`nu`	variance parameter
`eps`	numerical nugget

Details

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

IMSPE_optim *IMSPE optimization*

Description

Search for the best value of the IMSPE criterion, possibly using a h-steps lookahead strategy to favor designs with replication

Usage

```
IMSPE_optim(
  model,
  h = 2,
  Xcand = NULL,
 control = list(tol_dist = 1e-06, tol_diff = 1e-06, multi.start = 20, maxit = 100),
  Wijs = NULL,
  seed = NULL,
  ncores = 1
)
```

Arguments

model	homGP or hetGP model
h	horizon for multi-step ahead framework. The decision is made between: • sequential crit search starting by a new design (optimized first) then adding h replicates • sequential crit searches starting by 1 to h replicates before adding a new point Use h = 0 for the myopic criterion, i.e., not looking ahead.
Xcand	optional discrete set of candidates (otherwise a maximin LHS is used to initialise continuous 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 IMSPE 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
ncores	number of CPU available (> 1 mean parallel TRUE), see mclapply

Details

The domain needs to be [0, 1]^d for now.

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. B. Gramacy, M. Ludkovski (2019), Replication or exploration? Sequential design for stochastic simulation experiments, Technometrics, 61(1), 7-23.
Preprint available on arXiv:1710.03206.

Examples

```
################################################################################
## Bi-variate example (myopic version)
################################################################################

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)

IMSPE_grid <- apply(Xgrid, 1, crit_IMSPE, model = model)
filled.contour(x = xgrid, y = xgrid, matrix(IMSPE_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 <- IMSPE_optim(model, control = list(multi.start = 30, maxit = 30))
  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)

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

##############################################################################
## Bi-variate example (look-ahead version)
##############################################################################
## Not run:
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))

nsteps <- 5 # Increase for more steps

# To use parallel computation (turn off on Windows)
library(parallel)
parallel <- FALSE #TRUE #
if(parallel) ncores <- detectCores() else ncores <- 1

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

  # 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)

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

## End(Not run)
```

`LOO_preds` *Leave one out predictions*

Description

Provide leave one out predictions, e.g., for model testing and diagnostics. This is used in the method plot available on GP and TP models.

Usage

```
LOO_preds(model, ids = NULL)
```

Arguments

`model`	`homGP` or `hetGP` model, TP version is not considered at this point
`ids`	vector of indices of the unique design point considered (default to all)

Value

list with mean and variance predictions at x_i assuming this point has not been evaluated

Note

For TP models, `psi` is considered fixed.

References

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

F. Bachoc (2013), Cross Validation and Maximum Likelihood estimations of hyper-parameters of Gaussian processes with model misspecification, Computational Statistics & Data Analysis, 55–69.

Examples

```
set.seed(32)
## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
nvar <- 1

## Model fitting
model <- mleHomGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(10, nvar),
                  covtype = "Matern5_2", known = list(beta0 = 0))
LOO_p <- LOO_preds(model)

# model minus observation(s) at x_i
d_mot <- find_reps(X, Z)

LOO_ref <- matrix(NA, nrow(d_mot$X0), 2)
for(i in 1:nrow(d_mot$X0)){
 model_i <- mleHomGP(X = list(X0 = d_mot$X0[-i,, drop = FALSE], Z0 = d_mot$Z0[-i],
                     mult = d_mot$mult[-i]), Z = unlist(d_mot$Zlist[-i]),
                   lower = rep(0.1, nvar), upper = rep(50, nvar), covtype = "Matern5_2",
               known = list(theta = model$theta, k_theta_g = model$k_theta_g, g = model$g,
                                  beta0 = 0))
 model_i$nu_hat <- model$nu_hat
 p_i <- predict(model_i, d_mot$X0[i,,drop = FALSE])
 LOO_ref[i,] <- c(p_i$mean, p_i$sd2)
}

# Compare results

range(LOO_ref[,1] - LOO_p$mean)
range(LOO_ref[,2] - LOO_p$sd2)

# Use of LOO for diagnostics
plot(model)
```

---

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 = NULL,
  upper = NULL,
 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, return.hom = FALSE, factr
    = 1e+09),
  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`	optional 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).
- 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 to include the inverse covariance matrix in the object for further use (e.g., prediction).
- return.hom boolean to include homoskedastic GP models used for initialization (i.e., modHom and modNugs).
- factr (default to 1e9) and pgtol are available to be passed to control for L-BFGS-B in optim.

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 nu_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 nu_hat * (C + Lambda * diag(1/mult)) = nu_hat * K, 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. nu_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$theta, initial lengthscales are taken at 10% of the range determined with lower and upper, while init$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$nu_hat). Note that when specifying settings$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,
- nu_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,
- nu_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 (not scaled by nu_hat and nu_hat_var),
- X0, Z0, Z, eps, logN, covtype: values given in input,
- time: time to train the model, in seconds.

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on arXiv:1611.05902.

See Also

predict.hetGP for predictions, update.hetGP for updating an existing model. summary and plot functions are 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 = NULL,
  upper = NULL,
 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$double.eps),
    10000)),
 settings = list(linkThetas = "joint", logN = TRUE, initStrategy = "residuals", checkHom
  = TRUE, penalty = TRUE, trace = 0, return.matrices = TRUE, return.hom = FALSE, factr
    = 1e+09),
  covtype = c("Gaussian", "Matern5_2", "Matern3_2"),
  maxit = 100,
  known = list(beta0 = 0),
  init = list(nu = 3),
  eps = sqrt(.Machine$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).
- `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).
- Arguments `factr` (default to 1e9) and `pgtol` are available to be passed to `control` for L-BFGS-B in `optim`.

`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 nu_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$theta, initial lengthscales are taken at 10% of the range determined with lower and upper, while init$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$nu_hat). Note that when specifying settings$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,
- sigma2: 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,
- nu_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,
- time: time to train the model, in seconds.

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on 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. `summary` and `plot` functions are 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 <- 100
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 = NULL,
  upper = NULL,
  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, factr = 1e+07)
)
```

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	optional 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 10% of the range determined with lower and upper) • g_init initial value of the nugget parameter to be optimized over (based on the variance at replicates if there are any, else 0.1)
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). Arguments factr (default to 1e9) and pgtol are available to be passed to control for L-BFGS-B in optim (for the joint likelihood only).

Details

The global covariance matrix of the model is parameterized as nu_hat * (C + g * diag(1/mult)) = nu_hat * K, 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. nu_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,

- nu_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 (not scaled by nu_hat) (if return.Ki is TRUE in settings)
- time: time to train the model, in seconds.

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on arXiv:1611.05902.

See Also

predict.homGP for predictions, update.homGP for updating an existing model. summary and plot functions are 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 = NULL,
  upper = NULL,
  known = list(beta0 = 0),
 noiseControl = list(g_bounds = c(sqrt(.Machine$double.eps), 10000), nu_bounds = c(2 +
    0.001, 30), sigma2_bounds = c(sqrt(.Machine$double.eps), 10000)),
  init = list(nu = 3),
  covtype = c("Gaussian", "Matern5_2", "Matern3_2"),
  maxit = 100,
  eps = sqrt(.Machine$double.eps),
  settings = list(return.Ki = TRUE, factr = 1e+09)
)
```

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 10% of the range determined with lower and upper)
- g_init initial value of the nugget parameter to be optimized over (based on the variance at replicates if there are any, else 10% of the variance)
- 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). Arguments factr (default to 1e9) and pgtol are available to be passed to control for L-BFGS-B in optim.

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)
- time: time to train the model, in seconds.

References

M. Binois, Robert B. Gramacy, M. Ludkovski (2018), Practical heteroskedastic Gaussian process modeling for large simulation experiments, Journal of Computational and Graphical Statistics, 27(4), 808–821.
Preprint available on arXiv:1611.05902.

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

M. Chung, M. Binois, RB Gramacy, DJ Moquin, AP Smith, AM Smith (2019). Parameter and Uncertainty Estimation for Dynamical Systems Using Surrogate Stochastic Processes. SIAM Journal on Scientific Computing, 41(4), 2212-2238.
Preprint available on arXiv:1802.00852.

See Also

`predict.homTP` for predictions. `summary` and `plot` functions are 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 prediction
- `sd2_var`: (returned if `noise.var = TRUE`) kriging variance of the noise process (i.e., on log-variances if `logN = TRUE`)
- `cov`: (returned if `xprime` is given) 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 (i.e., on log-variances if `logN = TRUE`)
- `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

pred_noisy_input *Gaussian process prediction prediction at a noisy input* x*, with centered Gaussian noise of variance* sigma_x*. Several options are available, with different efficiency/accuracy tradeoffs.*

Description

Gaussian process prediction prediction at a noisy input x, with centered Gaussian noise of variance sigma_x. Several options are available, with different efficiency/accuracy tradeoffs.

Usage

```
pred_noisy_input(x, model, sigma_x, type = c("simple", "taylor", "exact"))
```

Arguments

x	design considered
model	GP
sigma_x	input variance
type	available options include • simple relying on a corrective term, see (McHutchon2011); • taylor based on a Taylor expansion, see, e.g., (Girard2003); • exact for exact moments (only for the Gaussian covariance).

Note

Beta version.

References

A. McHutchon and C.E. Rasmussen (2011), Gaussian process training with input noise, Advances in Neural Information Processing Systems, 1341-1349.

A. Girard, C.E. Rasmussen, J.Q. Candela and R. Murray-Smith (2003), Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting, Advances in Neural Information Processing Systems, 545-552.

Examples

```
##############################################################################
### Illustration of prediction with input noise
##############################################################################

## noise std deviation function defined in [0,1]
noiseFun <- function(x, coef = 1.1, scale = 0.25){
  if(is.null(nrow(x))) x <- matrix(x, nrow = 1)
  return(scale*(coef + sin(x * 2 * pi)))
}

## data generating function combining mean and noise fields
ftest <- function(x, scale = 0.25){
if(is.null(nrow(x))) x <- matrix(x, ncol = 1)
  return(f1d(x) + rnorm(nrow(x), mean = 0, sd = noiseFun(x, scale = scale)))
}

ntest <- 101; xgrid <- seq(0,1, length.out = ntest); Xgrid <- matrix(xgrid, ncol = 1)
set.seed(42)
Xpred <- Xgrid[rep(1:ntest, each = 100),,drop = FALSE]
Zpred <- matrix(ftest(Xpred), byrow = TRUE, nrow = ntest)
n <- 10
```

```
N <- 20
X <- matrix(seq(0, 1, length.out = n))
if(N > n) X <- rbind(X, X[sample(1:n, N-n, replace = TRUE),,drop = FALSE])
X <- X[order(X[,1]),,drop = FALSE]

Z <- apply(X, 1, ftest)
par(mfrow = c(1, 2))
plot(X, Z, ylim = c(-10,15), xlim = c(-0.1,1.1))
lines(xgrid, f1d(xgrid))
lines(xgrid, drop(f1d(xgrid)) + 2*noiseFun(xgrid), lty = 3)
lines(xgrid, drop(f1d(xgrid)) - 2*noiseFun(xgrid), lty = 3)
model <- mleHomGP(X, Z, known = list(beta0 = 0))
preds <- predict(model, Xgrid)
lines(xgrid, preds$mean, col = "red", lwd = 2)
lines(xgrid, preds$mean - 2*sqrt(preds$sd2), col = "blue")
lines(xgrid, preds$mean + 2*sqrt(preds$sd2), col = "blue")
lines(xgrid, preds$mean - 2*sqrt(preds$sd2 + preds$nugs), col = "blue", lty = 2)
lines(xgrid, preds$mean + 2*sqrt(preds$sd2 + preds$nugs), col = "blue", lty = 2)

sigmax <- 0.1
X1 <- matrix(0.5)

lines(xgrid, dnorm(xgrid, X1, sigmax) - 10, col = "darkgreen")

# MC experiment
nmc <- 1000
XX <- matrix(rnorm(nmc, X1, sigmax))
pxx <- predict(model, XX)
YXX <- rnorm(nmc, mean = pxx$mean, sd = sqrt(pxx$sd2 + pxx$nugs))
points(XX, YXX, pch = '.')

hh <- hist(YXX, breaks = 51, plot = FALSE)
dd <- density(YXX)
plot(hh$density, hh$mids, ylim = c(-10, 15))
lines(dd$y, dd$x)

# GP predictions
pin1 <- pred_noisy_input(X1, model, sigmax^2, type = "exact")
pin2 <- pred_noisy_input(X1, model, sigmax^2, type = "taylor")
pin3 <- pred_noisy_input(X1, model, sigmax^2, type = "simple")
ygrid <- seq(-10, 15,, ntest)
lines(dnorm(ygrid, pin1$mean, sqrt(pin1$sd2)), ygrid, lty = 2, col = "orange")
lines(dnorm(ygrid, pin2$mean, sqrt(pin2$sd2)), ygrid, lty = 2, col = "violet")
lines(dnorm(ygrid, pin3$mean, sqrt(pin3$sd2)), ygrid, lty = 2, col = "grey")
abline(h = mean(YXX), col = "red") # empirical mean

par(mfrow = c(1, 1))
```

---

rebuild *Import and export of hetGP objects*

Description

Functions to make hetGP objects lighter before exporting them, and to reverse this after import. The rebuild function may also be used to obtain more robust inverse of covariance matrices using ginv.

Usage

```
rebuild(object, robust)

## S3 method for class 'homGP'
rebuild(object, robust = FALSE)

strip(object)

## S3 method for class 'hetGP'
rebuild(object, robust = FALSE)

## S3 method for class 'homTP'
rebuild(object, robust = FALSE)

## S3 method for class 'hetTP'
rebuild(object, robust = FALSE)
```

Arguments

object	homGP or homTP model without slot Ki (inverse covariance matrix), or hetGP or hetTP model without slot Ki or Kgi
robust	if TRUE ginv is used for matrix inversion, otherwise it is done via Cholesky.

Value

object with additional or removed slots.

Examples

```
set.seed(32)
## motorcycle data
library(MASS)
X <- matrix(mcycle$times, ncol = 1)
Z <- mcycle$accel
## Model fitting
model <- mleHetGP(X = X, Z = Z, lower = 0.1, upper = 50)

# Check size
object.size(model)

# Remove internal elements, e.g., to save it
model <- strip(model)

# Check new size
```

```
object.size(model)

# Now rebuild model, and use ginv instead
model <- rebuild(model, robust = TRUE)
object.size(model)
```

`scores` *Score and RMSE function To asses the performance of the prediction, this function computes the root mean squared error and proper score function (also known as negative log-probability density).*

Description

Score and RMSE function To asses the performance of the prediction, this function computes the root mean squared error and proper score function (also known as negative log-probability density).

Usage

```
scores(model, Xtest, Ztest, return.rmse = FALSE)
```

Arguments

`model`	`homGP` or `hetGP` model, including inverse matrices
`Xtest`	matrix of new design locations
`Ztest`	corresponding vector of observations, or alternatively, a matrix of size [nrow(Xtest) x number of replicates], a list of size nrow(Xtest) with a least one value per element
`return.rmse`	if `TRUE`, return the root mean squared error

References

T. Gneiting, and A. Raftery (2007). Strictly Proper Scoring Rules, Prediction, and Estimation, Journal of the American Statistical Association, 102(477), 359-378.

`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 initialization. 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
testpts <- 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)

## Visualizing initial predictive surface
preds <- predict(x = testpts, model_init)
plot(X, Z)
lines(testpts, 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 <- testpts[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
p_fin <- predict(x=testpts, model)

## Visualizing the result by augmenting earlier plot
lines(testpts, p_fin$mean, col = "blue")
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2 + p_fin$nugs)),
  col = "blue", lty = 3)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2 + p_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,
                          lower = rep(0.1, nvar), upper = rep(50, nvar),
                 init = list(theta = model_init$theta, k_theta_g = model_init$k_theta_g))
p_dir <- predict(x = testpts, model_direct)
```

```
print(model_direct$nit_opt)
lines(testpts, p_dir$mean, col = "green")
lines(testpts, qnorm(0.05, p_dir$mean, sqrt(p_dir$sd2)), col = "green",
  lty = 2)
lines(testpts, qnorm(0.95, p_dir$mean, sqrt(p_dir$sd2)), col = "green",
  lty = 2)
lines(testpts, qnorm(0.05, p_dir$mean, sqrt(p_dir$sd2 + p_dir$nugs)),
  col = "green", lty = 3)
lines(testpts, qnorm(0.95, p_dir$mean, sqrt(p_dir$sd2 + p_dir$nugs)),
  col = "green", lty = 3)
lines(testpts, sin(testpts), col = "red", lty = 2)

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

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

Description

Fast update of existing `hetTP` model with new observations.

Usage

```
## S3 method for class 'hetTP'
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 `"hetTP"`-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 mleHetTP. 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, mleHetTP 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) + noisefun(X) * rt(length(X), df = 10)

## Initial fit
testpts <- matrix(seq(0, 2*pi, length = 10*n), ncol = 1)
```

```
model <- model_init <- mleHetTP(X = X, Z = Z, lower = rep(0.1, nvar),
  upper = rep(50, nvar), maxit = 1000)

## Visualizing initial predictive surface
preds <- predict(x = testpts, model_init)
plot(X, Z)
lines(testpts, 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 <- testpts[newIds, drop = FALSE]
  newZ <- sin(newX) + noisefun(newX) * rt(length(newX), df = 10)
  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
p_fin <- predict(x=testpts, model)

## Visualizing the result by augmenting earlier plot
lines(testpts, p_fin$mean, col = "blue")
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2 + p_fin$nugs)),
  col = "blue", lty = 3)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2 + p_fin$nugs)),
  col = "blue", lty = 3)

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

testpts <- 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 = testpts, object = model_init)
plot(X, Z)
lines(testpts, preds$mean, col = "red")


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

  newX <- testpts[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$nit_opt)
}
p_fin <- predict(x = testpts, object = model)
lines(testpts, p_fin$mean, col = "blue")
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2)), col = "blue", lty = 2)
lines(testpts, qnorm(0.05, p_fin$mean, sqrt(p_fin$sd2 + p_fin$nugs)),
      col = "blue", lty = 3)
lines(testpts, qnorm(0.95, p_fin$mean, sqrt(p_fin$sd2 + p_fin$nugs)),
      col = "blue", lty = 3)

model_direct <-  mleHomGP(X = X, Z = Z, lower = rep(0.1, nvar), upper = rep(50, nvar))
p_dir <- predict(x = testpts, object = model_direct)
print(model_direct$nit_opt)
```

```
lines(testpts, p_dir$mean, col = "green")
lines(testpts, qnorm(0.05, p_dir$mean, sqrt(p_dir$sd2)), col = "green", lty = 2)
lines(testpts, qnorm(0.95, p_dir$mean, sqrt(p_dir$sd2)), col = "green", lty = 2)
lines(testpts, qnorm(0.05, p_dir$mean, sqrt(p_dir$sd2 + p_dir$nugs)),
      col = "green", lty = 3)
lines(testpts, qnorm(0.95, p_dir$mean, sqrt(p_dir$sd2 + p_dir$nugs)),
      col = "green", lty = 3)

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

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



## End(Not run)
```

---

update.homTP *Fast* homTP-*update*

Description

Update existing homTP model with new observations

Usage

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

Arguments

object	initial model of class homTP
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 `mleHomTP`. 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 TP moding
##------------------------------------------------------------
set.seed(42)

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

df_noise <- 3
nvar <- 1
n <- 10
X <- matrix(seq(0, 2 * pi, length=n), ncol = 1)
mult <- sample(1:50, n, replace = TRUE)
X <- rep(X, mult)
Z <- sin(X) + noisefun(X) * rt(length(X), df = df_noise)

testpts <- matrix(seq(0, 2*pi, length = 10*n), ncol = 1)
mod <- mod_init <- mleHomTP(X = X, Z = Z, covtype = "Matern5_2",
                            lower = rep(0.1, nvar), upper = rep(50, nvar))
preds <- predict(x = testpts, object = mod_init)
plot(X, Z)
lines(testpts, preds$mean, col = "red")


nsteps <- 10
for(i in 1:nsteps){
  newIds <- sort(sample(1:(10*n), 5))

 newX <- testpts[rep(newIds, times = sample(1:50, length(newIds), replace = TRUE)), drop = FALSE]
  newZ <- sin(newX) + noisefun(newX) * rt(length(newX), df = df_noise)
  points(newX, newZ, col = "blue", pch = 20)
  mod <- update(object = mod, newX, newZ)
  X <- c(X, newX)
  Z <- c(Z, newZ)
  plot(X, Z)
```

```
  print(mod$nit_opt)
}
p_fin <- predict(x = testpts, object = mod)
lines(testpts, p_fin$mean, col = "blue")
lines(testpts, p_fin$mean + sqrt(p_fin$sd2) * qt(0.05, df = mod$nu + length(Z)),
      col = "blue", lty = 2)
lines(testpts, p_fin$mean + sqrt(p_fin$sd2) * qt(0.95, df = mod$nu + length(Z)),
      col = "blue", lty = 2)
lines(testpts, p_fin$mean + sqrt(p_fin$sd2 + p_fin$nugs) * qt(0.05, df = mod$nu + length(Z)),
      col = "blue", lty = 3)
lines(testpts, p_fin$mean + sqrt(p_fin$sd2 + p_fin$nugs) * qt(0.95, df = mod$nu + length(Z)),
      col = "blue", lty = 3)

mod_dir <-  mleHomTP(X = X, Z = Z, covtype = "Matern5_2",
                          lower = rep(0.1, nvar), upper = rep(50, nvar))
p_dir <- predict(x = testpts, object = mod_dir)
print(mod_dir$nit_opt)
lines(testpts, p_dir$mean, col = "green")
lines(testpts, p_dir$mean + sqrt(p_dir$sd2) * qt(0.05, df = mod_dir$nu + length(Z)),
      col = "green", lty = 2)
lines(testpts, p_dir$mean + sqrt(p_dir$sd2) * qt(0.95, df = mod_dir$nu + length(Z)),
      col = "green", lty = 2)
lines(testpts, p_dir$mean + sqrt(p_dir$sd2 + p_dir$nugs) * qt(0.05, df = mod_dir$nu + length(Z)),
      col = "green", lty = 3)
lines(testpts, p_dir$mean + sqrt(p_dir$sd2 + p_dir$nugs) * qt(0.95, df = mod_dir$nu + length(Z)),
      col = "green", lty = 3)

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

## Compare outputs
summary(mod_init)
summary(mod)
summary(mod_dir)



## End(Not run)
```

`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. B. Gramacy, M. Ludkovski (2019), Replication or exploration? Sequential design for stochastic simulation experiments, Technometrics, 61(1), 7-23.
Preprint available on arXiv:1710.03206.

Index