

Package ‘RobustGaSP’

September 17, 2016

Type Package

Title Robust Gaussian Stochastic Process Emulation

Version 0.5.2

Date/Publication 2016-09-17 00:46:55

Maintainer Mengyang Gu <mg211@stat.duke.edu>

Author Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Description Robust parameter estimation and prediction of Gaussian stochastic process emulators.
Important functions : rgasp(), predict.rgasp().

License GPL (>= 2)

Depends methods

Imports Rcpp (>= 0.12.3)

Suggests nloptr (>= 1.0.4)

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Repository CRAN

RoxygenNote 5.0.1

R topics documented:

| | |
|------------------------------|-----------|
| RobustGaSP-package | 2 |
| findInertInputs | 4 |
| predict | 6 |
| rgasp | 11 |
| rgasp-class | 15 |
| Sample | 16 |
| show | 18 |
| Index | 20 |

RobustGaSP-package *Robust Gaussian Stochastic Process Emulation*

Description

Robust parameter estimation and prediction of Gaussian stochastic process emulators. Important functions : `rgasp()`, `predict.rgasp()`.

Details

The DESCRIPTION file:

```
Package:           RobustGaSP
Type:              Package
Title:             Robust Gaussian Stochastic Process Emulation
Version:           0.5.2
Date/Publication:  2016-02-08 08:08:08
Authors@R:         c(person(given="Mengyang",family="Gu",role=c("aut","cre"),email="mg211@stat.duke.edu"), person(
Maintainer:        Mengyang Gu <mg211@stat.duke.edu>
Author:            Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]
Description:       Robust parameter estimation and prediction of Gaussian stochastic process emulators. Important functio
License:           GPL (>= 2)
Depends:           methods
Imports:           Rcpp (>= 0.12.3)
Suggests:          nloptr (>= 1.0.4)
LinkingTo:         Rcpp, RcppEigen
NeedsCompilation: yes
Repository:        CRAN
Packaged:          2016-06-24 12:06:44 UTC; jpm
RoxygenNote:      5.0.1
```

Index of help topics:

```
RobustGaSP-package  Robust Gaussian Stochastic Process Emulation
Sample              Sample for Robust GaSP model
findInertInputs     find inert inputs with the posterior mode
predict             Prediction for Robust GaSP model
rgasp               Setting up the robust GaSP model
rgasp-class         Robust GaSP class
show                Show Robust GaSP object
```

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Maintainer: Mengyang Gu <mg211@stat.duke.edu>

References

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

See Also

[RobustGaSP](#)

Examples

```
#-----
# a 3 dimensional example
#-----
# dimensional of the inputs
dim_inputs <- 3
# number of the inputs
num_obs <- 30
# uniform samples of design
input <- matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)

# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) ##maximin lhd sample

####
# outputs from the 3 dim detpep10curv function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-detpep10curv(input[i,])
}

# use constant mean basis, with no constraint on optimization
m1<- rgasp(design = input, response = output, lower_bound=FALSE)

# the following use constraints on optimization
# m1<- rgasp(design = input, response = output, lower_bound=TRUE)

# the following use a single start on optimization
# m1<- rgasp(design = input, response = output, lower_bound=FALSE, multiple_starts=FALSE)

# number of points to be predicted
num_testing_input <- 5000
# generate points to be predicted
testing_input <- matrix(runif(num_testing_input*dim_inputs),num_testing_input,dim_inputs)
# Perform prediction
m1.predict<-predict(m1, testing_input)
# Predictive mean
m1.predict$mean

# The following tests how good the prediction is
testing_output <- matrix(0,num_testing_input,1)
```

```

for(i in 1:num_testing_input){
  testing_output[i]<-detpep10curv(testing_input[i,])
}

# compute the MSE, average coverage and average length
# out of sample MSE
MSE_emulator <- sum((m1.predict$mean-testing_output)^2)/(num_testing_input)

# proportion covered by 95% posterior predictive credible interval
prop_emulator <- length(which((m1.predict$lower95<=testing_output)
  &(m1.predict$upper95>=testing_output)))/num_testing_input

# average length of posterior predictive credible interval
length_emulator <- sum(m1.predict$upper95-m1.predict$lower95)/num_testing_input

# output of prediction
MSE_emulator
prop_emulator
length_emulator
# normalized RMSE
sqrt(MSE_emulator/mean((testing_output-mean(output))^2 ))

```

findInertInputs

find inert inputs with the posterior mode

Description

The function tests for inert inputs (inputs that barely affect the outputs) using the posterior mode.

Usage

```
findInertInputs(object, threshold=0.1)
```

Arguments

| | |
|-----------|--|
| object | an object of class rgasp. |
| threshold | a threshold between 0 to 1. If the normalized inverse parameter of an input is smaller this value, it is classified as inert inputs. |

Details

This function utilizes the following quantity

$$\text{object}@p * \text{object}@\text{beta_hat} * \text{object}@CL / \text{sum}(\text{object}@\text{beta_hat} * \text{object}@CL)$$

for each input to identify the inert outputs. The average estimated normalized inverse range parameters will be 1. If the estimated normalized inverse range parameters of an input is close to 0, it means this input might be an inert input.

In this method, a prior that has shrinkage effects is suggested to use, .e.g the jointly robust prior (i.e. one should set prior_choice='ref_approx' in rgasp()) to obtain the use codergasp object before using

this function). Moreover, one may not add a lower bound of the range parameters to perform this method (i.e. one should set `lower_bound=F` in `rgasp()`). For more details see Chapter 4 in

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

Value

A vector that has the same dimension of the number of inputs indicating how likely the inputs are inert. The average value is 1. When a value is very close to zero, it tends to be an inert inputs.

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Maintainer: Mengyang Gu <mg211@stat.duke.edu>

References

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

Examples

```
#-----
# test for inert inputs in the Borehole function
#-----
# dimensional of the inputs
dim_inputs <- 8
# number of the inputs
num_obs <- 40

# uniform samples of design
set.seed(0)
input <- matrix(runif(num_obs*dim_inputs), num_obs, dim_inputs)
# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) # maximin lhd sample

# rescale the design to the domain
input[,1]<-0.05+(0.15-0.05)*input[,1];
input[,2]<-100+(50000-100)*input[,2];
input[,3]<-63070+(115600-63070)*input[,3];
input[,4]<-990+(1110-990)*input[,4];
input[,5]<-63.1+(116-63.1)*input[,5];
input[,6]<-700+(820-700)*input[,6];
input[,7]<-1120+(1680-1120)*input[,7];
input[,8]<-9855+(12045-9855)*input[,8];

# outputs from the 8 dim Borehole function

output=matrix(0,num_obs,1)
for(i in 1:num_obs){
```

```

    output[i]=borehole(input[i,])
  }

# use constant mean basis with trend, with no constraint on optimization
m3<- rgasp(design = input, response = output, lower_bound=FALSE)

P=findInertInputs(m3)

```

| | |
|---------|---|
| predict | <i>Prediction for Robust GaSP model</i> |
|---------|---|

Description

Function to make prediction on Robust GaSP models after the Robust GaSP model has been constructed.

Usage

```

## S4 method for signature 'rgasp'
predict(object, testing_input, testing_trend= matrix(1,dim(testing_input)[1],1),...)

```

Arguments

| | |
|---------------|--|
| object | an object of class rgasp. |
| testing_input | a matrix containing the inputs where the rgasp is to perform prediction. |
| testing_trend | a matrix of mean/trend for prediction. |
| ... | Extra arguments to be passed to the function (not implemented yet). |

Value

The returned value is a list with

| | |
|---------|---|
| mean | predictive mean for the testing inputs. |
| lower95 | lower bound of the 95% posterior credible interval. |
| upper95 | upper bound of the 95% posterior credible interval. |
| sd | standard deviation of each testing_input. |

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]
 Maintainer: Mengyang Gu <mg211@stat.duke.edu>

References

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

Examples

```
#-----
# a 3 dimensional example
#-----
# dimensional of the inputs
dim_inputs <- 3
# number of the inputs
num_obs <- 30
# uniform samples of design
input <- matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)

# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) ##maximin lhd sample

# outputs from the 3 dim detpep10curv function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-detpep10curv (input[i,])
}

# use constant mean basis, with no constraint on optimization
m1<- rgasp(design = input, response = output, lower_bound=FALSE)

# the following use constraints on optimization
# m1<- rgasp(design = input, response = output, lower_bound=TRUE)

# the following use a single start on optimization
# m1<- rgasp(design = input, response = output, lower_bound=FALSE, multiple_starts=FALSE)

# number of points to be predicted
num_testing_input <- 5000
# generate points to be predicted
testing_input <- matrix(runif(num_testing_input*dim_inputs),num_testing_input,dim_inputs)
# Perform prediction
m1.predict<-predict(m1, testing_input)
# Predictive mean
m1.predict$mean

# The following tests how good the prediction is
testing_output <- matrix(0,num_testing_input,1)
for(i in 1:num_testing_input){
  testing_output[i]<-detpep10curv(testing_input[i,])
}

# compute the MSE, average coverage and average length
```

```

# out of sample MSE
MSE_emulator <- sum((m1.predict$mean-testing_output)^2)/(num_testing_input)

# proportion covered by 95% posterior predictive credible interval
prop_emulator <- length(which((m1.predict$lower95<=testing_output)
                             &(m1.predict$upper95>=testing_output)))/num_testing_input

# average length of posterior predictive credible interval
length_emulator <- sum(m1.predict$upper95-m1.predict$lower95)/num_testing_input

# output of prediction
MSE_emulator
prop_emulator
length_emulator
# normalized RMSE
sqrt(MSE_emulator/mean((testing_output-mean(output))^2 ))

#-----
# a 2 dimensional example with trend
#-----
# dimensional of the inputs
dim_inputs <- 2
# number of the inputs
num_obs <- 20

# uniform samples of design
input <-matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)
# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) ##maximin lhd sample

# outputs from the 2 dim Brainin function

output <- matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-limetal02non (input[i,])
}

#mean basis (trend)
X<-cbind(rep(1,num_obs), input )

# use constant mean basis with trend, with no constraint on optimization
m2<- rgasp(design = input, response = output,trend =X, lower_bound=FALSE,multiple_start=TRUE)

# number of points to be predicted
num_testing_input <- 5000
# generate points to be predicted
testing_input <- matrix(runif(num_testing_input*dim_inputs),num_testing_input,dim_inputs)

# trend of testing

```



```

testing_X<-cbind(rep(1,num_testing_input), testing_input )

# Perform prediction
m2.predict<-predict(m2, testing_input,testing_trend=testing_X)
# Predictive mean
#m2.predict$mean

# The following tests how good the prediction is
testing_output <- matrix(0,num_testing_input,1)
for(i in 1:num_testing_input){
  testing_output[i]<-limetal02non(testing_input[i,])
}

# compute the MSE, average coverage and average length
# out of sample MSE
MSE_emulator <- sum((m2.predict$mean-testing_output)^2)/(num_testing_input)

# proportion covered by 95% posterior predictive credible interval
prop_emulator <- length(which((m2.predict$lower95<=testing_output)
&(m2.predict$upper95>=testing_output)))/num_testing_input

# average length of posterior predictive credible interval
length_emulator <- sum(m2.predict$upper95-m2.predict$lower95)/num_testing_input

# output of prediction
MSE_emulator
prop_emulator
length_emulator
# normalized RMSE
sqrt(MSE_emulator/mean((testing_output-mean(output))^2 ))

#-----
# an 8 dimensional example using only a subset inputs and a noise with unknown variance
#-----
# dimensional of the inputs
dim_inputs <- 8
# number of the inputs
num_obs <- 30

# uniform samples of design
input <-matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)
# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) # maximin lhd sample

# rescale the design to the domain
input[,1]<-0.05+(0.15-0.05)*input[,1];
input[,2]<-100+(50000-100)*input[,2];
input[,3]<-63070+(115600-63070)*input[,3];
input[,4]<-990+(1110-990)*input[,4];
input[,5]<-63.1+(116-63.1)*input[,5];

```

```

input[,6]<-700+(820-700)*input[,6];
input[,7]<-1120+(1680-1120)*input[,7];
input[,8]<-9855+(12045-9855)*input[,8];

# outputs from the 8 dim Borehole function

output=matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]=borehole(input[i,])
}

# use constant mean basis with trend, with no constraint on optimization
m3<- rgaspl(design = input[,c(1,4,6,7,8)], response = output,
           nugget.est=TRUE, lower_bound=FALSE,multiple_start=TRUE)

# number of points to be predicted
num_testing_input <- 5000
# generate points to be predicted
testing_input <- matrix(runif(num_testing_input*dim_inputs),num_testing_input,dim_inputs)

# rescale the points to the region to be predict
testing_input[,1]<-0.05+(0.15-0.05)*testing_input[,1];
testing_input[,2]<-100+(50000-100)*testing_input[,2];
testing_input[,3]<-63070+(115600-63070)*testing_input[,3];
testing_input[,4]<-990+(1110-990)*testing_input[,4];
testing_input[,5]<-63.1+(116-63.1)*testing_input[,5];
testing_input[,6]<-700+(820-700)*testing_input[,6];
testing_input[,7]<-1120+(1680-1120)*testing_input[,7];
testing_input[,8]<-9855+(12045-9855)*testing_input[,8];

# Perform prediction
m3.predict<-predict(m3, testing_input[,c(1,4,6,7,8)])
# Predictive mean
#m3.predict$mean

# The following tests how good the prediction is
testing_output <- matrix(0,num_testing_input,1)
for(i in 1:num_testing_input){
  testing_output[i]<-borehole(testing_input[i,])
}

# compute the MSE, average coverage and average length
# out of sample MSE
MSE_emulator <- sum((m3.predict$mean-testing_output)^2)/(num_testing_input)

# proportion covered by 95% posterior predictive credible interval
prop_emulator <- length(which((m3.predict$lower95<=testing_output)

```

```

      &(m3.predict$upper95>=testing_output))/num_testing_input

# average length of posterior predictive credible interval
length_emulator <- sum(m3.predict$upper95-m3.predict$lower95)/num_testing_input

# output of sample prediction
MSE_emulator
prop_emulator
length_emulator
# normalized RMSE
sqrt(MSE_emulator/mean((testing_output-mean(output))^2 ))

```

 rgasp

Setting up the robust GaSP model

Description

Setting up the robust GaSP model for estimating the parameters (if the parameters are not given).

Usage

```

rgasp(design, response, trend = matrix(1, length(response), 1),
      nugget = 0, nugget.est = F, range.par = NA, prior_choice="ref_approx",
      a = 0.2, b = 1/(length(response))^{1/dim(as.matrix(design))[2]}*(a+
      dim(as.matrix(design))[2]),
      kernel_type = "matern_5_2",
      alpha = rep(1.9, dim(as.matrix(design))[2]),
      multiple_starts=T, lower_bound=T, max_eval=30, xtol_rel=1e-5)

```

Arguments

| | |
|------------|---|
| design | a matrix of inputs. |
| response | a matrix of outputs. |
| trend | the mean/trend matrix of inputs. The default value is a vector of ones. |
| nugget | numerical value of the nugget variance ratio. If nugget is equal to 0, it means there is either no nugget or the nugget is estimated. If the nugget is not equal to 0, it means a fixed nugget. The default value is 0. |
| nugget.est | boolean value. T means nugget should be estimated and F means nugget is fixed or not estimated. The default value is F F. |
| range.par | either NA or a vector. If it is NA, it means range parameters are estimated; otherwise range parameters are given. The default value is NA. |

| | |
|-----------------|---|
| prior_choice | the choice of prior for range parameters and noise-variance parameters. <code>ref_xi</code> and <code>ref_gamma</code> means the reference prior with reference prior with the log of inverse range parameterization ξ or range parameterization γ . <code>ref_approx</code> uses the jointly robust prior to approximate the reference prior. The default choice is <code>ref_approx</code> . |
| a | prior parameters in the jointly robust prior. The default value is 0.2. |
| b | prior parameters in the jointly robust prior. The default value is $n^{-1/p}(a+p)$ where n is the number of runs and p is the dimension of the input vector. |
| kernel_type | type of kernel. <code>matern_3_2</code> and <code>matern_5_2</code> are Matern correlation with roughness parameter 3/2 and 5/2 respectively. <code>pow_exp</code> is power exponential correlation with roughness parameter alpha. If <code>pow_exp</code> is to be used, one needs to specify its roughness parameter alpha. The default choice is <code>matern_5_2</code> . |
| alpha | roughness parameters in the <code>pow_exp</code> correlation functions. The default choice is a vector with each entry being 1.9. |
| multiple_starts | boolean value. T means it uses multiple default initial points to search the optimal range and nugget parameters and F means it use one default points to search the optimal parameters. The default value is T. |
| lower_bound | boolean value. T means the default lower bounds of the inverse range parameters are used to constrained the optimization and F means the optimization is unconstrained. The default value is T and we also suggest to use F in various scenarios. |
| max_eval | the maximum number of steps to estimate the range and nugget parameters. The default choice is 30. |
| xtol_rel | the torerance bound to stop the searching algorithm in estimation of the range and nugget parameters. The default choice is 10^{-5} . |

Value

`rgasp` returns a S4 object of class `rgasp` (see `rgasp-class`).

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Maintainer: Mengyang Gu <mg211@stat.duke.edu>

References

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

Examples

```
library(RobustGaSP)
#-----
# a 3 dimensional example
```

```

#-----
# dimensional of the inputs
dim_inputs <- 3
# number of the inputs
num_obs <- 30
# uniform samples of design
input <- matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)

# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) ##maximin lhd sample

####
# outputs from the 3 dim detpep10curv function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-detpep10curv (input[i,])
}

# use constant mean basis, with no constraint on optimization
m1<- rgasp(design = input, response = output, lower_bound=FALSE)

# the following use constraints on optimization
# m1<- rgasp(design = input, response = output, lower_bound=TRUE)

# the following use a single start on optimization
# m1<- rgasp(design = input, response = output, lower_bound=FALSE, multiple_starts=FALSE)

show(m1)      #####show this rgasp object

m1@beta_hat   #####estimated inverse range parameters

#-----
# a 2 dimensional example with trend
#-----
# dimensional of the inputs
dim_inputs <- 2
# number of the inputs
num_obs <- 20

# uniform samples of design
input <-matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)
# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) # maximin lhd sample

# outputs from the 2 dim Brainin function

output <- matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-limetal02non (input[i,])
}

```

```

#####trend or mean basis
X<-cbind(rep(1,num_obs), input )

# use constant mean basis with trend, with no constraint on optimization
m2<- rgasp(design = input, response = output,trend =X, lower_bound=FALSE,multiple_start=TRUE)

show(m1)      # show this rgasp object

m1@beta_hat   # estimated inverse range parameters
m1@theta_hat

#-----
# an 8 dimensional example using only a subset inputs and a noise with unknown variance
#-----
# dimensional of the inputs
dim_inputs <- 8
# number of the inputs
num_obs <- 30

# uniform samples of design
input <-matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)
# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) # maximin lhs sample

# rescale the design to the domain
input[,1]<-0.05+(0.15-0.05)*input[,1];
input[,2]<-100+(50000-100)*input[,2];
input[,3]<-63070+(115600-63070)*input[,3];
input[,4]<-990+(1110-990)*input[,4];
input[,5]<-63.1+(116-63.1)*input[,5];
input[,6]<-700+(820-700)*input[,6];
input[,7]<-1120+(1680-1120)*input[,7];
input[,8]<-9855+(12045-9855)*input[,8];

# outputs from the 8 dim Borehole function

output=matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]=borehole(input[i,])
}

# use constant mean basis with trend, with no constraint on optimization
m3<- rgasp(design = input[,c(1,4,6,7,8)], response = output,
           nugget.est=TRUE, lower_bound=FALSE,multiple_start=TRUE)

m3@beta_hat   # estimated inverse range parameters

```

m3@nugget

rgasp-class

Robust GaSP class

Description

S4 class for Robust GaSP if the range and noise-variance ratio parameters are given and/or have been estimated.

Objects from the Class

Objects of this class are created and initialized with the function `rgasp` that computes the calculations needed for setting up the analysis.

Slots

p: Object of class `integer`. The dimensions of the inputs.

num_obs: Object of class `integer`. The number of observations.

input: Object of class `matrix` with dimension $n \times p$. The design of experiments.

output: Object of class `matrix` with dimension $n \times 1$. The Observations or output vector.

X: Object of class `matrix` of with dimension $n \times q$. The mean basis function, i.e. the trend function.

q: Object of class `integer`. The number of mean basis.

LB: Object of class `vector` with dimension $p \times 1$. The lower bound for inverse range parameters `beta`.

beta_initial: Object of class `vector` with the initial values of inverse range parameters $p \times 1$.

beta_hat: Object of class `vector` with dimension $p \times 1$. The inverse-range parameters.

log_post: Object of class `numeric` with the logarithm of marginal posterior.

R0: Object of class `list` of matrices where the j -th matrix is an absolute difference matrix of the j -th input vector.

theta_hat: Object of class `vector` with dimension $q \times 1$. The the mean (trend) parameter.

L: Object of class `matrix` with dimension $n \times n$. The Cholesky decomposition of the correlation matrix `R`, i.e.

$$L \% * \%t(L) = R$$

sigma2_hat: Object of the class `matrix`. The estimated variance parameter.

LX: Object of the class `matrix` with dimension $q \times q$. The Cholesky decomposition of the correlation matrix

$$t(X) \% * \%R^{-1} \% * \%X$$

CL: Object of the class `vector` used for the lower bound and the prior.

nugget: A numeric object used for the noise-variance ratio parameter.
nugget.est: A logical object of whether the nugget is estimated (T) or fixed (F).
kernel_type: A character to specify the type of kernel to use.
alpha: A numeric parameter for the roughness in the kernel.
call: The call to `rgasp` function to create the object.

Methods

show Prints the main slots of the object.
predict See [predict](#).

Note

The number of experiments `p` must be larger or equal than the spatial dimension `num_obs`. The response output must have one dimension. The number of observations in `input` must be equal to the number of experiments output.

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]
 Maintainer: Mengyang Gu <mg211@stat.duke.edu>

See Also

[RobustGaSP](#) for more details about how to create a RobustGaSP object.

Sample

Sample for Robust GaSP model

Description

Function to sample Robust GaSP after the Robust GaSP model has been constructed.

Usage

```
## S4 method for signature 'rgasp'
Sample(object, testing_input, num_sample=1,
testing_trend= matrix(1,dim(testing_input)[1],1),...)
```

Arguments

| | |
|----------------------------|---|
| <code>object</code> | an object of class <code>rgasp</code> . |
| <code>testing_input</code> | a matrix containing the inputs where the <code>rgasp</code> is to sample. |
| <code>num_sample</code> | number of samples one wants. |
| <code>testing_trend</code> | a matrix of mean/trend for prediction. |
| <code>...</code> | Extra arguments to be passed to the function (not implemented yet). |

Value

The returned value is a matrix where each column is a sample on the prespecified inputs.

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Maintainer: Mengyang Gu <mg211@stat.duke.edu>

References

Mengyang Gu. (2016). Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output. Ph.D. thesis. Duke University.

Examples

```
#-----
# a 1 dimensional example
#-----

#####1dim hig02
p1 = 1    ###dimensional of the inputs
dim_inputs1 <- p1
n1 = 15   ###sample size or number of training computer runs you have
num_obs1 <- n1
input1 = 10*matrix(runif(num_obs1*dim_inputs1), num_obs1,dim_inputs1) ##uniform
#####lhs is better
#library(lhs)
#input1 = 10*maximinLHS(n=num_obs1, k=dim_inputs1) ##maximin lhd sample
output1 = matrix(0,num_obs1,1)
for(i in 1:num_obs1){
  output1[i]=hig02 (input1[i])
}

m1<- rgasp(design = input1, response = output1, lower_bound=FALSE)

#####locations to samples
testing_input1 = seq(0,10,1/50)
testing_input1=as.matrix(testing_input1)
#####draw 10 samples
m1_sample=Sample(m1,testing_input1,num_sample=10)

#####plot these samples
matplot(testing_input1,m1_sample, type='l',xlab='input',ylab='output')
lines(input1,output1,type='p')
```

 show

Show Robust GaSP object

Description

Function to print Robust GaSP models after the Robust GaSP model has been constructed.

Usage

```
## S4 method for signature 'rgasp'
show(object)
```

Arguments

object an object of class rgasp.

Author(s)

Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut]

Maintainer: Mengyang Gu <mg211@stat.duke.edu>

Examples

```
#-----
# a 3 dimensional example
#-----
# dimensional of the inputs
dim_inputs <- 3
# number of the inputs
num_obs <- 30
# uniform samples of design
input <- matrix(runif(num_obs*dim_inputs), num_obs,dim_inputs)

# Following codes use maximin Latin Hypercube Design, which is typically better than uniform
# library(lhs)
# input <- maximinLHS(n=num_obs, k=dim_inputs) ##maximin lhd sample

####
# outputs from the 3 dim detpep10curv function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-detpep10curv (input[i,])
}

# use constant mean basis, with no constraint on optimization
m1<- rgasp(design = input, response = output, lower_bound=FALSE)

# the following use constraints on optimization
```

```
# m1<- rgasp(design = input, response = output, lower_bound=TRUE)

# the following use a single start on optimization
# m1<- rgasp(design = input, response = output, lower_bound=FALSE, multiple_starts=FALSE)

show(m1)
```

Index

*Topic **classes**

rgasp-class, [15](#)

*Topic **computer model**

RobustGaSP-package, [2](#)

*Topic **emulation**

RobustGaSP-package, [2](#)

*Topic **package**

RobustGaSP-package, [2](#)

*Topic **simulation**

RobustGaSP-package, [2](#)

findInertInputs, [4](#)

predict, [6](#), [16](#)

predict,rgasp-method (predict), [6](#)

predict.rgasp (predict), [6](#)

predict.rgasp-class (predict), [6](#)

rgasp, [11](#), [15](#)

rgasp-class, [15](#)

rgasp-method (rgasp), [11](#)

RobustGaSP, [3](#), [16](#)

RobustGaSP (RobustGaSP-package), [2](#)

RobustGaSP-package, [2](#)

Sample, [16](#)

Sample,rgasp-method (Sample), [16](#)

Sample.rgasp (Sample), [16](#)

Sample.rgasp-class (Sample), [16](#)

show, [18](#)

show,rgasp-method (show), [18](#)

show.rgasp (show), [18](#)

show.rgasp-class (show), [18](#)