# Package ‘RobustGaSP’

February 9, 2024

**Type** Package

**Title** Robust Gaussian Stochastic Process Emulation

**Version** 0.6.6

**Date/Publication** 2024-02-09 00:30:22 UTC

**Maintainer** Mengyang Gu <mengyang@pstat.ucsb.edu>

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

**Description** Robust parameter estimation and prediction of Gaussian stochastic process emulators. It allows for robust parameter estimation and prediction using Gaussian stochastic process emulator. It also implements the parallel partial Gaussian stochastic process emulator for computer model with massive outputs See the reference: Mengyang Gu and Jim Berger, 2016, Annals of Applied Statistics; Mengyang Gu, Xiaojing Wang and Jim Berger, 2018, Annals of Statistics.

**License** GPL-2 | GPL-3

**LazyData** true

**Depends** R (>= 3.5.0), methods

**Imports** Rcpp (>= 0.12.3), nloptr (>= 1.0.4)

**LinkingTo** Rcpp, RcppEigen

**NeedsCompilation** yes

**Repository** CRAN

**RoxygenNote** 5.0.1

## R topics documented:

---

`RobustGaSP-package` *Robust Gaussian Stochastic Process Emulation*

---

## Description

Robust parameter estimation and prediction of Gaussian stochastic process emulators. It allows for robust parameter estimation and prediction using Gaussian stochastic process emulator. It also implements the parallel partial Gaussian stochastic process emulator for computer model with massive outputs See the reference: Mengyang Gu and Jim Berger, 2016, Annals of Applied Statistics; Mengyang Gu, Xiaojing Wang and Jim Berger, 2018, Annals of Statistics.

## Details

The DESCRIPTION file:

| | |
|---|---|
| Package: | RobustGaSP |
| Type: | Package |
| Title: | Robust Gaussian Stochastic Process Emulation |
| Version: | 0.6.6 |
| Date/Publication: | 2024-01-14 08:10:03 UTC |
| Authors@R: | c(person(given="Mengyang",family="Gu",role=c("aut","cre"), email="mengyang@pstat.ucsb.edu"), pe |
| Maintainer: | Mengyang Gu <mengyang@pstat.ucsb.edu> |
| Author: | Mengyang Gu [aut, cre], Jesus Palomo [aut], James Berger [aut] |
| Description: | Robust parameter estimation and prediction of Gaussian stochastic process emulators. It allows for robu |
| License: | GPL-2 \| GPL-3 |
| LazyData: | true |
| Depends: | R (>= 3.5.0), methods |
| Imports: | Rcpp (>= 0.12.3), nloptr (>= 1.0.4) |
| LinkingTo: | Rcpp, RcppEigen |
| NeedsCompilation: | yes |
| Repository: | CRAN |
| Packaged: | 2019-06-05 02:09:17 UTC; gumengyang |
| RoxygenNote: | 5.0.1 |

Index of help topics:

```
RobustGaSP-package      Robust Gaussian Stochastic Process Emulation
findInertInputs         find inert inputs with the posterior mode
humanity.X              data from the humanity model
leave_one_out_rgasp     leave-one-out fitted values and standard
                        deviation for robust GaSP model
plot                    Plot for Robust GaSP model
ppgasp                  Setting up the parallel partial GaSP model
ppgasp-class            PP GaSP class
predict.ppgasp          Prediction for PP GaSP model
predict.rgasp           Prediction for Robust GaSP model
predppgasp-class        Predicted PP GaSP class
predrgasp-class         Predictive robust GaSP class
rgasp                   Setting up the robust GaSP model
rgasp-class             Robust GaSP class
show                    Show Robust GaSP object
show.ppgasp             Show parllel partial Gaussian stochastic
                        process (PP GaSP) object
simulate                Sample for Robust GaSP model
```

## Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

## References

J.O. Berger, V. De Oliveira and B. Sanso (2001), *Objective Bayesian analysis of spatially correlated data*, *Journal of the American Statistical Association*, 96, 1361-1374.

M. Gu. and J.O. Berger (2016). Parallel partial Gaussian process emulation for computer models with massive output. *Annals of Applied Statistics*, 10(3), 1317-1347.

M. Gu. (2016). Robust uncertainty quantification and scalable computation for computer models with massive output. Ph.D. thesis. Duke University.

M. Gu, X. Wang and J.O. Berger (2018), *Robust Gaussian stochastic process emulation*, *Annals of Statistics*, 46(6A), 3038-3066.

M. Gu (2018), *Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection*, arXiv:1804.09329.

R. Paulo (2005), *Default priors for Gaussian processes*, *Annals of statistics*, 33(2), 556-582.

J. Sacks, W.J. Welch, T.J. Mitchell, and H.P. Wynn (1989), *Design and analysis of computer experiments*, *Statistical Science*, **4**, 409-435.

## 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 dettepepel.3.data function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-dettepepel.3.data(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)

# 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, outasS3 = FALSE)
# 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]<-dettepepel.3.data(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` or the `ppgasp`. |
| `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

object@p*object@beta_hat*object@CL/sum(object@beta_hat*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 `rgasp` 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 the reference below.

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 inerts. 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 <mengyang@pstat.ucsb.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)
```

humanity_model *data from the humanity model*

## Description

This data set provides the training data and testing data from the 'diplomatic and military operations in a non-warfighting domain' (DIAMOND) simulator. It porduces the number of casualties during the second day to sixth day after the earthquake and volcanic eruption in Giarre and Catania. See (Overstall and Woods (2016)) for details.

## Usage

```
data(humanity_model)
```

## Format

Four data frame with observations on the following variables.

`humanity.X` A matrix of the training inputs.

`humanity.Y` A matrix of the output of the calsualties from the second to sixth day after the the earthquake and volcanic eruption for each set of training inputs.

`humanity.Xt` A matrix of the test inputs.

`humanity.Yt` A matrix of the test output of the calsualties.

## References

A. M. Overstall and D. C. Woods (2016). Multivariate emulation of computer simulators: model selection and diagnostics with application to a humanitarian relief model. Journal of the Royal Statistical Society: Series C (Applied Statistics), 65(4):483-505.

B. Taylor and A. Lane. Development of a novel family of military campaign simulation models. Journal of the Operational Research Society, 55(4):333-339, 2004.

`leave_one_out_rgasp` *leave-one-out fitted values and standard deviation for robust GaSP model*

## Description

A function to calculate leave-one-out fitted values and the standard deviation of the prediction on robust GaSP models after the robust GaSP model has been constructed.

## Usage

```
leave_one_out_rgasp(object)
```

## Arguments

| | |
|---|---|
| `object` | an object of class `rgasp`. |

## Value

A list of 2 elements with

| | |
|---|---|
| `mean` | leave one out fitted values. |
| `sd` | standard deviation of each prediction. |

## Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

## References

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

## See Also

rgasp

## 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 dettepepel.3.data function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-dettepepel.3.data (input[i,])
}

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

##leave one out predict
leave_one_out_m1=leave_one_out_rgasp(m1)

##predictive mean
leave_one_out_m1$mean
##standard deviation
leave_one_out_m1$sd
##standardized error
(leave_one_out_m1$mean-output)/leave_one_out_m1$sd
```

---

plot *Plot for Robust GaSP model*

---

## Description

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

## Usage

```
## S4 method for signature 'rgasp'
plot(x,y, ...)
```

## Arguments

| | |
|---|---|
| `x` | an object of class `rgasp`. |
| `y` | not used. |
| `...` | additional arguments not implemented yet. |

## Value

Three plots: the leave-one-out fitted values versus exact values, standardized residuals and QQ plot.

### Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

### References

M. 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 dettepepel.3.data function

 output = matrix(0,num_obs,1)
 for(i in 1:num_obs){
   output[i]<-dettepepel.3.data (input[i,])
 }

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

 # plot
 plot(m1)
```

---

ppgasp *Setting up the parallel partial GaSP model*

---

### Description

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

### Usage

```
ppgasp(design,response,trend=matrix(1,dim(response)[1],1),zero.mean="No",nugget=0,
 nugget.est=F,range.par=NA,method='post_mode',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',isotropic=F,R0=NA,
  optimization='lbfgs',
  alpha=rep(1.9,dim(as.matrix(design))[2]),lower_bound=T,
 max_eval=max(30,20+5*dim(design)[2]),initial_values=NA,num_initial_values=2)
```

### Arguments

| | |
|---|---|
| `design` | a matrix of inputs. |
| `response` | a matrix of outputs where each row is one runs of the computer model output. |
| `trend` | the mean/trend matrix of inputs. The default value is a vector of ones. |
| `zero.mean` | it has zero mean or not. The default value is FALSE meaning the mean is not zero. TRUE means the mean is zero. |
| `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`. |
| `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`. |
| `method` | method of parameter estimation. `post_mode` means the marginal posterior mode is used for estimation. `mle` means the maximum likelihood estimation is used. `mmle` means the maximum marginal likelihood estimation is used. The `post_mode` is the default method. |
| `prior_choice` | the choice of prior for range parameters and noise-variance parameters. `ref_xi` and `ref_gamma` means the reference prior with reference prior with the log of inverse range parameterization $\xi$ or range parameterization $\gamma$. `ref_approx` uses the jointly robust prior to approximate the reference prior. The default choice is `ref_approx`. |
| `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` | A vector specifying the type of kernels of each coordinate of the input. `matern_3_2` and `matern_5_2` are `Matern correlation` with roughness parameter 3/2 and 5/2 respectively. `pow_exp` is power exponential correlation with roughness parameter alpha. If `pow_exp` is to be used, one needs to specify its roughness parameter alpha. The default choice is `matern_5_2`. |
| `isotropic` | a boolean value. `T` means the isotropic kernel will be used and `F` means the separable kernel will be used. The default choice is the separable kernel. |

| | |
|---|---|
| `R0` | the distance between inputs. If the value is `NA`, it will be calculated later. It can also be specified by the user. If specified by user, it is either a `matrix` or `list`. The default value is `NA`. |
| `optimization` | the method for numerically optimization of the kernel parameters. Currently three methods are implemented. `lbfgs` is the low-storage version of the Broyden-Fletcher-Goldfarb-Shanno method. `nelder-mead` is the Nelder and Mead method. `brent` is the Brent method for one-dimensional problems. |
| `alpha` | roughness parameters in the `pow_exp` correlation functions. The default choice is a vector with each entry being 1.9. |
| `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. |
| `initial_values` | a matrix of initial values of the kernel parameters to be optimized numerically, where each row of the matrix contains a set of the log inverse range parameters and the log nugget parameter. |
| `num_initial_values` | the number of initial values of the kernel parameters in optimization. |

## Value

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

## Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

## References

M. Gu. and J.O. Berger (2016). Parallel partial Gaussian process emulation for computer models with massive output. *Annals of Applied Statistics*, 10(3), 1317-1347.

M. Gu, X. Wang and J.O. Berger (2018), *Robust Gaussian stochastic process emulation*, *Annals of Statistics*, 46(6A), 3038-3066.

M. Gu (2018), *Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection*, arXiv:1804.09329.

J. Nocedal (1980), Updating quasi-Newton matrices with limited storage, *Math. Comput.*, 35, 773-782.

D. C. Liu and J. Nocedal (1989), On the limited memory BFGS method for large scale optimization, *Math. Programming*, 45, p. 503-528.

Brent, R. (1973), Algorithms for Minimization without Derivatives. Englewood Cliffs N.J.: Prentice-Hall.

## Examples

```
library(RobustGaSP)

###parallel partial Gaussian stochastic process (PP GaSP) model
##for the humanity model
data(humanity_model)
##120 runs. The input has 13 variables and output is 5 dimensional.
##PP GaSP Emulator
m.ppgasp=ppgasp(design=humanity.X,response=humanity.Y,nugget.est= TRUE)
show(m.ppgasp)

##make predictions
m_pred=predict(m.ppgasp,humanity.Xt)
sqrt(mean((m_pred$mean-humanity.Yt)^2))
mean(m_pred$upper95>humanity.Yt & humanity.Yt>m_pred$lower95)
mean(m_pred$upper95-m_pred$lower95)
sqrt( mean( (mean(humanity.Y)-humanity.Yt)^2 ))

##with a linear trend on the selected input performs better
## Not run:
  ###PP GaSP Emulation with a linear trend for the humanity model
  data(humanity_model)
  ##pp gasp with trend
  n<-dim(humanity.Y)[1]
  n_testing=dim(humanity.Yt)[1]
  H=cbind(matrix(1,n,1),humanity.X$foodC)
  H_testing=cbind(matrix(1,n_testing,1),humanity.Xt$foodC)
  m.ppgasp_trend=ppgasp(design=humanity.X,response=humanity.Y,trend=H,
  nugget.est= TRUE)

  show(m.ppgasp_trend)

  ##make predictions
  m_pred_trend=predict(m.ppgasp_trend,humanity.Xt,testing_trend=H_testing)
  sqrt(mean((m_pred_trend$mean-humanity.Yt)^2))
  mean(m_pred_trend$upper95>humanity.Yt & humanity.Yt>m_pred_trend$lower95)
  mean(m_pred_trend$upper95-m_pred_trend$lower95)

## End(Not run)
```

---

`ppgasp-class` *PP GaSP class*

---

## Description

S4 class for PP GaSP model 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 ppgasp 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.

**k:** Object of class integer. The number of outputs in each computer model run.

**input:** Object of class matrix with dimension n x p. The design of experiments.

**output:** Object of class matrix with dimension n x k. Each row denotes a output vector in each run of the computer model.

**X:** Object of class matrix of with dimension n x q. The mean basis function, i.e. the trend function.

**zero_mean:** A character to specify whether the mean is zero or not. "Yes" means it has zero mean and "No"" means the mean is not zero.

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

**LB:** Object of class vector with dimension p x 1. The lower bound for inverse range parameters beta.

**beta_initial:** Object of class vector with the initial values of inverse range parameters p x 1.

**beta_hat:** Object of class vector with dimension p x 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 x 1. The the mean (trend) parameter.

**L:** Object of class matrix with dimension n x 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 of each output.

**LX:** Object of the class matrix with dimension q x 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 vector of character to specify the type of kernel to use.

**alpha:** Object of class vector with dimension p x 1 for the roughness parameters in the kernel.

**method:** Object of class character to specify the method of parameter estimation. There are three values: post_mode, mle and mmle.

**isotropic:** Object of class logical to specify whether the kernel is isotropic.

**call:** The call to ppgasp function to create the object.

### Methods

**show** Prints the main slots of the object.

**predict** See predict.

### Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

### See Also

RobustGaSP for more details about how to create a RobustGaSP object.

---

## predict.ppgasp *Prediction for PP GaSP model*

---

### Description

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

### Usage

```
## S4 method for signature 'ppgasp'
predict(object, testing_input,
testing_trend= matrix(1,dim(testing_input)[1],1),r0=NA,
interval_data=T,
outasS3 = T,loc_index=NA, ...)
```

### Arguments

| | |
|---|---|
| object | an object of class ppgasp. |
| testing_input | a matrix containing the inputs where the rgasp is to perform prediction. |
| testing_trend | a matrix of mean/trend for prediction. |
| r0 | the distance between input and testing input. If the value is NA, it will be calculated later. It can also be specified by the user. If specified by user, it is either a matrix or list. The default value is NA. |
| interval_data | a boolean value. If T, the interval of the data will be calculated. Otherwise, the interval of the mean of the data will be calculted. |
| outasS3 | a boolean parameter indicating whether the output of the function should be as an S3 object. |
| loc_index | specified coodinate index of the prediction. The default value is NA and prediction will be computed for all coordinates. If e.g. loc_index=c(3,5), it means the prediction will be computed on only the third and fifth coordinates, corresponding the coordinates of the third and fifth columns of the output matrix. |
| ... | Extra arguments to be passed to the function (not implemented yet). |

## Value

If the parameter outasS3=F, then the returned value is a S4 object of class predppgasp-class with

| | |
|---|---|
| call: | call to predict.ppgasp function where the returned object has been created. |
| 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. |

If the parameter outasS3=T, then 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 <mengyang@pstat.ucsb.edu>

## References

M. Gu. and J.O. Berger (2016). Parallel partial Gaussian process emulation for computer models with massive output. *Annals of Applied Statistics*, 10(3), 1317-1347.

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

## Examples

```
library(RobustGaSP)
#---------------------------------
# an example of environmental model
#---------------------------------

set.seed(1)
#Here the sample size is very small. Consider to use more observations
n=80
p=4
##using the latin hypercube will be better
#library(lhs)
#input_samples=maximinLHS(n,p)
input_samples=matrix(runif(n*p),n,p)
input=matrix(0,n,p)
input[,1]=7+input_samples[,1]*6
input[,2]=0.02+input_samples[,2]*1
input[,3]=0.01+input_samples[,3]*2.99
```

```
input[,4]=30.01+input_samples[,4]*0.285

k=300
output=matrix(0,n,k)
##environ.4.data is an environmental model on a spatial-time vector
##? environ.4.data
for(i in 1:n){
  output[i,]=environ.4.data(input[i,],s=seq(0.15,3,0.15),t=seq(4,60,4)  )
}

##samples some test inputs
n_star=1000
sample_unif=matrix(runif(n_star*p),n_star,p)

testing_input=matrix(0,n_star,p)
testing_input[,1]=7+sample_unif[,1]*6
testing_input[,2]=0.02+sample_unif[,2]*1
testing_input[,3]=0.01+sample_unif[,3]*2.99
testing_input[,4]=30.01+sample_unif[,4]*0.285


testing_output=matrix(0,n_star,k)

s=seq(0.15,3,0.15)
t=seq(4,60,4)

for(i in 1:n_star){
  testing_output[i,]=environ.4.data(testing_input[i,],s=s,t=t )
}

##we do a transformation of the output
##one can change the number of initial values to test
log_output_1=log(output+1)
#since we have lots of output, we use 'nelder-mead' for optimization
m.ppgasp=ppgasp(design=input,response=log_output_1,kernel_type
                ='pow_exp',num_initial_values=2,optimization='nelder-mead')

m_pred.ppgasp=predict(m.ppgasp,testing_input)
##we transform back for the prediction
m_pred_ppgasp_median=exp(m_pred.ppgasp$mean)-1
##mean squared error
mean( (m_pred_ppgasp_median-testing_output)^2)
##variance of the testing outputs
var(as.numeric(testing_output))

##makes plots for the testing
par(mfrow=c(1,2))
testing_plot_1=matrix(testing_output[1,],  length(t), length(s) )

max_testing_plot_1=max(testing_plot_1)
min_testing_plot_1=min(testing_plot_1)

image(x=t,y=s,testing_plot_1,  col = hcl.colors(100, "terrain"),main='test outputs')
```

```
contour(x=t,y=s,testing_plot_1, levels = seq(min_testing_plot_1, max_testing_plot_1,
                                      by = (max_testing_plot_1-min_testing_plot_1)/5),
        add = TRUE, col = "brown")

ppgasp_plot_1=matrix(m_pred_ppgasp_median[1,],  length(t), length(s) )
max_ppgasp_plot_1=max(ppgasp_plot_1)
min_ppgasp_plot_1=min(ppgasp_plot_1)

image(x=t,y=s,ppgasp_plot_1,  col = hcl.colors(100, "terrain"),main='prediction')
contour(x=t,y=s,ppgasp_plot_1, levels = seq(min_testing_plot_1, max_ppgasp_plot_1,
                                        by = (max_ppgasp_plot_1-min_ppgasp_plot_1)/5),
        add = TRUE, col = "brown")
dev.off()
```

---

predict.rgasp *Prediction for Robust GaSP model*

---

## Description

Function to make prediction on the robust GaSP model 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),
r0=NA,interval_data=T,
outasS3 = T,...)
```

## 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. |
| r0 | the distance between input and testing input. If the value is NA, it will be calculated later. It can also be specified by the user. If specified by user, it is either a matrix or list. The default value is NA. |
| interval_data | a boolean value. If T, the interval of the data will be calculated. Otherwise, the interval of the mean of the data will be calculted. |
| outasS3 | a boolean parameter indicating whether the output of the function should be as an S3 object. |
| ... | Extra arguments to be passed to the function (not implemented yet). |

## Value

If the parameter `outasS3=F`, then the returned value is a `S4 object` of class `predrgasp-class` with

**`call`:** `call` to `predict.rgasp` function where the returned object has been created.

**`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`.

If the parameter `outasS3=T`, then 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 <mengyang@pstat.ucsb.edu>

## References

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

M. Gu. and J.O. Berger (2016). Parallel partial Gaussian process emulation for computer models with massive output. *Annals of Applied Statistics*, 10(3), 1317-1347.

M. Gu, X. Wang and J.O. Berger (2018), *Robust Gaussian Stochastic Process Emulation*, *Annals of Statistics*, 46(6A), 3038-3066.

M. Gu (2018), *Jointly Robust Prior for Gaussian Stochastic Process in Emulation, Calibration and Variable Selection*, arXiv:1804.09329.

## 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 dettepepel.3.data function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-dettepepel.3.data (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=FALS)

# 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]<-dettepepel.3.data(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]<-limetal.2.data (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)


# 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]<-limetal.2.data(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 ))


   ###here try the isotropic kernel (a function of Euclidean distance)
 m2_isotropic<- rgasp(design = input, response = output,trend =X,
            lower_bound=FALSE,isotropic=TRUE)

 m2_isotropic.predict<-predict(m2_isotropic, testing_input,testing_trend=testing_X)

 # compute the MSE, average coverage and average length
 # out of sample MSE
MSE_emulator_isotropic <- sum((m2_isotropic.predict$mean-testing_output)^2)/(num_testing_input)

 # proportion covered by 95% posterior predictive credible interval
 prop_emulator_isotropic <- length(which((m2_isotropic.predict$lower95<=testing_output)
                    &(m2_isotropic.predict$upper95>=testing_output)))/num_testing_input

 # average length of  posterior predictive credible interval
 length_emulator_isotropic <- sum(m2_isotropic.predict$upper95-
 m2_isotropic.predict$lower95)/num_testing_input

 MSE_emulator_isotropic
 prop_emulator_isotropic
 length_emulator_isotropic
 ##the result of isotropic kernel is not as good as the product kernel for this example


#--------------------------------------------------------------------------------------
 # an 8 dimensional example using only a subset inputs and a noise with unknown variance
#--------------------------------------------------------------------------------------
 set.seed(1)
 # dimensional of the inputs
 dim_inputs <- 8
 # number of the inputs
 num_obs <- 50

 # 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<- rgasp(design = input[,c(1,4,6,7,8)], response = output,
           nugget.est=TRUE, lower_bound=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)

# resale 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 ))
```

---

`predppgasp-class` *Predicted PP GaSP class*

---

### Description

S4 class for the prediction of a PP GaSP model

### Objects from the Class

Objects of this class are created and initialized with the function `predict.ppgasp` that computes the prediction on the PP GaSP model after the PP GaSP model has been constructed.

### Slots

`call`**:** `call` to `predict.ppgasp` function where the returned object has been created.

`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 <mengyang@pstat.ucsb.edu>

### See Also

`predict.ppgasp` for more details about how to make predictions based on a ppgasp object.

---

`predrgasp-class` *Predictive robust GaSP class*

---

## Description

S4 class for the prediction of a Robust GaSP

## Objects from the Class

Objects of this class are created and initialized with the function `predict.rgasp` that computes the prediction on Robust GaSP models after the Robust GaSP model has been constructed.

## Slots

**`call`:** `call` to `predict.rgasp` function where the returned object has been created.

**`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 <mengyang@pstat.ucsb.edu>

## See Also

`predict.rgasp` for more details about how to make predictions based on a `rgasp` object.

---

`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),zero.mean="No",nugget=0,
 nugget.est=F,range.par=NA,method='post_mode',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',isotropic=F,R0=NA,
  optimization='lbfgs', alpha=rep(1.9,dim(as.matrix(design))[2]),
  lower_bound=T,max_eval=max(30,20+5*dim(design)[2]),
  initial_values=NA,num_initial_values=2)
```

### 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. |
| `zero.mean` | it has zero mean or not. The default value is `NO` meaning the mean is not zero. `Yes` means the mean is zero. |
| `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`. |
| `method` | method of parameter estimation. `post_mode` means the marginal posterior mode is used for estimation. `mle` means the maximum likelihood estimation is used. `mmle` means the maximum marginal likelihood estimation is used. The `post_mode` is the default method. |
| `prior_choice` | the choice of prior for range parameters and noise-variance parameters. `ref_xi` and `ref_gamma` means the reference prior with reference prior with the log of inverse range parameterization $\xi$ or range parameterization $\gamma$. `ref_approx` uses the jointly robust prior to approximate the reference prior. The default choice is `ref_approx`. |
| `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` | A vector specifying the type of kernels of each coordinate of the input. `matern_3_2` and `matern_5_2` are `Matern correlation` with roughness parameter 3/2 and 5/2 respectively. `pow_exp` is power exponential correlation with roughness parameter alpha. If `pow_exp` is to be used, one needs to specify its roughness parameter alpha. The default choice is `matern_5_2`. The `periodic_gauss` means the Gaussian kernel with periodic folding method with be used. The `periodic_exp` means the exponential kernel with periodic folding method will be used. |
| `isotropic` | a boolean value. `T` means the isotropic kernel will be used and `F` means the separable kernel will be used. The default choice is the separable kernel. |
| `R0` | the distance between inputs. If the value is `NA`, it will be calculated later. It can also be specified by the user. If specified by user, it is either a `matrix` or `list`. The default value is `NA`. |
| `optimization` | the method for numerically optimization of the kernel parameters. Currently three methods are implemented. `lbfgs` is the low-storage version of the Broyden-Fletcher-Goldfarb-Shanno method. `nelder-mead` is the Nelder and Mead method. `brent` is the Brent method for one-dimensional problems. |
| `alpha` | roughness parameters in the `pow_exp` correlation functions. The default choice is a vector with each entry being 1.9. |

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.

initial_values
: a matrix of initial values of the kernel parameters to be optimized numerically, where each row of the matrix contains a set of the log inverse range parameters and the log nugget parameter.

num_initial_values
: the number of initial values of the kernel parameters in optimization.

## 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 <mengyang@pstat.ucsb.edu>

## References

M. Gu, X. Wang and J.O. Berger (2018), Robust Gaussian stochastic process emulation, *Annals of Statistics*, 46(6A), 3038-3066.

M. Gu (2018), Jointly robust prior for Gaussian stochastic process in emulation, calibration and variable selection, arXiv:1804.09329.

M. Gu. (2016). Robust uncertainty quantification and scalable computation for computer models with massive output. Ph.D. thesis. Duke University.

M. Gu. and J.O. Berger (2016). Parallel partial Gaussian process emulation for computer models with massive output. *Annals of Applied Statistics*, 10(3), 1317-1347.

E.T. Spiller, M.J. Bayarri, J.O. Berger and E.S. Calder and A.K. Patra and E.B. Pitman, and R.L. Wolpert (2014), Automating emulator construction for geophysical hazard maps. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1), 126-152.

J. Nocedal (1980), Updating quasi-Newton matrices with limited storage, *Math. Comput.*, 35, 773-782.

D. C. Liu and J. Nocedal (1989), On the limited memory BFGS method for large scale optimization, *Math. Programming*, 45, p. 503-528.

Brent, R. (1973), Algorithms for Minimization without Derivatives. Englewood Cliffs N.J.: Prentice-Hall.

## Examples

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

```
#------------------------
# dimensional of the inputs
dim_inputs <- 3
# number of the inputs
num_obs <- 50
# 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 dettepepel.3.data function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-dettepepel.3.data (input[i,])
}

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

# you can use specify the estimation as maximum likelihood estimation (MLE)
m2<- rgasp(design = input, response = output, method='mle',lower_bound=FALSE)

##let's do some comparison on prediction
n_testing=1000
testing_input=matrix(runif(n_testing*dim_inputs),n_testing,dim_inputs)

m1_pred=predict(m1,testing_input=testing_input)
m2_pred=predict(m2,testing_input=testing_input)


##root of mean square error and interval
test_output = matrix(0,n_testing,1)
for(i in 1:n_testing){
  test_output[i]<-dettepepel.3.data (testing_input[i,])
}

##root of mean square error
sqrt(mean( (m1_pred$mean-test_output)^2))
sqrt(mean( (m2_pred$mean-test_output)^2))
sd(test_output)
#---------------------------------------
# a 1 dimensional example with zero mean
#---------------------------------------


input=10*seq(0,1,1/14)
output<-higdon.1.data(input)
#the following code fit a GaSP with zero mean by setting zero.mean="Yes"
```

```
model<- rgasp(design = input, response = output, zero.mean="Yes")
model

testing_input = as.matrix(seq(0,10,1/100))
model.predict<-predict(model,testing_input)
names(model.predict)

#########plot predictive distribution
testing_output=higdon.1.data(testing_input)
plot(testing_input,model.predict$mean,type='l',col='blue',
     xlab='input',ylab='output')
polygon( c(testing_input,rev(testing_input)),c(model.predict$lower95,
      rev(model.predict$upper95)),col =  "grey80", border = FALSE)
lines(testing_input, testing_output)
lines(testing_input,model.predict$mean,type='l',col='blue')
lines(input, output,type='p')

## mean square erros
mean((model.predict$mean-testing_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 a 2 dim function

output <- matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-limetal.2.data (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)

show(m2)      # show this rgasp object

m2@beta_hat       # estimated inverse range parameters
m2@theta_hat      # estimated trend parameters
```

```
#--------------------------------------------------------------------------------------
# an 8 dimensional example using only a subset inputs and a noise with unknown variance
#--------------------------------------------------------------------------------------
set.seed(1)
# dimensional of the inputs
dim_inputs <- 8
# number of the inputs
num_obs <- 50

# 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<- rgasp(design = input[,c(1,4,6,7,8)], response = output,
          nugget.est=TRUE, lower_bound=FALSE)

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 x p. The design of experiments.

**output:** Object of class matrix with dimension n x 1. The Observations or output vector.

**X:** Object of class matrix of with dimension n x q. The mean basis function, i.e. the trend function.

**zero_mean:** A character to specify whether the mean is zero or not. "Yes" means it has zero mean and "No"" means the mean is not zero.

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

**LB:** Object of class vector with dimension p x 1. The lower bound for inverse range parameters beta.

**beta_initial:** Object of class vector with the initial values of inverse range parameters p x 1.

**beta_hat:** Object of class vector with dimension p x 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 x 1. The the mean (trend) parameter.

**L:** Object of class matrix with dimension n x n. The Cholesky decomposition of the correlation matrix R, i.e.

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

**sigma2_hat:** Object of the class numeric. The estimated variance parameter.

**LX:** Object of the class matrix with dimension q x 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 vector of character to specify the type of kernel to use.

**alpha:** Object of class vector with dimension p x 1 for the roughness parameters in the kernel.

**method:** Object of class character to specify the method of parameter estimation. There are three values: post_mode, mle and mmle.

**isotropic:** Object of class logical to specify whether the kernel is isotropic.

**call:** The call to rgasp function to create the object.

### Methods

**show** Prints the main slots of the object.

**predict** See `predict`.

### Note

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 <mengyang@pstat.ucsb.edu>

### See Also

`RobustGaSP` for more details about how to create a `RobustGaSP` object.

---

## 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 <mengyang@pstat.ucsb.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 dettepepel.3.data function

output = matrix(0,num_obs,1)
for(i in 1:num_obs){
  output[i]<-dettepepel.3.data (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)


show(m1)
```

---

`show.ppgasp` *Show parllel partial Gaussian stochastic process (PP GaSP) object*

---

## Description

Function to print the PP GaSP model after the PP GaSP model has been constructed.

## Usage

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

## Arguments

| | |
|---|---|
| `object` | an object of class ppgasp. |

## Author(s)

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

Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

## Examples

```
library(RobustGaSP)

###PP GaSP model for the humanity model
data(humanity_model)
##pp gasp
m.ppgasp=ppgasp(design=humanity.X,response=humanity.Y,nugget.est= TRUE)
show(m.ppgasp)
```

---

simulate *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'
simulate(object, testing_input, num_sample=1,
testing_trend= matrix(1,dim(testing_input)[1],1),
r0=NA,rr0=NA,sample_data=T,...)
```

## Arguments

| | |
|---|---|
| `object` | an object of class `rgasp`. |
| `testing_input` | a matrix containing the inputs where the `rgasp` is to sample. |
| `num_sample` | number of samples one wants. |
| `testing_trend` | a matrix of mean/trend for prediction. |
| `r0` | the distance between input and testing input. If the value is `NA`, it will be calculated later. It can also be specified by the user. If specified by user, it is either a `matrix` or `list`. The default value is `NA`. |
| `rr0` | the distance between testing input and testing input. If the value is `NA`, it will be calculated later. It can also be specified by the user. If specified by user, it is either a `matrix` or `list`. The default value is `NA`. |
| `sample_data` | a boolean value. If `T`, the interval of the data will be calculated. Otherwise, the interval of the mean of the data will be calculted. |
| `...` | 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 <mengyang@pstat.ucsb.edu>

## References

M. 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 higdon.1.data
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]=higdon.1.data (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=simulate(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')
```

# Index