Package 'FastGaSP'

September 3, 2021

Type Package

Title Fast and Exact Computation of Gaussian Stochastic Process

Version 0.5.2

Date 2021-08-28

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

Author Mengyang Gu [aut, cre]

Description
Implements fast and exact computation of Gaussian stochastic process with the Matern kernel using forward filtering and backward smoothing algorithm. It allows for the cases with or without a noise. See the reference: Mengyang Gu and Yanxun Xu (2017), <arXiv:1711.11501>.

License GPL (>= 2)

Depends methods

Imports Rcpp, RobustGaSP

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Repository CRAN

RoxygenNote 5.0.1

Date/Publication 2021-09-02 23:20:08 UTC

R topics documented:

FastGaSP-package *Fast and Exact Computation of Gaussian Stochastic Process*

Description

Implements fast and exact computation of Gaussian stochastic process with the Matern kernel using forward filtering and backward smoothing algorithm. It allows for the cases with or without a noise. See the reference: Mengyang Gu and Yanxun Xu (2017), <arXiv:1711.11501>.

Details

The DESCRIPTION file:

Package:	FastGaSP
Type:	Package
Title:	Fast and Exact Computation of Gaussian Stochastic Process
Version:	0.5.2
Date:	2021-08-28
Authors@R:	c(person(given="Mengyang",family="Gu",role=c("aut","cre"),email="mengyang@pstat.ucsb.edu"))
Maintainer:	Mengyang Gu <mengyang@pstat.ucsb.edu>
Author:	Mengyang Gu [aut, cre]
Description:	Implements fast and exact computation of Gaussian stochastic process with the Matern kernel using forv
License:	GPL (>= 2)
Depends:	methods
Imports:	Rcpp, RobustGaSP
LinkingTo:	Rcpp, RcppEigen
NeedsCompilation:	yes
Repository:	CRAN
Packaged:	2021-08-28 19:14:39 UTC; gumengyang
RoxygenNote:	5.0.1
Date/Publication:	2018-12-11 14:41:24 UTC

Index of help topics:

Fast computational algorithms for Gaussian stochastic process with Matern kernels by the forward filtering and backward smoothing algorithm.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu and Y. Xu (2020), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, *Fast Nonseparable Gaussian Stochastic Process With Application to Methylation Level Interpolation*, *Journal of Computational and Graphical Statistics*, **29**, 250-260.

M. Gu and W. Shen (2020), *Generalized probabilistic principal component analysis of correlated data*, *Journal of Machine Learning Research*, **21**, 13-1.

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

See Also

FastGaSP

Examples

```
library(FastGaSP)

#--------------------------------------------------------------------------------
# Example 1 : fast computation algorithm for noisy data
#--------------------------------------------------------------------------------

y_R<-function(x){
  sin(2*pi*x)
}

###let's test for 2000 observations
set.seed(1)
num_obs=2000
input=runif(num_obs)
output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)

##constucting the fgasp.model
fgasp.model=fgasp(input, output)

##range and noise-variance ratio (nugget) parameters
param=c( log(1),log(.02))
## the log lik
```

```
log_lik(param,fgasp.model)
##time cost to compute the likelihood
time_cost=system.time(log_lik(param,fgasp.model))
time_cost[1]

##consider a nonparametric regression setting
##estimate the parameter by maximum likelihood estimation

est_all<-optim(c(log(1),log(.02)),log_lik,object=fgasp.model,method="L-BFGS-B",
              control = list(fnscale=-1))

##estimated log inverse range parameter and log nugget
est_all$par

##estimate variance
est.var=Get_log_det_S2(est_all$par,fgasp.model@have_noise,fgasp.model@delta_x,
                    fgasp.model@output,fgasp.model@kernel_type)[[2]]/fgasp.model@num_obs
est.var

###1. Do some interpolation test
num_test=5000
testing_input=runif(num_test) ##there are the input where you don't have observations
pred.model=predict(param=est_all$par,object=fgasp.model,testing_input=testing_input)

lb=pred.model@mean+qnorm(0.025)*sqrt(pred.model@var)
ub=pred.model@mean+qnorm(0.975)*sqrt(pred.model@var)

## calculate lb for the mean function
pred.model2=predict(param=est_all$par,object=fgasp.model,testing_input=testing_input,var_data=FALSE)
lb_mean_funct=pred.model2@mean+qnorm(0.025)*sqrt(pred.model2@var)
ub_mean_funct=pred.model2@mean+qnorm(0.975)*sqrt(pred.model2@var)

## plot the prediction
min_val=min(lb,output)
max_val=max(ub,output)

plot(pred.model@testing_input,pred.model@mean,type='l',col='blue',
     ylim=c(min_val,max_val),
     xlab='x',ylab='y')
polygon(c(pred.model@testing_input,rev(pred.model@testing_input)),
        c(lb,rev(ub)),col = "grey80", border = FALSE)
lines(pred.model@testing_input,pred.model@mean,type='l',col='blue')
lines(pred.model@testing_input,y_R(pred.model@testing_input),type='l',col='black')
lines(pred.model2@testing_input,lb_mean_funct,col='blue',lty=2)
lines(pred.model2@testing_input,ub_mean_funct,col='blue',lty=2)
lines(input,output,type='p',pch=16,col='black',cex=0.4) #one can plot data

legend("bottomleft", legend=c("predictive mean","95% predictive interval","truth"),
       col=c("blue","blue","black"), lty=c(1,2,1), cex=.8)

#-------------------------------------------------------------
# Example 2: example that one does not have a noise in the data
#-------------------------------------------------------------
```

```
## Here is a function in the Sobolev Space with order 3
y_R<-function(x){
  j_seq=seq(1,200,1)
  record_y_R=0
  for(i_j in 1:200){
    record_y_R=record_y_R+2*j_seq[i_j]^{-2*3}*sin(j_seq[i_j])*cos(pi*(j_seq[i_j]-0.5)*x)

  }
  record_y_R
}


##generate some data without noise
num_obs=50
input=seq(0,1,1/(num_obs-1))

output=y_R(input)


##constucting the fgasp.model
fgasp.model=fgasp(input, output,have_noise=FALSE)

##range and noise-variance ratio (nugget) parameters
param=c( log(1))
## the log lik
log_lik(param,fgasp.model)



#if one does not have noise one may need to give a lower bound or use a penalty
#(e.g. induced by a prior) to make the estimation more robust
est_all<-optimize(log_lik,interval=c(0,10),maximum=TRUE,fgasp.model)

##Do some interpolation test for comparison
num_test=1000
testing_input=runif(num_test) ##there are the input where you don't have observations

pred.model=predict(param=est_all$maximum,object=fgasp.model,testing_input=testing_input)



#This is the 95 posterior credible interval for the outcomes which contain the estimated
#variance of the noise
#sometimes there are numerical instability is one does not have noise or error
lb=pred.model@mean+qnorm(0.025)*sqrt(abs(pred.model@var))
ub=pred.model@mean+qnorm(0.975)*sqrt(abs(pred.model@var))

## plot the prediction
min_val=min(lb,output)
max_val=max(ub,output)

plot(pred.model@testing_input,pred.model@mean,type='l',col='blue',
     ylim=c(min_val,max_val),
```

```
     xlab='x',ylab='y')
polygon( c(pred.model@testing_input,rev(pred.model@testing_input)),
         c(lb,rev(ub)),col = "grey80", border = FALSE)
lines(pred.model@testing_input,pred.model@mean,type='l',col='blue')
lines(pred.model@testing_input,y_R(pred.model@testing_input),type='l',col='black')
lines(input,output,type='p',pch=16,col='black')
legend("bottomleft", legend=c("predictive mean","95% predictive interval","truth"),
       col=c("blue","blue","black"), lty=c(1,2,1), cex=.8)


##mean square error for all inputs
mean((pred.model@mean- y_R(pred.model@testing_input))^2)
```

`fgasp` *Setting up the Fast GaSP model*

Description

Creating an `fgasp` class for a GaSP model with matern covariance.

Usage

```
fgasp(input, output, have_noise=TRUE, kernel_type='matern_5_2')
```

Arguments

`input`	a vector with dimension num_obs x 1 for the sorted input locations.
`output`	a vector with dimension n x 1 for the observations at the sorted input locations.
`have_noise`	a bool value. If it is true, it means the model contains a noise.
`kernel_type`	a `character` to specify the type of kernel to use. The current version supports kernel_type to be "matern_5_2" or "exp", meaning that the matern kernel with roughness parameter being 2.5 or 0.5 (exponent kernel), respectively.

Value

`fgasp` returns an S4 object of class `fgasp` (see `fgasp`).

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

Examples

```
library(FastGaSP)

#-------------------------------------
# Example 1: a simple example with noise
#-------------------------------------

y_R<-function(x){
  cos(2*pi*x)
}

###let's test for 2000 observations
set.seed(1)
num_obs=2000
input=runif(num_obs)

output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)

##constucting the fgasp.model
fgasp.model=fgasp(input, output)
show(fgasp.model)

#-----------------------------------------
# Example 2: a simple example with no noise
#-----------------------------------------

y_R<-function(x){
  sin(2*pi*x)
}


##generate some data without noise
num_obs=50
input=seq(0,1,1/(num_obs-1))

output=y_R(input)


##constucting the fgasp.model
fgasp.model=fgasp(input, output,have_noise=FALSE)
```

```
show(fgasp.model)
```

`fgasp-class` *Fast GaSP class*

Description

S4 class for fast computation of the Gaussian stochastic process (GaSP) model with the Matern kernel function with or without a noise.

Objects from the Class

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

Slots

`num_obs`**:** object of class `integer`. The number of experimental observations.

`have_noise`**:** object of class `logical` to specify whether the the model has a noise or not. "TRUE" means the model contains a noise and "FALSE" means the model does not contain a noise.

`kernel_type`**:** a `character` to specify the type of kernel to use.The current version supports kernel_type to be "matern_5_2" or "exp", meaning that the matern kernel with roughness parameter being 2.5 or 0.5 (exponent kernel), respectively.

`input`**:** object of class `vector` with dimension num_obs x 1 for the sorted input locations.

`delta_x`**:** object of class `vector` with dimension (num_obs-1) x 1 for the differences between the sorted input locations.

`output`**:** object of class `vector` with dimension num_obs x 1 for the observations at the sorted input locations.

Methods

show Prints the main slots of the object.

predict See `predict`.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

See Also

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

`log_lik` *Natural logarithm of profile likelihood by the fast computing algorithm*

Description

This function computes the natural logarithm of the profile likelihood for the range and nugget parameter (if there is one) after plugging the closed form maximum likelihood estimator for the variance parameter.

Usage

```
log_lik(param, object)
```

Arguments

`param`	a vector of parameters. The first parameter is the natural logarithm of the inverse range parameter in the kernel function. If the data contain noise, the second parameter is the logarithm of the nugget-variance ratio parameter.
`object`	an object of class `fgasp`.

Value

The numerical value of natural logarithm of the profile likelihood.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

Examples

```
library(FastGaSP)
#--------------------------------------------------------------------------
# Example 1: comparing the fast and slow algorithms to compute the likelihood
# of the Gaussian stochastic process for data with noises
#--------------------------------------------------------------------------

y_R<-function(x){
  sin(2*pi*x)
}

###let's test for 1000 observations
set.seed(1)
num_obs=1000
input=runif(num_obs)
output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)

##constucting the fgasp.model
fgasp.model=fgasp(input, output)

##range and noise-variance ratio (nugget) parameters
param=c( log(1),log(.02))
## the log lik
log_lik(param,fgasp.model)
##time cost to compute the likelihood
time_cost=system.time(log_lik(param,fgasp.model))
time_cost[1]

##now I am comparing to the one with straightforward inversion

matern_5_2_kernel<-function(d,beta){
  res=(1+sqrt(5)*beta*d + 5*beta^2*d^2/3 )*exp(-sqrt(5)*beta*d)
  res
}

##A function for computing the likelihood by the GaSP in a straightforward way
log_lik_GaSP_slow<-function(param,have_noise=TRUE,input,output){
  n=length(output)
  beta=exp(param[1])
  eta=0
  if(have_noise){
```

```
    eta=exp(param[2])
  }
  R00=abs(outer(input,input,'-'))
  R=matern_5_2_kernel(R00,beta=beta)
  R_tilde=R+eta*diag(n)
  #use Cholesky and one backsolver and one forward solver so it is more stable
  L=t(chol(R_tilde))
  output_t_R.inv= t(backsolve(t(L),forwardsolve(L,output )))
  S_2=output_t_R.inv%*%output

  -sum(log(diag(L)))-n/2*log(S_2)
}



##range and noise-variance ratio (nugget) parameters
param=c( log(1),log(.02))
## the log lik
log_lik(param,fgasp.model)
log_lik_GaSP_slow(param,have_noise=TRUE,input=input,output=output)

##time cost to compute the likelihood
##More number of inputs mean larger differences
time_cost=system.time(log_lik(param,fgasp.model))
time_cost

time_cost_slow=system.time(log_lik_GaSP_slow(param,have_noise=TRUE,input=input,output=output))
time_cost_slow


#-------------------------------------------------------------------------
# Example 2: comparing the fast and slow algorithms to compute the likelihood
# of the Gaussian stochastic process for data without a noise
#-------------------------------------------------------------------------
## Here is a function in the Sobolev Space with order 3
y_R<-function(x){
  j_seq=seq(1,200,1)
  record_y_R=0
  for(i_j in 1:200){
    record_y_R=record_y_R+2*j_seq[i_j]^{-2*3}*sin(j_seq[i_j])*cos(pi*(j_seq[i_j]-0.5)*x)

  }
  record_y_R
}


##generate some data without noise
num_obs=50
input=seq(0,1,1/(num_obs-1))

output=y_R(input)
```

```
##constucting the fgasp.model
fgasp.model=fgasp(input, output,have_noise=FALSE)

##range and noise-variance ratio (nugget) parameters
param=c( log(1))
## the log lik
log_lik(param,fgasp.model)
log_lik_GaSP_slow(param,have_noise=FALSE,input=input,output=output)
```

`predict` *Prediction and uncertainty quantification on the testing input using a GaSP model.*

Description

This function computes the predictive mean and variance on the given testing input using a GaSP model.

Usage

```
## S4 method for signature 'fgasp'
predict(param,object, testing_input, var_data=TRUE, sigma_2=NULL)
```

Arguments

`param`	a vector of parameters. The first parameter is the natural logarithm of the inverse range parameter in the kernel function. If the data contain noise, the second parameter is the logarithm of the nugget-variance ratio parameter.
`object`	an object of class `fgasp`.
`testing_input`	a vector of testing input for prediction.
`var_data`	a bool valueto decide whether the noise of the data is included for computing the posterior predictive variance.
`sigma_2`	a numerical value specifying the variance of the kernel function. If given, the package uses this parameter for prediction.

Value

The returned value is a S4 Class `predictobj.fgasp`.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

Examples

```
library(FastGaSP)

#--------------------------------------------------------------------------------
# Example 1: a simple example with noise to show fast computation algorithm
#--------------------------------------------------------------------------------

y_R<-function(x){
  cos(2*pi*x)
}

###let's test for 2000 observations
set.seed(1)
num_obs=2000
input=runif(num_obs)

output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)

##constucting the fgasp.model
fgasp.model=fgasp(input, output)

##range and noise-variance ratio (nugget) parameters
param=c( log(1),log(.02))
## the log lik
log_lik(param,fgasp.model)
##time cost to compute the likelihood
time_cost=system.time(log_lik(param,fgasp.model))
time_cost[1]

##consider a nonparametric regression setting
##estimate the parameter by maximum likelihood estimation

est_all<-optim(c(log(1),log(.02)),log_lik,object=fgasp.model,method="L-BFGS-B",
               control = list(fnscale=-1))

##estimated log inverse range parameter and log nugget
est_all$par

##estimate variance
est.var=Get_log_det_S2(est_all$par,fgasp.model@have_noise,fgasp.model@delta_x,
```

```
                    fgasp.model@output,fgasp.model@kernel_type)[[2]]/fgasp.model@num_obs
est.var

###1. Do some interpolation test
num_test=5000
testing_input=runif(num_test) ##there are the input where you don't have observations
pred.model=predict(param=est_all$par,object=fgasp.model,testing_input=testing_input)

lb=pred.model@mean+qnorm(0.025)*sqrt(pred.model@var)
ub=pred.model@mean+qnorm(0.975)*sqrt(pred.model@var)

## calculate lb for the mean function
pred.model2=predict(param=est_all$par,object=fgasp.model,testing_input=testing_input,var_data=FALSE)
lb_mean_funct=pred.model2@mean+qnorm(0.025)*sqrt(pred.model2@var)
ub_mean_funct=pred.model2@mean+qnorm(0.975)*sqrt(pred.model2@var)

## plot the prediction
min_val=min(lb,output)
max_val=max(ub,output)

plot(pred.model@testing_input,pred.model@mean,type='l',col='blue',
     ylim=c(min_val,max_val),
     xlab='x',ylab='y')
polygon( c(pred.model@testing_input,rev(pred.model@testing_input)),
    c(lb,rev(ub)),col = "grey80", border = FALSE)
lines(pred.model@testing_input,pred.model@mean,type='l',col='blue')
lines(pred.model@testing_input,y_R(pred.model@testing_input),type='l',col='black')
lines(pred.model2@testing_input,lb_mean_funct,col='blue',lty=2)
lines(pred.model2@testing_input,ub_mean_funct,col='blue',lty=2)
lines(input,output,type='p',pch=16,col='black',cex=0.4) #one can plot data

legend("bottomleft", legend=c("predictive mean","95% predictive interval","truth"),
       col=c("blue","blue","black"), lty=c(1,2,1), cex=.8)


##mean square error for all inputs
mean((pred.model@mean- y_R(pred.model@testing_input))^2)
##coverage for the mean
length(which(y_R(pred.model@testing_input)>lb_mean_funct &
               y_R(pred.model@testing_input)<ub_mean_funct))/pred.model@num_testing
##average length of the invertal for the mean
mean(abs(ub_mean_funct-lb_mean_funct))
##average length of the invertal for the data
mean(abs(ub-lb))

#-------------------------------------------------------------------------------
# Example 2: numerical comparison with the GaSP by inverting the covariance matrix
#-------------------------------------------------------------------------------
##matern correlation with smoothness parameter being 2.5
matern_5_2_kernel<-function(d,beta){
  res=(1+sqrt(5)*beta*d + 5*beta^2*d^2/3 )*exp(-sqrt(5)*beta*d)
  res
}
```

```
##A function for computing the likelihood by the GaSP in a straightforward way
log_lik_GaSP_slow<-function(param,have_noise=TRUE,input,output){
  n=length(output)
  beta=exp(param[1])
  eta=0
  if(have_noise){
    eta=exp(param[2])
  }
  R00=abs(outer(input,input,'-'))
  R=matern_5_2_kernel(R00,beta=beta)
  R_tilde=R+eta*diag(n)
  #use Cholesky and one backsolver and one forward solver so it is more stable
  L=t(chol(R_tilde))
  output_t_R.inv= t(backsolve(t(L),forwardsolve(L,output )))
  S_2=output_t_R.inv%*%output

  -sum(log(diag(L)))-n/2*log(S_2)
}


pred_GaSP_slow<-function(param,have_noise=TRUE,input,output,testing_input){
  beta=exp(param[1])
  R00=abs(outer(input,input,'-'))
  eta=0
  if(have_noise){
    eta=exp(param[2])
  }
  R=matern_5_2_kernel(R00,beta=beta)
  R_tilde=R+eta*diag(length(output))

  ##I invert it here but one can still use cholesky to make it more stable
  R_tilde_inv=solve(R_tilde)

  r0=abs(outer(input,testing_input,'-'))
  r=matern_5_2_kernel(r0,beta=beta)

  S_2=t(output)%*%(R_tilde_inv%*%output)

  sigma_2_hat=as.numeric(S_2/num_obs)

  pred_mean=t(r)%*%(R_tilde_inv%*%output)
  pred_var=rep(0,length(testing_input))


  for(i in 1:length(testing_input)){
    pred_var[i]=-t(r[,i])%*%R_tilde_inv%*%r[,i]
  }
  pred_var=pred_var+1+eta
  list=list()
  list$mean=pred_mean
  list$var=pred_var*sigma_2_hat
  list
```

```
}

##let's test sin function
y_R<-function(x){
  sin(2*pi*x)
}


###let's test for 800 observations
set.seed(1)
num_obs=800
input=runif(num_obs)
output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)


##constucting the fgasp.model
fgasp.model=fgasp(input, output)

##range and noise-variance ratio (nugget) parameters
param=c( log(1),log(.02))
## the log lik
log_lik(param,fgasp.model)
log_lik_GaSP_slow(param,have_noise=TRUE,input=input,output=output)

##time cost to compute the likelihood
##More number of inputs mean larger differences
time_cost=system.time(log_lik(param,fgasp.model))
time_cost

time_cost_slow=system.time(log_lik_GaSP_slow(param,have_noise=TRUE,input=input,output=output))
time_cost_slow

est_all<-optim(c(log(1),log(.02)),log_lik,object=fgasp.model,method="L-BFGS-B",
              control = list(fnscale=-1))
##estimated log inverse range parameter and log nugget
est_all$par


##Do some interpolation test for comparison
num_test=200
testing_input=runif(num_test) ##there are the input where you don't have observations

##one may sort the testing_input or not, and the prediction will be on the sorted testing_input
##testing_input=sort(testing_input)

## two ways of prediction
pred.model=predict(param=est_all$par,object=fgasp.model,testing_input=testing_input)
pred_slow=pred_GaSP_slow(param=est_all$par,have_noise=TRUE,input,output,sort(testing_input))
## look at the difference
max(abs(pred.model@mean-pred_slow$mean))
max(abs(pred.model@var-pred_slow$var))
```

predictobj.fgasp-class

Predictive results for the Fast GaSP class

Description

S4 class for prediction for a Fast GaSP model with or without a noise.

Objects from the Class

Objects of this class are created and initialized with the function `predict` that computes the prediction and the uncertainty quantification.

Slots

`num_testing`: object of class `integer`. Number of testing inputs.

`testing_input`: object of class `vector`. The testing input locations.

param a vector of parameters. The first parameter is the natural logarithm of the inverse range parameter in the kernel function. If the data contain noise, the second parameter is the logarithm of the nugget-variance ratio parameter.

`mean`: object of class `vector`. The predictive mean at testing inputs.

`var`: object of class `vector`. The predictive variance at testing inputs. If the `var_data` is true, the predictive variance of the data is calculated. Otherwise, the predictive variance of the mean is calculated.

`var_data`: object of class `logical`. If the `var_data` is true, the predictive variance of the data is calculated for `var`. Otherwise, the predictive variance of the mean is calculated for `var`.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

See Also

`predict.fgasp` for more details about how to do prediction for a `fgasp` object.

show *Show an* fgasp *object.*

Description

Function to print the codefgasp object.

Usage

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

Arguments

object	an object of class fgasp.

Author(s)

Mengyang Gu [aut, cre]

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

References

Hartikainen, J. and Sarkka, S. (2010). *Kalman filtering and smoothing solutions to temporal gaussian process regression models*, *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop*, 379-384.

M. Gu, Y. Xu (2017), *Nonseparable Gaussian stochastic process: a unified view and computational strategy*, arXiv:1711.11501.

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

Examples

```
#-------------------------------------
# Example: a simple example with noise
#-------------------------------------

y_R<-function(x){
  cos(2*pi*x)
}

###let's test for 2000 observations
set.seed(1)
num_obs=2000
input=runif(num_obs)

output=y_R(input)+rnorm(num_obs,mean=0,sd=0.1)
```

```
##constucting the fgasp.model
fgasp.model=fgasp(input, output)
show(fgasp.model)
```

Index