Package 'FastGaSP'

June 6, 2025

Type Package

Title Fast and Exact Computation of Gaussian Stochastic Process

Version 0.6.1

Date 2025-06-04

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

Author Mengyang Gu [aut, cre],
Xinyi Fang [aut],
Yizi Lin [aut]

Description
Implements fast and exact computation of Gaussian stochastic process with the Matern kernel using forward filtering and backward smoothing algorithm. It includes efficient implementations of the inverse Kalman filter, with applications such as estimating particle interaction functions. These tools support models with or without noise. Additionally, the package offers algorithms for fast parameter estimation in latent factor models, where the factor loading matrix is orthogonal, and latent processes are modeled by Gaussian processes. See the references: 1) Mengyang Gu and Yanxun Xu (2020), Journal of Computational and Graphical Statistics; 2) Xinyi Fang and Mengyang Gu (2024), <doi:10.48550/arXiv.2407.10089>; 3) Mengyang Gu and Weining Shen (2020), Journal of Machine Learning Research; 4) Yizi Lin, Xubo Liu, Paul Segall and Mengyang Gu (2025), <doi:10.48550/arXiv.2501.01324>.

License GPL (>= 2)

Depends methods

Imports Rcpp,rstiefel

LinkingTo Rcpp, RcppEigen

NeedsCompilation yes

Repository CRAN

RoxygenNote 7.3.2

Date/Publication 2025-06-06 00:10:02 UTC

Contents

`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 includes efficient implementations of the inverse Kalman filter, with applications such as estimating particle interaction functions. These tools support models with or without noise. Additionally, the package offers algorithms for fast parameter estimation in latent factor models, where the factor loading matrix is orthogonal, and latent processes are modeled by Gaussian processes. See the references: 1) Mengyang Gu and Yanxun Xu (2020), Journal of Computational and Graphical Statistics; 2) Xinyi Fang and Mengyang Gu (2024), <doi:10.48550/arXiv.2407.10089>; 3) Mengyang Gu and Weining Shen (2020), Journal of Machine Learning Research; 4) Yizi Lin, Xubo Liu, Paul Segall and Mengyang Gu (2025), <doi:10.48550/arXiv.2501.01324>.

Details

The DESCRIPTION file:

Package:	FastGaSP
Type:	Package
Title:	Fast and Exact Computation of Gaussian Stochastic Process
Version:	0.6.1
Date:	2025-06-04
Authors@R:	c(person(given="Mengyang", family="Gu", role=c("aut", "cre"), email="mengyang@pstat.ucsb.edu"), p
Maintainer:	Mengyang Gu <mengyang@pstat.ucsb.edu>
Author:	Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]
Description:	Implements fast and exact computation of Gaussian stochastic process with the Matern kernel using forv
License:	GPL (>= 2)
Depends:	methods
Imports:	Rcpp,rstiefel
LinkingTo:	Rcpp, RcppEigen
NeedsCompilation:	yes
Repository:	CRAN
Packaged:	2025-06-05 23:08:41 UTC; xinyifang
RoxygenNote:	7.3.2
Date/Publication:	2025-02-12 12:20:02 UTC

Index of help topics:

```
                        the testing input using a GaSP model.
predict.fmou            Prediction and uncertainty quantification on
                        the future observations using a FMOU model.
predict.gppca           Prediction and uncertainty quantification on
                        the future observations using GPPCA.
predictobj.fgasp-class
                        Predictive results for the Fast GaSP class
show,fgasp-method       Show an 'fgasp' object.
show,particle.est-method
                        Show method for particle estimation class
show.particle.data      Show method for particle data class
simulate_particle       Simulate particle trajectories
trajectory_data         Convert experimental particle tracking data to
                        particle.data object
```

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

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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)
```

`extract_time_window` *Extract time window from particle data*

Description

Extracts a specified time window from a particle.data object while preserving all relevant tracking information and parameters. Works with both simulation and experimental data.

Usage

```
extract_time_window(data_obj, first_frame, last_frame)
```

Arguments

`data_obj`	An object of class `particle.data`.
`first_frame`	Integer specifying the first frame to include (must be >= 1).
`last_frame`	Integer specifying the last frame to include (must be less than the total number of frames).

Value

Returns a new `particle.data` object containing:

px_list, py_list Position data for the selected time window

vx_list, vy_list Velocity data for the selected time window

theta_list Angle data if present in original object

particle_tracking Tracking information for the selected frames (experimental data)

data_type Original data type ("simulation" or "experiment")

n_particles Particle counts (constant for simulation, time series for experimental)

T_time Number of time steps in the extracted window

model, sigma_0, radius Original model parameters (for simulation data)

D_y Original dimension of the output space

`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], Xinyi Fang [aut], Yizi Lin [aut]

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], Xinyi Fang [aut], Yizi Lin [aut]

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.

`fit` *Fit Particle Interaction Models*

Description

Generic function for fitting particle interaction models to data.

Usage

```
fit(object, ...)
```

Arguments

`object`	Object containing data to be fit
`...`	Additional arguments passed to fitting methods

See Also

`fit.particle.data` for fitting particle data models

`fit.fmou` *The fast EM algorithm of multivariate Ornstein-Uhlenbeck processes*

Description

This function implements an efficient EM algorithm to estimate the parameters in the FMOU model, a latent factor model with a fixed or estimated orthogonal factor loading matrix, where each latent factor is modeled as an O-U (Ornstein-Uhlenbeck) process.

Usage

```
## S4 method for signature 'fmou'
fit.fmou(object, M=50, threshold=1e-4,
         track_iterations=FALSE,track_neg_log_lik=FALSE,
         U_init=NULL, rho_init=NULL, sigma2_init=NULL, d_ub=NULL)
```

Arguments

object	an objecft of class fmou.
M	number of iterations in the EM algorithm, default is 50.
threshold	stopping criteria with respect to predictive mean of observations, default is 1e-4.
track_iterations	a bool value, default is FALSE. If TRUE, the estimations in each EM iteration will be recorded and returned.
track_neg_log_lik	a bool value, default is FALSE. If TRUE, the negative log marginal likelihood of the output in each EM iteration will be recorded and returned.
U_init	user-specified initial factor loading matrix in the EM algorithm. Default is NULL. The dimension is k*d, where k is the length of observations at each time step and d is the number of latent factors.
rho_init	user-specified initial correlation parameters in the EM algorithm. Default is NULL. The length is equal to the number of latent factors.
sigma2_init	user-specified initial variance parameters in the EM algorithm. Default is NULL. The length is equal to the number of latent factors.
d_ub	upper bound of d when d is estimated. Default is null.

Value

output	the observation matrix.
U	the estimated (or fixed) factor loading matrix.
post_z_mean	the posterior mean of latent factors.
post_z_var	the posterior variance of latent factors.
post_z_cov	the posterior covariance between two consecutive time steps of a latent process.
mean_obs	the predictive mean of the observations.
mean_obs_95lb	the lower bound of the 95% posterior credible intervals of predictive mean.
mean_obs_95ub	the upper bound of the 95% posterior credible intervals of predictive mean.
sigma0_2	estimated variance of noise.
rho	estimated correlation parameters.
sigma2	estimated variance parameters
num_iterations	number of iterations in the EM algorithm.
d	the estimated (or fixed) number of latent factors.
record_sigma0_2	estimated variance of noise in each iteration.
record_rho	estimated correlation parameters in each iteration.
record_sigma2	estimation variance parameters in each iteration.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Lin, Y., Liu, X., Segall, P., & Gu, M. (2025). Fast data inversion for high-dimensional dynamical systems from noisy measurements. arXiv preprint arXiv:2501.01324.

Examples

```
## generate simulated data
library(FastGaSP)
library(rstiefel)

d = 5  # number of latent factors
k = 20 # length of observation at each time step
n = 500 # number time step
noise_level = 1 # variance of noise

U = rustiefel(k, k) # factor loading matrix
z = matrix(NA, d, n)
sigma_2 = runif(d, 0.5, 1)
rho = runif(d, 0.95, 1)
for(l in 1:d){
  R = matrix(NA, n, n)
  diag(R) = 1
  for(ir in 1:n){
    for(ic in 1:n){
      R[ir, ic] = rho[l]^(abs(ir-ic)) * R[ir, ir]
    }
  }
  R = (sigma_2[l]/(1-rho[l]^2) )* R
  z[l, ] = t(chol(R)) %*% rnorm(n)
}

signal = U[,1:d] %*% z
y = signal + matrix(rnorm(n*k,mean=0,sd=sqrt(noise_level)),k,n)

##constucting the fmou.model
fmou.model=fmou(output=y, d=d, est_U0=TRUE, est_sigma0_2=TRUE)

## estimate the parameters
em_alg <- fit.fmou(fmou.model, M=500)

## root mean square error (RMSE) of predictive mean of observations
sqrt(mean((em_alg$mean_obs-signal)^2))

## standard deviation of (truth) mean of observations
sd(signal)
```

```
## estimated variance of noise
em_alg$sigma0_2
```

`fit.gppca` *Parameter estimation for generalized probabilistic principal component analysis of correlated data.*

Description

This function estimates the parameters for generalized probabilistic principal component analysis of correlated data.

Usage

```
## S4 method for signature 'gppca'
fit.gppca(object, sigma0_2=NULL, d_ub=NULL)
```

Arguments

`object`	an object of class `gppca`.
`sigma0_2`	variance of noise. Default is `NULL`. User should specify a value when it is known in real data.
`d_ub`	upper bound of d when d is estimated. Default is `NULL`.

Value

`est_A`	the estimated factor loading matrix.
`est_beta`	the estimated inversed range parameter.
`est_sigma0_2`	the estimated variance of noise.
`est_sigma2`	the estimated variance parameter.
`mean_obs`	the predictive mean of the observations.
`mean_obs_95lb`	the lower bound of the 95% posterior credible intervals of predictive mean.
`mean_obs_95ub`	the upper bound of the 95% posterior credible intervals of predictive mean.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Gu, M., & Shen, W. (2020), Generalized probabilistic principal component analysis of correlated data, *Journal of Machine Learning Research, 21*(13), 1-41.

Examples

```
library(FastGaSP)
library(rstiefel)

matern_5_2_funct <- function(d, beta_i) {
  cnst <- sqrt(5.0)
  matOnes <- matrix(1, nrow = nrow(d), ncol = ncol(d))
  result <- cnst * beta_i * d
  res <- (matOnes + result + (result^2) / 3) * exp(-result)
  return(res)
}

n=200
k=8
d=4

beta_real=0.01
sigma_real=1
sigma_0_real=sqrt(.01)
input=seq(1,n,1)
R0_00=as.matrix(abs(outer(input,input,'-')))
R_r = matern_5_2_funct(R0_00, beta_real)
L_sample = t(chol(R_r))
kernel_type='matern_5_2'


input=sort(input)
delta_x=input[2:length(input)]-input[1:(length(input)-1)]
A=rustiefel(k, d)  ##sample from Stiefel manifold
Factor=matrix(0,d,n)
for(i in 1: d){
  Factor[i,]=sigma_real^2*L_sample%*%rnorm(n)
}
output=A%*%Factor+matrix(rnorm(n*k,mean=0,sd=sigma_0_real),k,n)

##constucting the gppca.model
gppca_obj <- gppca(input, output, d, shared_params = TRUE,est_d=FALSE)
## estimate the parameters
gppca_fit <- fit.gppca(gppca_obj)

## MSE between predictive mean of observations and true mean
Y_mean=A%*%Factor
mean((gppca_fit$mean_obs-Y_mean)^2)
sd(Y_mean)

## coverage of 95% posterior credible intervals
sum(gppca_fit$mean_obs_95lb<=Y_mean & gppca_fit$mean_obs_95ub>=Y_mean)/(n*k)

## length of 95% posterior credible intervals
mean(abs(gppca_fit$mean_obs_95ub-gppca_fit$mean_obs_95lb))
```

`fit.particle.data` *Fit method for particle data*

Description

Estimates interaction parameters for particle systems using trajectory data with the IKF-CG (Inverse Kalman Filter - Conjugate Gradient) approach. Supports both simulation and experimental data.

Usage

```
  ## S4 method for signature 'particle.data'
fit(
    object, param, cut_r_max=1, est_param = TRUE, nx=NULL, ny=NULL,
    kernel_type = "matern_5_2", tilde_nu = 0.1, tol = 1e-6,
    maxIte = 1000, output = NULL, ho_output = NULL,
   testing_inputs=NULL, compute_CI = TRUE, num_interaction = (length(param)-1)/2,
    data_type = object@data_type, model = object@model,
    apolar_vicsek = FALSE, direction = NULL
  )
```

Arguments

`object`	An object of class `particle.data` containing the trajectory data.
`param`	Numeric vector of parameters. Should contain 2*num_interaction + 1 elements: first num_interaction elements are log of inverse range parameters (beta), next num_interaction elements are log of variance-noise ratios (tau), and the final element is log(radius/(cut_r_max-radius)) where radius is the interaction radius.
`cut_r_max`	Numeric value specifying the maximum interaction radius to consider during estimation (default: 1).
`est_param`	If TRUE, param is used as initial values for parameter optimization. If FALSE, param is treated as fixed parameters for prediction (default: TRUE).
`nx`	An integer specifying the number of grid points along the x-axis (horizontal direction). If NULL, automatically calculated as floor((px_max-px_min)/cut_r_max), where px_max and px_min represent the maximum and minimum x-coordinates of all particles.
`ny`	An integer specifying the number of grid points along the y-axis (vertical direction). If NULL, automatically calculated as floor((py_max-py_min)/cut_r_max), where py_max and py_min represent the maximum and minimum y-coordinates of all particles.
`kernel_type`	Character string specifying the kernel type: 'matern_5_2' (default) or 'exp'.
`tilde_nu`	Numeric value for stabilizing the IKF computation (default: 0.1).
`tol`	Numeric value specifying convergence tolerance for the conjugate gradient algorithm (default: 1e-6).
`maxIte`	Integer specifying maximum iterations for the conjugate gradient algorithm (default: 1000).

output	Numerical vector (default = NULL). Used for residual bootstrap when different outputs but same inputs are needed.
ho_output	Numerical vector (default = NULL). Used for residual bootstrap when different hold-out outputs but same inputs are needed.
testing_inputs	Matrix of inputs for prediction (NULL if only performing parameter estimation). Each column represents testing inputs for one interaction.
compute_CI	When TRUE, computes the 95% credible interval for testing_inputs (default: TRUE).
num_interaction	Integer specifying number of interactions to predict (default: (length(param_ini)-1)/2).
data_type	Character string indicating data type ("simulation" or "experiment").
model	Character string specifying the model type (e.g., "unnormalized_Vicsek").
apolar_vicsek	When TRUE, considers only neighboring particles moving in the same direction (default: FALSE).
direction	Modeling direction ('x' or 'y') for experimental data analysis.

Value

Returns an object of class particle.est. See particle.est-class for details.

References

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

Examples

```
# Simulate data
vx_abs <- 0.5
vy_abs <- 0.5
v_abs <- sqrt(vx_abs^2+vy_abs^2)
sim <- simulate_particle(v_abs=v_abs, model = 'unnormalized_Vicsek')
show(sim)

# Set up testing inputs and initial parameters
testing_n <- 200
testing_inputs <- matrix(as.numeric(seq(-1,1,length.out=testing_n)),nr=1)
cut_r_max=1.5
param_ini <- log(c(0.3,1000,0.3/(cut_r_max-0.3)))  # Initial parameter values

# Fit model to simulation data
est <- fit(sim,param=param_ini,cut_r_max=1.5, testing_inputs = testing_inputs)
show(est)
```

fmou *Setting up the FMOU model*

Description

Creating an fmou class for fmou, a latent factor model with a fixed or estimated orthogonal factor loading matrix, where each latent factor is modeled as an O-U (Ornstein-Uhlenbeck) process.

Usage

```
fmou(output, d, est_d=FALSE, est_U0=TRUE, est_sigma0_2=TRUE, U0=NULL, sigma0_2=NULL)
```

Arguments

output	a k*n observation matrix, where k is the length of observations at each time step and n is the number of time steps.
d	number of latent factors.
est_d	a bool value, default is FALSE. If TRUE, d will be estimated by either variance matching (when noise level is given) or information criteria (when noise level is unknown). Otherwise, d is fixed, and users must assign a value to argument d.
est_U0	a bool value, default is TRUE. If TRUE, the factor loading matrix (U0) will be estimated. Otherwise, U0 is fixed.
est_sigma0_2	a bool value, default is TRUE . If TRUE, the variance of the noise will be estimated. Otherwise, it is fixed.
U0	the fixed factor loading matrix. Users should assign a k*d matrix to it when est_U0=FALSE.
sigma0_2	variance of noise. User should assign a value to it when est_sigma0_2=FALSE.

Value

fmou returns an S4 object of class fmou.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Lin, Y., Liu, X., Segall, P., & Gu, M. (2025). Fast data inversion for high-dimensional dynamical systems from noisy measurements. arXiv preprint arXiv:2501.01324.

Examples

```
## generate simulated data
library(FastGaSP)
library(rstiefel)

d = 5  # number of latent factors
k = 20 # length of observation at each time step
n = 100 # number time step
noise_level = 1 # variance of noise

U = rustiefel(k, k) # factor loading matrix
z = matrix(NA, d, n)
sigma_2 = runif(d, 0.5, 1)
rho = runif(d, 0.95, 1)
for(l in 1:d){
  R = matrix(NA, n, n)
  diag(R) = 1
  for(ir in 1:n){
    for(ic in 1:n){
      R[ir, ic] = rho[l]^(abs(ir-ic)) * R[ir, ir]
    }
  }
  R = (sigma_2[l]/(1-rho[l]^2) )* R
  z[l, ] = t(chol(R)) %*% rnorm(n)
}

signal = U[,1:d] %*% z
y = signal + matrix(rnorm(n*k,mean=0,sd=sqrt(noise_level)),k,n)

##constucting the fmou.model
fmou.model=fmou(output=y, d=d, est_U0=TRUE, est_sigma0_2=TRUE)
```

`fmou-class` *FMOU class*

Description

An S4 class for fast parameter estimation in the FMOU model, a latent factor model with a fixed or estimated orthogonal factor loading matrix, where each latent factor is modeled as an O-U (Ornstein-Uhlenbeck) process.

Objects from the Class

Objects of this class are created and initialized using the `fmou` function to set up the estimation.

Slots

`output:` object of class `matrix`. The observation matrix.

`d` object of class `integer` to specify the number of latent factors.

`est_d` object of class `logical`, default is `FALSE`. If `TRUE`, d will be estimated by either variance matching (when noise level is given) or information criteria (when noise level is unknown). Otherwise, d is fixed, and users must assign a value to `d`.

`est_U0` object of class `logical`, default is `TRUE`. If `TRUE`, the factor loading matrix (U0) will be estimated. Otherwise, U0 is fixed.

`est_sigma0_2` object of class `logical`, default is `TRUE` . If `TRUE`, the variance of the noise will be estimated. Otherwise, it is fixed.

`U0` object of class `matrix`. The fixed factor loading matrix. Users should assign a k*d matrix to it when `est_U0=False`. Here k is the length of observations at each time step.

`sigma0_2` object of class `numeric`. Variance of noise. User should assign a value to it when `est_sigma0_2=False`.

Methods

fit.fmou See `fit.fmou`.

predict.fmou See `predict.fmou`.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Lin, Y., Liu, X., Segall, P., & Gu, M. (2025). Fast data inversion for high-dimensional dynamical systems from noisy measurements. arXiv preprint arXiv:2501.01324.

See Also

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

---

gppca *Setting up the GPPCA model*

Description

Creating an `gppca` class for generalized probabilistic principal component analysis of correlated data.

Usage

```
gppca(input,output, d, est_d=FALSE, shared_params=TRUE, kernel_type="matern_5_2")
```

Arguments

input	a vector for the sorted input locations. The length is equal to the number of observations.
output	a k*d matrix for the observations at the sorted input locations. Here k is the number of locations and n is the number of observations.
d	number of latent factors.
est_d	a bool value, default is FALSE. If TRUE, d will be estimated by either variance matching (when noise level is given) or information criteria (when noise level is unknown). Otherwise, d is fixed, and users must assign a value to argument d.
shared_params	a bool value, default is TRUE. If TRUE, the latent processes share the correlation and variance parameters. Otherwise, each latent process has distinct parameters.
kernel_type	a character to specify the type of kernel to use. The current version supports kernel_type to be "matern_5_2" or "exponential", meaning that the matern kernel with roughness parameter being 2.5 or 0.5 (exponent kernel), respectively.

Value

gppca returns an S4 object of class gppca.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut] Maintainer: Mengyang Gu <mengyang@pstat.ucsb.edu>

References

Gu, M., & Shen, W. (2020), Generalized probabilistic principal component analysis of correlated data, *Journal of Machine Learning Research, 21(13)*, 1-41.

Examples

```
library(FastGaSP)
library(rstiefel)

matern_5_2_funct <- function(d, beta_i) {
  cnst <- sqrt(5.0)
  matOnes <- matrix(1, nrow = nrow(d), ncol = ncol(d))
  result <- cnst * beta_i * d
  res <- (matOnes + result + (result^2) / 3) * exp(-result)
  return(res)
}

n=200
k=8
d=4

beta_real=0.01
sigma_real=1
sigma_0_real=sqrt(.01)
input=seq(1,n,1)
```

```
R0_00=as.matrix(abs(outer(input,input,'-')))
R_r = matern_5_2_funct(R0_00, beta_real)
L_sample = t(chol(R_r))
kernel_type='matern_5_2'


input=sort(input)
delta_x=input[2:length(input)]-input[1:(length(input)-1)]
A=rustiefel(k, d)  ##sample from Stiefel manifold
Factor=matrix(0,d,n)
for(i in 1: d){
  Factor[i,]=sigma_real^2*L_sample%*%rnorm(n)
}
output=A

##constucting the gppca.model
gppca_obj <- gppca(input, output, d, shared_params = TRUE,est_d=FALSE)
```

---

gppca-class *GPPCA class*

Description

An S4 class for generalized probabilistic principal component analysis of correlated data.

Objects from the Class

Objects of this class are created and initialized using the gppca function to set up the estimation.

Slots

`input`**:** object of class `vector`, the length is equivalent to the number of observations.

`output`**:** object of class `matrix`. The observation matrix.

`d`**:** object of class `integer` to specify the number of latent factors.

`est_d`**:** object of class `logical`, default is `FALSE`. If `TRUE`, d will be estimated by either variance matching (when noise level is given) or information criteria (when noise level is unknown). Otherwise, d is fixed, and users must assign a value to `d`.

`shared_params`**:** object of class `logical`, default is `TRUE`. If `TRUE`, the latent processes share the correlation and variance parameters. Otherwise, each latent process has distinct parameters.

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

Methods

fit.gppca See `fit.gppca` for details.

predict.gppca See `predict.gppca` for details.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Gu, M., & Shen, W. (2020), Generalized probabilistic principal component analysis of correlated data, *Journal of Machine Learning Research, 21*(13), 1-41.

See Also

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

IKF *Inverse Kalman Filter - The multiplication of R with y with given kernel type*

Description

This function computes the product of the R matrix and the output vector for a given kernel type, where R is the correlation matrix for a dynamic linear model (DLM). Instead of explicitly forming the Cholesky decomposition of R, this function computes the product as $L(L^T y)$, where L is the Cholesky decomposition of R. This is achieved using the forward filtering algorithm for efficient computation.

Usage

```
IKF(beta, tilde_nu, delta_x, output, kernel_type='matern_5_2')
```

Arguments

beta	A scalar representing the inverse range parameter in the kernel function.
tilde_nu	A numerical value representing the nugget or the stabilizing parameter.
delta_x	A numeric vector of successive differences of inputs.
output	The output vector for which the product with R is computed.
kernel_type	A string specifying the kernel type. Must be either "matern_5_2" or "exp".

Value

A vector representing the product of the R matrix and the output vector

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

See Also

Get_R_y for computing the product of R and y with G, Q, and K matrices.

Examples

```
# Helper function for Matern 2.5 correlation
matern25_correlation <- function(d, beta) {
  #' Compute Matern 2.5 correlation matrix given distance matrix and beta parameter
  #'
  #' @param d Distance matrix or array of any dimensions
  #' @param beta Inverse range parameter
  #'
  #' @return Correlation matrix with same dimensions as d
  #'
  #' @details
  #' Formula: k(d) = (1 + sqrt(5)*beta*d + 5*beta^2*d^2/3) * exp(-sqrt(5)*beta*d)

  # Compute sqrt(5)*beta*d
  sqrt5_beta_d <- sqrt(5) * beta * d

  # Compute the correlation
  R <- (1 + sqrt5_beta_d + (5 * beta^2 * d^2) / 3) * exp(-sqrt5_beta_d)

  return(R)
}

# Example: Comparing IKF with direct matrix computation
n <- 2000                           # number of observations
input <- sort(runif(n))             # sorted input points
delta_x <- input[-1] - input[-n]    # consecutive differences
u <- rnorm(n)                       # random output vector
beta <- 10                          # inverse range parameter

# Test 1: Noise-free scenario
# Non-robust IKF (no stabilization parameter)
x_non_robust_IKF <- IKF(beta = beta, tilde_nu = 0, delta_x = delta_x,
                        output = u, kernel_type = 'matern_5_2')

# Robust IKF (with stabilization parameter)
tilde_nu <- 0.1  # stabilization parameter
x_IKF <- IKF(beta = beta, tilde_nu = tilde_nu, delta_x = delta_x,
             output = u, kernel_type = 'matern_5_2') - tilde_nu * u

# Direct matrix computation for comparison
R0 <- abs(outer(input, input, '-'))  # distance matrix
R <- matern25_correlation(R0, beta)  # correlation matrix
x_direct <- R %*% u
```

```
# Compare results (should be nearly identical)
cat("Maximum difference (non-robust IKF vs direct):",
    max(abs(x_direct - x_non_robust_IKF)), "\n")
cat("Maximum difference (robust IKF vs direct):",
    max(abs(x_direct - x_IKF)), "\n")

# Test 2: With noise
V <- 0.2  # noise variance
x_IKF_noisy <- IKF(beta = beta, tilde_nu = V, delta_x = delta_x,
                   output = u, kernel_type = 'matern_5_2')

# Direct computation with noise
x_direct_noisy <- (R + V * diag(n)) %*% u

# Compare results
cat("Maximum difference (IKF vs direct):",
    max(abs(x_direct_noisy - x_IKF_noisy)), "\n")
```

`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], Xinyi Fang [aut], Yizi Lin [aut]

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)
```

`particle.data-class` *Particle trajectory data class*

Description

S4 class for storing and analyzing particle trajectory data from both simulations and experimental observations. This class supports different models including Vicsek and can handle both position and velocity data along with optional angle information and particle tracking capabilities for experimental data.

Objects from the Class

Objects of this class can be created in two ways:

- For simulation data: Using `simulate_particle` that computes particle trajectories under physical models
- For experimental data: Using `trajectory_data` to save particle trajectories while handling varying numbers of particles between time steps

Slots

`px_list`**:** Object of class `list`. List of x-positions at each time step.

`py_list`**:** Object of class `list`. List of y-positions at each time step.

`vx_list`**:** Object of class `list`. List of x-velocities at each time step.

`vy_list`**:** Object of class `list`. List of y-velocities at each time step.

`theta_list`**:** Object of class `listOrNULL`. Optional list of particle velocity angles at each time step.

`particle_tracking`**:** Object of class `listOrNULL`. List of data frames containing particle mappings between consecutive frames (primarily for experimental data).

`data_type`**:** Object of class `character`. Type of data: either "simulation" or "experiment".

`n_particles`**:** Object of class `numeric`. Number of particles (constant for simulation data, or a vector recording the number of particles at each time step for experimental data).

`T_time`**:** Object of class `numeric`. Total number of time steps.

`D_y`**:** Object of class `numeric`. Dimension of the output space.

`model`: Object of class `characterOrNULL`. Type of particle interaction model (e.g., "Vicsek"). NULL for experimental data.

`sigma_0`: Object of class `numericOrNULL`. Noise variance parameter used in the model. NULL for experimental data.

`radius`: Object of class `numericOrNULL`. Interaction radius between particles. NULL for experimental data.

Methods

show: Method for displaying summary information about the particle.data object.

fit: Method for fitting the latent factor model to data using the IKF-CG algorithm, which returns a `particle.est` object containing estimated parameters and predictions. See `fit.particle.data` for detailed documentation.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Vicsek, T., Czirok, A., Ben-Jacob, E., Cohen, I., & Shochet, O. (1995). *Novel type of phase transition in a system of self-driven particles*, *Physical Review Letters*, **75**(6), 1226.

Chat'e, H., Ginelli, F., Gr'egoire, G., Peruani, F., & Raynaud, F. (2008). *Modeling collective motion: variations on the Vicsek model*, *The European Physical Journal B*, **64**(3), 451-456.

See Also

`simulate_particle` for simulating particle trajectories, `trajectory_data` for saving experimantal particle trajectories, `fit.particle.data` for model fitting methods

---

`particle.est-class` *Particle interaction estimation class*

Description

S4 class for storing estimated parameters and predictions for particle interaction models.

Objects from the Class

Objects of this class are created by the `fit.particle.data` (via `fit`) method when applied to `particle.data` objects to estimate interaction parameters and make predictions.

Slots

`data_type`: Object of class `character`. Specifies the type of data ("simulation" or "experiment").

`model`: Object of class `characterOrNULL`. Specifies the model type for simulation data (e.g., "Vicsek" or "two_interactions_Vicsek"). NULL for experimental data.

`D_y`: Object of class `numeric`. Dimension of the output space.

`num_interaction`: Object of class `numeric`. Number of interactions.

`parameters`: Object of class `numeric`. Vector of estimated parameters with length 2*D_y + 1:

- First D_y elements: beta (inverse range parameters)
- Next D_y elements: tau (variance-noise ratios)
- Last element: interaction radius

`sigma_2_0_est`: Object of class `numeric`. Estimated noise variance.

`predictions`: object of class `listOrNULL`. Contains predicted means and 95% confidence intervals (lower and upper bounds) for the particle interactions if testing inputs are given.

`training_data`: Object of class `list`. Contains the training data used in the GP model, obtained using the estimated interaction radius.

`gp_weights`: Object of class `matrix`. Contains the weights from the GP computation (A^T_j Sigma_y^(-1) y) used for prediction, with each column corresponding to a type of interaction j.

Methods

show: Method for displaying summary information about the estimated parameters.

References

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

See Also

`fit.particle.data` for more details about how to create a `particle.est` object. `particle.data-class` for the input data structure

---

`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], Xinyi Fang [aut], Yizi Lin [aut]

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))
```

`predict.fmou` *Prediction and uncertainty quantification on the future observations using a FMOU model.*

Description

This function predicts the future observations using a FMOU model. Uncertainty quantification is available.

Usage

```
## S4 method for signature 'fmou'
predict.fmou(object, param_est, step=1, interval=FALSE, interval_data=TRUE)
```

Arguments

`object`	an objecft of class `fmou`.
`param_est`	estimated parameters in the FMOU model. Obtained by the results of `fit.fmou()`.
`step`	a vector. Number of steps to be predicted. Default is 1.
`interval`	a bool value, default is `FALSE`. If `TRUE`, the 95% predictive intervals are computed.
`interval_data`	a bool value, default is `TRUE`. If `TRUE`, the 95% predictive intervals of the observations are computed. Otherwise, the 95% predictive intervals of the mean of the observation are computed.

Value

`pred_mean` the predictive mean.

`pred_interval_95lb` the 95% lower bound of the interval.

`pred_interval_95ub` the 95% upper bound of the interval.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Lin, Y., Liu, X., Segall, P., & Gu, M. (2025). Fast data inversion for high-dimensional dynamical systems from noisy measurements. arXiv preprint arXiv:2501.01324.

Examples

```
## generate simulated data
library(FastGaSP)
library(rstiefel)

d = 5  # number of latent factors
k = 20 # length of observation at each time step
n = 500 # number time step
noise_level = 1 # variance of noise

U = rustiefel(k, k) # factor loading matrix
z = matrix(NA, d, n)
sigma_2 = runif(d, 0.5, 1)
rho = runif(d, 0.95, 1)
for(l in 1:d){
  R = matrix(NA, n, n)
  diag(R) = 1
  for(ir in 1:n){
    for(ic in 1:n){
      R[ir, ic] = rho[l]^(abs(ir-ic)) * R[ir, ir]
    }
  }
  R = (sigma_2[l]/(1-rho[l]^2) )* R
  z[l, ] = t(chol(R)) %*% rnorm(n)
}

signal = U[,1:d] %*% z
y = signal + matrix(rnorm(n*k,mean=0,sd=sqrt(noise_level)),k,n)

##constucting the fmou.model
fmou.model=fmou(output=y, d=d, est_U0=TRUE, est_sigma0_2=TRUE)

## estimate the parameters
```

```
em_alg <- fit.fmou(fmou.model, M=500)

## two-step-ahead prediction
pred_2step <- predict.fmou(fmou.model,em_alg, step=c(1:2))
```

`predict.gppca` *Prediction and uncertainty quantification on the future observations using GPPCA.*

Description

This function predicts the future observations using a GPPCA model. Uncertainty quantification is available.

Usage

```
## S4 method for signature 'gppca'
predict.gppca(object, param, A_hat, step=1, interval=FALSE, interval_data=TRUE)
```

Arguments

`object`	an object of class `gppca`.
`param`	estimated parameters (beta, sigma2, sigma0_2).
`A_hat`	estimated factor loading matrix.
`step`	a vector. Number of steps to be predicted. Default is 1.
`interval`	a bool value, default is `FALSE`. If `TRUE`, the 95% predictive intervals are computed.
`interval_data`	a bool value, default is `TRUE`. If `TRUE`, the 95% predictive intervals of the observations are computed. Otherwise, the 95% predictive intervals of the mean of the observation are computed.

Value

`pred_mean`	the predictive mean.
`pred_interval_95lb`	the 95% lower bound of the interval.
`pred_interval_95ub`	the 95% upper bound of the interval.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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

References

Gu, M., & Shen, W. (2020), Generalized probabilistic principal component analysis of correlated data, *Journal of Machine Learning Research, 21*(13), 1-41.

Examples

```
library(FastGaSP)
library(rstiefel)

matern_5_2_funct <- function(d, beta_i) {
  cnst <- sqrt(5.0)
  matOnes <- matrix(1, nrow = nrow(d), ncol = ncol(d))
  result <- cnst * beta_i * d
  res <- (matOnes + result + (result^2) / 3) * exp(-result)
  return(res)
}

n=200
k=8
d=4

beta_real=0.01
sigma_real=1
sigma_0_real=sqrt(.01)
input=seq(1,n,1)
R0_00=as.matrix(abs(outer(input,input,'-')))
R_r = matern_5_2_funct(R0_00, beta_real)
L_sample = t(chol(R_r))
kernel_type='matern_5_2'


input=sort(input)
delta_x=input[2:length(input)]-input[1:(length(input)-1)]
A=rustiefel(k, d)  ##sample from Stiefel manifold
Factor=matrix(0,d,n)
for(i in 1: d){
  Factor[i,]=sigma_real^2*L_sample%*%rnorm(n)
}
output=A%*%Factor+matrix(rnorm(n*k,mean=0,sd=sigma_0_real),k,n)

##constucting the gppca.model
gppca_obj <- gppca(input, output, d, shared_params = TRUE,est_d=FALSE)
## estimate the parameters
gppca_fit <- fit.gppca(gppca_obj)

## two-step-ahead prediction
param <- c(gppca_fit$est_beta, gppca_fit$est_sigma2, gppca_fit$est_sigma0_2)
gppca_pred <- predict.gppca(gppca_obj, param, gppca_fit$est_A, step=1:3)
gppca_pred$pred_mean
```

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], Xinyi Fang [aut], Yizi Lin [aut]

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.fgasp *Show an* fgasp *object.*

Description

Function to print the fgasp object.

Usage

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

Arguments

object	an object of class fgasp.

Author(s)

Mengyang Gu [aut, cre], Xinyi Fang [aut], Yizi Lin [aut]

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)
```

`show.particle.data` *Show method for particle data class*

Description

Display method for objects of class `particle.data`, which prints a summary of the key parameters and characteristics of the particle system based on its data type (simulation or experimental).

Usage

```
## S4 method for signature 'particle.data'
show(object)
```

Arguments

`object`	An object of class `particle.data`.

Details

This method displays essential information about the particle system. The output varies based on the data type:

For simulation data:

- Type of particle interaction model
- Number of time steps
- Number of particles in the system
- Noise variance parameter (sigma_0)
- Interaction radius between particles

For experimental data:

- Number of time steps
- Average number of particles across all time steps

References

Vicsek, T., Czirok, A., Ben-Jacob, E., Cohen, I., & Shochet, O. (1995). *Novel type of phase transition in a system of self-driven particles*, *Physical Review Letters*, **75**(6), 1226.

Chat'e, H., Ginelli, F., Gr'egoire, G., Peruani, F., & Raynaud, F. (2008). *Modeling collective motion: variations on the Vicsek model*, *The European Physical Journal B*, **64**(3), 451-456.

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

See Also

particle.data-class for the full class description, simulate_particle for creating simulation data objects, trajectory_data for creating experimental data objects

---

show.particle.est *Show method for particle estimation class*

Description

Display method for objects of class particle.est, which prints a summary of the estimated parameters for the particle interaction model.

Usage

```
  ## S4 method for signature 'particle.est'
show(object)
```

Arguments

object	An object of class particle.est.

Details

This method displays essential information about the estimated model parameters, including:

- Data type (simulation or experiment)
- Model type (for simulation data only)
- Dimension of output space
- Estimated parameters:
  - beta parameters (inverse range)
  - tau parameters (variance-noise ratio)
  - interaction radius

References

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

See Also

particle.est-class for details of the particle estimation class

`simulate_particle` *Simulate particle trajectories*

Description

Simulates particle trajectories using either the standard Vicsek model or a two-interaction variation of the Vicsek model.

Usage

```
simulate_particle(v_abs, n_t = 100, T_sim = 5, h = 0.1,
  cut_r = 0.5, sigma_0 = 0.1,
  noise_type = "Gaussian", model = "Vicsek")
```

Arguments

`v_abs`	Absolute velocity magnitude for all particles.
`n_t`	Number of particles (default: 100).
`T_sim`	Total simulation time steps (default: 5).
`h`	Time step size for numerical integration (default: 0.1).
`cut_r`	Radius of interaction between particles (default: 0.5).
`sigma_0`	Standard deviation of noise (default: 0.1).
`noise_type`	Distribution of noise: "Gaussian" or "Uniform" (default: "Gaussian").
`model`	Type of interaction model: "Vicsek", "unnormalized_Vicsek", or "two_interactions_Vicsek" (default: "Vicsek").

Value

Returns an S4 object of class `particle.data`. See `particle.data-class` for details of the returned object structure.

References

Vicsek, T., Czirok, A., Ben-Jacob, E., Cohen, I., & Shochet, O. (1995). *Novel type of phase transition in a system of self-driven particles*, *Physical Review Letters*, **75**(6), 1226.

Chat'e, H., Ginelli, F., Gr'egoire, G., Peruani, F., & Raynaud, F. (2008). *Modeling collective motion: variations on the Vicsek model*, *The European Physical Journal B*, **64**(3), 451-456.

Fang, X., & Gu, M. (2024). *The inverse Kalman filter*. arXiv:2407.10089.

See Also

`particle.data-class` for details on the particle.data class structure

Examples

```
#-------------------------------------------------
# Example: Simulate using standard Vicsek model
#-------------------------------------------------
vx_abs=0.5
vy_abs=0.5
v_abs=sqrt(vx_abs^2+vy_abs^2)
sim1 <- simulate_particle(v_abs=v_abs)

#-------------------------------------------------
# Example: Simulate using unnormalized variation
#-------------------------------------------------
sim2 <- simulate_particle(v_abs=v_abs, model = 'unnormalized_Vicsek')

#-------------------------------------------------
# Example: Simulate using two-interaction variation
#-------------------------------------------------
sim3 <- simulate_particle(v_abs=v_abs, model = 'two_interactions_Vicsek')
```

---

trajectory_data *Convert experimental particle tracking data to particle.data object*

Description

Processes experimental particle tracking data and creates a standardized particle.data object. This function handles time series data with varying numbers of particles across time steps and maintains particle identity tracking between consecutive frames.

Usage

```
trajectory_data(particle_data)
```

Arguments

`particle_data`	A data frame containing particle tracking data with the following required columns: • `px`, `py`: Particle positions in x and y coordinates • `vx`, `vy`: Particle velocities in x and y directions • `time`: Time step identifier (integer). Must be consecutive integers starting from the minimum time value • `particleID`: Unique particle identifier for tracking across frames

Value

Returns an S4 object of class `particle.data` with:

px_list, py_list Lists of particle x and y positions at each time step

vx_list, vy_list Lists of particle x and y velocities at each time step

theta_list List of particle angles computed from velocities

particle_tracking List of data frames containing particle mappings between consecutive frames

data_type "experiment"

n_particles Vector recording number of particles at each time step

T_time Total number of time steps

D_y Dimension of the output space (set to 1)

See Also

particle.data-class for details on the particle.data class structure

Examples

```
# Create sample tracking data
sample_data <- data.frame(time = rep(1:3, each = 5),
                          particleID = rep(1:5, 3),
                          px = rnorm(15),py = rnorm(15),
                          vx = rnorm(15),vy = rnorm(15)
)
# Convert to particle.data object
traj <- trajectory_data(sample_data)
# Display summary
show(traj)
```

Index