Package 'GauPro'

November 24, 2022

Type Package

Title Gaussian Process Fitting

Version 0.2.6

Author Collin Erickson

Maintainer Collin Erickson <collinberickson@gmail.com>

Description Fits a Gaussian process model to data. Gaussian processes
are commonly used in computer experiments to fit an interpolating model.
The model is stored as an 'R6' object and can be easily updated with new
data. There are options to run in parallel, and 'Rcpp'
has been used to speed up calculations.
For more info about Gaussian process software, see Erickson et al. (2018)
<doi:10.1016/j.ejor.2017.10.002>.

License GPL-3

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, R6, lbfgs

RoxygenNote 7.2.2

Suggests testthat, knitr, rmarkdown, microbenchmark, numDeriv,
ContourFunctions, dplyr, ggplot2, ggrepel, lhs, mixopt (>
0.1.0), tidyr, MASS

VignetteBuilder knitr

URL https://github.com/CollinErickson/GauPro

BugReports https://github.com/CollinErickson/GauPro/issues

Encoding UTF-8

NeedsCompilation yes

Repository CRAN

Date/Publication 2022-11-24 08:40:02 UTC

R topics documented:

`*.GauPro_kernel` *Kernel product*

Description

Kernel product

Usage

```
## S3 method for class 'GauPro_kernel'
k1 * k2
```

Arguments

k1	First kernel
k2	Second kernel

Value

Kernel which is product of two kernels

Examples

```
k1 <- Exponential$new(beta=1)
k2 <- Matern32$new(beta=0)
k <- k1 * k2
k$k(matrix(c(2,1), ncol=1))
```

+.GauPro_kernel *Kernel sum*

Description

Kernel sum

Usage

```
## S3 method for class 'GauPro_kernel'
k1 + k2
```

Arguments

k1	First kernel
k2	Second kernel

Value

Kernel which is sum of two kernels

Examples

```
k1 <- Exponential$new(beta=1)
k2 <- Matern32$new(beta=0)
k <- k1 + k2
k$k(matrix(c(2,1), ncol=1))
```

arma_mult_cube_vec *Cube multiply over first dimension*

Description

The result is transposed since that is what apply will give you

Usage

```
arma_mult_cube_vec(cub, v)
```

Arguments

cub	A cube (3D array)
v	A vector

Value

Transpose of multiplication over first dimension of cub time v

Examples

```
d1 <- 10
d2 <- 1e2
d3 <- 2e2
aa <- array(data = rnorm(d1*d2*d3), dim = c(d1, d2, d3))
bb <- rnorm(d3)
t1 <- apply(aa, 1, function(U) {U%*%bb})
t2 <- arma_mult_cube_vec(aa, bb)
dd <- t1 - t2

summary(dd)
image(dd)
table(dd)
# microbenchmark::microbenchmark(apply(aa, 1, function(U) {U%*%bb}),
#                                arma_mult_cube_vec(aa, bb))
```

---

corr_exponential_matrix_symC

Correlation Gaussian matrix in C (symmetric)

Description

Correlation Gaussian matrix in C (symmetric)

Usage

```
corr_exponential_matrix_symC(x, theta)
```

Arguments

x	Matrix x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_gauss_matrix_symC(matrix(c(1,0,0,1),2,2),c(1,1))
```

corr_gauss_dCdX *Correlation Gaussian matrix gradient in C using Armadillo*

Description

Correlation Gaussian matrix gradient in C using Armadillo

Usage

```
corr_gauss_dCdX(XX, X, theta, s2)
```

Arguments

XX	Matrix XX to get gradient for
X	Matrix X GP was fit to
theta	Theta vector
s2	Variance parameter

Value

3-dim array of correlation derivative

Examples

```
# corr_gauss_dCdX(matrix(c(1,0,0,1),2,2),c(1,1))
```

corr_gauss_matrix *Gaussian correlation*

Description

Gaussian correlation

Usage

```
corr_gauss_matrix(x, x2 = NULL, theta)
```

Arguments

x	First data matrix
x2	Second data matrix
theta	Correlation parameter

Value

Correlation matrix

Examples

```
corr_gauss_matrix(matrix(1:10,ncol=1), matrix(6:15,ncol=1), 1e-2)
```

`corr_gauss_matrixC` *Correlation Gaussian matrix in C using Rcpp*

Description

Correlation Gaussian matrix in C using Rcpp

Usage

```
corr_gauss_matrixC(x, y, theta)
```

Arguments

x	Matrix x
y	Matrix y, must have same number of columns as x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_gauss_matrixC(matrix(c(1,0,0,1),2,2), matrix(c(1,0,1,1),2,2), c(1,1))
```

`corr_gauss_matrix_armaC`
Correlation Gaussian matrix in C using Armadillo

Description

20-25

Usage

```
corr_gauss_matrix_armaC(x, y, theta, s2 = 1)
```

Arguments

x	Matrix x
y	Matrix y, must have same number of columns as x
theta	Theta vector
s2	Variance to multiply matrix by

Value

Correlation matrix

Examples

```
corr_gauss_matrix_armaC(matrix(c(1,0,0,1),2,2),matrix(c(1,0,1,1),2,2),c(1,1))

x1 <- matrix(runif(100*6), nrow=100, ncol=6)
x2 <- matrix(runif(1e4*6), ncol=6)
th <- runif(6)
t1 <- corr_gauss_matrixC(x1, x2, th)
t2 <- corr_gauss_matrix_armaC(x1, x2, th)
identical(t1, t2)
# microbenchmark::microbenchmark(corr_gauss_matrixC(x1, x2, th),
#                                corr_gauss_matrix_armaC(x1, x2, th))
```

---

corr_gauss_matrix_symC

Correlation Gaussian matrix in C (symmetric)

Description

Correlation Gaussian matrix in C (symmetric)

Usage

```
corr_gauss_matrix_symC(x, theta)
```

Arguments

x	Matrix x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_gauss_matrix_symC(matrix(c(1,0,0,1),2,2),c(1,1))
```

corr_gauss_matrix_sym_armaC

Correlation Gaussian matrix in C using Armadillo (symmetric)

Description

About 30

Usage

```
corr_gauss_matrix_sym_armaC(x, theta)
```

Arguments

x	Matrix x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_gauss_matrix_sym_armaC(matrix(c(1,0,0,1),2,2),c(1,1))

x3 <- matrix(runif(1e3*6), ncol=6)
th <- runif(6)
t3 <- corr_gauss_matrix_symC(x3, th)
t4 <- corr_gauss_matrix_sym_armaC(x3, th)
identical(t3, t4)
# microbenchmark::microbenchmark(corr_gauss_matrix_symC(x3, th),
#                     corr_gauss_matrix_sym_armaC(x3, th), times=50)
```

corr_latentfactor_matrixmatrixC

Correlation Latent factor matrix in C (symmetric)

Description

Correlation Latent factor matrix in C (symmetric)

Usage

```
corr_latentfactor_matrixmatrixC(x, y, theta, xindex, latentdim, offdiagequal)
```

Arguments

x	Matrix x
y	Matrix y
theta	Theta vector
xindex	Index to use
latentdim	Number of latent dimensions
offdiagequal	What to set off-diagonal values with matching values to.

Value

Correlation matrix

Examples

```
corr_latentfactor_matrixmatrixC(matrix(c(1,.5, 2,1.6, 1,0),ncol=2,byrow=TRUE),
                                matrix(c(2,1.6, 1,0),ncol=2,byrow=TRUE),
                                c(1.5,1.8), 1, 1, 1-1e-6)
corr_latentfactor_matrixmatrixC(matrix(c(0,0,0,1,0,0,0,2,0,0,0,3,0,0,0,4),
                                  ncol=4, byrow=TRUE),
                                matrix(c(0,0,0,2,0,0,0,4,0,0,0,1),
                                  ncol=4, byrow=TRUE),
  c(0.101, -0.714, 0.114, -0.755, 0.117, -0.76, 0.116, -0.752),
  4, 2, 1-1e-6) * 6.85
```

---

corr_latentfactor_matrix_symC

Correlation Latent factor matrix in C (symmetric)

Description

Correlation Latent factor matrix in C (symmetric)

Usage

```
corr_latentfactor_matrix_symC(x, theta, xindex, latentdim, offdiagequal)
```

Arguments

x	Matrix x
theta	Theta vector
xindex	Index to use
latentdim	Number of latent dimensions
offdiagequal	What to set off-diagonal values with matching values to.

Value

Correlation matrix

Examples

```
corr_latentfactor_matrix_symC(matrix(c(1,.5, 2,1.6, 1,0),ncol=2,byrow=TRUE),
                              c(1.5,1.8), 1, 1, 1-1e-6)
corr_latentfactor_matrix_symC(matrix(c(0,0,0,1,0,0,0,2,0,0,0,3,0,0,0,4),
                                     ncol=4, byrow=TRUE),
  c(0.101, -0.714, 0.114, -0.755, 0.117, -0.76, 0.116, -0.752),
  4, 2, 1-1e-6) * 6.85
```

---

corr_matern32_matrix_symC

Correlation Matern 3/2 matrix in C (symmetric)

Description

Correlation Matern 3/2 matrix in C (symmetric)

Usage

```
corr_matern32_matrix_symC(x, theta)
```

Arguments

x	Matrix x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_gauss_matrix_symC(matrix(c(1,0,0,1),2,2),c(1,1))
```

corr_matern52_matrix_symC *Correlation Gaussian matrix in C (symmetric)*

Description

Correlation Gaussian matrix in C (symmetric)

Usage

```
corr_matern52_matrix_symC(x, theta)
```

Arguments

x	Matrix x
theta	Theta vector

Value

Correlation matrix

Examples

```
corr_matern52_matrix_symC(matrix(c(1,0,0,1),2,2),c(1,1))
```

Cubic *Cubic Kernel R6 class*

Description

Cubic Kernel R6 class

Cubic Kernel R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_Cubic

Methods

Public methods:

- Cubic$k()
- Cubic$kone()
- Cubic$dC_dparams()
- Cubic$dC_dx()
- Cubic$print()
- Cubic$clone()

Method `k()`**:** Calculate covariance between two points

Usage:

```
Cubic$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Cubic$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Cubic$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Cubic$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Cubic$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Cubic$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Cubic$new(beta=runif(6)-.5)
plot(k1)

n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Cubic$new(1),
                              parallel=FALSE, restarts=0)
gp$predict(.454)
gp$cool1Dplot()
```

Exponential *Exponential Kernel R6 class*

Description

Exponential Kernel R6 class

Exponential Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_Exponential

Methods

Public methods:

- Exponential$k()
- Exponential$kone()
- Exponential$dC_dparams()
- Exponential$dC_dx()
- Exponential$print()
- Exponential$clone()

Method `k()`**:** Calculate covariance between two points

Usage:

```
Exponential$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Exponential$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Exponential$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Exponential$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Exponential$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Exponential$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Exponential$new(beta=0)
```

`FactorKernel` *Factor Kernel R6 class*

Description

Factor Kernel R6 class

Factor Kernel R6 class

Format

`R6Class` object.

Details

For a factor that has been converted to its indices. Each factor will need a separate kernel.

Value

Object of `R6Class` with methods for fitting GP model.

Super class

GauPro::GauPro_kernel -> GauPro_kernel_FactorKernel

Public fields

p Parameter for correlation

p_est Should p be estimated?

p_lower Lower bound of logp

p_upper Upper bound of logp

p_length length of p

s2 variance

s2_est Is s2 estimated?

logs2 Log of s2

logs2_lower Lower bound of logs2

logs2_upper Upper bound of logs2

xindex Index of the factor (which column of X)

nlevels Number of levels for the factor

Methods

Public methods:

- FactorKernel$new()
- FactorKernel$k()
- FactorKernel$kone()
- FactorKernel$dC_dparams()
- FactorKernel$C_dC_dparams()
- FactorKernel$dC_dx()
- FactorKernel$param_optim_start()
- FactorKernel$param_optim_start0()
- FactorKernel$param_optim_lower()
- FactorKernel$param_optim_upper()
- FactorKernel$set_params_from_optim()
- FactorKernel$s2_from_params()
- FactorKernel$plot()
- FactorKernel$print()
- FactorKernel$clone()

Method new(): Initialize kernel object

Usage:

```
FactorKernel$new(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 1,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`s2` Initial variance

`D` Number of input dimensions of data

`nlevels` Number of levels for the factor

`xindex` Index of the factor (which column of X)

`p_lower` Lower bound for p

`p_upper` Upper bound for p

`p_est` Should p be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

`p` Periodic parameter

`alpha` Periodic parameter

Method `k()`**:** Calculate covariance between two points

Usage:

```
FactorKernel$k(x, y = NULL, p = self$p, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`p` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
FactorKernel$kone(x, y, p, s2, isdiag = TRUE, offdiagequal = 1 - 1e-06)
```

Arguments:

`x` vector

`y` vector

`p` correlation parameters on regular scale

`s2` Variance parameter

`isdiag` Is this on the diagonal of the covariance?

`offdiagequal` What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
FactorKernel$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
FactorKernel$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
FactorKernel$dC_dx(
  XX,
  X,
  logp = self$logp,
  logalpha = self$logalpha,
  s2 = self$s2
)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`logp` log of p

`logalpha` log of alpha

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
FactorKernel$param_optim_start(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
FactorKernel$param_optim_start0(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
FactorKernel$param_optim_lower(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
FactorKernel$param_optim_upper(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `set_params_from_optim()`: Set parameters from optimization output

Usage:

```
FactorKernel$set_params_from_optim(
  optim_out,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `s2_from_params()`: Get s2 from params vector

Usage:

```
FactorKernel$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `plot()`:

Usage:

```
FactorKernel$plot(...)
```

Arguments:

`...` Not used.

Method `print()`: Print this object

Usage:

```
FactorKernel$print()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
FactorKernel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
kk <- FactorKernel$new(D=1, nlevels=5, xindex=1)
kk$p <- (1:10)/100
kmat <- outer(1:5, 1:5, Vectorize(kk$k))
kmat
kk$plot()
```

```
# 2D, Gaussian on 1D, index on 2nd dim
library(dplyr)
n <- 20
X <- cbind(matrix(runif(n,2,6), ncol=1),
           matrix(sample(1:2, size=n, replace=TRUE), ncol=1))
X <- rbind(X, c(3.3,3))
n <- nrow(X)
Z <- X[,1] - (X[,2]-1.8)^2 + rnorm(n,0,.1)
tibble(X=X, Z) %>% arrange(X,Z)
k2a <- IgnoreIndsKernel$new(k=Gaussian$new(D=1), ignoreinds = 2)
k2b <- FactorKernel$new(D=2, nlevels=3, xind=2)
k2 <- k2a * k2b
k2b$p_upper <- .65*k2b$p_upper
gp <- GauPro_kernel_model$new(X=X, Z=Z, kernel = k2, verbose = 5,
                              nug.min=1e-2, restarts=0)
gp$kernel$k1$kernel$beta
gp$kernel$k2$p
gp$kernel$k(x = gp$X)
tibble(X=X, Z=Z, pred=gp$predict(X)) %>% arrange(X, Z)
tibble(X=X[,2], Z) %>% group_by(X) %>% summarize(n=n(), mean(Z))
curve(gp$pred(cbind(matrix(x,ncol=1),1)),2,6, ylim=c(min(Z), max(Z)))
points(X[X[,2]==1,1], Z[X[,2]==1])
curve(gp$pred(cbind(matrix(x,ncol=1),2)), add=TRUE, col=2)
points(X[X[,2]==2,1], Z[X[,2]==2], col=2)
curve(gp$pred(cbind(matrix(x,ncol=1),3)), add=TRUE, col=3)
points(X[X[,2]==3,1], Z[X[,2]==3], col=3)
legend(legend=1:3, fill=1:3, x="topleft")
# See which points affect (5.5, 3 themost)
data.frame(X, cov=gp$kernel$k(X, c(5.5,3))) %>% arrange(-cov)
plot(k2b)
```

GauPro *GauPro_selector*

Description

GauPro_selector

Usage

```
GauPro(..., type = "Gauss")
```

Arguments

...	Pass on
type	Type of Gaussian process, or the kind of correlation function.

Value

A GauPro object

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
#y <- sin(2*pi*x) + rnorm(n,0,1e-1)
y <- (2*x) %%1
gp <- GauPro(X=x, Z=y, parallel=FALSE)
```

GauPro_base *Class providing object with methods for fitting a GP model*

Description

Class providing object with methods for fitting a GP model

Class providing object with methods for fitting a GP model

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Methods

`new(X, Z, corr="Gauss", verbose=0, separable=T, useC=F,useGrad=T, parallel=T, nug.est=T, ...)`
: This method is used to create object of this class with `X` and `Z` as the data.

`update(Xnew=NULL, Znew=NULL, Xall=NULL, Zall=NULL, restarts = 5, param_update = T, nug.update = self$nug.`
: This method updates the model, adding new data if given, then running optimization again.

Public fields

`X` Design matrix

`Z` Responses

`N` Number of data points

`D` Dimension of data

`corr` Type of correlation function

`nug.min` Minimum value of nugget

`nug` Value of the nugget, is estimated unless told otherwise

`separable` Are the dimensions separable?

`verbose` 0 means nothing printed, 1 prints some, 2 prints most.

`useGrad` Should grad be used?

`useC` Should C code be used?

`parallel` Should the code be run in parallel?

`parallel_cores` How many cores are there? It will self detect, do not set yourself.

Active bindings

`corr` Type of correlation function

`separable` Are the dimensions separable?

Methods

Public methods:

- `GauPro_base$corr_func()`
- `GauPro_base$new()`
- `GauPro_base$initialize_GauPr()`
- `GauPro_base$fit()`
- `GauPro_base$update_K_and_estimates()`
- `GauPro_base$predict()`
- `GauPro_base$pred()`
- `GauPro_base$pred_one_matrix()`
- `GauPro_base$pred_mean()`
- `GauPro_base$pred_meanC()`
- `GauPro_base$pred_var()`
- `GauPro_base$pred_LOO()`
- `GauPro_base$plot()`
- `GauPro_base$cool1Dplot()`
- `GauPro_base$plot1D()`
- `GauPro_base$plot2D()`
- `GauPro_base$loglikelihood()`
- `GauPro_base$optim()`
- `GauPro_base$optimRestart()`
- `GauPro_base$update()`
- `GauPro_base$update_data()`
- `GauPro_base$update_corrparams()`
- `GauPro_base$update_nugget()`
- `GauPro_base$deviance_searchnug()`
- `GauPro_base$nugget_update()`
- `GauPro_base$grad_norm()`
- `GauPro_base$sample()`
- `GauPro_base$print()`
- `GauPro_base$clone()`

Method `corr_func()`:

Usage:

```
GauPro_base$corr_func(...)
```

Method `new()`**:**

Usage:

```
GauPro_base$new(
  X,
  Z,
  verbose = 0,
  useC = F,
  useGrad = T,
  parallel = FALSE,
  nug = 1e-06,
  nug.min = 1e-08,
  nug.est = T,
  param.est = TRUE,
  ...
)
```

Method `initialize_GauPr()`**:**

Usage:

```
GauPro_base$initialize_GauPr()
```

Method `fit()`**:**

Usage:

```
GauPro_base$fit(X, Z)
```

Method `update_K_and_estimates()`**:**

Usage:

```
GauPro_base$update_K_and_estimates()
```

Method `predict()`**:**

Usage:

```
GauPro_base$predict(XX, se.fit = F, covmat = F, split_speed = T)
```

Method `pred()`**:**

Usage:

```
GauPro_base$pred(XX, se.fit = F, covmat = F, split_speed = T)
```

Method `pred_one_matrix()`**:**

Usage:

```
GauPro_base$pred_one_matrix(XX, se.fit = F, covmat = F)
```

Method `pred_mean()`**:**

Usage:

```
GauPro_base$pred_mean(XX, kx.xx)
```

Method `pred_meanC()`**:**

Usage:

```
GauPro_base$pred_meanC(XX, kx.xx)
```

Method `pred_var()`**:**

Usage:

```
GauPro_base$pred_var(XX, kxx, kx.xx, covmat = F)
```

Method `pred_LOO()`**:**

Usage:

```
GauPro_base$pred_LOO(se.fit = FALSE)
```

Method `plot()`**:** Plot the object

Usage:

```
GauPro_base$plot(...)
```

Arguments:

... Parameters passed to cool1Dplot(), plot2D(), or plotmarginal()

Method `cool1Dplot()`**:**

Usage:

```
GauPro_base$cool1Dplot(
  n2 = 20,
  nn = 201,
  col2 = "gray",
  xlab = "x",
  ylab = "y",
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL
)
```

Method `plot1D()`**:**

Usage:

```
GauPro_base$plot1D(
  n2 = 20,
  nn = 201,
  col2 = 2,
  xlab = "x",
  ylab = "y",
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL
)
```

Method `plot2D()`**:**

Usage:

```
GauPro_base$plot2D()
```

Method `loglikelihood()`**:**

Usage:

```
GauPro_base$loglikelihood(mu = self$mu_hat, s2 = self$s2_hat)
```

Method `optim()`**:**

Usage:

```
GauPro_base$optim(
  restarts = 5,
  param_update = T,
  nug.update = self$nug.est,
  parallel = self$parallel,
  parallel_cores = self$parallel_cores
)
```

Method `optimRestart()`**:**

Usage:

```
GauPro_base$optimRestart(
  start.par,
  start.par0,
  param_update,
  nug.update,
  optim.func,
  optim.grad,
  optim.fngr,
  lower,
  upper,
  jit = T
)
```

Method `update()`**:**

Usage:

```
GauPro_base$update(
  Xnew = NULL,
  Znew = NULL,
  Xall = NULL,
  Zall = NULL,
  restarts = 5,
  param_update = self$param.est,
  nug.update = self$nug.est,
  no_update = FALSE
)
```

Method `update_data()`**:**

Usage:

```
GauPro_base$update_data(Xnew = NULL, Znew = NULL, Xall = NULL, Zall = NULL)
```

Method `update_corrparams()`**:**

Usage:

```
GauPro_base$update_corrparams(...)
```

Method `update_nugget()`**:**

Usage:

```
GauPro_base$update_nugget(...)
```

Method `deviance_searchnug()`**:**

Usage:

```
GauPro_base$deviance_searchnug()
```

Method `nugget_update()`**:**

Usage:

```
GauPro_base$nugget_update()
```

Method `grad_norm()`**:**

Usage:

```
GauPro_base$grad_norm(XX)
```

Method `sample()`**:**

Usage:

```
GauPro_base$sample(XX, n = 1)
```

Method `print()`**:**

Usage:

```
GauPro_base$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_base$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#n <- 12
#x <- matrix(seq(0,1,length.out = n), ncol=1)
#y <- sin(2*pi*x) + rnorm(n,0,1e-1)
#gp <- GauPro(X=x, Z=y, parallel=FALSE)
```

GauPro_Gauss *Corr Gauss GP using inherited optim*

Description

Corr Gauss GP using inherited optim

Corr Gauss GP using inherited optim

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro -> GauPro_Gauss

Methods

Public methods:

- GauPro_Gauss$new()
- GauPro_Gauss$corr_func()
- GauPro_Gauss$deviance_theta()
- GauPro_Gauss$deviance_theta_log()
- GauPro_Gauss$deviance()
- GauPro_Gauss$deviance_grad()
- GauPro_Gauss$deviance_fngr()
- GauPro_Gauss$deviance_log()
- GauPro_Gauss$deviance_log2()
- GauPro_Gauss$deviance_log_grad()
- GauPro_Gauss$deviance_log2_grad()
- GauPro_Gauss$deviance_log2_fngr()
- GauPro_Gauss$get_optim_functions()
- GauPro_Gauss$param_optim_lower()
- GauPro_Gauss$param_optim_upper()
- GauPro_Gauss$param_optim_start()
- GauPro_Gauss$param_optim_start0()
- GauPro_Gauss$param_optim_jitter()
- GauPro_Gauss$update_params()
- GauPro_Gauss$grad()
- GauPro_Gauss$grad_dist()

- GauPro_Gauss$hessian()
- GauPro_Gauss$print()
- GauPro_Gauss$clone()

Method `new()`**:**

Usage:

```
GauPro_Gauss$new(
  X,
  Z,
  verbose = 0,
  separable = T,
  useC = F,
  useGrad = T,
  parallel = FALSE,
  nug = 1e-06,
  nug.min = 1e-08,
  nug.est = T,
  param.est = T,
  theta = NULL,
  theta_short = NULL,
  theta_map = NULL,
  ...
)
```

Method `corr_func()`**:**

Usage:

```
GauPro_Gauss$corr_func(x, x2 = NULL, theta = self$theta)
```

Method `deviance_theta()`**:**

Usage:

```
GauPro_Gauss$deviance_theta(theta)
```

Method `deviance_theta_log()`**:**

Usage:

```
GauPro_Gauss$deviance_theta_log(beta)
```

Method `deviance()`**:**

Usage:

```
GauPro_Gauss$deviance(theta = self$theta, nug = self$nug)
```

Method `deviance_grad()`**:**

Usage:

```
GauPro_Gauss$deviance_grad(
  theta = NULL,
  nug = self$nug,
  joint = NULL,
  overwhat = if (self$nug.est) "joint" else "theta"
)
```

Method `deviance_fngr()`**:**

Usage:

```
GauPro_Gauss$deviance_fngr(
  theta = NULL,
  nug = NULL,
  overwhat = if (self$nug.est) "joint" else "theta"
)
```

Method `deviance_log()`**:**

Usage:

```
GauPro_Gauss$deviance_log(beta = NULL, nug = self$nug, joint = NULL)
```

Method `deviance_log2()`**:**

Usage:

```
GauPro_Gauss$deviance_log2(beta = NULL, lognug = NULL, joint = NULL)
```

Method `deviance_log_grad()`**:**

Usage:

```
GauPro_Gauss$deviance_log_grad(
  beta = NULL,
  nug = self$nug,
  joint = NULL,
  overwhat = if (self$nug.est) "joint" else "theta"
)
```

Method `deviance_log2_grad()`**:**

Usage:

```
GauPro_Gauss$deviance_log2_grad(
  beta = NULL,
  lognug = NULL,
  joint = NULL,
  overwhat = if (self$nug.est) "joint" else "theta"
)
```

Method `deviance_log2_fngr()`**:**

Usage:

```
GauPro_Gauss$deviance_log2_fngr(
  beta = NULL,
  lognug = NULL,
  joint = NULL,
  overwhat = if (self$nug.est) "joint" else "theta"
)
```

Method `get_optim_functions()`**:**

Usage:

```
GauPro_Gauss$get_optim_functions(param_update, nug.update)
```

Method `param_optim_lower()`:

Usage:

```
GauPro_Gauss$param_optim_lower()
```

Method `param_optim_upper()`:

Usage:

```
GauPro_Gauss$param_optim_upper()
```

Method `param_optim_start()`:

Usage:

```
GauPro_Gauss$param_optim_start()
```

Method `param_optim_start0()`:

Usage:

```
GauPro_Gauss$param_optim_start0()
```

Method `param_optim_jitter()`:

Usage:

```
GauPro_Gauss$param_optim_jitter(param_value)
```

Method `update_params()`:

Usage:

```
GauPro_Gauss$update_params(restarts, param_update, nug.update)
```

Method `grad()`:

Usage:

```
GauPro_Gauss$grad(XX)
```

Method `grad_dist()`:

Usage:

```
GauPro_Gauss$grad_dist(XX)
```

Method `hessian()`:

Usage:

```
GauPro_Gauss$hessian(XX, useC = self$useC)
```

Method `print()`:

Usage:

```
GauPro_Gauss$print()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
GauPro_Gauss$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_Gauss$new(X=x, Z=y, parallel=FALSE)
```

GauPro_Gauss_LOO *Corr Gauss GP using inherited optim*

Description

Corr Gauss GP using inherited optim

Corr Gauss GP using inherited optim

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super classes

GauPro::GauPro -> GauPro::GauPro_Gauss -> GauPro_Gauss_LOO

Methods

Public methods:

- GauPro_Gauss_LOO$update()
- GauPro_Gauss_LOO$pred_one_matrix()
- GauPro_Gauss_LOO$print()
- GauPro_Gauss_LOO$clone()

Method update():

Usage:

```
GauPro_Gauss_LOO$update(
  Xnew = NULL,
  Znew = NULL,
  Xall = NULL,
  Zall = NULL,
  restarts = 5,
  param_update = self$param.est,
  nug.update = self$nug.est,
  no_update = FALSE
)
```

Method `pred_one_matrix()`**:**

Usage:

```
GauPro_Gauss_LOO$pred_one_matrix(XX, se.fit = F, covmat = F)
```

Method `print()`**:** Print this object

Usage:

```
GauPro_Gauss_LOO$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_Gauss_LOO$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_Gauss_LOO$new(X=x, Z=y, parallel=FALSE)
```

`GauPro_kernel` *Kernel R6 class*

Description

Kernel R6 class

Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Public fields

`D` Number of input dimensions of data

`useC` Should C code be used when possible? Can be much faster.

Methods

Public methods:

- GauPro_kernel$plot()
- GauPro_kernel$print()
- GauPro_kernel$clone()

Method `plot()`**:** Plot kernel decay.

Usage:

```
GauPro_kernel$plot(X = NULL)
```

Arguments:

`X` Matrix of points the kernel is used with. Some will be used to demonstrate how the covariance changes.

Method `print()`**:** Print this object

Usage:

```
GauPro_kernel$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_kernel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#k <- GauPro_kernel$new()
```

`GauPro_kernel_beta` *Beta Kernel R6 class*

Description

Beta Kernel R6 class

Beta Kernel R6 class

Format

`R6Class` object.

Details

This is the base structure for a kernel that uses beta = log10(theta) for the lengthscale parameter. It standardizes the params because they all use the same underlying structure. Kernels that inherit this only need to implement kone and dC_dparams.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro_kernel -> GauPro_kernel_beta

Public fields

beta Parameter for correlation. Log of theta.

beta_est Should beta be estimated?

beta_lower Lower bound of beta

beta_upper Upper bound of beta

beta_length length of beta

s2 variance

logs2 Log of s2

logs2_lower Lower bound of logs2

logs2_upper Upper bound of logs2

s2_est Should s2 be estimated?

useC Should C code used? Much faster.

Methods

Public methods:

- GauPro_kernel_beta$new()
- GauPro_kernel_beta$k()
- GauPro_kernel_beta$kone()
- GauPro_kernel_beta$param_optim_start()
- GauPro_kernel_beta$param_optim_start0()
- GauPro_kernel_beta$param_optim_lower()
- GauPro_kernel_beta$param_optim_upper()
- GauPro_kernel_beta$set_params_from_optim()
- GauPro_kernel_beta$C_dC_dparams()
- GauPro_kernel_beta$s2_from_params()
- GauPro_kernel_beta$clone()

Method new(): Initialize kernel object

Usage:

```
GauPro_kernel_beta$new(
  beta,
  s2 = 1,
  D,
  beta_lower = -8,
```

```
  beta_upper = 6,
  beta_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = TRUE
)
```

Arguments:

`beta` Initial beta value

`s2` Initial variance

`D` Number of input dimensions of data

`beta_lower` Lower bound for beta

`beta_upper` Upper bound for beta

`beta_est` Should beta be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

`useC` Should C code used? Much faster.

Method `k()`**:** Calculate covariance between two points

Usage:

```
GauPro_kernel_beta$k(
  x,
  y = NULL,
  beta = self$beta,
  s2 = self$s2,
  params = NULL
)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters. Log of theta.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Calculate covariance between two points

Usage:

```
GauPro_kernel_beta$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector.

`y` vector.

`beta` Correlation parameters. Log of theta.

`theta` Correlation parameters.

`s2` Variance parameter.

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
GauPro_kernel_beta$param_optim_start(
  jitter = F,
  y,
  beta_est = self$beta_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
GauPro_kernel_beta$param_optim_start0(
  jitter = F,
  y,
  beta_est = self$beta_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_lower()`**:** Upper bounds of parameters for optimization

Usage:

```
GauPro_kernel_beta$param_optim_lower(
  beta_est = self$beta_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`s2_est` Is s2 being estimated?

`p_est` Is p being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
GauPro_kernel_beta$param_optim_upper(
  beta_est = self$beta_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`s2_est` Is s2 being estimated?

`p_est` Is p being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
GauPro_kernel_beta$set_params_from_optim(
  optim_out,
  beta_est = self$beta_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`beta_est` Is beta being estimated?

`s2_est` Is s2 being estimated?

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
GauPro_kernel_beta$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
GauPro_kernel_beta$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_kernel_beta$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#k1 <- Matern52$new(beta=0)
```

GauPro_kernel_model *GauPro model that uses kernels*

Description

GauPro model that uses kernels

GauPro model that uses kernels

Format

R6Class object.

Details

Class providing object with methods for fitting a GP model. Allows for different kernel and trend functions to be used.

Value

Object of R6Class with methods for fitting GP model.

Methods

`new(X, Z, corr="Gauss", verbose=0, separable=T, useC=F, useGrad=T, parallel=T, nug.est=T, ...)`
This method is used to create object of this class with X and Z as the data.

`update(Xnew=NULL, Znew=NULL, Xall=NULL, Zall=NULL, restarts = 0, param_update = T, nug.update = self$nug.`
This method updates the model, adding new data if given, then running optimization again.

Public fields

`X` Design matrix

`Z` Responses

`N` Number of data points

`D` Dimension of data

`nug.min` Minimum value of nugget

`nug.max` Maximum value of the nugget.

`nug.est` Should the nugget be estimated?

`nug` Value of the nugget, is estimated unless told otherwise

`param.est` Should the kernel parameters be estimated?

`verbose` 0 means nothing printed, 1 prints some, 2 prints most.

`useGrad` Should grad be used?

`useC` Should C code be used?

`parallel` Should the code be run in parallel?

`parallel_cores` How many cores are there? By default it detects.

`kernel` The kernel to determine the correlations.

`trend` The trend.

`mu_hatX` Predicted trend value for each point in X.

`s2_hat` Variance parameter estimate

`K` Covariance matrix

`Kchol` Cholesky factorization of K

`Kinv` Inverse of K

`Kinv_Z_minus_mu_hatX` K inverse times Z minus the predicted trend at X.

`restarts` Number of optimization restarts to do when updating.

`normalize` Should the inputs be normalized?

`normalize_mean` If using normalize, the mean of each column.

`normalize_sd` If using normalize, the standard deviation of each column.

`optimizer` What algorithm should be used to optimize the parameters.

`track_optim` Should it track the parameters evaluated while optimizing?

`track_optim_inputs` If track_optim is TRUE, this will keep a list of parameters evaluated. View them with plot_track_optim.

`track_optim_dev` If track_optim is TRUE, this will keep a vector of the deviance values calculated while optimizing parameters. View them with plot_track_optim.

`formula` Formula

`convert_formula_data` List for storing data to convert data using the formula

Methods

Public methods:

- `GauPro_kernel_model$new()`
- `GauPro_kernel_model$fit()`
- `GauPro_kernel_model$update_K_and_estimates()`
- `GauPro_kernel_model$predict()`
- `GauPro_kernel_model$pred()`
- `GauPro_kernel_model$pred_one_matrix()`
- `GauPro_kernel_model$pred_mean()`
- `GauPro_kernel_model$pred_meanC()`
- `GauPro_kernel_model$pred_var()`
- `GauPro_kernel_model$pred_LOO()`
- `GauPro_kernel_model$pred_var_after_adding_points()`
- `GauPro_kernel_model$pred_var_after_adding_points_sep()`
- `GauPro_kernel_model$pred_var_reduction()`
- `GauPro_kernel_model$pred_var_reductions()`
- `GauPro_kernel_model$plot()`
- `GauPro_kernel_model$cool1Dplot()`

- GauPro_kernel_model$plot1D()
- GauPro_kernel_model$plot2D()
- GauPro_kernel_model$plotmarginal()
- GauPro_kernel_model$plotmarginalrandom()
- GauPro_kernel_model$plotLOO()
- GauPro_kernel_model$plot_track_optim()
- GauPro_kernel_model$loglikelihood()
- GauPro_kernel_model$get_optim_functions()
- GauPro_kernel_model$param_optim_lower()
- GauPro_kernel_model$param_optim_upper()
- GauPro_kernel_model$param_optim_start()
- GauPro_kernel_model$param_optim_start0()
- GauPro_kernel_model$param_optim_start_mat()
- GauPro_kernel_model$optim()
- GauPro_kernel_model$optimRestart()
- GauPro_kernel_model$update()
- GauPro_kernel_model$update_fast()
- GauPro_kernel_model$update_params()
- GauPro_kernel_model$update_data()
- GauPro_kernel_model$update_corrparams()
- GauPro_kernel_model$update_nugget()
- GauPro_kernel_model$deviance()
- GauPro_kernel_model$deviance_grad()
- GauPro_kernel_model$deviance_fngr()
- GauPro_kernel_model$grad()
- GauPro_kernel_model$grad_norm()
- GauPro_kernel_model$grad_dist()
- GauPro_kernel_model$grad_sample()
- GauPro_kernel_model$grad_norm2_mean()
- GauPro_kernel_model$grad_norm2_dist()
- GauPro_kernel_model$grad_norm2_sample()
- GauPro_kernel_model$hessian()
- GauPro_kernel_model$sample()
- GauPro_kernel_model$EI()
- GauPro_kernel_model$maxEI()
- GauPro_kernel_model$maxEIwithfactors()
- GauPro_kernel_model$maxEIwithfactorsordiscrete()
- GauPro_kernel_model$maxqEI()
- GauPro_kernel_model$KG()
- GauPro_kernel_model$importance()
- GauPro_kernel_model$print()
- GauPro_kernel_model$summary()

- GauPro_kernel_model$clone()

Method `new()`**:** Create kernel_model object

Usage:

```
GauPro_kernel_model$new(
  X,
  Z,
  kernel,
  trend,
  verbose = 0,
  useC = TRUE,
  useGrad = TRUE,
  parallel = FALSE,
  parallel_cores = "detect",
  nug = 1e-06,
  nug.min = 1e-08,
  nug.max = 100,
  nug.est = TRUE,
  param.est = TRUE,
  restarts = 0,
  normalize = FALSE,
  optimizer = "L-BFGS-B",
  track_optim = FALSE,
  formula,
  data,
  ...
)
```

Arguments:

`X` Matrix whose rows are the input points

`Z` Output points corresponding to X

`kernel` The kernel to use. E.g., Gaussian$new().

`trend` Trend to use. E.g., trend_constant$new().

`verbose` Amount of stuff to print. 0 is little, 2 is a lot.

`useC` Should C code be used when possible? Should be faster.

`useGrad` Should the gradient be used?

`parallel` Should code be run in parallel? Make optimization faster but uses more computer resources.

`parallel_cores` When using parallel, how many cores should be used?

`nug` Value for the nugget. The starting value if estimating it.

`nug.min` Minimum allowable value for the nugget.

`nug.max` Maximum allowable value for the nugget.

`nug.est` Should the nugget be estimated?

`param.est` Should the kernel parameters be estimated?

`restarts` How many optimization restarts should be used when estimating parameters?

`normalize` Should the data be normalized?

`optimizer` What algorithm should be used to optimize the parameters.

`track_optim` Should it track the parameters evaluated while optimizing?

`...` Not used

Method `fit()`**:** Fit model

Usage:

```
GauPro_kernel_model$fit(X, Z)
```

Arguments:

`X` Inputs

`Z` Outputs

Method `update_K_and_estimates()`**:** Update covariance matrix and estimates

Usage:

```
GauPro_kernel_model$update_K_and_estimates()
```

Method `predict()`**:** Predict for a matrix of points

Usage:

```
GauPro_kernel_model$predict(
  XX,
  se.fit = F,
  covmat = F,
  split_speed = F,
  mean_dist = FALSE
)
```

Arguments:

`XX` points to predict at

`se.fit` Should standard error be returned?

`covmat` Should covariance matrix be returned?

`split_speed` Should the matrix be split for faster predictions?

`mean_dist` Should the error be for the distribution of the mean?

Method `pred()`**:** Predict for a matrix of points

Usage:

```
GauPro_kernel_model$pred(
  XX,
  se.fit = F,
  covmat = F,
  split_speed = F,
  mean_dist = FALSE
)
```

Arguments:

`XX` points to predict at

`se.fit` Should standard error be returned?

`covmat` Should covariance matrix be returned?

`split_speed` Should the matrix be split for faster predictions?

`mean_dist` Should the error be for the distribution of the mean?

Method `pred_one_matrix()`**:** Predict for a matrix of points

Usage:

```
GauPro_kernel_model$pred_one_matrix(
  XX,
  se.fit = F,
  covmat = F,
  return_df = FALSE,
  mean_dist = FALSE
)
```

Arguments:

`XX` points to predict at

`se.fit` Should standard error be returned?

`covmat` Should covariance matrix be returned?

`return_df` When returning se.fit, should it be returned in a data frame?

`mean_dist` Should the error be for the distribution of the mean?

Method `pred_mean()`**:** Predict mean

Usage:

```
GauPro_kernel_model$pred_mean(XX, kx.xx)
```

Arguments:

`XX` points to predict at

`kx.xx` Covariance of X with XX

Method `pred_meanC()`**:** Predict mean using C

Usage:

```
GauPro_kernel_model$pred_meanC(XX, kx.xx)
```

Arguments:

`XX` points to predict at

`kx.xx` Covariance of X with XX

Method `pred_var()`**:** Predict variance

Usage:

```
GauPro_kernel_model$pred_var(XX, kxx, kx.xx, covmat = F)
```

Arguments:

`XX` points to predict at

`kxx` Covariance of XX with itself

`kx.xx` Covariance of X with XX

`covmat` Should the covariance matrix be returned?

Method `pred_LOO()`**:** leave one out predictions

Usage:

```
GauPro_kernel_model$pred_LOO(se.fit = FALSE)
```

Arguments:

`se.fit` Should standard errors be included?

Method `pred_var_after_adding_points()`**:** Predict variance after adding points

Usage:

```
GauPro_kernel_model$pred_var_after_adding_points(add_points, pred_points)
```

Arguments:

`add_points` Points to add

`pred_points` Points to predict at

Method `pred_var_after_adding_points_sep()`**:** Predict variance reductions after adding each point separately

Usage:

```
GauPro_kernel_model$pred_var_after_adding_points_sep(add_points, pred_points)
```

Arguments:

`add_points` Points to add

`pred_points` Points to predict at

Method `pred_var_reduction()`**:** Predict variance reduction for a single point

Usage:

```
GauPro_kernel_model$pred_var_reduction(add_point, pred_points)
```

Arguments:

`add_point` Point to add

`pred_points` Points to predict at

Method `pred_var_reductions()`**:** Predict variance reductions

Usage:

```
GauPro_kernel_model$pred_var_reductions(add_points, pred_points)
```

Arguments:

`add_points` Points to add

`pred_points` Points to predict at

Method `plot()`**:** Plot the object

Usage:

```
GauPro_kernel_model$plot(...)
```

Arguments:

`...` Parameters passed to cool1Dplot(), plot2D(), or plotmarginal()

Method `cool1Dplot()`**:** Make cool 1D plot

Usage:

```
GauPro_kernel_model$cool1Dplot(
  n2 = 20,
  nn = 201,
  col2 = "gray",
  xlab = "x",
  ylab = "y",
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL
)
```

Arguments:

`n2` Number of things to plot

`nn` Number of things to plot

`col2` color

`xlab` x label

`ylab` y label

`xmin` xmin

`xmax` xmax

`ymin` ymin

`ymax` ymax

Method `plot1D()`**:** Make 1D plot

Usage:

```
GauPro_kernel_model$plot1D(
  n2 = 20,
  nn = 201,
  col2 = 2,
  col3 = 3,
  xlab = "x",
  ylab = "y",
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL
)
```

Arguments:

`n2` Number of things to plot

`nn` Number of things to plot

`col2` Color of the prediction interval

`col3` Color of the interval for the mean

`xlab` x label

`ylab` y label

`xmin` xmin

`xmax` xmax

`ymin` ymin

`ymax` ymax

Method `plot2D()`**:** Make 2D plot

Usage:

```
GauPro_kernel_model$plot2D()
```

Method `plotmarginal()`**:** Plot marginal. For each input, hold all others at a constant value and adjust it along it's range to see how the prediction changes.

Usage:

```
GauPro_kernel_model$plotmarginal(pt = colMeans(self$X))
```

Arguments:

`pt` What point to use as a center. All values except the one being plotted are held constant at this value.

Method `plotmarginalrandom()`**:** Plot marginal prediction for random sample of inputs

Usage:

```
GauPro_kernel_model$plotmarginalrandom(n = 100)
```

Arguments:

`n` Number of random points to evaluate

Method `plotLOO()`**:** Plot leave one out predictions for design points

Usage:

```
GauPro_kernel_model$plotLOO()
```

Method `plot_track_optim()`**:** If track_optim, this will plot the parameters in the order they were evaluated.

Usage:

```
GauPro_kernel_model$plot_track_optim(minindex = NULL)
```

Arguments:

`minindex` Minimum index to plot.

Method `loglikelihood()`**:** Calculate loglikelihood of parameters

Usage:

```
GauPro_kernel_model$loglikelihood(mu = self$mu_hatX, s2 = self$s2_hat)
```

Arguments:

`mu` Mean parameters

`s2` Variance parameter

Method `get_optim_functions()`**:** Get optimization functions

Usage:

```
GauPro_kernel_model$get_optim_functions(param_update, nug.update)
```

Arguments:

`param_update` Should parameters be updated?

`nug.update` Should nugget be updated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
GauPro_kernel_model$param_optim_lower(nug.update)
```

Arguments:

`nug.update` Is the nugget being updated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
GauPro_kernel_model$param_optim_upper(nug.update)
```

Arguments:

`nug.update` Is the nugget being updated?

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
GauPro_kernel_model$param_optim_start(nug.update, jitter)
```

Arguments:

`nug.update` Is nugget being updated?

`jitter` Should there be a jitter?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
GauPro_kernel_model$param_optim_start0(nug.update, jitter)
```

Arguments:

`nug.update` Is nugget being updated?

`jitter` Should there be a jitter?

Method `param_optim_start_mat()`**:** Get matrix for starting points of optimization

Usage:

```
GauPro_kernel_model$param_optim_start_mat(restarts, nug.update, l)
```

Arguments:

`restarts` Number of restarts to use

`nug.update` Is nugget being updated?

`l` Not used

Method `optim()`**:** Optimize parameters

Usage:

```
GauPro_kernel_model$optim(
  restarts = self$restarts,
  n0 = 5 * self$D,
  param_update = T,
  nug.update = self$nug.est,
  parallel = self$parallel,
  parallel_cores = self$parallel_cores
)
```

Arguments:

`restarts` Number of restarts to do

`n0` This many starting parameters are chosen and evaluated. The best ones are used as the starting points for optimization.

`param_update` Should parameters be updated?

`nug.update` Should nugget be updated?

`parallel` Should restarts be done in parallel?

`parallel_cores` If running parallel, how many cores should be used?

Method `optimRestart()`**:** Run a single optimization restart.

Usage:

```
GauPro_kernel_model$optimRestart(
  start.par,
  start.par0,
  param_update,
  nug.update,
  optim.func,
  optim.grad,
  optim.fngr,
  lower,
  upper,
  jit = T,
  start.par.i
)
```

Arguments:

`start.par` Starting parameters

`start.par0` Starting parameters

`param_update` Should parameters be updated?

`nug.update` Should nugget be updated?

`optim.func` Function to optimize.

`optim.grad` Gradient of function to optimize.

`optim.fngr` Function that returns the function value and its gradient.

`lower` Lower bounds for optimization

`upper` Upper bounds for optimization

`jit` Is jitter being used?

`start.par.i` Starting parameters for this restart

Method `update()`: Update the model. Should only give in (Xnew and Znew) or (Xall and Zall).

Usage:

```
GauPro_kernel_model$update(
  Xnew = NULL,
  Znew = NULL,
  Xall = NULL,
  Zall = NULL,
  restarts = self$restarts,
  param_update = self$param.est,
  nug.update = self$nug.est,
  no_update = FALSE
)
```

Arguments:

`Xnew` New X values to add.

`Znew` New Z values to add.

`Xall` All X values to be used. Will replace existing X.

`Zall` All Z values to be used. Will replace existing Z.

`restarts` Number of optimization restarts.

`param_update` Are the parameters being updated?

`nug.update` Is the nugget being updated?

`no_update` Are no parameters being updated?

Method `update_fast()`: Fast update when adding new data.

Usage:

```
GauPro_kernel_model$update_fast(Xnew = NULL, Znew = NULL)
```

Arguments:

`Xnew` New X values to add.

`Znew` New Z values to add.

Method `update_params()`: Update the parameters.

Usage:

```
GauPro_kernel_model$update_params(..., nug.update)
```

Arguments:

`...` Passed to optim.

`nug.update` Is the nugget being updated?

Method `update_data()`: Update the data. Should only give in (Xnew and Znew) or (Xall and Zall).

Usage:

```
GauPro_kernel_model$update_data(
  Xnew = NULL,
  Znew = NULL,
  Xall = NULL,
  Zall = NULL
)
```

Arguments:

`Xnew` New X values to add.

`Znew` New Z values to add.

`Xall` All X values to be used. Will replace existing X.

`Zall` All Z values to be used. Will replace existing Z.

Method `update_corrparams()`**:** Update correlation parameters. Not the nugget.

Usage:

```
GauPro_kernel_model$update_corrparams(...)
```

Arguments:

`...` Passed to self$update()

Method `update_nugget()`**:** Update nugget Not the correlation parameters.

Usage:

```
GauPro_kernel_model$update_nugget(...)
```

Arguments:

`...` Passed to self$update()

Method `deviance()`**:** Calculate the deviance.

Usage:

```
GauPro_kernel_model$deviance(
  params = NULL,
  nug = self$nug,
  nuglog,
  trend_params = NULL
)
```

Arguments:

`params` Kernel parameters

`nug` Nugget

`nuglog` Log of nugget. Only give in nug or nuglog.

`trend_params` Parameters for the trend.

Method `deviance_grad()`**:** Calculate the gradient of the deviance.

Usage:

```
GauPro_kernel_model$deviance_grad(
  params = NULL,
  kernel_update = TRUE,
  X = self$X,
  nug = self$nug,
  nug.update,
  nuglog,
  trend_params = NULL,
  trend_update = TRUE
)
```

Arguments:

`params` Kernel parameters

`kernel_update` Is the kernel being updated? If yes, it's part of the gradient.

`X` Input matrix

`nug` Nugget

`nug.update` Is the nugget being updated? If yes, it's part of the gradient.

`nuglog` Log of the nugget.

`trend_params` Trend parameters

`trend_update` Is the trend being updated? If yes, it's part of the gradient.

Method `deviance_fngr()`**:** Calculate the deviance along with its gradient.

Usage:

```
GauPro_kernel_model$deviance_fngr(
  params = NULL,
  kernel_update = TRUE,
  X = self$X,
  nug = self$nug,
  nug.update,
  nuglog,
  trend_params = NULL,
  trend_update = TRUE
)
```

Arguments:

`params` Kernel parameters

`kernel_update` Is the kernel being updated? If yes, it's part of the gradient.

`X` Input matrix

`nug` Nugget

`nug.update` Is the nugget being updated? If yes, it's part of the gradient.

`nuglog` Log of the nugget.

`trend_params` Trend parameters

`trend_update` Is the trend being updated? If yes, it's part of the gradient.

Method `grad()`**:** Calculate gradient

Usage:

```
GauPro_kernel_model$grad(XX, X = self$X, Z = self$Z)
```

Arguments:

`XX` points to calculate at

`X` X points

`Z` output points

Method `grad_norm()`**:** Calculate norm of gradient

Usage:

```
GauPro_kernel_model$grad_norm(XX)
```

Arguments:

`XX` points to calculate at

Method `grad_dist()`**:** Calculate distribution of gradient

Usage:

```
GauPro_kernel_model$grad_dist(XX)
```

Arguments:

`XX` points to calculate at

Method `grad_sample()`**:** Sample gradient at points

Usage:

```
GauPro_kernel_model$grad_sample(XX, n)
```

Arguments:

`XX` points to calculate at

`n` Number of samples

Method `grad_norm2_mean()`**:** Calculate mean of gradient norm squared

Usage:

```
GauPro_kernel_model$grad_norm2_mean(XX)
```

Arguments:

`XX` points to calculate at

Method `grad_norm2_dist()`**:** Calculate distribution of gradient norm squared

Usage:

```
GauPro_kernel_model$grad_norm2_dist(XX)
```

Arguments:

`XX` points to calculate at

Method `grad_norm2_sample()`**:** Get samples of squared norm of gradient

Usage:

```
GauPro_kernel_model$grad_norm2_sample(XX, n)
```

Arguments:

`XX` points to sample at

`n` Number of samples

Method `hessian()`**:** Calculate Hessian

Usage:

```
GauPro_kernel_model$hessian(XX, as_array = FALSE)
```

Arguments:

`XX` Points to calculate Hessian at

`as_array` Should result be an array?

Method `sample()`**:** Sample at rows of XX

Usage:

```
GauPro_kernel_model$sample(XX, n = 1)
```

Arguments:

`XX` Input matrix

`n` Number of samples

Method `EI()`**:** Calculate expected improvement

Usage:

```
GauPro_kernel_model$EI(x, minimize = FALSE, eps = 0)
```

Arguments:

`x` Vector to calculate EI of, or matrix for whose rows it should be calculated

`minimize` Are you trying to minimize the output?

`eps` Exploration parameter

Method `maxEI()`**:** Find the point that maximizes the expected improvement

Usage:

```
GauPro_kernel_model$maxEI(
  lower = apply(self$X, 2, min),
  upper = apply(self$X, 2, max),
  n0 = 100,
  minimize = FALSE,
  eps = 0,
  dontconvertback = FALSE,
  discreteinputs = NULL,
  mopar = NULL
)
```

Arguments:

`lower` Lower bounds to search within

`upper` Upper bounds to search within

`n0` Number of points to evaluate in initial stage

`minimize` Are you trying to minimize the output?

`eps` Exploration parameter

`dontconvertback` If data was given in with a formula, should it converted back to the original scale?

`mopar` List of parameters using mixopt

Method `maxEIwithfactors()`**:** Find the point that maximizes the expected improvement. Used whenever one of the inputs is a factor (can only take values 1:n).

Usage:

```
GauPro_kernel_model$maxEIwithfactors(
  lower = apply(self$X, 2, min),
  upper = apply(self$X, 2, max),
  n0 = 100,
  minimize = FALSE,
  eps = 0
)
```

Arguments:

`lower` Lower bounds to search within

`upper` Upper bounds to search within

`n0` Number of points to evaluate in initial stage

`minimize` Are you trying to minimize the output?

`eps` Exploration parameter

Method `maxEIwithfactorsordiscrete()`**:**

Usage:

```
GauPro_kernel_model$maxEIwithfactorsordiscrete(
  lower = apply(self$X, 2, min),
  upper = apply(self$X, 2, max),
  n0 = 100,
  minimize = FALSE,
  eps = 0,
  dontconvertback = FALSE,
  discreteinputs = NULL
)
```

Method `maxqEI()`**:** Find the multiple points that maximize the expected improvement. Currently only implements the constant liar method.

Usage:

```
GauPro_kernel_model$maxqEI(
  npoints,
  method = "CL",
  lower = apply(self$X, 2, min),
  upper = apply(self$X, 2, max),
  n0 = 100,
  minimize = FALSE,
  eps = 0,
  dontconvertback = FALSE,
  mopar = NULL
)
```

Arguments:

`npoints` Number of points to add

`method` Method to use. Can only be "CL" for constant liar.

`lower` Lower bounds to search within

`upper` Upper bounds to search within

`n0` Number of points to evaluate in initial stage

`minimize` Are you trying to minimize the output?

`eps` Exploration parameter

`dontconvertback` If data was given in with a formula, should it converted back to the original scale?

`mopar` List of parameters using mixopt

Method `KG()`**:** Calculate Knowledge Gradient

Usage:

```
GauPro_kernel_model$KG(x, minimize = FALSE, eps = 0, current_extreme = NULL)
```

Arguments:

`x` Point to calculate at

`minimize` Is the objective to minimize?

`eps` Exploration parameter

`current_extreme` Used for recursive solving

Method `importance()`**:** Feature importance

Usage:

```
GauPro_kernel_model$importance(plot = TRUE)
```

Arguments:

`plot` Should the plot be made?

Method `print()`**:** Print this object

Usage:

```
GauPro_kernel_model$print()
```

Method `summary()`**:** Summary

Usage:

```
GauPro_kernel_model$summary(...)
```

Arguments:

`...` Additional arguments

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_kernel_model$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://scikit-learn.org/stable/modules/permutation_importance.html#id2

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Gaussian$new(1),
                              parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()

n <- 200
```

```
d <- 7
x <- matrix(runif(n*d), ncol=d)
f <- function(x) {x[1]*x[2] + cos(x[3]) + x[4]^2}
y <- apply(x, 1, f)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Gaussian)
```

GauPro_kernel_model_LOO *Corr Gauss GP using inherited optim*

Description

Corr Gauss GP using inherited optim

Corr Gauss GP using inherited optim

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro -> GauPro_kernel_model_LOO

Public fields

tmod A second GP model for the t-values of leave-one-out predictions

use_LOO Should the leave-one-out error corrections be used?

Methods

Public methods:

- GauPro_kernel_model_LOO$new()
- GauPro_kernel_model_LOO$update()
- GauPro_kernel_model_LOO$pred_one_matrix()
- GauPro_kernel_model_LOO$clone()

Method new(): Create a kernel model that uses a leave-one-out GP model to fix the standard error predictions.

Usage:

```
GauPro_kernel_model_LOO$new(..., LOO_kernel, LOO_options = list())
```

Arguments:

... Passed to super$initialize.

`LOO_kernel` The kernel that should be used for the leave-one-out model. Shouldn't be too smooth.

`LOO_options` Options passed to the leave-one-out model.

Method `update()`**:** Update the model. Should only give in (Xnew and Znew) or (Xall and Zall).

Usage:

```
GauPro_kernel_model_LOO$update(
  Xnew = NULL,
  Znew = NULL,
  Xall = NULL,
  Zall = NULL,
  restarts = 5,
  param_update = self$param.est,
  nug.update = self$nug.est,
  no_update = FALSE
)
```

Arguments:

`Xnew` New X values to add.

`Znew` New Z values to add.

`Xall` All X values to be used. Will replace existing X.

`Zall` All Z values to be used. Will replace existing Z.

`restarts` Number of optimization restarts.

`param_update` Are the parameters being updated?

`nug.update` Is the nugget being updated?

`no_update` Are no parameters being updated?

Method `pred_one_matrix()`**:** Predict for a matrix of points

Usage:

```
GauPro_kernel_model_LOO$pred_one_matrix(
  XX,
  se.fit = F,
  covmat = F,
  return_df = FALSE,
  mean_dist = FALSE
)
```

Arguments:

`XX` points to predict at

`se.fit` Should standard error be returned?

`covmat` Should covariance matrix be returned?

`return_df` When returning se.fit, should it be returned in a data frame?

`mean_dist` Should mean distribution be returned?

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_kernel_model_LOO$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model_LOO$new(X=x, Z=y, kernel=Gaussian)
y <- x^2 * sin(2*pi*x) + rnorm(n,0,1e-3)
gp <- GauPro_kernel_model_LOO$new(X=x, Z=y, kernel=Matern52)
y <- exp(-1.4*x)*cos(7*pi*x/2)
gp <- GauPro_kernel_model_LOO$new(X=x, Z=y, kernel=Matern52)
```

GauPro_trend *Trend R6 class*

Description

Trend R6 class

Trend R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Public fields

`D` Number of input dimensions of data

Methods

Public methods:

- `GauPro_trend$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
GauPro_trend$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#k <- GauPro_trend$new()
```

Gaussian *Gaussian Kernel R6 class*

Description

Gaussian Kernel R6 class

Gaussian Kernel R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_Gaussian

Methods

Public methods:

- Gaussian$k()
- Gaussian$kone()
- Gaussian$dC_dparams()
- Gaussian$C_dC_dparams()
- Gaussian$dC_dx()
- Gaussian$d2C_dx2()
- Gaussian$d2C_dudv()
- Gaussian$d2C_dudv_ueqvrows()
- Gaussian$print()
- Gaussian$clone()

Method `k()`**:** Calculate covariance between two points

Usage:

```
Gaussian$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Gaussian$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Gaussian$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
Gaussian$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Gaussian$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `d2C_dx2()`**:** Second derivative of covariance with respect to X

Usage:

```
Gaussian$d2C_dx2(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `d2C_dudv()`**:** Second derivative of covariance with respect to X and XX each once.

Usage:

```
Gaussian$d2C_dudv(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `d2C_dudv_ueqvrows()`**:** Second derivative of covariance with respect to X and XX when they equal the same value

Usage:

```
Gaussian$d2C_dudv_ueqvrows(XX, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Gaussian$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Gaussian$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Gaussian$new(beta=0)
plot(k1)
k1 <- Gaussian$new(beta=c(0,-1, 1))
plot(k1)
```

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Gaussian$new(1),
                              parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()
```

`Gaussian_devianceC` *Calculate the Gaussian deviance in C*

Description

Calculate the Gaussian deviance in C

Usage

```
Gaussian_devianceC(theta, nug, X, Z)
```

Arguments

`theta`	Theta vector
`nug`	Nugget
`X`	Matrix X
`Z`	Matrix Z

Value

Correlation matrix

Examples

```
Gaussian_devianceC(c(1,1), 1e-8, matrix(c(1,0,0,1),2,2), matrix(c(1,0),2,1))
```

Gaussian_hessianC *Calculate Hessian for a GP with Gaussian correlation*

Description

Calculate Hessian for a GP with Gaussian correlation

Usage

```
Gaussian_hessianC(XX, X, Z, Kinv, mu_hat, theta)
```

Arguments

XX	The vector at which to calculate the Hessian
X	The input points
Z	The output values
Kinv	The inverse of the correlation matrix
mu_hat	Estimate of mu
theta	Theta parameters for the correlation

Value

Matrix, the Hessian at XX

Examples

```
set.seed(0)
n <- 40
x <- matrix(runif(n*2), ncol=2)
f1 <- function(a) {sin(2*pi*a[1]) + sin(6*pi*a[2])}
y <- apply(x,1,f1) + rnorm(n,0,.01)
gp <- GauPro(x,y, verbose=2, parallel=FALSE);gp$theta
gp$hessian(c(.2,.75), useC=TRUE) # Should be -38.3, -5.96, -5.96, -389.4 as 2x2 matrix
```

Gaussian_hessianCC *Gaussian hessian in C*

Description

Gaussian hessian in C

Usage

```
Gaussian_hessianCC(XX, X, Z, Kinv, mu_hat, theta)
```

Arguments

XX	point to find Hessian at
X	matrix of data points
Z	matrix of output
Kinv	inverse of correlation matrix
mu_hat	mean estimate
theta	correlation parameters

Value

Hessian matrix

Gaussian_hessianR *Calculate Hessian for a GP with Gaussian correlation*

Description

Calculate Hessian for a GP with Gaussian correlation

Usage

```
Gaussian_hessianR(XX, X, Z, Kinv, mu_hat, theta)
```

Arguments

XX	The vector at which to calculate the Hessian
X	The input points
Z	The output values
Kinv	The inverse of the correlation matrix
mu_hat	Estimate of mu
theta	Theta parameters for the correlation

Value

Matrix, the Hessian at XX

Examples

```
set.seed(0)
n <- 40
x <- matrix(runif(n*2), ncol=2)
f1 <- function(a) {sin(2*pi*a[1]) + sin(6*pi*a[2])}
y <- apply(x,1,f1) + rnorm(n,0,.01)
gp <- GauPro(x,y, verbose=2, parallel=FALSE);gp$theta
gp$hessian(c(.2,.75), useC=FALSE) # Should be -38.3, -5.96, -5.96, -389.4 as 2x2 matrix
```

gradfuncarray *Calculate gradfunc in optimization to speed up. NEEDS TO APERM dC_dparams Doesn't need to be exported, should only be useful in functions.*

Description

Calculate gradfunc in optimization to speed up. NEEDS TO APERM dC_dparams Doesn't need to be exported, should only be useful in functions.

Usage

```
gradfuncarray(dC_dparams, Cinv, Cinv_yminusmu)
```

Arguments

dC_dparams	Derivative matrix for covariance function wrt kernel parameters
Cinv	Inverse of covariance matrix
Cinv_yminusmu	Vector that is the inverse of C times y minus the mean.

Value

Vector, one value for each parameter

Examples

```
gradfuncarray(array(dim=c(2,4,4), data=rnorm(32)), matrix(rnorm(16),4,4), rnorm(4))
```

gradfuncarrayR *Calculate gradfunc in optimization to speed up. NEEDS TO APERM dC_dparams Doesn't need to be exported, should only be useful in functions.*

Description

Calculate gradfunc in optimization to speed up. NEEDS TO APERM dC_dparams Doesn't need to be exported, should only be useful in functions.

Usage

```
gradfuncarrayR(dC_dparams, Cinv, Cinv_yminusmu)
```

Arguments

dC_dparams	Derivative matrix for covariance function wrt kernel parameters
Cinv	Inverse of covariance matrix
Cinv_yminusmu	Vector that is the inverse of C times y minus the mean.

Value

Vector, one value for each parameter

Examples

```
a1 <- array(dim=c(2,4,4), data=rnorm(32))
a2 <- matrix(rnorm(16),4,4)
a3 <- rnorm(4)
#gradfuncarray(a1, a2, a3)
#gradfuncarrayR(a1, a2, a3)
```

IgnoreIndsKernel *Kernel R6 class*

Description

Kernel R6 class

Kernel R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro_kernel -> GauPro_kernel_IgnoreInds

Public fields

D Number of input dimensions of data

kernel Kernel to use on indices that aren't ignored

ignoreinds Indices to ignore. For a matrix X, these are the columns to ignore. For example, when those dimensions will be given a different kernel, such as for factors.

Active bindings

`s2_est` Is s2 being estimated?

`s2` Value of s2 (variance)

Methods

Public methods:

- `IgnoreIndsKernel$new()`
- `IgnoreIndsKernel$k()`
- `IgnoreIndsKernel$kone()`
- `IgnoreIndsKernel$dC_dparams()`
- `IgnoreIndsKernel$C_dC_dparams()`
- `IgnoreIndsKernel$param_optim_start()`
- `IgnoreIndsKernel$param_optim_start0()`
- `IgnoreIndsKernel$param_optim_lower()`
- `IgnoreIndsKernel$param_optim_upper()`
- `IgnoreIndsKernel$set_params_from_optim()`
- `IgnoreIndsKernel$s2_from_params()`
- `IgnoreIndsKernel$print()`
- `IgnoreIndsKernel$clone()`

Method `new()`**:** Initialize kernel object

Usage:

```
IgnoreIndsKernel$new(k, ignoreinds)
```

Arguments:

`k` Kernel to use on the non-ignored indices

`ignoreinds` Indices of columns of X to ignore.

Method `k()`**:** Calculate covariance between two points

Usage:

```
IgnoreIndsKernel$k(x, y = NULL, ...)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`...` Passed to kernel

Method `kone()`**:** Find covariance of two points

Usage:

```
IgnoreIndsKernel$kone(x, y, ...)
```

Arguments:

`x` vector

`y` vector

... Passed to kernel

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
IgnoreIndsKernel$dC_dparams(params = NULL, X, ...)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

... Passed to kernel

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
IgnoreIndsKernel$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
IgnoreIndsKernel$param_optim_start(...)
```

Arguments:

... Passed to kernel

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
IgnoreIndsKernel$param_optim_start0(...)
```

Arguments:

... Passed to kernel

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
IgnoreIndsKernel$param_optim_lower(...)
```

Arguments:

... Passed to kernel

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
IgnoreIndsKernel$param_optim_upper(...)
```

Arguments:

... Passed to kernel

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
IgnoreIndsKernel$set_params_from_optim(...)
```

Arguments:

`...` Passed to kernel

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
IgnoreIndsKernel$s2_from_params(...)
```

Arguments:

`...` Passed to kernel

Method `print()`**:** Print this object

Usage:

```
IgnoreIndsKernel$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
IgnoreIndsKernel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
kg <- Gaussian$new(D=3)
kig <- GauPro::IgnoreIndsKernel$new(k = Gaussian$new(D=3), ignoreinds = 2)
Xtmp <- as.matrix(expand.grid(1:2, 1:2, 1:2))
cbind(Xtmp, kig$k(Xtmp))
cbind(Xtmp, kg$k(Xtmp))
```

---

kernel_exponential_dC *Derivative of Matern 5/2 kernel covariance matrix in C*

Description

Derivative of Matern 5/2 kernel covariance matrix in C

Usage

```
kernel_exponential_dC(
  x,
  theta,
  C_nonug,
  s2_est,
  beta_est,
  lenparams_D,
  s2_nug,
  s2
)
```

Arguments

x	Matrix x
theta	Theta vector
C_nonug	cov mat without nugget
s2_est	whether s2 is being estimated
beta_est	Whether theta/beta is being estimated
lenparams_D	Number of parameters the derivative is being calculated for
s2_nug	s2 times the nug
s2	s2 parameter

Value

Correlation matrix

---

kernel_gauss_dC *Derivative of Gaussian kernel covariance matrix in C*

Description

Derivative of Gaussian kernel covariance matrix in C

Usage

```
kernel_gauss_dC(x, theta, C_nonug, s2_est, beta_est, lenparams_D, s2_nug)
```

Arguments

x	Matrix x
theta	Theta vector
C_nonug	cov mat without nugget
s2_est	whether s2 is being estimated

beta_est	Whether theta/beta is being estimated
lenparams_D	Number of parameters the derivative is being calculated for
s2_nug	s2 times the nug

Value

Correlation matrix

`kernel_latentFactor_dC` *Derivative of covariance matrix of X with respect to kernel parameters for the Latent Factor Kernel*

Description

Derivative of covariance matrix of X with respect to kernel parameters for the Latent Factor Kernel

Usage

```
kernel_latentFactor_dC(
  x,
  pf,
  C_nonug,
  s2_est,
  p_est,
  lenparams_D,
  s2_nug,
  latentdim,
  xindex,
  nlevels,
  s2
)
```

Arguments

x	Matrix x
pf	pf vector
C_nonug	cov mat without nugget
s2_est	whether s2 is being estimated
p_est	Whether theta/beta is being estimated
lenparams_D	Number of parameters the derivative is being calculated for
s2_nug	s2 times the nug
latentdim	Number of latent dimensions
xindex	Which column of x is the indexing variable
nlevels	Number of levels
s2	Value of s2

Value

Correlation matrix

`kernel_matern32_dC` *Derivative of Matern 5/2 kernel covariance matrix in C*

Description

Derivative of Matern 5/2 kernel covariance matrix in C

Usage

```
kernel_matern32_dC(x, theta, C_nonug, s2_est, beta_est, lenparams_D, s2_nug)
```

Arguments

`x`	Matrix x
`theta`	Theta vector
`C_nonug`	cov mat without nugget
`s2_est`	whether s2 is being estimated
`beta_est`	Whether theta/beta is being estimated
`lenparams_D`	Number of parameters the derivative is being calculated for
`s2_nug`	s2 times the nug

Value

Correlation matrix

`kernel_matern52_dC` *Derivative of Matern 5/2 kernel covariance matrix in C*

Description

Derivative of Matern 5/2 kernel covariance matrix in C

Usage

```
kernel_matern52_dC(x, theta, C_nonug, s2_est, beta_est, lenparams_D, s2_nug)
```

Arguments

x	Matrix x
theta	Theta vector
C_nonug	cov mat without nugget
s2_est	whether s2 is being estimated
beta_est	Whether theta/beta is being estimated
lenparams_D	Number of parameters the derivative is being calculated for
s2_nug	s2 times the nug

Value

Correlation matrix

---

kernel_product *Gaussian Kernel R6 class*

Description

Gaussian Kernel R6 class

Gaussian Kernel R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro_kernel -> GauPro_kernel_product

Public fields

k1 kernel 1

k2 kernel 2

s2 Variance

Active bindings

k1pl param length of kernel 1

k2pl param length of kernel 2

s2_est Is s2 being estimated?

Methods

Public methods:

- kernel_product$new()
- kernel_product$k()
- kernel_product$param_optim_start()
- kernel_product$param_optim_start0()
- kernel_product$param_optim_lower()
- kernel_product$param_optim_upper()
- kernel_product$set_params_from_optim()
- kernel_product$dC_dparams()
- kernel_product$C_dC_dparams()
- kernel_product$dC_dx()
- kernel_product$s2_from_params()
- kernel_product$print()
- kernel_product$clone()

Method `new()`**:** Is s2 being estimated?
Length of the parameters of k1
Length of the parameters of k2
Initialize kernel

Usage:

```
kernel_product$new(k1, k2)
```

Arguments:

`k1` Kernel 1
`k2` Kernel 2

Method `k()`**:** Calculate covariance between two points

Usage:

```
kernel_product$k(x, y = NULL, params, ...)
```

Arguments:

`x` vector.
`y` vector, optional. If excluded, find correlation of x with itself.
`params` parameters to use instead of beta and s2.
`...` Not used

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
kernel_product$param_optim_start(jitter = F, y)
```

Arguments:

`jitter` Should there be a jitter?
`y` Output

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
kernel_product$param_optim_start0(jitter = F, y)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
kernel_product$param_optim_lower()
```

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
kernel_product$param_optim_upper()
```

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
kernel_product$set_params_from_optim(optim_out)
```

Arguments:

`optim_out` Output from optimization

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
kernel_product$dC_dparams(params = NULL, C, X, C_nonug, nug)
```

Arguments:

`params` Kernel parameters

`C` Covariance with nugget

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
kernel_product$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
kernel_product$dC_dx(XX, X)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
kernel_product$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `print()`**:** Print this object

Usage:

```
kernel_product$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
kernel_product$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Exponential$new(beta=1)
k2 <- Matern32$new(beta=2)
k <- k1 * k2
k$k(matrix(c(2,1), ncol=1))
```

`kernel_sum` *Gaussian Kernel R6 class*

Description

Gaussian Kernel R6 class

Gaussian Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_kernel` -> `GauPro_kernel_sum`

Public fields

`k1` kernel 1

`k2` kernel 2

`k1_param_length` param length of kernel 1

`k2_param_length` param length of kernel 2

`k1pl` param length of kernel 1

`k2pl` param length of kernel 2

`s2` variance

`s2_est` Is s2 being estimated?

Methods

Public methods:

- `kernel_sum$new()`
- `kernel_sum$k()`
- `kernel_sum$param_optim_start()`
- `kernel_sum$param_optim_start0()`
- `kernel_sum$param_optim_lower()`
- `kernel_sum$param_optim_upper()`
- `kernel_sum$set_params_from_optim()`
- `kernel_sum$dC_dparams()`
- `kernel_sum$C_dC_dparams()`
- `kernel_sum$dC_dx()`
- `kernel_sum$s2_from_params()`
- `kernel_sum$print()`
- `kernel_sum$clone()`

Method `new()`**:** Initialize kernel

Usage:

```
kernel_sum$new(k1, k2)
```

Arguments:

`k1` Kernel 1

`k2` Kernel 2

Method `k()`**:** Calculate covariance between two points

Usage:

```
kernel_sum$k(x, y = NULL, params, ...)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`params` parameters to use instead of beta and s2.

`...` Not used

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
kernel_sum$param_optim_start(jitter = F, y)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
kernel_sum$param_optim_start0(jitter = F, y)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
kernel_sum$param_optim_lower()
```

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
kernel_sum$param_optim_upper()
```

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
kernel_sum$set_params_from_optim(optim_out)
```

Arguments:

`optim_out` Output from optimization

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
kernel_sum$dC_dparams(params = NULL, C, X, C_nonug, nug)
```

Arguments:

`params` Kernel parameters

`C` Covariance with nugget

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
kernel_sum$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
kernel_sum$dC_dx(XX, X)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
kernel_sum$s2_from_params(params)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `print()`**:** Print this object

Usage:

```
kernel_sum$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
kernel_sum$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Exponential$new(beta=1)
k2 <- Matern32$new(beta=2)
k <- k1 + k2
k$k(matrix(c(2,1), ncol=1))
```

`LatentFactorKernel` *Latent Factor Kernel R6 class*

Description

Latent Factor Kernel R6 class

Latent Factor Kernel R6 class

Format

`R6Class` object.

Details

Used for factor variables, a single dimension. Each level of the factor gets mapped into a latent space, then the distances in that space determine their correlations.

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_kernel` -> `GauPro_kernel_LatentFactorKernel`

Public fields

`p` Parameter for correlation

`p_est` Should p be estimated?

`p_lower` Lower bound of logp

`p_upper` Upper bound of logp

`p_length` length of p

`s2` variance

`s2_est` Is s2 estimated?

`logs2` Log of s2

`logs2_lower` Lower bound of logs2

`logs2_upper` Upper bound of logs2

`xindex` Index of the factor (which column of X)

`nlevels` Number of levels for the factor

`latentdim` Dimension of embedding space

Methods

Public methods:

- LatentFactorKernel$new()
- LatentFactorKernel$k()
- LatentFactorKernel$kone()
- LatentFactorKernel$dC_dparams()
- LatentFactorKernel$C_dC_dparams()
- LatentFactorKernel$dC_dx()
- LatentFactorKernel$param_optim_start()
- LatentFactorKernel$param_optim_start0()
- LatentFactorKernel$param_optim_lower()
- LatentFactorKernel$param_optim_upper()
- LatentFactorKernel$set_params_from_optim()
- LatentFactorKernel$s2_from_params()
- LatentFactorKernel$plot()
- LatentFactorKernel$plotLatent()
- LatentFactorKernel$print()
- LatentFactorKernel$clone()

Method `new()`**:** Initialize kernel object

Usage:

```
LatentFactorKernel$new(
  s2 = 1,
  D,
  nlevels,
  xindex,
  latentdim,
  p_lower = 0,
  p_upper = 1,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`s2` Initial variance

`D` Number of input dimensions of data

`nlevels` Number of levels for the factor

`xindex` Index of X to use the kernel on

`latentdim` Dimension of embedding space

`p_lower` Lower bound for p

`p_upper` Upper bound for p

`p_est` Should p be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

`p` Periodic parameter

Method `k()`**:** Calculate covariance between two points

Usage:

```
LatentFactorKernel$k(x, y = NULL, p = self$p, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`p` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
LatentFactorKernel$kone(x, y, pf, s2, isdiag = TRUE, offdiagequal = 1 - 1e-06)
```

Arguments:

`x` vector

`y` vector

`pf` correlation parameters on regular scale, includes zeroes for first level.

`s2` Variance parameter

`isdiag` Is this on the diagonal of the covariance?

`offdiagequal` What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
LatentFactorKernel$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
LatentFactorKernel$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
LatentFactorKernel$dC_dx(
  XX,
  X,
  logp = self$logp,
  logalpha = self$logalpha,
  s2 = self$s2
)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`logp` log of p

`logalpha` log of alpha

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
LatentFactorKernel$param_optim_start(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
LatentFactorKernel$param_optim_start0(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
LatentFactorKernel$param_optim_lower(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
LatentFactorKernel$param_optim_upper(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
LatentFactorKernel$set_params_from_optim(
  optim_out,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
LatentFactorKernel$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `plot()`**:**

Usage:

```
LatentFactorKernel$plot(...)
```

Arguments:

`...` Not used.

Method `plotLatent()`**:** Plot the points in the latent space

Usage:

```
LatentFactorKernel$plotLatent()
```

Method `print()`**:** Print this object

Usage:

```
LatentFactorKernel$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LatentFactorKernel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://stackoverflow.com/questions/27086195/linear-index-upper-triangular-matrix

Examples

```
# Create a new kernel for a single factor with 5 levels,
#  mapped into two latent dimensions.
kk <- LatentFactorKernel$new(D=1, nlevels=5, xindex=1, latentdim=2)
# Random initial parameter values
kk$p
# Plots to understand
kk$plotLatent()
kk$plot()


# 5 levels, 1/4 are similar and 2/3/5 are similar
n <- 30
x <- matrix(sample(1:5, n, TRUE))
y <- c(ifelse(x == 1 | x == 4, 4, -3) + rnorm(n,0,.1))
plot(c(x), y)
m5 <- GauPro_kernel_model$new(
  X=x, Z=y,
  kernel=LatentFactorKernel$new(D=1, nlevels = 5, xindex = 1, latentdim = 2))
m5$kernel$p
# We should see 1/4 and 2/3/4 in separate clusters
m5$kernel$plotLatent()

library(dplyr)
n <- 20
X <- cbind(matrix(runif(n,2,6), ncol=1),
```

```
          matrix(sample(1:2, size=n, replace=TRUE), ncol=1))
X <- rbind(X, c(3.3,3), c(3.7,3))
n <- nrow(X)
Z <- X[,1] - (4-X[,2])^2 + rnorm(n,0,.1)
plot(X[,1], Z, col=X[,2])
tibble(X=X, Z) %>% arrange(X,Z)
k2a <- IgnoreIndsKernel$new(k=Gaussian$new(D=1), ignoreinds = 2)
k2b <- LatentFactorKernel$new(D=2, nlevels=3, xind=2, latentdim=2)
k2 <- k2a * k2b
k2b$p_upper <- .65*k2b$p_upper
gp <- GauPro_kernel_model$new(X=X, Z=Z, kernel = k2, verbose = 5,
  nug.min=1e-2, restarts=1)
gp$kernel$k1$kernel$beta
gp$kernel$k2$p
gp$kernel$k(x = gp$X)
tibble(X=X, Z=Z, pred=gp$predict(X)) %>% arrange(X, Z)
tibble(X=X[,2], Z) %>% group_by(X) %>% summarize(n=n(), mean(Z))
curve(gp$pred(cbind(matrix(x,ncol=1),1)),2,6, ylim=c(min(Z), max(Z)))
points(X[X[,2]==1,1], Z[X[,2]==1])
curve(gp$pred(cbind(matrix(x,ncol=1),2)), add=TRUE, col=2)
points(X[X[,2]==2,1], Z[X[,2]==2], col=2)
curve(gp$pred(cbind(matrix(x,ncol=1),3)), add=TRUE, col=3)
points(X[X[,2]==3,1], Z[X[,2]==3], col=3)
legend(legend=1:3, fill=1:3, x="topleft")
# See which points affect (5.5, 3 themost)
data.frame(X, cov=gp$kernel$k(X, c(5.5,3))) %>% arrange(-cov)
plot(k2b)
```

`Matern32` *Matern 3/2 Kernel R6 class*

Description

Matern 3/2 Kernel R6 class

Matern 3/2 Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super classes

`GauPro::GauPro_kernel` -> `GauPro::GauPro_kernel_beta` -> `GauPro_kernel_Matern32`

Public fields

`sqrt3` Saved value of square root of 3

Methods

Public methods:

- `Matern32$k()`
- `Matern32$kone()`
- `Matern32$dC_dparams()`
- `Matern32$dC_dx()`
- `Matern32$print()`
- `Matern32$clone()`

Method `k()`**:** Calculate covariance between two points

Usage:

```
Matern32$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Matern32$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Matern32$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Matern32$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Matern32$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Matern32$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Matern32$new(beta=0)
plot(k1)

n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Matern32$new(1),
                              parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()
```

`Matern52` *Matern 5/2 Kernel R6 class*

Description

Matern 5/2 Kernel R6 class

Matern 5/2 Kernel R6 class

Format

`R6Class` object.

Value

Object of R6Class with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_Matern52

Public fields

sqrt5 Saved value of square root of 5

Methods

Public methods:

- Matern52$k()
- Matern52$kone()
- Matern52$dC_dparams()
- Matern52$dC_dx()
- Matern52$print()
- Matern52$clone()

Method k(): Calculate covariance between two points

Usage:

```
Matern52$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

x vector.

y vector, optional. If excluded, find correlation of x with itself.

beta Correlation parameters.

s2 Variance parameter.

params parameters to use instead of beta and s2.

Method kone(): Find covariance of two points

Usage:

```
Matern52$kone(x, y, beta, theta, s2)
```

Arguments:

x vector

y vector

beta correlation parameters on log scale

theta correlation parameters on regular scale

s2 Variance parameter

Method dC_dparams(): Derivative of covariance with respect to parameters

Usage:

```
Matern52$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Matern52$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Matern52$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Matern52$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Matern52$new(beta=0)
plot(k1)

n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Matern52$new(1),
                              parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()
```

`OrderedFactorKernel` *Periodic Kernel R6 class*

Description

Periodic Kernel R6 class

Periodic Kernel R6 class

Format

`R6Class` object.

Details

p is the period for each dimension, a is a single number for scaling

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_kernel` -> `GauPro_kernel_OrderedFactorKernel`

Public fields

`p` Parameter for correlation

`p_est` Should p be estimated?

`p_lower` Lower bound of logp

`p_upper` Upper bound of logp

`p_length` length of p

`s2` variance

`s2_est` Is s2 estimated?

`logs2` Log of s2

`logs2_lower` Lower bound of logs2

`logs2_upper` Upper bound of logs2

`xindex` Index of the factor (which column of X)

`nlevels` Number of levels for the factor

Methods

Public methods:

- OrderedFactorKernel$new()
- OrderedFactorKernel$k()
- OrderedFactorKernel$kone()
- OrderedFactorKernel$dC_dparams()
- OrderedFactorKernel$C_dC_dparams()
- OrderedFactorKernel$dC_dx()
- OrderedFactorKernel$param_optim_start()
- OrderedFactorKernel$param_optim_start0()
- OrderedFactorKernel$param_optim_lower()
- OrderedFactorKernel$param_optim_upper()
- OrderedFactorKernel$set_params_from_optim()
- OrderedFactorKernel$s2_from_params()
- OrderedFactorKernel$plot()
- OrderedFactorKernel$print()
- OrderedFactorKernel$clone()

Method `new()`**:** Initialize kernel object

Usage:

```
OrderedFactorKernel$new(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 1,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`s2` Initial variance

`D` Number of input dimensions of data

`nlevels` Number of levels for the factor

`xindex` Index of X to use the kernel on

`p_lower` Lower bound for p

`p_upper` Upper bound for p

`p_est` Should p be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

p Periodic parameter

Method `k()`**:** Calculate covariance between two points

Usage:

```
OrderedFactorKernel$k(x, y = NULL, p = self$p, s2 = self$s2, params = NULL)
```

Arguments:

x vector.

y vector, optional. If excluded, find correlation of x with itself.

p Correlation parameters.

s2 Variance parameter.

params parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
OrderedFactorKernel$kone(x, y, p, s2, isdiag = TRUE, offdiagequal = 1 - 1e-06)
```

Arguments:

x vector

y vector

p correlation parameters on regular scale

s2 Variance parameter

isdiag Is this on the diagonal of the covariance?

offdiagequal What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
OrderedFactorKernel$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

params Kernel parameters

X matrix of points in rows

C_nonug Covariance without nugget added to diagonal

C Covariance with nugget

nug Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
OrderedFactorKernel$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

params Kernel parameters

X matrix of points in rows

nug Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
OrderedFactorKernel$dC_dx(
  XX,
  X,
  logp = self$logp,
  logalpha = self$logalpha,
  s2 = self$s2
)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`logp` log of p

`logalpha` log of alpha

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
OrderedFactorKernel$param_optim_start(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
OrderedFactorKernel$param_optim_start0(
  jitter = F,
  y,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
OrderedFactorKernel$param_optim_lower(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
OrderedFactorKernel$param_optim_upper(p_est = self$p_est, s2_est = self$s2_est)
```

Arguments:

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
OrderedFactorKernel$set_params_from_optim(
  optim_out,
  p_est = self$p_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`p_est` Is p being estimated?

`s2_est` Is s2 being estimated?

`alpha_est` Is alpha being estimated?

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
OrderedFactorKernel$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `plot()`**:**

Usage:

```
OrderedFactorKernel$plot(...)
```

Arguments:

`...` Not used.

Method `print()`**:** Print this object

Usage:

```
OrderedFactorKernel$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OrderedFactorKernel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://stackoverflow.com/questions/27086195/linear-index-upper-triangular-matrix

Examples

```
kk <- OrderedFactorKernel$new(D=1, nlevels=5, xindex=1)
kk$p <- (1:10)/100
kmat <- outer(1:5, 1:5, Vectorize(kk$k))
kmat


library(dplyr)
n <- 20
X <- cbind(matrix(runif(n,2,6), ncol=1),
           matrix(sample(1:2, size=n, replace=TRUE), ncol=1))
X <- rbind(X, c(3.3,3), c(3.7,3))
n <- nrow(X)
Z <- X[,1] - (4-X[,2])^2 + rnorm(n,0,.1)
plot(X[,1], Z, col=X[,2])
tibble(X=X, Z) %>% arrange(X,Z)
k2a <- IgnoreIndsKernel$new(k=Gaussian$new(D=1), ignoreinds = 2)
k2b <- OrderedFactorKernel$new(D=2, nlevels=3, xind=2)
k2 <- k2a * k2b
k2b$p_upper <- .65*k2b$p_upper
gp <- GauPro_kernel_model$new(X=X, Z=Z, kernel = k2, verbose = 5,
  nug.min=1e-2, restarts=0)
gp$kernel$k1$kernel$beta
gp$kernel$k2$p
gp$kernel$k(x = gp$X)
tibble(X=X, Z=Z, pred=gp$predict(X)) %>% arrange(X, Z)
tibble(X=X[,2], Z) %>% group_by(X) %>% summarize(n=n(), mean(Z))
curve(gp$pred(cbind(matrix(x,ncol=1),1)),2,6, ylim=c(min(Z), max(Z)))
points(X[X[,2]==1,1], Z[X[,2]==1])
curve(gp$pred(cbind(matrix(x,ncol=1),2)), add=TRUE, col=2)
points(X[X[,2]==2,1], Z[X[,2]==2], col=2)
curve(gp$pred(cbind(matrix(x,ncol=1),3)), add=TRUE, col=3)
points(X[X[,2]==3,1], Z[X[,2]==3], col=3)
legend(legend=1:3, fill=1:3, x="topleft")
# See which points affect (5.5, 3 themost)
data.frame(X, cov=gp$kernel$k(X, c(5.5,3))) %>% arrange(-cov)
```

```
plot(k2b)
```

`Periodic` *Periodic Kernel R6 class*

Description

Periodic Kernel R6 class

Periodic Kernel R6 class

Format

`R6Class` object.

Details

p is the period for each dimension, a is a single number for scaling

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_kernel` -> `GauPro_kernel_Periodic`

Public fields

`p` Parameter for correlation

`p_est` Should p be estimated?

`logp` Log of p

`logp_lower` Lower bound of logp

`logp_upper` Upper bound of logp

`p_length` length of p

`alpha` Parameter for correlation

`alpha_est` Should alpha be estimated?

`logalpha` Log of alpha

`logalpha_lower` Lower bound of logalpha

`logalpha_upper` Upper bound of logalpha

`s2` variance

`s2_est` Is s2 estimated?

`logs2` Log of s2

`logs2_lower` Lower bound of logs2

`logs2_upper` Upper bound of logs2

Methods

Public methods:

- Periodic$new()
- Periodic$k()
- Periodic$kone()
- Periodic$dC_dparams()
- Periodic$C_dC_dparams()
- Periodic$dC_dx()
- Periodic$param_optim_start()
- Periodic$param_optim_start0()
- Periodic$param_optim_lower()
- Periodic$param_optim_upper()
- Periodic$set_params_from_optim()
- Periodic$s2_from_params()
- Periodic$print()
- Periodic$clone()

Method `new()`**:** Initialize kernel object

Usage:

```
Periodic$new(
  p,
  alpha = 1,
  s2 = 1,
  D,
  p_lower = 0,
  p_upper = 100,
  p_est = TRUE,
  alpha_lower = 0,
  alpha_upper = 100,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`p` Periodic parameter

`alpha` Periodic parameter

`s2` Initial variance

`D` Number of input dimensions of data

`p_lower` Lower bound for p

`p_upper` Upper bound for p

`p_est` Should p be estimated?

`alpha_lower` Lower bound for alpha

`alpha_upper` Upper bound for alpha

`alpha_est` Should alpha be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

Method `k()`**:** Calculate covariance between two points

Usage:

```
Periodic$k(
  x,
  y = NULL,
  logp = self$logp,
  logalpha = self$logalpha,
  s2 = self$s2,
  params = NULL
)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`logp` Correlation parameters.

`logalpha` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Periodic$kone(x, y, logp, p, alpha, s2)
```

Arguments:

`x` vector

`y` vector

`logp` correlation parameters on log scale

`p` correlation parameters on regular scale

`alpha` correlation parameter

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Periodic$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
Periodic$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Periodic$dC_dx(XX, X, logp = self$logp, logalpha = self$logalpha, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`logp` log of p

`logalpha` log of alpha

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
Periodic$param_optim_start(
  jitter = F,
  y,
  p_est = self$p_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
Periodic$param_optim_start0(
  jitter = F,
  y,
  p_est = self$p_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`p_est` Is p being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
Periodic$param_optim_lower(
  p_est = self$p_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`p_est` Is p being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
Periodic$param_optim_upper(
  p_est = self$p_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`p_est` Is p being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
Periodic$set_params_from_optim(
  optim_out,
  p_est = self$p_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`p_est` Is p being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
Periodic$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector

`s2_est` Is s2 being estimated?

Method `print()`**:** Print this object

Usage:

```
Periodic$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Periodic$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Periodic$new(p=1, alpha=1)
plot(k1)

n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Periodic$new(D=1),
                              parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()
plot(gp$kernel)
```

`PowerExp` *Power Exponential Kernel R6 class*

Description

Power Exponential Kernel R6 class

Power Exponential Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_PowerExp

Public fields

`alpha` alpha value (the exponent). Between 0 and 2.

`alpha_lower` Lower bound for alpha

`alpha_upper` Upper bound for alpha

`alpha_est` Should alpha be estimated?

Methods

Public methods:

- PowerExp$new()
- PowerExp$k()
- PowerExp$kone()
- PowerExp$dC_dparams()
- PowerExp$dC_dx()
- PowerExp$param_optim_start()
- PowerExp$param_optim_start0()
- PowerExp$param_optim_lower()
- PowerExp$param_optim_upper()
- PowerExp$set_params_from_optim()
- PowerExp$print()
- PowerExp$clone()

Method `new()`**:** Initialize kernel object

Usage:

```
PowerExp$new(
  alpha = 1.95,
  beta,
  s2 = 1,
  D,
  beta_lower = -8,
  beta_upper = 6,
  beta_est = TRUE,
  alpha_lower = 0,
  alpha_upper = 2,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`alpha` Initial alpha value (the exponent). Between 0 and 2.

`beta` Initial beta value

`s2` Initial variance

`D` Number of input dimensions of data

`beta_lower` Lower bound for beta

`beta_upper` Upper bound for beta

`beta_est` Should beta be estimated?

`alpha_lower` Lower bound for alpha

`alpha_upper` Upper bound for alpha

`alpha_est` Should alpha be estimated?

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

Method `k()`**:** Calculate covariance between two points

Usage:

```
PowerExp$k(
  x,
  y = NULL,
  beta = self$beta,
  alpha = self$alpha,
  s2 = self$s2,
  params = NULL
)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`alpha` alpha value (the exponent). Between 0 and 2.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
PowerExp$kone(x, y, beta, theta, alpha, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`alpha` alpha value (the exponent). Between 0 and 2.

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
PowerExp$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
PowerExp$dC_dx(
  XX,
  X,
  theta,
  beta = self$beta,
  alpha = self$alpha,
  s2 = self$s2
)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`alpha` alpha value (the exponent). Between 0 and 2.

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
PowerExp$param_optim_start(
  jitter = F,
  y,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
PowerExp$param_optim_start0(
  jitter = F,
  y,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
PowerExp$param_optim_lower(
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
PowerExp$param_optim_upper(
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
PowerExp$set_params_from_optim(
  optim_out,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`beta_est` Is beta estimate?

`alpha_est` Is alpha estimated?

`s2_est` Is s2 estimated?

Method `print()`**:** Print this object

Usage:

```
PowerExp$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PowerExp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- PowerExp$new(beta=0, alpha=0)
```

`predict.GauPro` *Predict for class GauPro*

Description

Predict for class GauPro

Usage

```
## S3 method for class 'GauPro'
predict(object, XX, se.fit = F, covmat = F, split_speed = T, ...)
```

Arguments

`object`	Object of class GauPro
`XX`	new points to predict
`se.fit`	Should standard error be returned (and variance)?
`covmat`	Should the covariance matrix be returned?
`split_speed`	Should the calculation be split up to speed it up?
`...`	Additional parameters

Value

Prediction from object at XX

Examples

```
n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro(X=x, Z=y, parallel=FALSE)
predict(gp, .448)
```

---

print.summary.GauPro *Print summary.GauPro*

Description

Print summary.GauPro

Usage

```
## S3 method for class 'summary.GauPro'
print(x, ...)
```

Arguments

x	summary.GauPro object
...	Additional args

RatQuad *Rational Quadratic Kernel R6 class*

Description

Rational Quadratic Kernel R6 class

Rational Quadratic Kernel R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super classes

GauPro::GauPro_kernel -> GauPro::GauPro_kernel_beta -> GauPro_kernel_RatQuad

Public fields

`alpha` alpha value (the exponent). Between 0 and 2.

`logalpha` Log of alpha

`logalpha_lower` Lower bound for log of alpha

`logalpha_upper` Upper bound for log of alpha

`alpha_est` Should alpha be estimated?

Methods

Public methods:

- `RatQuad$new()`
- `RatQuad$k()`
- `RatQuad$kone()`
- `RatQuad$dC_dparams()`
- `RatQuad$dC_dx()`
- `RatQuad$param_optim_start()`
- `RatQuad$param_optim_start0()`
- `RatQuad$param_optim_lower()`
- `RatQuad$param_optim_upper()`
- `RatQuad$set_params_from_optim()`
- `RatQuad$print()`
- `RatQuad$clone()`

Method `new()`**:** Initialize kernel object

Usage:

```
RatQuad$new(
  beta,
  alpha = 1,
  s2 = 1,
  D,
  beta_lower = -8,
  beta_upper = 6,
  beta_est = TRUE,
  alpha_lower = 0,
  alpha_upper = Inf,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE
)
```

Arguments:

`beta` Initial beta value

`alpha` Initial alpha value

`s2` Initial variance

`D` Number of input dimensions of data
`beta_lower` Lower bound for beta
`beta_upper` Upper bound for beta
`beta_est` Should beta be estimated?
`alpha_lower` Lower bound for alpha
`alpha_upper` Upper bound for alpha
`alpha_est` Should alpha be estimated?
`s2_lower` Lower bound for s2
`s2_upper` Upper bound for s2
`s2_est` Should s2 be estimated?

Method `k()`**:** Calculate covariance between two points

Usage:

```
RatQuad$k(
  x,
  y = NULL,
  beta = self$beta,
  logalpha = self$logalpha,
  s2 = self$s2,
  params = NULL
)
```

Arguments:

`x` vector.
`y` vector, optional. If excluded, find correlation of x with itself.
`beta` Correlation parameters.
`logalpha` A correlation parameter
`s2` Variance parameter.
`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
RatQuad$kone(x, y, beta, theta, alpha, s2)
```

Arguments:

`x` vector
`y` vector
`beta` correlation parameters on log scale
`theta` correlation parameters on regular scale
`alpha` A correlation parameter
`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
RatQuad$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
RatQuad$dC_dx(XX, X, theta, beta = self$beta, alpha = self$alpha, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`alpha` parameter

`s2` Variance parameter

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
RatQuad$param_optim_start(
  jitter = F,
  y,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
RatQuad$param_optim_start0(
  jitter = F,
  y,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
RatQuad$param_optim_lower(
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
RatQuad$param_optim_upper(
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
RatQuad$set_params_from_optim(
  optim_out,
  beta_est = self$beta_est,
  alpha_est = self$alpha_est,
  s2_est = self$s2_est
)
```

Arguments:

`optim_out` Output from optimization

`beta_est` Is beta being estimated?

`alpha_est` Is alpha being estimated?

`s2_est` Is s2 being estimated?

Method `print()`**:** Print this object

Usage:

```
RatQuad$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RatQuad$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- RatQuad$new(beta=0, alpha=0)
```

`sqrt_matrix` *Find the square root of a matrix*

Description

Same thing as 'expm::sqrtm', but faster.

Usage

```
sqrt_matrix(mat, symmetric)
```

Arguments

`mat`	Matrix to find square root matrix of
`symmetric`	Is it symmetric? Passed to eigen.

Value

Square root of mat

Examples

```
mat <- matrix(c(1,.1,.1,1), 2, 2)
smat <- sqrt_matrix(mat=mat, symmetric=TRUE)
smat %*% smat
```

summary.GauPro *Summary for GauPro object*

Description

Summary for GauPro object

Usage

```
## S3 method for class 'GauPro'
summary(object, ...)
```

Arguments

object	GauPro R6 object
...	Additional arguments passed to summary

Value

Summary

trend_0 *Trend R6 class*

Description

Trend R6 class

Trend R6 class

Format

R6Class object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro_trend -> GauPro_trend_0

Public fields

m Trend parameters

m_lower m lower bound

m_upper m upper bound

m_est Should m be estimated?

Methods

Public methods:

- trend_0$new()
- trend_0$Z()
- trend_0$dZ_dparams()
- trend_0$dZ_dx()
- trend_0$param_optim_start()
- trend_0$param_optim_start0()
- trend_0$param_optim_lower()
- trend_0$param_optim_upper()
- trend_0$set_params_from_optim()
- trend_0$clone()

Method `new()`**:** Initialize trend object

Usage:

```
trend_0$new(m = 0, m_lower = 0, m_upper = 0, m_est = FALSE, D = NA)
```

Arguments:

`m` trend initial parameters

`m_lower` trend lower bounds

`m_upper` trend upper bounds

`m_est` Logical of whether each param should be estimated

`D` Number of input dimensions of data

Method `Z()`**:** Get trend value for given matrix X

Usage:

```
trend_0$Z(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend parameters

`params` trend parameters

Method `dZ_dparams()`**:** Derivative of trend with respect to trend parameters

Usage:

```
trend_0$dZ_dparams(X, m = m$est, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`params` overrides m

Method `dZ_dx()`**:** Derivative of trend with respect to X

Usage:

```
trend_0$dZ_dx(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`params` overrides m

Method `param_optim_start()`**:** Get parameter initial point for optimization

Usage:

```
trend_0$param_optim_start(jitter, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `param_optim_start0()`**:** Get parameter initial point for optimization

Usage:

```
trend_0$param_optim_start0(jitter, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `param_optim_lower()`**:** Get parameter lower bounds for optimization

Usage:

```
trend_0$param_optim_lower(jitter, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `param_optim_upper()`**:** Get parameter upper bounds for optimization

Usage:

```
trend_0$param_optim_upper(jitter, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `set_params_from_optim()`**:** Set parameters after optimization

Usage:

```
trend_0$set_params_from_optim(optim_out)
```

Arguments:

`optim_out` Output from optim

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
trend_0$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
t1 <- trend_0$new()
```

`trend_c` *Trend R6 class*

Description

Trend R6 class

Trend R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_trend` -> `GauPro_trend_c`

Public fields

`m` Trend parameters

`m_lower` m lower bound

`m_upper` m upper bound

`m_est` Should m be estimated?

Methods

Public methods:

- `trend_c$new()`
- `trend_c$Z()`
- `trend_c$dZ_dparams()`
- `trend_c$dZ_dx()`
- `trend_c$param_optim_start()`
- `trend_c$param_optim_start0()`
- `trend_c$param_optim_lower()`
- `trend_c$param_optim_upper()`
- `trend_c$set_params_from_optim()`
- `trend_c$clone()`

Method `new()`**:** Initialize trend object

Usage:

```
trend_c$new(m = 0, m_lower = -Inf, m_upper = Inf, m_est = TRUE, D = NA)
```

Arguments:

`m` trend initial parameters

`m_lower` trend lower bounds

`m_upper` trend upper bounds

`m_est` Logical of whether each param should be estimated

`D` Number of input dimensions of data

Method `Z()`**:** Get trend value for given matrix X

Usage:

```
trend_c$Z(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend parameters

`params` trend parameters

Method `dZ_dparams()`**:** Derivative of trend with respect to trend parameters

Usage:

```
trend_c$dZ_dparams(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`params` overrides m

Method `dZ_dx()`**:** Derivative of trend with respect to X

Usage:

```
trend_c$dZ_dx(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`params` overrides m

Method `param_optim_start()`**:** Get parameter initial point for optimization

Usage:

```
trend_c$param_optim_start(jitter, trend_est = self$m_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `param_optim_start0()`**:** Get parameter initial point for optimization

Usage:

```
trend_c$param_optim_start0(jitter, trend_est = self$m_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimate.

Method `param_optim_lower()`**:** Get parameter lower bounds for optimization

Usage:

```
trend_c$param_optim_lower(trend_est = self$m_est)
```

Arguments:

`trend_est` If the trend should be estimate.

Method `param_optim_upper()`**:** Get parameter upper bounds for optimization

Usage:

```
trend_c$param_optim_upper(trend_est = self$m_est)
```

Arguments:

`trend_est` If the trend should be estimate.

Method `set_params_from_optim()`**:** Set parameters after optimization

Usage:

```
trend_c$set_params_from_optim(optim_out)
```

Arguments:

`optim_out` Output from optim

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
trend_c$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
t1 <- trend_c$new()
```

`trend_LM` *Trend R6 class*

Description

Trend R6 class

Trend R6 class

Format

`R6Class` object.

Value

Object of R6Class with methods for fitting GP model.

Super class

GauPro::GauPro_trend -> GauPro_trend_LM

Public fields

m Trend parameters

m_lower m lower bound

m_upper m upper bound

m_est Should m be estimated?

b trend parameter

b_lower trend lower bounds

b_upper trend upper bounds

b_est Should b be estimated?

Methods

Public methods:

- trend_LM$new()
- trend_LM$Z()
- trend_LM$dZ_dparams()
- trend_LM$dZ_dx()
- trend_LM$param_optim_start()
- trend_LM$param_optim_start0()
- trend_LM$param_optim_lower()
- trend_LM$param_optim_upper()
- trend_LM$set_params_from_optim()
- trend_LM$clone()

Method new(): Initialize trend object

Usage:

```
trend_LM$new(
  D,
  m = rep(0, D),
  m_lower = rep(-Inf, D),
  m_upper = rep(Inf, D),
  m_est = rep(TRUE, D),
  b = 0,
  b_lower = -Inf,
  b_upper = Inf,
  b_est = TRUE
)
```

Arguments:

`D` Number of input dimensions of data

`m` trend initial parameters

`m_lower` trend lower bounds

`m_upper` trend upper bounds

`m_est` Logical of whether each param should be estimated

`b` trend parameter

`b_lower` trend lower bounds

`b_upper` trend upper bounds

`b_est` Should b be estimated?

Method `Z()`**:** Get trend value for given matrix X

Usage:

```
trend_LM$Z(X, m = self$m, b = self$b, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend parameters

`b` trend parameters (slopes)

`params` trend parameters

Method `dZ_dparams()`**:** Derivative of trend with respect to trend parameters

Usage:

```
trend_LM$dZ_dparams(X, m = self$m_est, b = self$b_est, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`b` trend intercept

`params` overrides m

Method `dZ_dx()`**:** Derivative of trend with respect to X

Usage:

```
trend_LM$dZ_dx(X, m = self$m, params = NULL)
```

Arguments:

`X` matrix of points

`m` trend values

`params` overrides m

Method `param_optim_start()`**:** Get parameter initial point for optimization

Usage:

```
trend_LM$param_optim_start(jitter = FALSE, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimated.

Method `param_optim_start0()`**:** Get parameter initial point for optimization

Usage:

```
trend_LM$param_optim_start0(jitter, trend_est)
```

Arguments:

`jitter` Not used

`trend_est` If the trend should be estimated.

Method `param_optim_lower()`**:** Get parameter lower bounds for optimization

Usage:

```
trend_LM$param_optim_lower(trend_est)
```

Arguments:

`trend_est` If the trend should be estimated.

Method `param_optim_upper()`**:** Get parameter upper bounds for optimization

Usage:

```
trend_LM$param_optim_upper(trend_est)
```

Arguments:

`trend_est` If the trend should be estimated.

Method `set_params_from_optim()`**:** Set parameters after optimization

Usage:

```
trend_LM$set_params_from_optim(optim_out)
```

Arguments:

`optim_out` Output from optim

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
trend_LM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
t1 <- trend_LM$new(D=2)
```

Triangle *Triangle Kernel R6 class*

Description

Triangle Kernel R6 class

Triangle Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super classes

`GauPro::GauPro_kernel` -> `GauPro::GauPro_kernel_beta` -> `GauPro_kernel_Triangle`

Methods

Public methods:

- `Triangle$k()`
- `Triangle$kone()`
- `Triangle$dC_dparams()`
- `Triangle$dC_dx()`
- `Triangle$print()`
- `Triangle$clone()`

Method `k()`**:** Calculate covariance between two points

Usage:

```
Triangle$k(x, y = NULL, beta = self$beta, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`beta` Correlation parameters.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
Triangle$kone(x, y, beta, theta, s2)
```

Arguments:

`x` vector

`y` vector

`beta` correlation parameters on log scale

`theta` correlation parameters on regular scale

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
Triangle$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
Triangle$dC_dx(XX, X, theta, beta = self$beta, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`theta` Correlation parameters

`beta` log of theta

`s2` Variance parameter

Method `print()`**:** Print this object

Usage:

```
Triangle$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Triangle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- Triangle$new(beta=0)
plot(k1)

n <- 12
x <- matrix(seq(0,1,length.out = n), ncol=1)
y <- sin(2*pi*x) + rnorm(n,0,1e-1)
gp <- GauPro_kernel_model$new(X=x, Z=y, kernel=Triangle$new(1),
```

```
                                parallel=FALSE)
gp$predict(.454)
gp$plot1D()
gp$cool1Dplot()
```

`White` *White noise Kernel R6 class*

Description

White noise Kernel R6 class

White noise Kernel R6 class

Format

`R6Class` object.

Value

Object of `R6Class` with methods for fitting GP model.

Super class

`GauPro::GauPro_kernel` -> `GauPro_kernel_White`

Public fields

`s2` variance

`logs2` Log of s2

`logs2_lower` Lower bound of logs2

`logs2_upper` Upper bound of logs2

`s2_est` Should s2 be estimated?

Methods

Public methods:

- `White$new()`
- `White$k()`
- `White$kone()`
- `White$dC_dparams()`
- `White$C_dC_dparams()`
- `White$dC_dx()`
- `White$param_optim_start()`
- `White$param_optim_start0()`
- `White$param_optim_lower()`

- White$param_optim_upper()
- White$set_params_from_optim()
- White$s2_from_params()
- White$print()
- White$clone()

Method `new()`**:** Initialize kernel object

Usage:

```
White$new(s2 = 1, D, s2_lower = 1e-08, s2_upper = 1e+08, s2_est = TRUE)
```

Arguments:

`s2` Initial variance

`D` Number of input dimensions of data

`s2_lower` Lower bound for s2

`s2_upper` Upper bound for s2

`s2_est` Should s2 be estimated?

Method `k()`**:** Calculate covariance between two points

Usage:

```
White$k(x, y = NULL, s2 = self$s2, params = NULL)
```

Arguments:

`x` vector.

`y` vector, optional. If excluded, find correlation of x with itself.

`s2` Variance parameter.

`params` parameters to use instead of beta and s2.

Method `kone()`**:** Find covariance of two points

Usage:

```
White$kone(x, y, s2)
```

Arguments:

`x` vector

`y` vector

`s2` Variance parameter

Method `dC_dparams()`**:** Derivative of covariance with respect to parameters

Usage:

```
White$dC_dparams(params = NULL, X, C_nonug, C, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`C_nonug` Covariance without nugget added to diagonal

`C` Covariance with nugget

`nug` Value of nugget

Method `C_dC_dparams()`**:** Calculate covariance matrix and its derivative with respect to parameters

Usage:

```
White$C_dC_dparams(params = NULL, X, nug)
```

Arguments:

`params` Kernel parameters

`X` matrix of points in rows

`nug` Value of nugget

Method `dC_dx()`**:** Derivative of covariance with respect to X

Usage:

```
White$dC_dx(XX, X, s2 = self$s2)
```

Arguments:

`XX` matrix of points

`X` matrix of points to take derivative with respect to

`s2` Variance parameter

`theta` Correlation parameters

`beta` log of theta

Method `param_optim_start()`**:** Starting point for parameters for optimization

Usage:

```
White$param_optim_start(jitter = F, y, s2_est = self$s2_est)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`s2_est` Is s2 being estimated?

Method `param_optim_start0()`**:** Starting point for parameters for optimization

Usage:

```
White$param_optim_start0(jitter = F, y, s2_est = self$s2_est)
```

Arguments:

`jitter` Should there be a jitter?

`y` Output

`s2_est` Is s2 being estimated?

Method `param_optim_lower()`**:** Lower bounds of parameters for optimization

Usage:

```
White$param_optim_lower(s2_est = self$s2_est)
```

Arguments:

`s2_est` Is s2 being estimated?

Method `param_optim_upper()`**:** Upper bounds of parameters for optimization

Usage:

```
White$param_optim_upper(s2_est = self$s2_est)
```

Arguments:

`s2_est` Is s2 being estimated?

Method `set_params_from_optim()`**:** Set parameters from optimization output

Usage:

```
White$set_params_from_optim(optim_out, s2_est = self$s2_est)
```

Arguments:

`optim_out` Output from optimization
`s2_est` s2 estimate

Method `s2_from_params()`**:** Get s2 from params vector

Usage:

```
White$s2_from_params(params, s2_est = self$s2_est)
```

Arguments:

`params` parameter vector
`s2_est` Is s2 being estimated?

Method `print()`**:** Print this object

Usage:

```
White$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
White$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
k1 <- White$new(s2=1e-8)
```

Index