

# Package ‘GauPro’

June 10, 2024

**Type** Package

**Title** Gaussian Process Fitting

**Version** 0.2.12

**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](https://doi.org/10.1016/j.ejor.2017.10.002)>.

**License** GPL-3

**LinkingTo** Rcpp, RcppArmadillo

**Imports** Rcpp, R6, lbfgs

**RoxygenNote** 7.3.1

**Suggests** testthat, knitr, rmarkdown, microbenchmark, numDeriv, ContourFunctions, dplyr, ggplot2, ggrepel, gridExtra, lhs, mixopt (> 0.1.0), rlang, splitfngr, 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** 2024-06-10 08:00:02 UTC

## Contents

*.GauPro_kernel . . . . .	3
+.GauPro_kernel . . . . .	4
arma_mult_cube_vec . . . . .	4
corr_cubic_matrix_symC . . . . .	5
corr_exponential_matrix_symC . . . . .	6
corr_gauss_dCdX . . . . .	6
corr_gauss_matrix . . . . .	7
corr_gauss_matrixC . . . . .	7
corr_gauss_matrix_armaC . . . . .	8
corr_gauss_matrix_symC . . . . .	9
corr_gauss_matrix_sym_armaC . . . . .	9
corr_latentfactor_matrixmatrixC . . . . .	10
corr_latentfactor_matrix_symC . . . . .	11
corr_matern32_matrix_symC . . . . .	11
corr_matern52_matrix_symC . . . . .	12
corr_orderedfactor_matrixmatrixC . . . . .	13
corr_orderedfactor_matrix_symC . . . . .	13
Cubic . . . . .	14
Exponential . . . . .	17
FactorKernel . . . . .	19
GauPro . . . . .	26
GauPro_base . . . . .	27
GauPro_Gauss . . . . .	35
GauPro_Gauss_LOO . . . . .	42
GauPro_kernel . . . . .	43
GauPro_kernel_beta . . . . .	45
GauPro_kernel_model . . . . .	49
GauPro_kernel_model_LOO . . . . .	69
GauPro_trend . . . . .	71
Gaussian . . . . .	72
Gaussian_devianceC . . . . .	76
Gaussian_hessianC . . . . .	77
Gaussian_hessianCC . . . . .	77
Gaussian_hessianR . . . . .	78
GowerFactorKernel . . . . .	79
gpkm . . . . .	85
gradfuncarray . . . . .	87
gradfuncarrayR . . . . .	88
IgnoreIndsKernel . . . . .	88
kernel_cubic_dC . . . . .	92
kernel_exponential_dC . . . . .	93
kernel_gauss_dC . . . . .	94
kernel_latentFactor_dC . . . . .	94
kernel_matern32_dC . . . . .	95
kernel_matern52_dC . . . . .	96
kernel_orderedFactor_dC . . . . .	96

kernel_product . . . . .	97
kernel_sum . . . . .	101
LatentFactorKernel . . . . .	104
Matern32 . . . . .	111
Matern52 . . . . .	114
OrderedFactorKernel . . . . .	117
Periodic . . . . .	124
PowerExp . . . . .	130
predict.GauPro . . . . .	136
print.summary.GauPro . . . . .	137
RatQuad . . . . .	137
sqrt_matrix . . . . .	143
summary.GauPro . . . . .	143
trend_0 . . . . .	144
trend_c . . . . .	146
trend_LM . . . . .	149
Triangle . . . . .	153
White . . . . .	155

**Index****160**


---

*.GauPro_kernel	<i>Kernel product</i>
-----------------	-----------------------

---

**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))
```

---

<code>+.GauPro_kernel</code>	<i>Kernel sum</i>
------------------------------	-------------------

---

**Description**

Kernel sum

**Usage**

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

**Arguments**

<code>k1</code>	First kernel
<code>k2</code>	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))
```

---

<code>arma_mult_cube_vec</code>	<i>Cube multiply over first dimension</i>
---------------------------------	---

---

**Description**

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

**Usage**

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

**Arguments**

<code>cub</code>	A cube (3D array)
<code>v</code>	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\_cubic\_matrix\_symC

*Correlation Cubic matrix in C (symmetric)*

---

**Description**

Correlation Cubic matrix in C (symmetric)

**Usage**

```
corr_cubic_matrix_symC(x, theta)
```

**Arguments**

x	Matrix x
theta	Theta vector

**Value**

Correlation matrix

**Examples**

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

---

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	<i>Gaussian correlation</i>
-------------------	-----------------------------

---

**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	<i>Correlation Gaussian matrix in C using Rcpp</i>
--------------------	--

---

**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





---

`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**

<code>x</code>	Matrix <code>x</code>
<code>theta</code>	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**

<code>x</code>	Matrix <code>x</code>
<code>theta</code>	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))
```

---

 corr\_orderedfactor\_matrixmatrixC

*Correlation ordered factor matrix in C (symmetric)*


---

**Description**

Correlation ordered factor matrix in C (symmetric)

**Usage**

```
corr_orderedfactor_matrixmatrixC(x, y, theta, xindex, offdiagequal)
```

**Arguments**

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

**Value**

Correlation matrix

**Examples**

```
corr_orderedfactor_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, 1-1e-6)
corr_orderedfactor_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, 1-1e-6) * 6.85
```

---

 corr\_orderedfactor\_matrix\_symC

*Correlation ordered factor matrix in C (symmetric)*


---

**Description**

Correlation ordered factor matrix in C (symmetric)

**Usage**

```
corr_orderedfactor_matrix_symC(x, theta, xindex, offdiagequal)
```

**Arguments**

x	Matrix x
theta	Theta vector
xindex	Index to use
offdiagequal	What to set off-diagonal values with matching values to.

**Value**

Correlation matrix

**Examples**

```
corr_orderedfactor_matrix_symC(matrix(c(1,.5, 2,1.6, 1,0),ncol=2,byrow=TRUE),
                                c(1.5,1.8), 1, 1-1e-6)
corr_orderedfactor_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, 1-1e-6) * 6.85
```

---

Cubic

*Cubic Kernel R6 class*

---

**Description**

Cubic Kernel R6 class

Cubic Kernel R6 class

**Usage**

```
k_Cubic(
  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.

**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)

```

Exponential

*Exponential Kernel R6 class***Description**

Exponential Kernel R6 class

Exponential Kernel R6 class

**Usage**

```

k_Exponential(
  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.

**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**

Initialize kernel object

**Usage**

```
k_FactorKernel(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 0.9,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  p,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

**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	Vector of correlations
useC	Should C code used? Not implemented for FactorKernel yet.
offdiagequal	What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

**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 p

`p_upper` Upper bound of p

`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

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

**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$print()`
- `FactorKernel$clone()`

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

*Usage:*

```
FactorKernel$new(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 0.9,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  p,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

*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 Vector of correlations

useC Should C code used? Not implemented for FactorKernel yet.

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** 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 = self\$offdiagequal)

*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, ...)

*Arguments:*

XX matrix of points

X matrix of points to take derivative with respect to

... Additional args, not used

**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?

**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?

**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?

**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?

**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?

**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** 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

nug.min Minimum value of nugget

nug Value of the nugget, is estimated unless told otherwise

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.

nug.est Should the nugget be estimated?

param.est Should the parameters be estimated?

mu\_hat Mean estimate

s2\_hat Variance estimate

K Covariance matrix

Kchol Cholesky factorization of K

Kinv Inverse of K

## 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_L00()`
- `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()`: Correlation function

*Usage:*

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

*Arguments:*

... Does nothing

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

*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,
  ...
)
```

*Arguments:*

X Matrix whose rows are the input points

Z Output points corresponding to X

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.

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

nug.min Minimum allowable value for the nugget.

nug.est Should the nugget be estimated?

param.est Should the kernel parameters be estimated?

... Not used

**Method** initialize\_GauPr(): Not used

*Usage:*

```
GauPro_base$initialize_GauPr()
```

**Method** fit(): Fit the model, never use this function

*Usage:*

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

*Arguments:*

X Not used

Z Not used

**Method** update\_K\_and\_estimates(): Update Covariance matrix and estimated parameters

*Usage:*

```
GauPro_base$update_K_and_estimates()
```

**Method** predict(): Predict mean and se for given matrix

*Usage:*

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

*Arguments:*

XX Points to predict at

se.fit Should the se be returned?

covmat Should the covariance matrix be returned?

split\_speed Should the predictions be split up for speed

**Method** `pred()`: Predict mean and se for given matrix

*Usage:*

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

*Arguments:*

XX Points to predict at

se.fit Should the se be returned?

covmat Should the covariance matrix be returned?

split\_speed Should the predictions be split up for speed

**Method** `pred_one_matrix()`: Predict mean and se for given matrix

*Usage:*

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

*Arguments:*

XX Points to predict at

se.fit Should the se be returned?

covmat Should the covariance matrix be returned?

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

*Usage:*

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

*Arguments:*

XX Points to predict at

kx.xx Covariance matrix between X and XX

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

*Usage:*

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

*Arguments:*

XX Points to predict at

kx.xx Covariance matrix between X and XX

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

*Usage:*

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

*Arguments:*

XX Points to predict at

kxx Covariance matrix of XX with itself  
 kx.xx Covariance matrix between X and XX  
 covmat Not used

**Method** `pred_L00()`: Predict at X using leave-one-out. Can use for diagnostics.

*Usage:*

```
GauPro_base$pred_L00(se.fit = FALSE)
```

*Arguments:*

`se.fit` Should the standard error and t values be returned?

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

*Usage:*

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

*Arguments:*

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

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

*Usage:*

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

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

*Usage:*

```
GauPro_base$plot2D()
```

**Method loglikelihood():** Calculate the log likelihood, don't use this

*Usage:*

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

*Arguments:*

mu Mean vector  
 s2 s2 param

**Method optim():** Optimize parameters

*Usage:*

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

*Arguments:*

restarts Number of restarts to do



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

*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?

**Method** update(): Update the model, can be data and parameters

*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
)
```

*Arguments:*

Xnew New X matrix

Znew New Z values  
Xall Matrix with all X values  
Zall All Z values  
restarts Number of optimization restarts  
param\_update Should the parameters be updated?  
nug.update Should the nugget be updated?  
no\_update Should none of the parameters/nugget be updated?

**Method** update\_data(): Update the data

*Usage:*

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

*Arguments:*

Xnew New X matrix  
Znew New Z values  
Xall Matrix with all X values  
Zall All Z values

**Method** update\_corrparams(): Update the correlation parameters

*Usage:*

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

*Arguments:*

... Args passed to update

**Method** update\_nugget(): Update the nugget

*Usage:*

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

*Arguments:*

... Args passed to update

**Method** deviance\_searchnug(): Optimize deviance for nugget

*Usage:*

```
GauPro_base$deviance_searchnug()
```

**Method** nugget\_update(): Update the nugget

*Usage:*

```
GauPro_base$nugget_update()
```

**Method** grad\_norm(): Calculate the norm of the gradient at XX

*Usage:*

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

*Arguments:*

XX Points to calculate at

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

*Usage:*

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

*Arguments:*

XX Input points to sample at

n Number of samples

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

*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

**Public fields**

corr Name of correlation  
 theta Correlation parameters  
 theta\_length Length of theta  
 theta\_map Map for theta  
 theta\_short Short vector for theta  
 separable Are the dimensions separable?

**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()`: Create GauPro object

*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,
    ...
)

```

*Arguments:*

*X* Matrix whose rows are the input points

*Z* Output points corresponding to *X*

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

*separable* Are dimensions separable?

*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.

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

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

*nug.est* Should the nugget be estimated?

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

*theta* Correlation parameters

*theta\_short* Correlation parameters, not recommended

*theta\_map* Correlation parameters, not recommended

... Not used

**Method** `corr_func()`: Correlation function*Usage:*

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

*Arguments:*

*x* First point

*x2* Second point

*theta* Correlation parameter

**Method** `deviance_theta()`: Calculate deviance*Usage:*

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

*Arguments:*

*theta* Correlation parameter

**Method** `deviance_theta_log()`: Calculate deviance

*Usage:*

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

*Arguments:*

beta Correlation parameter on log scale

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

*Usage:*

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

*Arguments:*

theta Correlation parameter

nug Nugget

**Method** `deviance_grad()`: Calculate deviance gradient

*Usage:*

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

*Arguments:*

theta Correlation parameter

nug Nugget

joint Calculate over theta and nug at same time?

overwhat Calculate over theta and nug at same time?

**Method** `deviance_fngr()`: Calculate deviance and gradient at same time

*Usage:*

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

*Arguments:*

theta Correlation parameter

nug Nugget

overwhat Calculate over theta and nug at same time?

joint Calculate over theta and nug at same time?

**Method** `deviance_log()`: Calculate deviance gradient

*Usage:*

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

*Arguments:*

beta Correlation parameter on log scale  
 nug Nugget  
 joint Calculate over theta and nug at same time?

**Method** deviance\_log2(): Calculate deviance on log scale

*Usage:*

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

*Arguments:*

beta Correlation parameter on log scale  
 lognug Log of nugget  
 joint Calculate over theta and nug at same time?

**Method** deviance\_log\_grad(): Calculate deviance gradient on log scale

*Usage:*

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

*Arguments:*

beta Correlation parameter  
 nug Nugget  
 joint Calculate over theta and nug at same time?  
 overwhat Calculate over theta and nug at same time?

**Method** deviance\_log2\_grad(): Calculate deviance gradient on log scale

*Usage:*

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

*Arguments:*

beta Correlation parameter  
 lognug Log of nugget  
 joint Calculate over theta and nug at same time?  
 overwhat Calculate over theta and nug at same time?

**Method** deviance\_log2\_fngr(): Calculate deviance and gradient on log scale

*Usage:*

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

*Arguments:*

beta Correlation parameter

lognug Log of nugget

joint Calculate over theta and nug at same time?

overwhat Calculate over theta and nug at same time?

**Method** get\_optim\_functions(): Get optimization functions

*Usage:*

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

*Arguments:*

param\_update Should the parameters be updated?

nug.update Should the nugget be updated?

**Method** param\_optim\_lower(): Lower bound of params

*Usage:*

```
GauPro_Gauss$param_optim_lower()
```

**Method** param\_optim\_upper(): Upper bound of params

*Usage:*

```
GauPro_Gauss$param_optim_upper()
```

**Method** param\_optim\_start(): Start value of params for optim

*Usage:*

```
GauPro_Gauss$param_optim_start()
```

**Method** param\_optim\_start0(): Start value of params for optim

*Usage:*

```
GauPro_Gauss$param_optim_start0()
```

**Method** param\_optim\_jitter(): Jitter value of params for optim

*Usage:*

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

*Arguments:*

param\_value param value to add jitter to

**Method** update\_params(): Update value of params after optim

*Usage:*

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



*Arguments:*

restarts Number of restarts  
param\_update Are the params being updated?  
nug.update Is the nugget being updated?

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

*Usage:*

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

*Arguments:*

XX Points to calculate grad at

**Method** grad\_dist(): Calculate the gradient distribution

*Usage:*

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

*Arguments:*

XX Points to calculate grad at

**Method** hessian(): Calculate the hessian

*Usage:*

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

*Arguments:*

XX Points to calculate grad at  
useC Should C code be used to speed up?

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

*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](#)

**Public fields**

use\_LOO Should the leave-one-out correction be used?

tmod Second GP model fit to the t-values of leave-one-out predictions

**Methods****Public methods:**

- [GauPro\\_Gauss\\_LOO\\$update\(\)](#)
- [GauPro\\_Gauss\\_LOO\\$pred\\_one\\_matrix\(\)](#)
- [GauPro\\_Gauss\\_LOO\\$print\(\)](#)
- [GauPro\\_Gauss\\_LOO\\$clone\(\)](#)

**Method** `update()`: Update the model, can be data and parameters

*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
)
```

*Arguments:*

Xnew New X matrix  
 Znew New Z values  
 Xall Matrix with all X values  
 Zall All Z values  
 restarts Number of optimization restarts  
 param\_update Should the parameters be updated?  
 nug.update Should the nugget be updated?  
 no\_update Should none of the parameters/nugget be updated?

**Method** `pred_one_matrix()`: Predict mean and se for given matrix

*Usage:*

```
GauPro_Gauss_L00$pred_one_matrix(XX, se.fit = F, covmat = F)
```

*Arguments:*

XX Points to predict at  
 se.fit Should the se be returned?  
 covmat Should the covariance matrix be returned?

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

*Usage:*

```
GauPro_Gauss_L00$print()
```

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

*Usage:*

```
GauPro_Gauss_L00$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_L00$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  $\beta = \log_{10}(\theta)$  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 *Gaussian process model with kernel*

---

**Description**

Class providing object with methods for fitting a GP model. Allows for different kernel and trend functions to be used. The object is an R6 object with many methods that can be called.

'gpkm()' is equivalent to 'GauPro\_kernel\_model\$new()', but is easier to type and gives parameter autocomplete suggestions.

**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 = 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$plotkernel()`
- `GauPro_kernel_model$plotLOO()`
- `GauPro_kernel_model$plot_track_optim()`
- `GauPro_kernel_model$loglikelihood()`
- `GauPro_kernel_model$AIC()`
- `GauPro_kernel_model$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$gradpredvar()`
- `GauPro_kernel_model$sample()`
- `GauPro_kernel_model$optimize_fn()`
- `GauPro_kernel_model$EI()`
- `GauPro_kernel_model$maxEI()`
- `GauPro_kernel_model$maxqEI()`
- `GauPro_kernel_model$KG()`
- `GauPro_kernel_model$AugmentedEI()`
- `GauPro_kernel_model$CorrectedEI()`
- `GauPro_kernel_model$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?  
*formula* Formula for the data if giving in a data frame.  
*data* Data frame of data. Use in conjunction with *formula*.  
 ... 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,
  return_df = TRUE
)
```

*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?

return\_df When returning se.fit, should it be returned in a data frame? Otherwise it will be a list, which is faster.

**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,
  return_df = TRUE
)
```

*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?

return\_df When returning se.fit, should it be returned in a data frame? Otherwise it will be a list, which is faster.

**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? Otherwise it will be a list, which is faster.  
`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 = "green",
  xlab = "x",
  ylab = "y",
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  gg = TRUE
)
```



*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  
 gg Should ggplot2 be used to make plot?

**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,
  gg = TRUE
)
```

*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  
 gg Should ggplot2 be used to make plot?

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

```
GauPro_kernel_model$plot2D(se = FALSE, mean = TRUE, horizontal = TRUE, n = 50)
```

*Arguments:*

se Should the standard error of prediction be plotted?  
 mean Should the mean be plotted?  
 horizontal If plotting mean and se, should they be next to each other?  
 n Number of points along each dimension

**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(npt = 5, ncol = NULL)
```

*Arguments:*

npt Number of lines to make. Each line represents changing a single variable while holding the others at the same values.  
 ncol Number of columnsfor the plot

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

*Usage:*

```
GauPro_kernel_model$plotmarginalrandom(npt = 100, ncol = NULL)
```

*Arguments:*

npt Number of random points to evaluate  
 ncol Number of columns in the plot

**Method** `plotkernel()`: Plot the kernel

*Usage:*

```
GauPro_kernel_model$plotkernel(X = self$X)
```

*Arguments:*

X X matrix for kernel plot

**Method** `plotL00()`: Plot leave one out predictions for design points

*Usage:*

```
GauPro_kernel_model$plotL00()
```

**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** AIC(): AIC (Akaike information criterion)

*Usage:*

GauPro\_kernel\_model\$AIC()

**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?  
 1 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** `gradpredvar()`: Calculate gradient of the predictive variance

*Usage:*

```
GauPro_kernel_model$gradpredvar(XX)
```

*Arguments:*

XX points to calculate at

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

*Usage:*

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

*Arguments:*

XX Input matrix

n Number of samples

**Method** `optimize_fn()`: Optimize any function of the GP prediction over the valid input space. If there are inputs that should only be optimized over a discrete set of values, specify 'mopar' for all parameters. Factor inputs will be handled automatically.

*Usage:*

```
GauPro_kernel_model$optimize_fn(
  fn = NULL,
  lower = apply(self$X, 2, min),
  upper = apply(self$X, 2, max),
  n0 = 100,
  minimize = FALSE,
  fn_args = NULL,
  gr = NULL,
  fngr = NULL,
  mopar = NULL,
  groupeval = FALSE
)
```

*Arguments:*

fn Function to optimize

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?

fn\_args Arguments to pass to the function fn.

gr Gradient of function to optimize.

`fngr` Function that returns list with names elements "fn" for the function value and "gr" for the gradient. Useful when it is slow to evaluate and fn/gr would duplicate calculations if done separately.

`mopar` List of parameters using `mixopt`

`groupeval` Can a matrix of points be evaluated? Otherwise just a single point at a time.

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

*Usage:*

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

*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

`return_grad` Should the gradient be returned?

... Additional args

**Method** `maxEI()`: Find the point that maximizes the expected improvement. If there are inputs that should only be optimized over a discrete set of values, specify 'mopar' for all parameters.

*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,
  EItype = "corrected",
  mopar = NULL,
  usegrad = FALSE
)
```

*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?

`EItype` Type of EI to calculate. One of "EI", "Augmented", or "Corrected"

`mopar` List of parameters using `mixopt`

`usegrad` Should the gradient be used when optimizing? Can make it faster.

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

*Usage:*

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

*Arguments:*

*npoints* Number of points to add

*method* Method to use for setting the output value for the points chosen as a placeholder. Can be one of: "CL" for constant liar, which uses the best value seen yet; or "pred", which uses the predicted value, also called the Believer method in literature.

*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

*EItype* Type of EI to calculate. One of "EI", "Augmented", or "Corrected"

*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 AugmentedEI():** Calculated Augmented EI*Usage:*

```
GauPro_kernel_model$AugmentedEI(
  x,
  minimize = FALSE,
  eps = 0,
  return_grad = F,
  ...
)
```

*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  
 return\_grad Should the gradient be returned?  
 ... Additional args  
 f The reference max, user shouldn't change this.

**Method** CorrectedEI(): Calculated Augmented EI*Usage:*

```
GauPro_kernel_model$CorrectedEI(
  x,
  minimize = FALSE,
  eps = 0,
  return_grad = F,
  ...
)
```

*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  
 return\_grad Should the gradient be returned?  
 ... Additional args

**Method** importance(): Feature importance*Usage:*

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

*Arguments:*

plot Should the plot be made?  
 printBars Should the importances be printed as bars?

**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](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$new(X=x, Z=y, kernel="gauss")
gp$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(..., L00_kernel, L00_options = list())
```

*Arguments:*

... Passed to `super$initialize`.

`L00_kernel` The kernel that should be used for the leave-one-out model. Shouldn't be too smooth.

`L00_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_L00$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_L00$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_L00$new(X=x, Z=y, kernel=Gaussian)
y <- x^2 * sin(2*pi*x) + rnorm(n,0,1e-3)
gp <- GauPro_kernel_model_L00$new(X=x, Z=y, kernel=Matern52)
y <- exp(-1.4*x)*cos(7*pi*x/2)
gp <- GauPro_kernel_model_L00$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

**Usage**

```
k_Gaussian(  
  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.

**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
```

---

GowerFactorKernel      *Gower factor Kernel R6 class*

---

### Description

Gower factor Kernel R6 class

Gower factor Kernel R6 class

### Usage

```
k_GowerFactorKernel(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 0.9,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  p,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

### 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	Vector of correlations
useC	Should C code used? Not implemented for FactorKernel yet.
offdiagequal	What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

**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\_GowerFactorKernel

**Public fields**

p Parameter for correlation  
 p\_est Should p be estimated?  
 p\_lower Lower bound of p  
 p\_upper Upper bound 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  
 offdiagequal What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

**Methods****Public methods:**

- GowerFactorKernel\$new()
- GowerFactorKernel\$k()
- GowerFactorKernel\$kone()
- GowerFactorKernel\$dC\_dparams()
- GowerFactorKernel\$C\_dC\_dparams()
- GowerFactorKernel\$dC\_dx()
- GowerFactorKernel\$param\_optim\_start()
- GowerFactorKernel\$param\_optim\_start0()
- GowerFactorKernel\$param\_optim\_lower()
- GowerFactorKernel\$param\_optim\_upper()



- `GowerFactorKernel$set_params_from_optim()`
- `GowerFactorKernel$s2_from_params()`
- `GowerFactorKernel$print()`
- `GowerFactorKernel$clone()`

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

*Usage:*

```
GowerFactorKernel$new(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 0,
  p_upper = 0.9,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  p,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

*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` Vector of correlations

`useC` Should C code used? Not implemented for FactorKernel yet.

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

*Usage:*

```
GowerFactorKernel$k(x, y = NULL, p = self$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:*

```
GowerFactorKernel$kone(
  x,
  y,
  p,
  s2,
  isdiag = TRUE,
  offdiagequal = self$offdiagequal
)
```

*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:*

```
GowerFactorKernel$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:*

```
GowerFactorKernel$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:*

```
GowerFactorKernel$dC_dx(XX, X, ...)
```

*Arguments:*

XX matrix of points

X matrix of points to take derivative with respect to

... Additional args, not used

**Method** param\_optim\_start(): Starting point for parameters for optimization

*Usage:*

```
GowerFactorKernel$param_optim_start(
  jitter = F,
  y,
  p_est = self$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:*

```
GowerFactorKernel$param_optim_start0(
  jitter = F,
  y,
  p_est = self$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:*

```
GowerFactorKernel$param_optim_lower(p_est = self$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:*

```
GowerFactorKernel$param_optim_upper(p_est = self$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:*

```
GowerFactorKernel$set_params_from_optim(  
  optim_out,  
  p_est = self$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:*

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

*Arguments:*

params parameter vector

s2\_est Is s2 being estimated?

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

*Usage:*

```
GowerFactorKernel$print()
```

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.

**Examples**

```

kk <- GowerFactorKernel$new(D=1, nlevels=5, xindex=1, p=.2)
kmat <- outer(1:5, 1:5, Vectorize(kk$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 <- GowerFactorKernel$new(D=2, nlevels=3, xind=2)
k2 <- k2a * k2b
k2b$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) the most
data.frame(X, cov=gp$kernel$k(X, c(5.5,3))) %>% arrange(-cov)
plot(k2b)

```

**Description**

Fits a Gaussian process regression model to data.

An R6 object is returned with many methods.

‘gpkm()’ is an alias for ‘GauPro\_kernel\_model\$new()’. For full documentation, see documentation for ‘GauPro\_kernel\_model’.

Standard methods that work include ‘plot()’, ‘summary()’, and ‘predict()’.

**Usage**

```

gpkm(
  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?
formula	Formula for the data if giving in a data frame.
data	Data frame of data. Use in conjunction with formula.
...	Not used

### Details

The default kernel is a Matern 5/2 kernel, but factor/character inputs will be given factor kernels.

---

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

---

### 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	<i>Calculate gradfunc in optimization to speed up. NEEDS TO APERM dC_dparams Doesn't need to be exported, should only be useful in functions.</i>
----------------	---

---

**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	<i>Kernel R6 class</i>
------------------	------------------------

---

**Description**

Kernel R6 class

Kernel R6 class

**Usage**

```
k_ignoreIndsKernel(k, ignoreinds, useC = TRUE)
```



**Arguments**

k	Kernel to use on the non-ignored indices
ignoreinds	Indices of columns of X to ignore.
useC	Should C code used? Not implemented for IgnoreInds.

**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\$dC\_dx()
- 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, useC = TRUE)
```

*Arguments:*

k Kernel to use on the non-ignored indices  
 ignoreinds Indices of columns of X to ignore.  
 useC Should C code used? Not implemented for IgnoreInds.

**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** dC\_dx(): Derivative of covariance with respect to X

*Usage:*

IgnoreIndsKernel\$dC\_dx(XX, X, ...)

*Arguments:*

XX matrix of points

X matrix of points to take derivative with respect to

... Additional arguments passed on to the kernel

**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\_cubic\_dC

*Derivative of cubic kernel covariance matrix in C*

---

### Description

Derivative of cubic kernel covariance matrix in C

### Usage

```
kernel_cubic_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

### Value

Correlation matrix

---

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	<i>Derivative of Matern 5/2 kernel covariance matrix in C</i>
--------------------	---

---

### 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\_orderedFactor\_dC  
*Derivative of covariance matrix of X with respect to kernel parameters  
for the Ordered Factor Kernel*

---

**Description**

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



**Usage**

```
kernel_orderedFactor_dC(
  x,
  pf,
  C_nonug,
  s2_est,
  p_est,
  lenparams_D,
  s2_nug,
  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
xindex	Which column of x is the indexing variable
nlevels	Number of levels
s2	Value of s2

**Value**

Correlation matrix

---

kernel_product	<i>Gaussian Kernel R6 class</i>
----------------	---------------------------------

---

**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**

k1p1 param length of kernel 1  
k2p1 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, useC = TRUE)`

*Arguments:*

k1 Kernel 1

k2 Kernel 2

useC Should C code used? Not applicable for kernel product.

**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	<i>Gaussian Kernel R6 class</i>
------------	---------------------------------

---

**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

k1p1 param length of kernel 1

k2p1 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, useC = TRUE)
```

*Arguments:*

k1 Kernel 1

k2 Kernel 2

useC Should C code used? Not applicable for kernel sum.

**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

### Usage

```
k_LatentFactorKernel(
  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,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

### Arguments

<code>s2</code>	Initial variance
<code>D</code>	Number of input dimensions of data
<code>nlevels</code>	Number of levels for the factor
<code>xindex</code>	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?
useC	Should C code used? Much faster.
offdiagequal	What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

**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 p  
 p\_upper Upper bound of p  
 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

`pf_to_p_log` Logical vector used to convert pf to p  
`p_to_pf_inds` Vector of indexes used to convert p to pf  
`offdiagequal` What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

## 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$set_params_from_optim()`
- `LatentFactorKernel$p_to_pf()`
- `LatentFactorKernel$s2_from_params()`
- `LatentFactorKernel$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,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
  
```

*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?  
 useC Should C code used? Much faster.  
 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** `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$kone(
  x,
  y,
  pf,
  s2,
  isdiag = TRUE,
  offdiagequal = self$offdiagequal
)
```

*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, ...)
```

*Arguments:*

XX matrix of points  
 X matrix of points to take derivative with respect to  
 ... Additional args, not used

**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?

**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?

**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?

**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?

**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?

**Method** p\_to\_pf(): Convert p (short parameter vector) to pf (long parameter vector with zeros).

*Usage:*

```
LatentFactorKernel$p_to_pf(p)
```

*Arguments:*

p Parameter vector

**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** `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) the most
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

### Usage

```

k_Matern32(
  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.

### 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

### Usage

```
k_Matern52(
  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.

**Format**

[R6Class](#) object.

**Details**

$k(x, y) = s2 * (1 + t1 + t1^2/3) * exp(-t1)$  where  $t1 = sqrt(5) * sqrt(sum(theta * (x - y)^2))$

**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    *Ordered Factor Kernel R6 class*

---

**Description**

Ordered Factor Kernel R6 class

Ordered Factor Kernel R6 class

**Usage**

```

k_OrderedFactorKernel(
  s2 = 1,
  D,
  nlevels,
  xindex,
  p_lower = 1e-08,
  p_upper = 5,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)

```

**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?
useC	Should C code used? Not implemented for FactorKernel yet.
offdiagequal	What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.

**Format**

[R6Class](#) object.

**Details**

Use for factor inputs that are considered to have an ordering

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

**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$plotLatent()`
- `OrderedFactorKernel$print()`
- `OrderedFactorKernel$clone()`

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

*Usage:*

```
OrderedFactorKernel$new(
  s2 = 1,
  D = NULL,
  nlevels,
  xindex,
  p_lower = 1e-08,
  p_upper = 5,
  p_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = TRUE,
  offdiagequal = 1 - 1e-06
)
```

*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?  
*useC* Should C code used? Much faster.  
*offdiagequal* What should offdiagonal values be set to when the indices are the same? Use to avoid decomposition errors, similar to adding a nugget.  
*p* Vector of distances in latent space

**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$kone(
  x,
  y,
  p,
  s2,
  isdiag = TRUE,
  offdiagequal = self$offdiagequal
)
```

*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, ...)
```

*Arguments:*

XX matrix of points  
 X matrix of points to take derivative with respect to  
 ... Additional args, not used

**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?

**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?

**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?

**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?

**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?

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

*Usage:*

```
OrderedFactorKernel$plotLatent()
```

**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) the most
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

**Usage**

```
k_Periodic(
  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,
  useC = 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?
useC	Should C code used? Much faster if implemented.

**Format**

[R6Class](#) object.

**Details**

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

$$k(x, y) = s^2 * \exp(-\text{sum}(\alpha * \sin(p * (x - y))^2))$$

$$k(x, y) = \sigma^2 * \exp(-\sum(\alpha_i * \sin(p * (x_i - y_i))^2))$$

**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?  
 $\log p$  Log of  $p$   
 $\log p\_lower$  Lower bound of  $\log p$   
 $\log p\_upper$  Upper bound of  $\log p$   
 $p\_length$  length of  $p$   
 $\alpha$  Parameter for correlation  
 $\alpha\_est$  Should  $\alpha$  be estimated?  
 $\log \alpha$  Log of  $\alpha$   
 $\log \alpha\_lower$  Lower bound of  $\log \alpha$   
 $\log \alpha\_upper$  Upper bound of  $\log \alpha$   
 $s^2$  variance  
 $s^2\_est$  Is  $s^2$  estimated?  
 $\log s^2$  Log of  $s^2$   
 $\log s^2\_lower$  Lower bound of  $\log s^2$   
 $\log s^2\_upper$  Upper bound of  $\log s^2$

**Methods****Public methods:**

- [Periodic\\$new\(\)](#)
- [Periodic\\$k\(\)](#)
- [Periodic\\$kone\(\)](#)
- [Periodic\\$dC\\_dparams\(\)](#)

- `Periodic$dC_dparams()`
- `Periodic$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,
  useC = 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?  
`useC` Should C code used? Much faster if implemented.

**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

**Usage**

```
k_PowerExp(
  alpha = 1.95,
  beta,
  s2 = 1,
  D,
  beta_lower = -8,
  beta_upper = 6,
```

```

    beta_est = TRUE,
    alpha_lower = 1e-08,
    alpha_upper = 2,
    alpha_est = TRUE,
    s2_lower = 1e-08,
    s2_upper = 1e+08,
    s2_est = TRUE,
    useC = 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?
useC	Should C code used? Much faster if implemented.

### 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 = 1e-08,
  alpha_upper = 2,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = 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?  
 useC Should C code used? Much faster if implemented.

**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	<i>Predict for class GauPro</i>
----------------	---------------------------------

---

### 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
```

**Value**

prints, returns invisible object

---

```
RatQuad          Rational Quadratic Kernel R6 class
```

---

**Description**

Rational Quadratic Kernel R6 class  
Rational Quadratic Kernel R6 class

**Usage**

```
k_RatQuad(
  beta,
  alpha = 1,
  s2 = 1,
  D,
  beta_lower = -8,
  beta_upper = 6,
  beta_est = TRUE,
  alpha_lower = 1e-08,
  alpha_upper = 100,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = 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?
useC	Should C code used? Much faster if implemented.

**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 = 1e-08,
  alpha_upper = 100,
  alpha_est = TRUE,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = 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?  
`useC` Should C code used? Much faster if implemented.

**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	<i>Find the square root of a matrix</i>
-------------	---

---

**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	<i>Summary for GauPro object</i>
----------------	----------------------------------

---

**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$param_optim_start(jitter = F, m_est = self$m_est)
```

*Arguments:*

jitter Not used  
m\_est If the trend should be estimate.

**Method** param\_optim\_start0(): Get parameter initial point for optimization

*Usage:*

```
trend_c$param_optim_start0(jitter = F, m_est = self$m_est)
```

*Arguments:*

jitter Not used  
m\_est If the trend should be estimate.

**Method** param\_optim\_lower(): Get parameter lower bounds for optimization

*Usage:*

```
trend_c$param_optim_lower(m_est = self$m_est)
```

*Arguments:*

m\_est If the trend should be estimate.

**Method** param\_optim\_upper(): Get parameter upper bounds for optimization

*Usage:*

```
trend_c$param_optim_upper(m_est = self$m_est)
```

*Arguments:*

m\_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$param_optim_start(  
  jitter = FALSE,  
  b_est = self$b_est,  
  m_est = self$m_est  
)
```

*Arguments:*

jitter Not used  
b\_est If the mean should be estimated.  
m\_est If the linear terms should be estimated.

**Method** param\_optim\_start0(): Get parameter initial point for optimization

*Usage:*

```
trend_LM$param_optim_start0(  
  jitter = FALSE,  
  b_est = self$b_est,  
  m_est = self$m_est  
)
```

*Arguments:*

jitter Not used

b\_est If the mean should be estimated.

m\_est If the linear terms should be estimated.

**Method** param\_optim\_lower(): Get parameter lower bounds for optimization

*Usage:*

```
trend_LM$param_optim_lower(b_est = self$b_est, m_est = self$m_est)
```

*Arguments:*

b\_est If the mean should be estimated.

m\_est If the linear terms should be estimated.

**Method** param\_optim\_upper(): Get parameter upper bounds for optimization

*Usage:*

```
trend_LM$param_optim_upper(b_est = self$b_est, m_est = self$m_est)
```

*Arguments:*

b\_est If the mean should be estimated.

m\_est If the linear terms 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

### Usage

```
k_Triangle(  
  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.

### 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()
```

## Description

Initialize kernel object

**Usage**

```
k_White(
  s2 = 1,
  D,
  s2_lower = 1e-08,
  s2_upper = 1e+08,
  s2_est = TRUE,
  useC = 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?
useC	Should C code used? Not implemented for White.

**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,
  useC = 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?  
`useC` Should C code used? Not implemented for White.

**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

\*.GauPro\_kernel, 3  
+.GauPro\_kernel, 4

arma\_mult\_cube\_vec, 4

corr\_cubic\_matrix\_symC, 5  
corr\_exponential\_matrix\_symC, 6  
corr\_gauss\_dCdX, 6  
corr\_gauss\_matrix, 7  
corr\_gauss\_matrix\_armaC, 8  
corr\_gauss\_matrix\_sym\_armaC, 9  
corr\_gauss\_matrix\_symC, 9  
corr\_gauss\_matrixC, 7  
corr\_latentfactor\_matrix\_symC, 11  
corr\_latentfactor\_matrixmatrixC, 10  
corr\_matern32\_matrix\_symC, 11  
corr\_matern52\_matrix\_symC, 12  
corr\_orderedfactor\_matrix\_symC, 13  
corr\_orderedfactor\_matrixmatrixC, 13  
Cubic, 14

Exponential, 17

FactorKernel, 19

GauPro, 26  
GauPro::GauPro, 35, 42, 69  
GauPro::GauPro\_Gauss, 42  
GauPro::GauPro\_kernel, 15, 18, 21, 45, 73, 80, 89, 98, 101, 105, 112, 115, 118, 125, 131, 138, 154, 156  
GauPro::GauPro\_kernel\_beta, 15, 18, 73, 112, 115, 131, 138, 154  
GauPro::GauPro\_trend, 144, 147, 149  
GauPro\_base, 27  
GauPro\_Gauss, 35  
GauPro\_Gauss\_L00, 42  
GauPro\_kernel, 43  
GauPro\_kernel\_beta, 45  
GauPro\_kernel\_model, 49  
GauPro\_kernel\_model\_L00, 69

GauPro\_trend, 71  
Gaussian, 72  
Gaussian\_devianceC, 76  
Gaussian\_hessianC, 77  
Gaussian\_hessianCC, 77  
Gaussian\_hessianR, 78  
GowerFactorKernel, 79  
gpkm, 85  
gradfuncarray, 87  
gradfuncarrayR, 88

IgnoreIndsKernel, 88

k\_Cubic (Cubic), 14  
k\_Exponential (Exponential), 17  
k\_FactorKernel (FactorKernel), 19  
k\_Gaussian (Gaussian), 72  
k\_GowerFactorKernel (GowerFactorKernel), 79  
k\_IgnoreIndsKernel (IgnoreIndsKernel), 88  
k\_LatentFactorKernel (LatentFactorKernel), 104  
k\_Matern32 (Matern32), 111  
k\_Matern52 (Matern52), 114  
k\_OrderedFactorKernel (OrderedFactorKernel), 117  
k\_Periodic (Periodic), 124  
k\_PowerExp (PowerExp), 130  
k\_RatQuad (RatQuad), 137  
k\_Triangle (Triangle), 153  
k\_White (White), 155  
kernel\_cubic\_dC, 92  
kernel\_exponential\_dC, 93  
kernel\_gauss\_dC, 94  
kernel\_latentFactor\_dC, 94  
kernel\_matern32\_dC, 95  
kernel\_matern52\_dC, 96  
kernel\_orderedFactor\_dC, 96  
kernel\_product, 97



kernel\_sum, [101](#)

LatentFactorKernel, [104](#)

Matern32, [111](#)  
Matern52, [114](#)

OrderedFactorKernel, [117](#)

Periodic, [124](#)  
PowerExp, [130](#)  
predict.GauPro, [136](#)  
print.summary.GauPro, [137](#)

R6Class, [15](#), [18](#), [20](#), [27](#), [35](#), [42](#), [44](#), [45](#), [49](#), [69](#),  
[71](#), [73](#), [80](#), [89](#), [97](#), [98](#), [101](#), [105](#), [112](#),  
[115](#), [118](#), [125](#), [131](#), [138](#), [144](#), [146](#),  
[147](#), [149](#), [153](#), [156](#)

RatQuad, [137](#)

sqrt\_matrix, [143](#)  
summary.GauPro, [143](#)

trend\_0, [144](#)  
trend\_c, [146](#)  
trend\_LM, [149](#)  
Triangle, [153](#)

White, [155](#)