
libKriging

Release 0.7

libKriging team

May 26, 2023

CONTENTS

| | | |
|----------|---------------------|-----------|
| 1 | Contents | 3 |
| | Bibliography | 89 |

libKriging is a C++ library for Kriging/Gaussian process regression.

Main features of libKriging are:

- **Standard implementation of most common kriging:**
 - ordinary/universal kriging
 - nugget (homoskedastic) or noise (heteroskedastic)
 - optimization of hyper-parameters (range, nugget, variance, ...) based on log-likelihood, leave-one-out, log-marginal-posterior
 - (pre-)normalization of conditional data
- **Port from and comparison/testing with some standard kriging libraries:**
 - <https://CRAN.R-project.org/package=DiceKriging>
 - <https://CRAN.R-project.org/package=RobustGaSP>
 - <https://github.com/stk-kriging>
- **Compatibility with commons OS/arch:**
 - Windows
 - Linux
 - OSX (intel & ARM)
- **(Almost) full wrapper availables for:**
 - Python: <https://pypi.org/project/pylibkriging/>
 - R: <https://github.com/libKriging/rlibkriging>
 - Octave
 - Matlab

Check out the *Usage* section for further information, and how to install the project.

Note: This project is under active development.

CONTENTS

1.1 Installation

libKriging may be installed directly from:

- Python, from PyPI:

```
pip3 install pylibkriging
```

- R

- from CRAN:

```
install.packages('rlibkriging')
```

- from GitHub (dev version):

```
devtools::install_github('libKriging/rlibkriging')
```

- Octave/Matlab, **download and uncompress** the archive for your system from libKriging latest release <https://github.com/libKriging/libKriging/releases/latest>, then:

```
addpath("mLibKriging")
```

1.2 Usage

libKriging may be used through:

- direct C++ access
- Python wrapper
- R wrapper
- Octave wrapper
- Matlab wrapper

The basic usage is almost the same whatever lang.:

```
# input design  
X = ...  
# output results
```

(continues on next page)

(continued from previous page)

```

y = ...

# load/import/... libKriging
...
# build & fit Kriging model
k = Kriging(y, X, "gauss")
# display model
print(k)

# setup another (dense) input sample
x = ...

# use kriging model to predict at x
p = k.predict(x, ...)

# and/or use kriging model to simulate at x
s = k.simulate(nsim = 10, seed = 123, x)

```

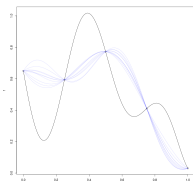
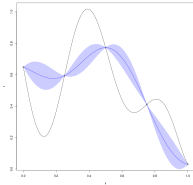
1.2.1 Basic demo

Sample the objective function

$$f : x \rightarrow 1 - \frac{1}{2} \left(\frac{\sin(12x)}{1+x} + 2\cos(7x)x^5 + 0.7 \right)$$

at $X = \{0.0, 0.25, 0.5, 0.75, 1.0\}$, then predict and simulate in $[0, 1]$.

This code, for Python, R or Matlab/Octave should return for both Python: , R: or Matlab/Octave :



1.2.2 SciKit-Learn wrapping

Implement SciKit-Learn BaseEstimator to plot gpr noisy targets (SciKit-Learn example: [https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_targets.py](#)), using libKriging:

```
from sklearn.base import BaseEstimator
import pylibkriging as lk
import numpy as np

class KrigingEstimator(BaseEstimator):
    def __init__(self, kernel="matern3_2", regmodel = "constant", normalize = False, optim_
    ↪="BFGS", objective = "LL", noise = None, parameters = None):
        self.kernel = kernel
        self.regmodel = regmodel
        self.normalize = normalize
        self.optim = optim
        self.objective = objective
        self.noise = noise
        self.parameters = parameters
        if self.parameters is None:
            self.parameters = {}
        if self.noise is None:
            self.kriging = lk.Kriging(self.kernel)
        elif type(self.noise) is float: # homoskedastic user-defined "noise"
            self.kriging = lk.NoiseKriging(self.kernel)
        else:
            raise Exception("noise type not supported:", type(self.noise))

    def fit(self, X, y):
        if self.noise is None:
            self.kriging.fit(y, X, self.regmodel, self.normalize, self.optim, self.objective,
    ↪self.parameters)
        elif type(self.noise) is float: # homoskedastic user-defined "noise"
            self.kriging.fit(y, np.repeat(self.noise, y.size), X, self.regmodel, self.
    ↪normalize, self.optim, self.objective, self.parameters)
        else:
            raise Exception("noise type not supported:", type(self.noise))

    def predict(self, X, return_std=False, return_cov=False):
        return self.kriging.predict(X, return_std, return_cov, False)

    def sample_y(self, X, n_samples = 1, random_state = 0):
        return self.kriging.simulate(nsim = n_samples, seed = random_state, x = X)

    def log_marginal_likelihood(self, theta=None, eval_gradient=False):
        if theta is None:
            return self.kriging.logLikeliHood()
        else:
            return self.kriging.logLikeliHoodFun(theta, eval_gradient)
```

1.3 API

Following API doc supports:

- Python wrapper
- R wrapper
- Octave wrapper
- Matlab wrapper

Python/R/Matlab/Octave syntaxes are almost identical, just diverging through some basic language elements:

| Python | R | Matlab/Octave |
|--------------------|------------------------|--------------------|
| <code>a = b</code> | <code>a <- b</code> | <code>a = b</code> |
| <code>True</code> | <code>TRUE</code> | <code>true</code> |
| <code>False</code> | <code>FALSE</code> | <code>false</code> |
| <code>None</code> | <code>NULL</code> | <code>[]</code> |
| <code>a.b()</code> | <code>a\$b()</code> | <code>a.b()</code> |

1.3.1 Contructors

Kriging

Description

Create a Kriging Object representing a Trend + GP Model

Usage

Just build the model:

```
Kriging(kernel)
# later, call fit(y,X,...)
```

or, build and fit at the same time:

```
Kriging(
  y,
  X,
  kernel,
  regmodel = "constant",
  normalize = FALSE,
  optim = "BFGS",
  objective = "LL",
  parameters = NULL
)
```

Arguments

| Argument | Description |
|-------------------------|--|
| <code>y</code> | Numeric vector of response values. |
| <code>X</code> | Numeric matrix of input design. |
| <code>kernel</code> | Character defining the covariance model: "gauss", "exp", "matern3_2", "matern5_2". |
| <code>regmodel</code> | Universal Kriging linear trend. |
| <code>normalize</code> | Logical. If TRUE both the input matrix <code>X</code> and the response <code>y</code> in normalized to take values in the interval $[0, 1]$. |
| <code>optim</code> | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS", "Newton" and "none", the later simply keeping the values given in <code>parameters</code> . The method "BFGS" uses the gradient of the objective. The method "Newton" uses both the gradient and the Hessian of the objective. |
| <code>objective</code> | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood, "LOO" for the Leave-One-Out sum of squares and "LMP" for the Log-Marginal Posterior. |
| <code>parameters</code> | Initial values for the hyper-parameters. When provided this must be named list with elements "sigma2" and "theta" containing the initial value(s) for the variance and for the range parameters. If <code>theta</code> is a matrix with more than one row, each row is used as a starting point for optimization. |

Details

The hyper-parameters (variance and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective`, using the method given in `optim`.

Value

An object "Kriging". Should be used with its `predict`, `simulate`, `update` methods.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)
## fit and print
k <- Kriging(y, X, kernel = "matern3_2")
k

x <- as.matrix(seq(from = 0, to = 1, length.out = 101))
p <- k$predict(x = x, stdev = TRUE, cov = FALSE)

plot(f)
points(X, y)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)),
border = NA, col = rgb(0, 0, 1, 0.2))
```

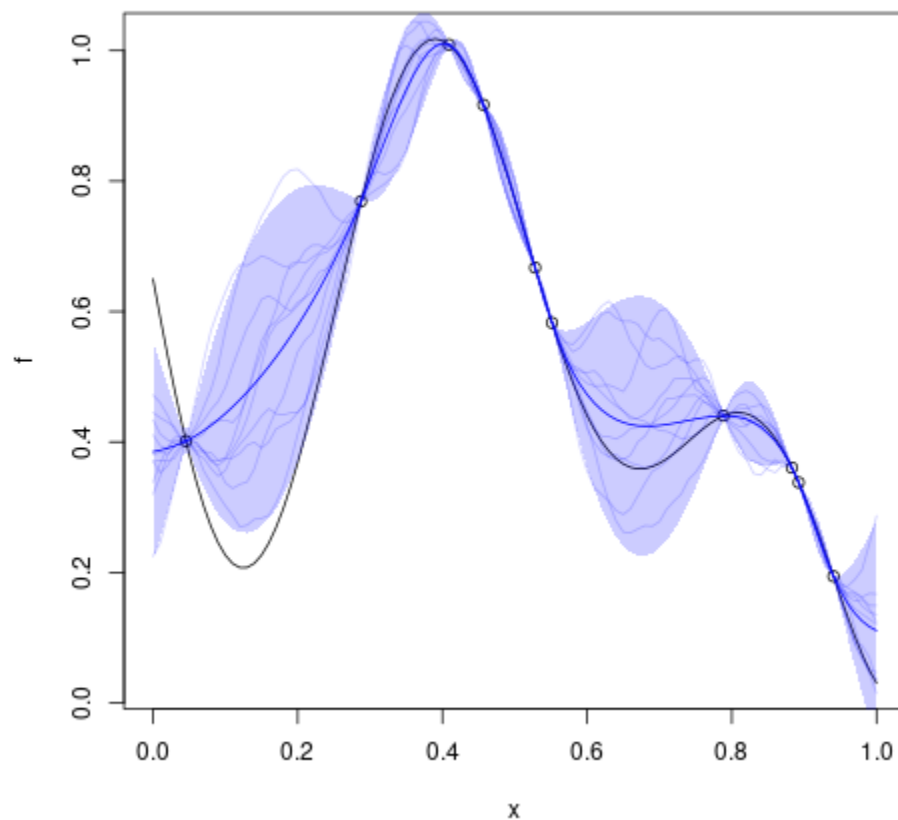
(continues on next page)

(continued from previous page)

```
s <- k$simulate(nsim = 10, seed = 123, x = x)
matlines(x, s, col = rgb(0, 0, 1, 0.2), type = "l", lty = 1)
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
* fit:
  * objective: LL
  * optim: BFGS
```



Kriging::update

Description

Update a Kriging model object with new points

Usage

- Python

```
# k = Kriging(...)
k.update(newy, newX)
```

- R

```
# k = Kriging(...)
k$update(newy, newX)
```

- Matlab/Octave

```
% k = Kriging(...)
k.update(newy, newX)
```

Arguments

| Argument | Description |
|----------|---|
| newy | Numeric vector of new responses (output). |
| newX | Numeric matrix of new input points. |

Examples

```
f <- function(x) 1- 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x)*x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)
points(X, y, col = "blue")

k <- Kriging(y, X, "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
p <- k$predict(x)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)), border = NA,
  ↪col = rgb(0, 0, 1, 0.2))

newX <- as.matrix(runif(3))
newy <- f(newX)
```

(continues on next page)

(continued from previous page)

```

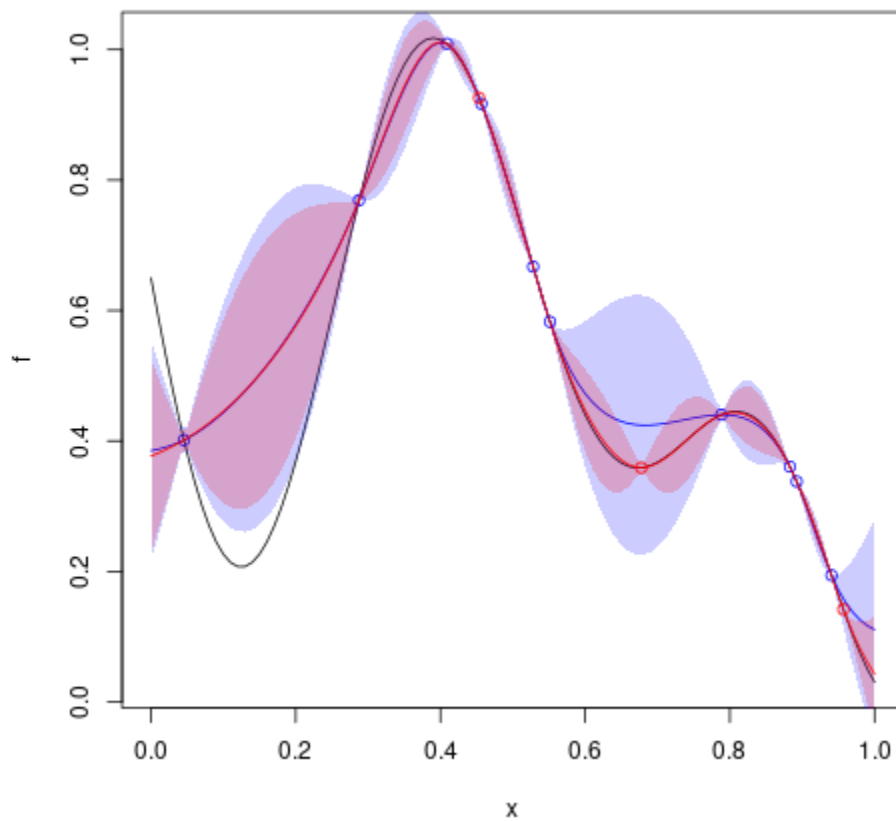
points(newX, newy, col = "red")

## change the content of the object 'k'
k$update(newy, newX)

x <- seq(from = 0, to = 1, length.out = 101)
p2 <- k$predict(x)
lines(x, p2$mean, col = "red")
polygon(c(x, rev(x)), c(p2$mean - 2 * p2$stdev, rev(p2$mean + 2 * p2$stdev)), border = NA, col = rgb(1, 0, 0, 0.2))

```

Results



Kriging::copy**Description**

Duplicate a Kriging Model

Usage

- Python

```
# k = Kriging(...)
k2 = k.copy()
```

- R

```
# k = Kriging(...)
k2 = k$copy()
```

- Matlab/Octave

```
% k = Kriging(...)
k2 = k.copy()
```

Value

The copy of object.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2")
k
k$copy()
```

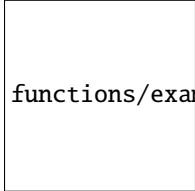
Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
* fit:
  * objective: LL
```

(continues on next page)

(continued from previous page)

```
* optim: BFGS
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
* fit:
  * objective: LL
  * optim: BFGS
```



functions/examples/copy.Kriging.md.png

Kriging::save & Kriging::load

Description

Save/Load a Kriging Model

Usage

- Python

```
# k = Kriging(...)
k.save("k.h5")
k2 = load("k.h5")
```

- R

```
# k = Kriging(...)
k$save("k.h5")
k2 = load("k.h5")
```

- Matlab/Octave

```
% k = Kriging(...)
k.save("k.h5")
k2 = load("k.h5")
```


Value

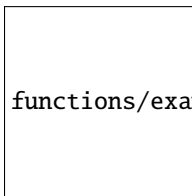
The loaded object.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2")
k
k$save("k.h5")
load("k.h5")
```

Results



functions/examples/saveload.Kriging.md.png

NuggetKriging

Description

Create an object "NuggetKriging" using the libKriging library.

Usage

Just build the model:

```
NuggetKriging(kernel)
# later, call fit(y,X,...)
```

or, build and fit at the same time:

```
NuggetKriging(
  y,
  X,
  kernel,
  regmodel = "constant",
  normalize = FALSE,
  optim = "BFGS",
  objective = "LL",
```

(continues on next page)

(continued from previous page)

```

parameters = NULL
)

```

Arguments

| Argument | Description |
|-------------------------|---|
| <code>y</code> | Numeric vector of response values. |
| <code>X</code> | Numeric matrix of input design. |
| <code>kernel</code> | Character defining the covariance model: "gauss", "exp", "matern3_2", "matern5_2". |
| <code>regmodel</code> | Universal NuggetKriging linear trend. |
| <code>normalize</code> | Logical. If TRUE both the input matrix <code>X</code> and the response <code>y</code> in normalized to take values in the interval $[0, 1]$. |
| <code>optim</code> | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS" and "none", the later simply keeping the values given in <code>parameters</code> . The method "BFGS" uses the gradient of the objective. |
| <code>objective</code> | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood and "LMP" for the Log-Marginal Posterior. |
| <code>parameters</code> | Initial values for the hyper-parameters. When provided this must be named list with some elements "sigma2", "theta", "nugget" containing the initial value(s) for the variance, range and nugget parameters. If <code>theta</code> is a matrix with more than one row, each row is used as a starting point for optimization. |

Details

The hyper-parameters (variance, nugget and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective`, using the method given in `optim`.

Value

An object "NuggetKriging". Should be used with its `predict`, `simulate`, `update` methods.

Examples

```

f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))
## fit and print
k <- NuggetKriging(y, X, kernel = "matern3_2")
k

x <- sort(c(X, as.matrix(seq(from = 0, to = 1, length.out = 101))))
p <- k$predict(x = x, stdev = TRUE, cov = FALSE)

plot(f)

```

(continues on next page)

(continued from previous page)

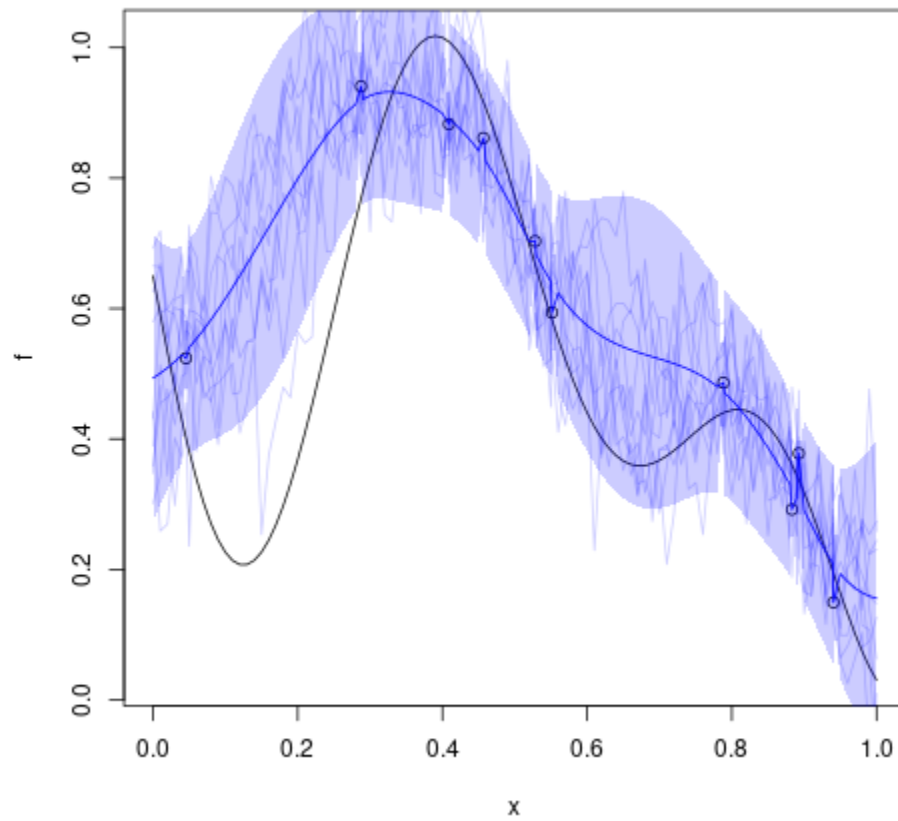
```
points(X, y)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)),
border = NA, col = rgb(0, 0, 1, 0.2))

s <- k$simulate(nsim = 10, seed = 123, x = x)

matlines(x, s, col = rgb(0, 0, 1, 0.2), type = "l", lty = 1)
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
* fit:
  * objective: LL
  * optim: BFGS
```



NuggetKriging::update

Description

Update a NuggetKriging model object with new points

Usage

- Python

```
# k = NuggetKriging(...)
k.update(newy, newX)
```

- R

```
# k = NuggetKriging(...)
k$update(newy, newX)
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.update(newy, newX)
```

Arguments

| Argument | Description |
|----------|---|
| newy | Numeric vector of new responses (output). |
| newX | Numeric matrix of new input points. |

Examples

```
f <- function(x) 1- 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x)*x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))
points(X, y, col = "blue")

k <- NuggetKriging(y, X, "matern3_2")

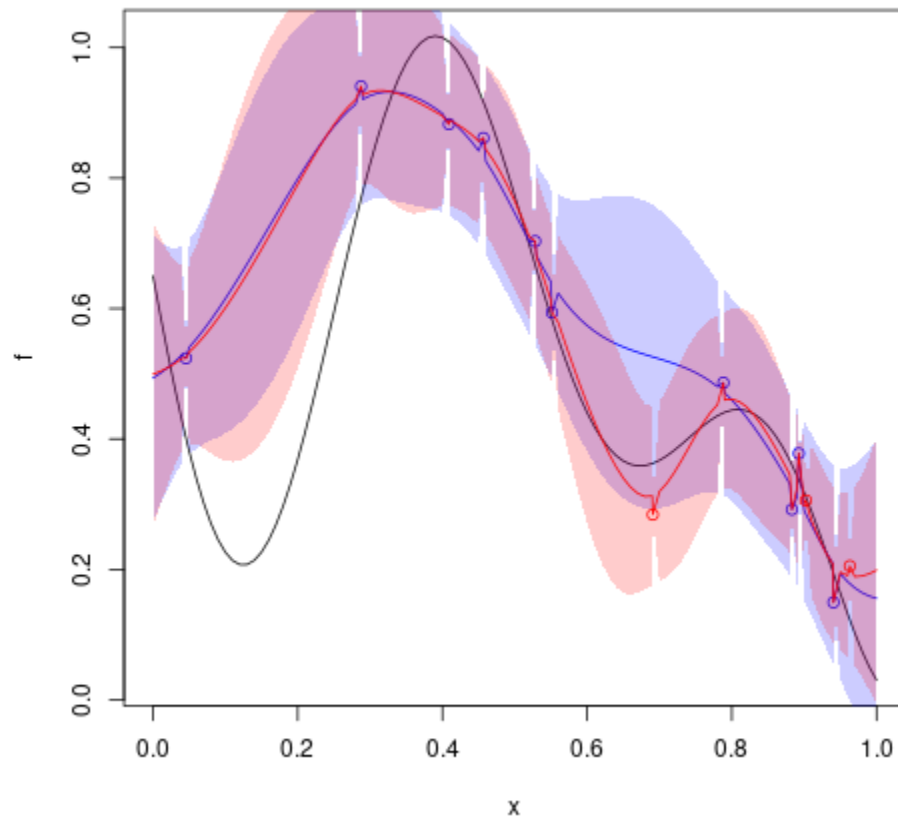
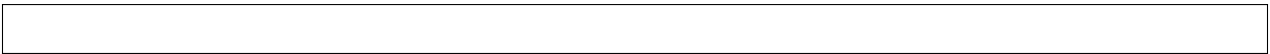
x <- sort(c(X,seq(from = 0, to = 1, length.out = 101))) # include design points to see
↪ interpolation
p <- k$predict(x)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)), border = NA,
↪ col = rgb(0, 0, 1, 0.2))

newX <- as.matrix(runif(3))
newy <- f(newX) + 0.1 * rnorm(nrow(newX))
points(newX, newy, col = "red")

## change the content of the object 'k'
k$update(newy, newX)

x <- sort(c(X,newX,seq(from = 0, to = 1, length.out = 101))) # include design points to
↪ see interpolation
p2 <- k$predict(x)
lines(x, p2$mean, col = "red")
polygon(c(x, rev(x)), c(p2$mean - 2 * p2$stdev, rev(p2$mean + 2 * p2$stdev)), border =
↪ NA, col = rgb(1, 0, 0, 0.2))
```

Results



`NuggetKriging::copy`

Description

Duplicate a `NuggetKriging` Model

Usage

- Python

```
# k = NuggetKriging(...)
k2 = k.copy()
```

- R

```
# k = NuggetKriging(...)
k2 = k$copy()
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k2 = k.copy()
```

Value

The copy of object.

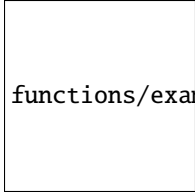
Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

k <- NuggetKriging(y, X, kernel = "matern3_2")
k
k$copy()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
  * fit:
    * objective: LL
    * optim: BFGS
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
  * fit:
    * objective: LL
    * optim: BFGS
```



functions/examples/copy.NuggetKriging.md.png

NuggetKriging::save & NuggetKriging::load

Description

Save/Load a NuggetKriging Model

Usage

- Python

```
# k = NuggetKriging(...)
k.save("k.h5")
k2 = load("k.h5")
```

- R

```
# k = NuggetKriging(...)
k$save("k.h5")
k2 = load("k.h5")
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.save("k.h5")
k2 = load("k.h5")
```

Value

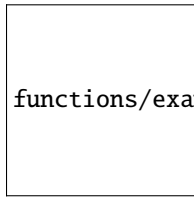
The loaded object.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

k <- NuggetKriging(y, X, kernel = "matern3_2")
k
k$save("k.h5")
load("k.h5")
```


Results



functions/examples/saveload.NuggetKriging.md.png

NoiseKriging

Description

Create an object "NoiseKriging" using the libKriging library.

Usage

Just build the model:

```
NoiseKriging(kernel)
# later, call fit(y,X,...)
```

or, build and fit at the same time:

```
NoiseKriging(
  y,
  noise,
  X,
  kernel,
  regmodel = "constant",
  normalize = FALSE,
  optim = "BFGS",
  objective = "LL",
  parameters = NULL
)
```

Arguments

| Argument | Description |
|-------------------------|---|
| <code>y</code> | Numeric vector of response values. |
| <code>noise</code> | Numeric vector of response variances. |
| <code>X</code> | Numeric matrix of input design. |
| <code>kernel</code> | Character defining the covariance model: "gauss", "exp", "matern3_2", "matern5_2". |
| <code>regmodel</code> | Universal NoiseKriging linear trend. |
| <code>normalize</code> | Logical. If TRUE both the input matrix <code>X</code> and the response <code>y</code> in normalized to take values in the interval $[0, 1]$. |
| <code>optim</code> | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS" and "none", the later simply keeping the values given in <code>parameters</code> . The method "BFGS" uses the gradient of the objective. |
| <code>objective</code> | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood. |
| <code>parameters</code> | Numerical values for the hyper-parameters. When provided this must be named list with elements "sigma2" and "theta" containing the initial value(s) for the variance and for the range parameters. If <code>theta</code> is a matrix with more than one row, each row is used as a starting point for optimization. |

Details

The hyper-parameters (variance and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective`, using the method given in `optim`.

Value

An object "NoiseKriging". Should be used with its `predict`, `simulate`, `update` methods.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X)) # add noise dep. on X
## fit and print
k <- NoiseKriging(y, noise=(X/10)^2, X, kernel = "matern3_2")
k

x <- as.matrix(seq(from = 0, to = 1, length.out = 101))
p <- k$predict(x = x, stdev = TRUE, cov = FALSE)

plot(f)
points(X, y)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)),
border = NA, col = rgb(0, 0, 1, 0.2))
```

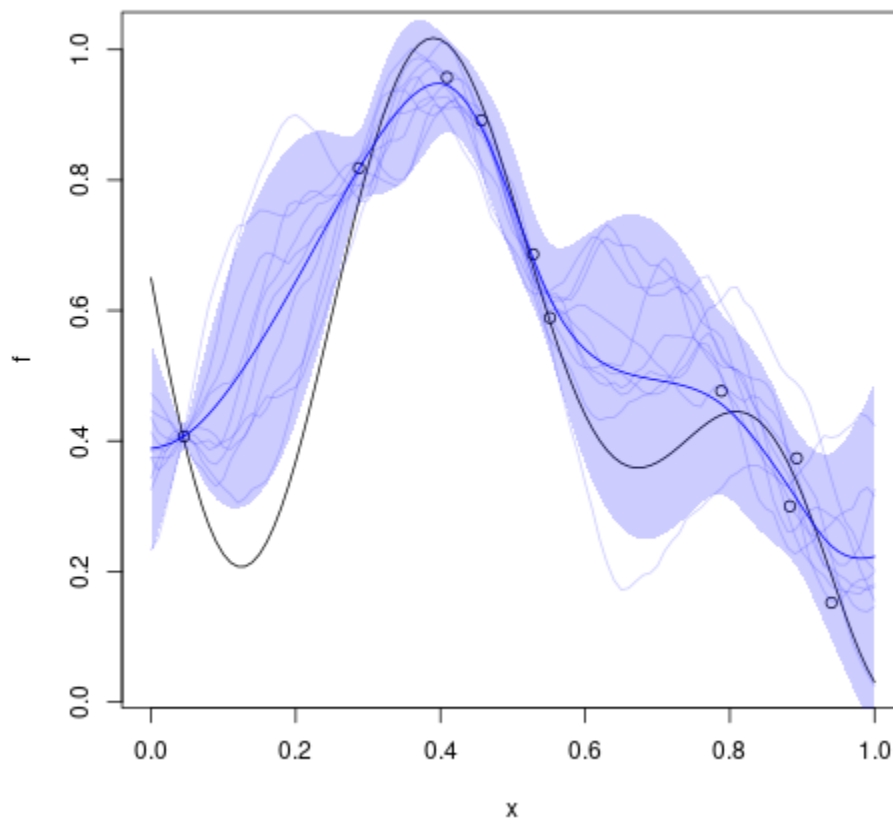
(continues on next page)

(continued from previous page)

```
s <- k$simulate(nsim = 10, seed = 123, x = x)
matlines(x, s, col = rgb(0, 0, 1, 0.2), type = "l", lty = 1)
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
  00278895, 0.00796412, 0.00304081, 0.00208497
* fit:
  * objective: LL
  * optim: BFGS
```



NoiseKriging::update

Description

Update a NoiseKriging model object with new points

Usage

- Python

```
# k = NoiseKriging(...)
k.update(newy, newnoise, newX)
```

- R

```
# k = NoiseKriging(...)
k$update(newy, newnoise, newX)
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k.update(newy, newnoise, newX)
```

Arguments

| Argument | Description |
|----------|---|
| newy | Numeric vector of new responses (output). |
| newnoise | Numeric vector of new noise variances (output). |
| newX | Numeric matrix of new input points. |

Examples

```
f <- function(x) 1- 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x)*x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X))
points(X, y, col = "blue")

k <- NoiseKriging(y, (X/10)^2, X, "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
p <- k$predict(x)
lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)), border = NA,
  ↪col = rgb(0, 0, 1, 0.2))

newX <- as.matrix(runif(3))
```

(continues on next page)

(continued from previous page)

```

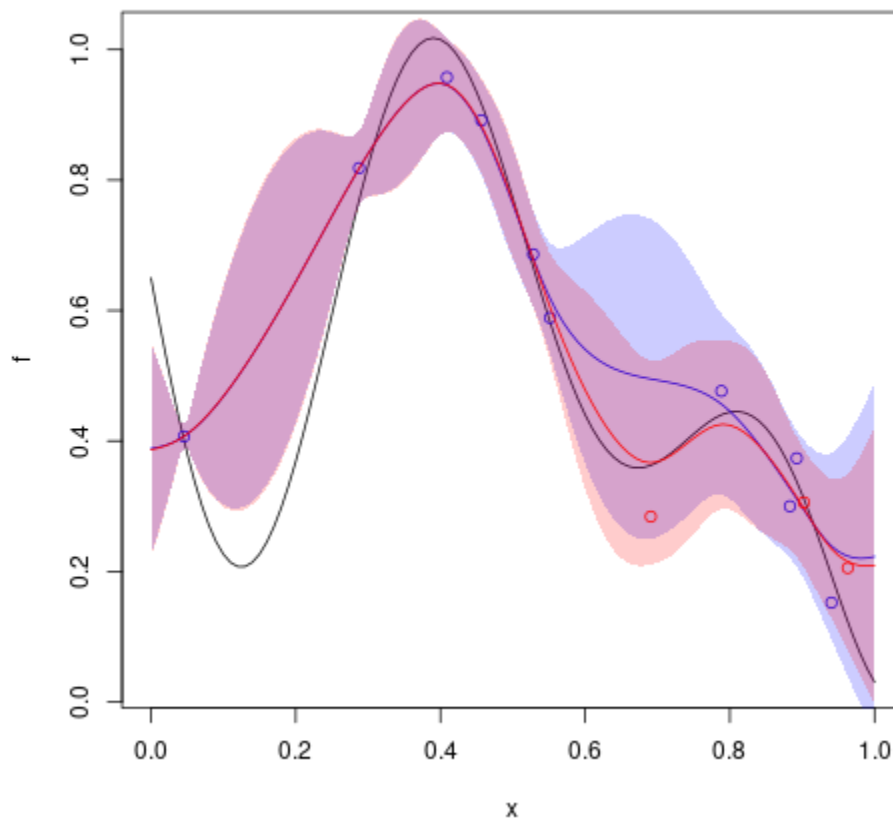
newy <- f(newX) + 0.1 * rnorm(nrow(newX))
points(newX, newy, col = "red")

## change the content of the object 'k'
k$update(newy, rep(0.1^2,3), newX)

x <- seq(from = 0, to = 1, length.out = 101)
p2 <- k$predict(x)
lines(x, p2$mean, col = "red")
polygon(c(x, rev(x)), c(p2$mean - 2 * p2$stdev, rev(p2$mean + 2 * p2$stdev)), border = NA, col = rgb(1, 0, 0, 0.2))

```

Results



NoiseKriging::copy

Description

Duplicate a NoiseKriging Model

Usage

- Python

```
# k = NoiseKriging(...)
k2 = k.copy()
```

- R

```
# k = NoiseKriging(...)
k2 = k$copy()
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k2 = k.copy()
```

Value

The copy of object.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X)) # add noise dep. on X

k <- NoiseKriging(y, noise=(X/10)^2, X, kernel = "matern3_2")
k
k$copy()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
↪ 0.00278895, 0.00796412, 0.00304081, 0.00208497
```

(continues on next page)

(continued from previous page)

```

* fit:
  * objective: LL
  * optim: BFGS
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
↪00278895, 0.00796412, 0.00304081, 0.00208497
* fit:
  * objective: LL
  * optim: BFGS

```

functions/examples/copy.NoiseKriging.md.png

NoiseKriging::save & NoiseKriging::load

Description

Save/Load a NoiseKriging Model

Usage

- Python

```

# k = NoiseKriging(...)
k.save("k.h5")
k2 = load("k.h5")

```

- R

```

# k = NoiseKriging(...)
k$save("k.h5")
k2 = load("k.h5")

```

- Matlab/Octave

```

% k = NoiseKriging(...)
k.save("k.h5")
k2 = load("k.h5")

```

Value

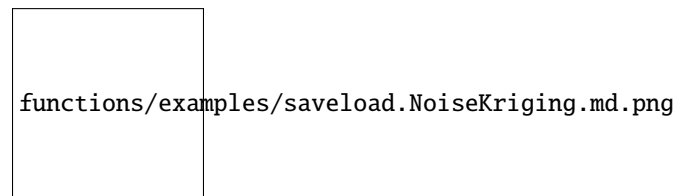
The loaded object.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X)) # add noise dep. on X

k <- NoiseKriging(y, noise=(X/10)^2, X, kernel = "matern3_2")
k
k$save("k.h5")
load("k.h5")
```

Results



1.3.2 Fit objective

Kriging::fit

Description

Fit a Kriging Object using Given Observations

Usage

- Python

```
# k = Kriging(kernel=...)
k.fit(y, X,
      regmodel = "constant",
      normalize = False,
      optim = "BFGS",
      objective = "LL",
      parameters = None)
```

- R


```
# k = Kriging(kernel=...)
k$fit(y, X,
      regmodel = "constant",
      normalize = FALSE,
      optim = "BFGS",
      objective = "LL",
      parameters = NULL)
```

- Matlab/Octave

```
% k = Kriging(kernel=...)
k.fit(y, X,
      regmodel = "constant",
      normalize = false,
      optim = "BFGS",
      objective = "LL",
      parameters = [])
```

Arguments

| Argument | Description |
|-------------------|--|
| y | Numeric vector of response values. |
| X | Numeric matrix of input design. |
| regmodel | Universal Kriging linear trend. |
| normalize | Logical. If TRUE both the input matrix X and the response y in normalized to take values in the interval $[0, 1]$. |
| optim | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS" , "Newton" and "none" , the later simply keeping the values given in parameters . The method "BFGS" uses the gradient of the objective. The method "Newton" uses both the gradient and the Hessian of the objective. |
| objective | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood, "LOO" for the Leave-One-Out sum of squares and "LMP" for the Log-Marginal Posterior. |
| parameters | Initial values for the hyper-parameters. When provided this must be named list with elements " sigma2 " and " theta " containing the initial value(s) for the variance and for the range parameters. If theta is a matrix with more than one row, each row is used as a starting point for optimization. |

Details

The hyper-parameters (variance and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective` , using the method given in `optim` .

Examples

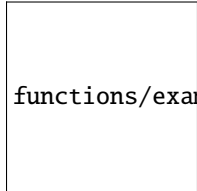
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging("matern3_2")
print("before fit")
print(k)

k$fit(y,X)
print("after fit")
print(k)
```

Results

```
[1] "before fit"
* covariance:
  * kernel: matern3_2
[1] "after fit"
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
  * fit:
    * objective: LL
    * optim: BFGS
```



functions/examples/fit.Kriging.md.png

Kriging::logLikelihood

Description

Get the Maximized Log-Likelihood of a Kriging Model Object

Usage

- Python

```
# k = Kriging(...)
k.logLikelihood()
```

- R

```
# k = Kriging(...)
k$logLikelihood()
```

- Matlab/Octave

```
% k = Kriging(...)
k.logLikelihood()
```

Details

See *logLikelihoodFun.Kriging* for more details on the profile log-likelihood function used in the maximization.

Value

The value of the maximized profile log-likelihood $\ell_{\text{prof}}(\hat{\theta})$. This is also the value $\ell(\hat{\theta}, \hat{\sigma}^2, \hat{\beta})$ of the maximized log-likelihood.

Examples

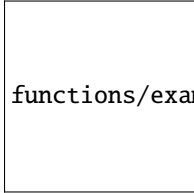
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2", objective="LL")
print(k)

k$logLikelihood()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
* fit:
  * objective: LL
  * optim: BFGS
[1] 8.62771
```



functions/examples/logLikelihood.Kriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/Kriging.cpp#L94>

Kriging::logLikelihoodFun

Description

Compute the Profile Log-Likelihood of a Kriging Model Object for a given Vector θ of Correlation Ranges

Usage

- Python

```
# k = Kriging(...)
k.logLikelihoodFun(theta)
```

- R

```
# k = Kriging(...)
k$logLikelihoodFun(theta)
```

- Matlab/Octave

```
% k = Kriging(...)
k.logLikelihoodFun(theta)
```

Arguments

| Argument | Description |
|----------|--|
| theta | A numeric vector of (positive) range parameters at which the profile log-likelihood will be evaluated. |
| grad | Logical. Should the function return the gradient? |
| hess | Logical. Should the function return Hessian? |

Details

The profile log-likelihood $\ell_{\text{prof}}(\theta)$ is obtained from the log-likelihood function $\ell(\theta, \sigma^2, \beta)$ by replacing the GP variance σ^2 and the vector β of trend coefficients by their ML estimates $\hat{\sigma}^2$ and $\hat{\beta}$ which are obtained by Generalized Least Squares. See [here](#) for more details.

Value

The value of the profile log-likelihood $\ell_{\text{prof}}(\theta)$ for the given vector θ of correlation ranges.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

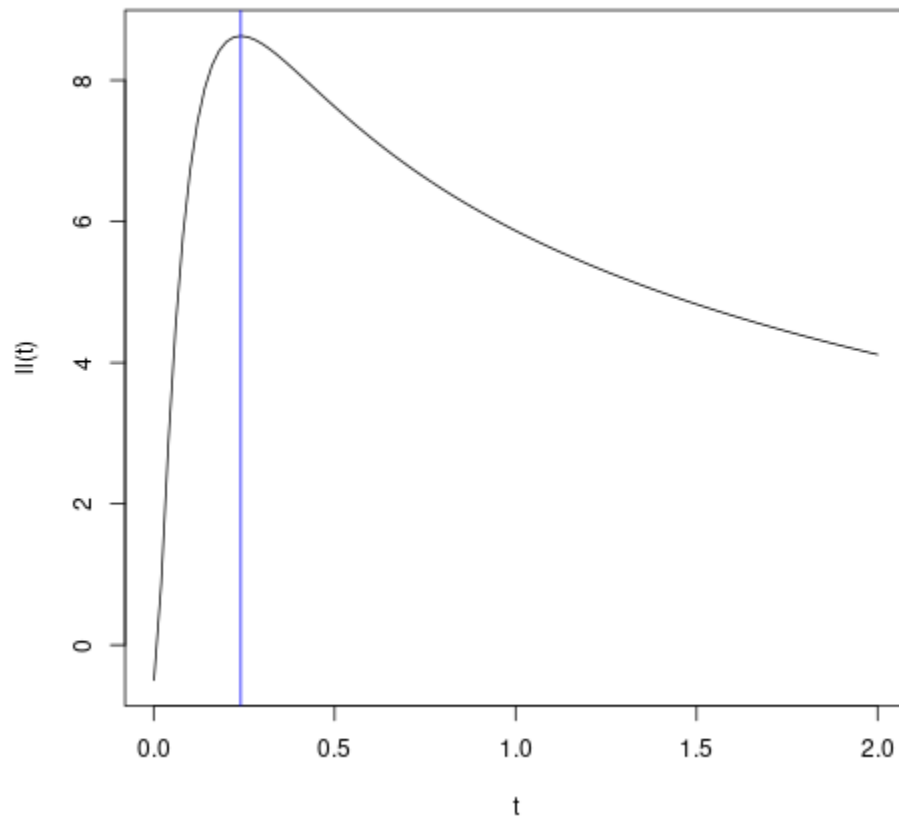
k <- Kriging(y, X, kernel = "matern3_2")
print(k)

ll <- function(theta) k$logLikelihoodFun(theta)$logLikelihood

t <- seq(from = 0.001, to = 2, length.out = 101)
plot(t, ll(t), type = 'l')
abline(v = k$theta(), col = "blue")
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.433954
* variance (est.): 0.0873685
* covariance:
  * kernel: matern3_2
  * range (est.): 0.240585
* fit:
  * objective: LL
  * optim: BFGS
```



Kriging::leaveOneOut

Description

Get the Minimized Leave-One-Out Sum of Squares of a Kriging Model

Usage

- Python

```
# k = Kriging(...)  
k.leaveOneOut()
```

- R

```
# k = Kriging(...)  
k$leaveOneOut()
```

- Matlab/Octave

```
% k = Kriging(...)
k.leaveOneOut()
```

Value

The *minimized* Leave-One-Out (LOO) sum of squares SSE_{LOO} , corresponding to the estimated value $\hat{\theta}$ of the vector of correlation ranges. See [leaveOneOutFun.Kriging](#) for more details.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2", objective="LOO")
print(k)

k$leaveOneOut()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.406331
* variance (est.): 0.0471509
* covariance:
  * kernel: matern3_2
  * range (est.): 0.284722
  * fit:
    * objective: LOO
    * optim: BFGS
[1] 0.003159176
```

functions/examples/leaveOneOut.Kriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/Kriging.cpp#L350>

Kriging::leaveOneOutFun

Description

Compute the Leave-One-Out (LOO) Sum of Squares of Errors for a `Kriging` Object and a Vector θ of Correlation Ranges

Usage

- Python

```
# k = Kriging(...)
k.logMargPostFun(theta, grad = FALSE)
```

- R

```
# k = Kriging(...)
k$logMargPostFun(theta, grad = FALSE)
```

- Matlab/Octave

```
% k = Kriging(...)
k.logMargPostFun(theta, grad = FALSE)
```

Arguments

| Argument | Description |
|----------|---|
| theta | A numeric vector of range parameters at which the LOO sum of squares will be evaluated. |
| grad | Logical. Should the gradient (w.r.t. theta) be returned? |

Details

The Leave-One-Out (LOO) sum of squares is defined by $SS_{\text{LOO}}(\theta) := \sum_{i=1}^n \{y_i - \hat{y}_{i|-i}\}^2$ where $\hat{y}_{i|-i}$ denotes the prediction of y_i based on the observations y_j with $j \neq i$. The vector \hat{y}_{LOO} of LOO predictions is computed efficiently, see [here](#) for details.

Value

The value $\text{SSE}_{\text{LOO}}(\theta)$ of the Leave-One-Out Sum of Squares for the given vector θ of correlation ranges.

Examples

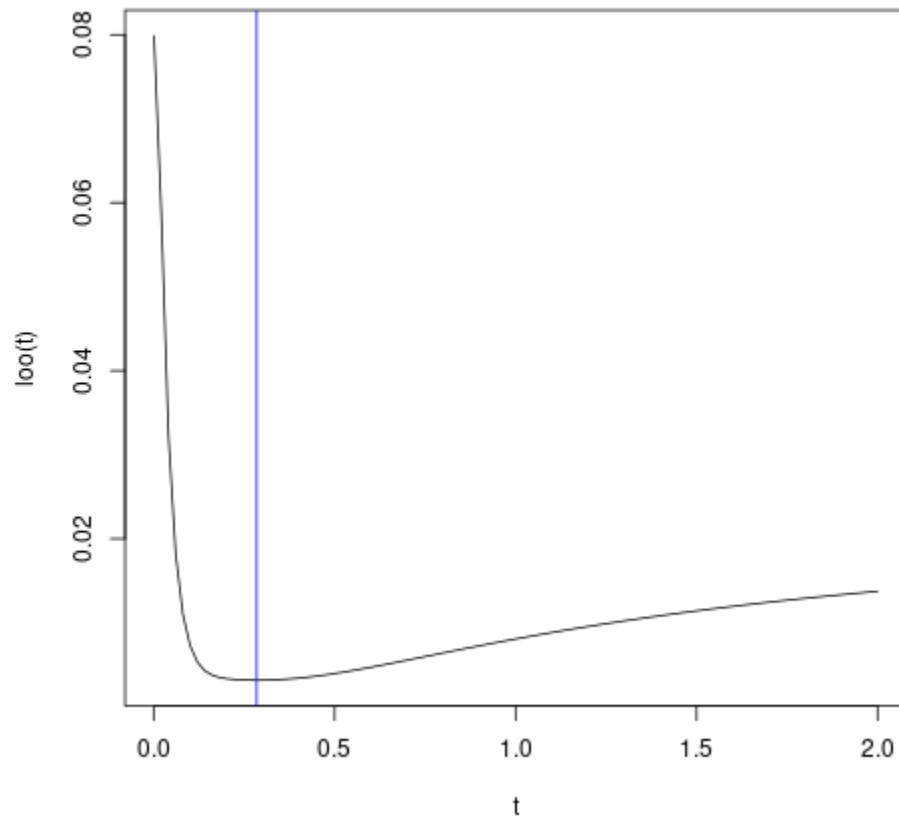
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2", objective = "LOO", optim="BFGS")
print(k)

loo <- function(theta) k$leaveOneOutFun(theta)$leaveOneOut
t <- seq(from = 0.001, to = 2, length.out = 101)
plot(t, loo(t), type = "l")
abline(v = k$theta(), col = "blue")
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.406331
* variance (est.): 0.0471509
* covariance:
  * kernel: matern3_2
  * range (est.): 0.284722
  * fit:
    * objective: LOO
    * optim: BFGS
```



Kriging::logMargPost

Description

Get the Maximized Log-Marginal Posterior Density of a Kriging Model

Usage

- Python

```
# k = Kriging(...)
k.logMargPost()
```

- R

```
# k = Kriging(...)
k$logMargPost()
```

- Matlab/Octave

```
% k = Kriging(...)
k.logMargPost()
```

Details

Using the jointly robust prior $\pi_{JR}(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta})$ the marginal or integrated posterior is the function of $\boldsymbol{\theta}$ obtained from the posterior density by marginalizing out the GP variance σ^2 and the vector $\boldsymbol{\beta}$ of trend coefficients. See [logMargPostFun.Kriging](#) for the log-marginal posterior density. By maximizing this function w.r.t. $\boldsymbol{\theta}$ we get estimated correlation ranges which are warranted to be positive and finite $0 < \theta_k < \infty$.

Value

The maximal value of the log-marginal posterior density, corresponding to the estimated value of the vector $\boldsymbol{\theta}$ of correlation ranges.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

k <- Kriging(y, X, kernel = "matern3_2", objective="LMP")
print(k)

k$logMargPost()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.388566
* variance (est.): 0.158896
* covariance:
  * kernel: matern3_2
  * range (est.): 0.313364
  * fit:
    * objective: LMP
    * optim: BFGS
[1] 10.64938
```

functions/examples/logMargPost.Kriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/Kriging.cpp#L494>
- The [RobustGaSP](#) R package

Kriging::logMargPostFun

Description

Compute the Log-Marginal Posterior Density of a Kriging Model Object for a given Vector θ of Correlation Ranges

Usage

- Python

```
# k = Kriging(...)
k.logMargPostFun(theta, grad = FALSE)
```

- R

```
# k = Kriging(...)
k$logMargPostFun(theta, grad = FALSE)
```

- Matlab/Octave

```
% k = Kriging(...)
k.logMargPostFun(theta, grad = FALSE)
```

Arguments

| Argument | Description |
|----------|--|
| theta | Numeric vector of correlation range parameters at which the function is to be evaluated. |
| grad | Logical. Should the function return the gradient (w.r.t theta)? |

Details

The log-marginal posterior density relates to the [jointly robust prior](#) $\pi_{\text{JR}}(\theta, \sigma^2, \beta) \propto \pi(\theta) \sigma^{-2}$. The marginal (or integrated) posterior is the function θ obtained by marginalizing out the GP variance σ^2 and the vector β of trend coefficients. Due to the form of the prior, the marginalization can be done on the likelihood $p_{\text{marg}}(\theta | \mathbf{y}) \propto \pi(\theta) \times L_{\text{marg}}(\theta; \mathbf{y})$.

Value

The value of the log-marginal posterior density $\log p_{\text{marg}}(\theta | y)$. By maximizing this function we should get the estimate of θ obtained when using `objective = "LMP"` in the `fit.Kriging` method.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)

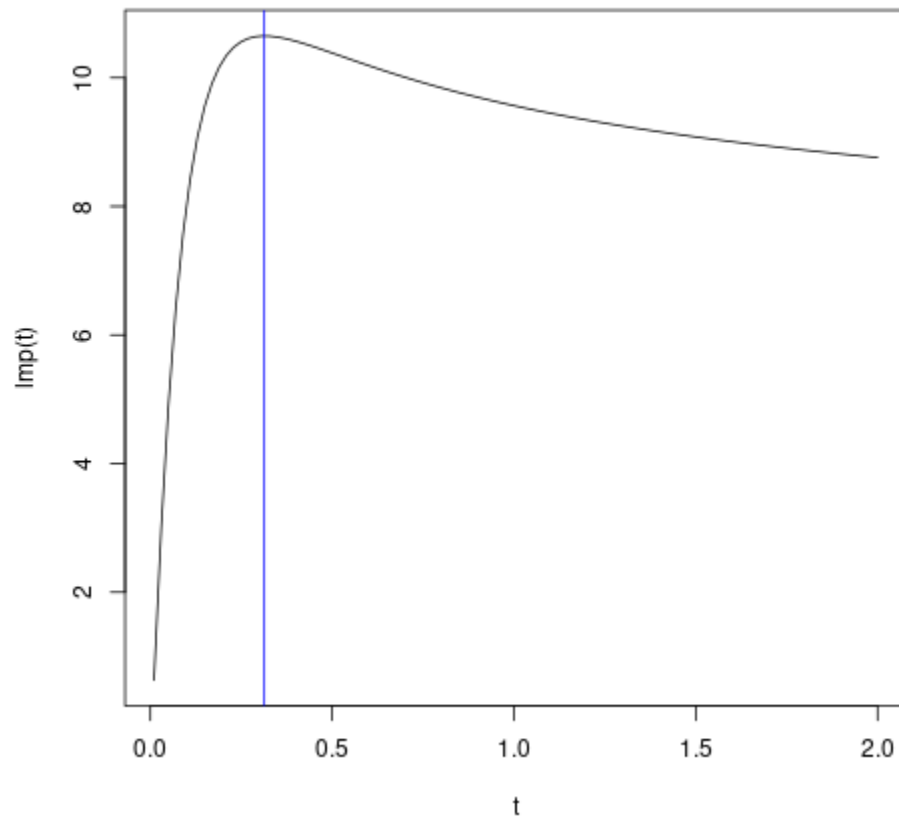
k <- Kriging(y, X, "matern3_2", objective="LMP")
print(k)

lmp <- function(theta) k$logMargPostFun(theta)$logMargPost

t <- seq(from = 0.01, to = 2, length.out = 101)
plot(t, lmp(t), type = "l")
abline(v = k$theta(), col = "blue")
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.194057,1.00912]
* trend constant (est.): 0.388566
* variance (est.): 0.158896
* covariance:
  * kernel: matern3_2
  * range (est.): 0.313364
  * fit:
    * objective: LMP
    * optim: BFGS
```



NuggetKriging::fit

Description

Fit a NuggetKriging Model Object using given Observations

Usage

- Python

```
# k = NuggetKriging(kernel=...)
k.fit(y, X,
      regmodel = "constant",
      normalize = False,
      optim = "BFGS",
      objective = "LL",
      parameters = None)
```

- R

```
# k = NuggetKriging(kernel=...)
k$fit(y, X,
      regmodel = "constant",
      normalize = FALSE,
      optim = "BFGS",
      objective = "LL",
      parameters = NULL)
```

- Matlab/Octave

```
% k = NuggetKriging(kernel=...)
k.fit(y, X,
      regmodel = "constant",
      normalize = false,
      optim = "BFGS",
      objective = "LL",
      parameters = [])
```

Arguments

| Argument | Description |
|------------|--|
| y | Numeric vector of response values. |
| X | Numeric matrix of input design. |
| regmodel | Universal NuggetKriging linear trend. |
| normalize | Logical. If TRUE both the input matrix X and the response y in normalized to take values in the interval [0, 1]. |
| optim | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS" and "none", the later simply keeping the values given in parameters . The method "BFGS" uses the gradient of the objective. |
| objective | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood and "LMP" for the Log-Marginal Posterior. |
| parameters | Initial values for the hyper-parameters. When provided this must be named list with some elements "sigma2", "theta", "nugget" containing the initial value(s) for the variance, range and nugget parameters. If theta is a matrix with more than one row, each row is used as a starting point for optimization. |
| kernel | Character defining the covariance model: "exp", "gauss", "matern3_2", "matern5_2" . |

Details

The hyper-parameters (variance and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective`, using the method given in `optim`.

Examples

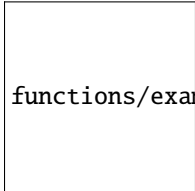
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

k <- NuggetKriging("matern3_2")
print("before fit")
print(k)

k$fit(y,X)
print("after fit")
print(k)
```

Results

```
[1] "before fit"
* covariance:
  * kernel: matern3_2
[1] "after fit"
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
  * fit:
    * objective: LL
    * optim: BFGS
```



functions/examples/fit.NuggetKriging.md.png

NuggetKriging::logLikelihood

Description

Get the Maximized Log-Likelihood of a NuggetKriging Model Object

Usage

- Python

```
# k = NuggetKriging(...)
k.logLikelihood()
```

- R

```
k$logLikelihood()
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.logLikelihood()
```

Details

See *logLikelihoodFun.NuggetKriging* for more details on the corresponding profile log-likelihood function.

Value

The value of the maximized profile log-likelihood $\ell_{\text{prof}}(\hat{\boldsymbol{\theta}}, \hat{\alpha})$ where $\alpha := \sigma^2 / (\sigma^2 + \nu^2)$ is the ratio of the variances σ^2 for the GP and $\sigma^2 + \nu^2$ for the GP + nugget. This is also the value $\ell(\hat{\boldsymbol{\theta}}, \hat{\alpha}, \hat{\sigma}^2, \hat{\boldsymbol{\beta}})$ or $\ell(\hat{\boldsymbol{\theta}}, \hat{\sigma}^2, \hat{\tau}^2, \hat{\boldsymbol{\beta}})$ of the maximized log-likelihood.

Examples

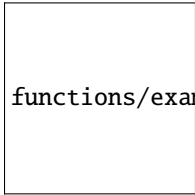
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

k <- NuggetKriging(y, X, kernel = "matern3_2", objective="LL")
print(k)

k$logLikelihood()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
* fit:
  * objective: LL
  * optim: BFGS
[1] 4.95114
```



functions/examples/logLikelihood.NuggetKriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NuggetKriging.cpp#L94>

NuggetKriging::logLikelihoodFun

Description

Compute the Profile Log-Likelihood of a NuggetKriging Model for given Vector θ of Correlation Ranges and a given Ratio of Variances GP/(GP + nugget)

Usage

- Python

```
# k = NuggetKriging(...)
k.logLikelihoodFun(theta_alpha, grad = FALSE)
```

- R

```
# k = NuggetKriging(...)
k$logLikelihoodFun(theta_alpha, grad = FALSE)
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.logLikelihoodFun(theta_alpha, grad = FALSE)
```

Arguments

| Argument | Description |
|-------------|--|
| theta_alpha | A numeric vector of (positive) range parameters and variance over nugget + variance at which the log-likelihood will be evaluated. |
| grad | Logical. Should the function return the gradient? |

Details

Consider the log-likelihood function $\ell(\boldsymbol{\theta}, \sigma^2, \tau^2, \boldsymbol{\beta})$ where σ^2 and τ^2 are the variances of the GP and the nugget components. A re-parameterization can be used with the two variances replaced by $\nu^2 := \sigma^2 + \tau^2$ and $\alpha := \sigma^2/(\sigma^2 + \tau^2)$. The profile log-likelihood is then obtained by replacing the variance $\nu^2 := \sigma^2 + \tau^2$ and the vector $\boldsymbol{\beta}$ of trend coefficients by their ML estimates $\hat{\nu}^2$ and $\hat{\boldsymbol{\beta}}$ which are obtained by Generalized Least Squares. See [here](#) for more details.

Value

The value of the profile log-likelihood $\ell_{\text{prof}}(\boldsymbol{\theta}, \alpha)$ for the given vector $\boldsymbol{\theta}$ of correlation ranges and the given variance ratio $\alpha := \sigma^2/(\sigma^2 + \tau^2)$ where σ^2 and τ^2 stand for the GP and the nugget variance. The parameters must be such that $\theta_k > 0$ for $k = 1, \dots, d$ and $0 < \alpha < 1$.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

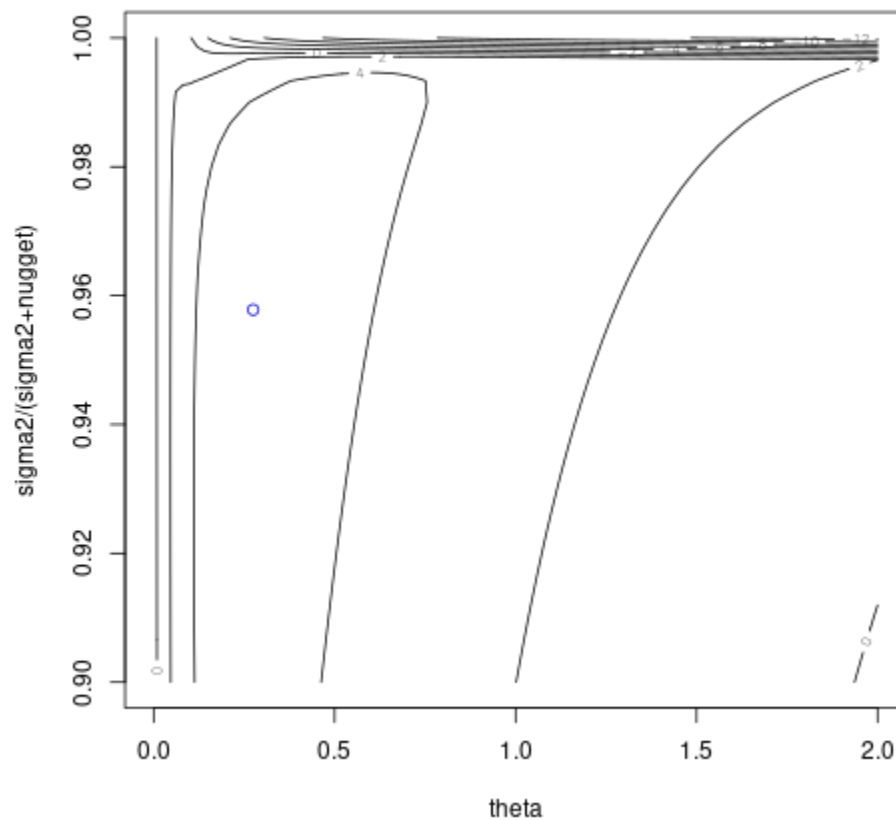
k <- NuggetKriging(y, X, kernel = "matern3_2")
print(k)

# theta0 = k$theta()
# ll_alpha <- function(alpha) k$logLikelihoodFun(cbind(theta0,alpha))$logLikelihood
# a <- seq(from = 0.9, to = 1.0, length.out = 101)
# plot(a, Vectorize(ll_alpha)(a), type = "l", xlim=c(0.9,1))
# abline(v = k$sigma2()/(k$sigma2()+k$nugget()), col = "blue")
#
# alpha0 = k$sigma2()/(k$sigma2()+k$nugget())
# ll_theta <- function(theta) k$logLikelihoodFun(cbind(theta,alpha0))$logLikelihood
# t <- seq(from = 0.001, to = 2, length.out = 101)
# plot(t, Vectorize(ll_theta)(t), type = 'l')
# abline(v = k$theta(), col = "blue")

ll <- function(theta_alpha) k$logLikelihoodFun(theta_alpha)$logLikelihood
a <- seq(from = 0.9, to = 1.0, length.out = 31)
t <- seq(from = 0.001, to = 2, length.out = 101)
contour(t,a,matrix(ncol=length(a),ll(expand.grid(t,a))),xlab="theta",ylab="sigma2/
↳ (sigma2+nugget)")
points(k$theta(),k$sigma2()/(k$sigma2()+k$nugget()),col='blue')
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.488124
* variance (est.): 0.0788813
* covariance:
  * kernel: matern3_2
  * range (est.): 0.275004
  * nugget (est.): 0.00347449
* fit:
  * objective: LL
  * optim: BFGS
```



NuggetKriging::logMargPost

Description

Get the Maximized Log-Marginal Posterior Density of a NuggetKriging Model

Usage

- Python

```
# k = NuggetKriging(...)
k.logMargPost()
```

- R

```
# k = NuggetKriging(...)
k$logMargPost()
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.logMargPost()
```

Details

Using the [jointly robust](#) prior $\pi_{JR}(\boldsymbol{\theta}, \alpha, \sigma^2, \boldsymbol{\beta})$ the marginal or integrated posterior is the function of $\boldsymbol{\theta}$ and α obtained from the posterior density by marginalizing out the GP variance σ^2 and the vector $\boldsymbol{\beta}$ of trend coefficients. See [logMargPostFun.NuggetKriging](#) for the log-marginal posterior density. By maximizing this function w.r.t. $\boldsymbol{\theta}$ and α we get estimated correlation ranges which are warranted to be positive and finite $0 < \theta_k < \infty$. The estimated variance ratio is such that $0 < \alpha < 1$.

Value

The maximal value of the log-marginal posterior density, corresponding to the estimated value of the vector $[\boldsymbol{\theta}, \alpha]$ where $\boldsymbol{\theta}$ is the vector of correlation ranges and $\alpha := \sigma^2 / (\sigma^2 + \tau^2)$ is the ratio of variance GP/(GP + nugget).

Examples

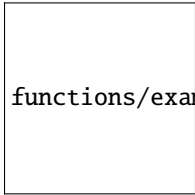
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

k <- NuggetKriging(y, X, kernel = "matern3_2", objective="LMP")
print(k)

k$logMargPost()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.389559
* variance (est.): 0.192207
* covariance:
  * kernel: matern3_2
  * range (est.): 0.434061
  * nugget (est.): 0.00330572
* fit:
  * objective: LMP
  * optim: BFGS
[1] 7.010943
```



functions/examples/logMargPost.NuggetKriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NuggetKriging.cpp#L494>
- The RobustGaSP R package

NuggetKriging::logMargPostFun

Description

Compute the Log-Marginal Posterior Density of a NuggetKriging Model for a given Vector θ of Correlation Ranges and a given Ratio $\sigma^2/(\sigma^2 + \tau^2)$ of Variances GP/(GP + nugget)

Usage

- Python

```
# k = Kriging(...)
k.logMargPostFun(theta_alpha, grad = FALSE)
```

- R

```
# k = Kriging(...)
k$logMargPostFun(theta_alpha, grad = FALSE)
```

- Matlab/Octave

```
% k = Kriging(...)
k.logMargPostFun(theta_alpha, grad = FALSE)
```

Arguments

| Argument | Description |
|-------------|--|
| theta_alpha | Numeric vector of correlation range and variance over nugget + variance parameters at which the function is to be evaluated. |
| grad | Logical. Should the function return the gradient (w.r.t theta_alpha)? |

Details

The log-marginal posterior density relates to the [jointly robust prior](#) $\pi_{JR}(\theta, \alpha, \sigma^2, \beta) \propto \pi(\theta, \alpha) \sigma^{-2}$. The marginal (or integrated) posterior is the function θ and α obtained by marginalizing out the GP variance σ^2 and the vector β of trend coefficients. Due to the form of the prior, the marginalization can be done on the likelihood $p_{\text{marg}}(\theta, \alpha | \mathbf{y}) \propto \pi(\theta, \alpha) \times L_{\text{marg}}(\theta, \alpha; \mathbf{y})$.

Value

The value of the log-marginal posterior density $\log p_{\text{marg}}(\theta, \alpha | \mathbf{y})$ where θ is the vector of correlation ranges and $\alpha = \sigma^2/(\sigma^2 + \tau^2)$ is the ratio of variances GP/(GP + nugget). By maximizing this function we should get the estimates of θ and α obtained when using `objective = "LMP"` in the [fit.NuggetKriging](#) method.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))

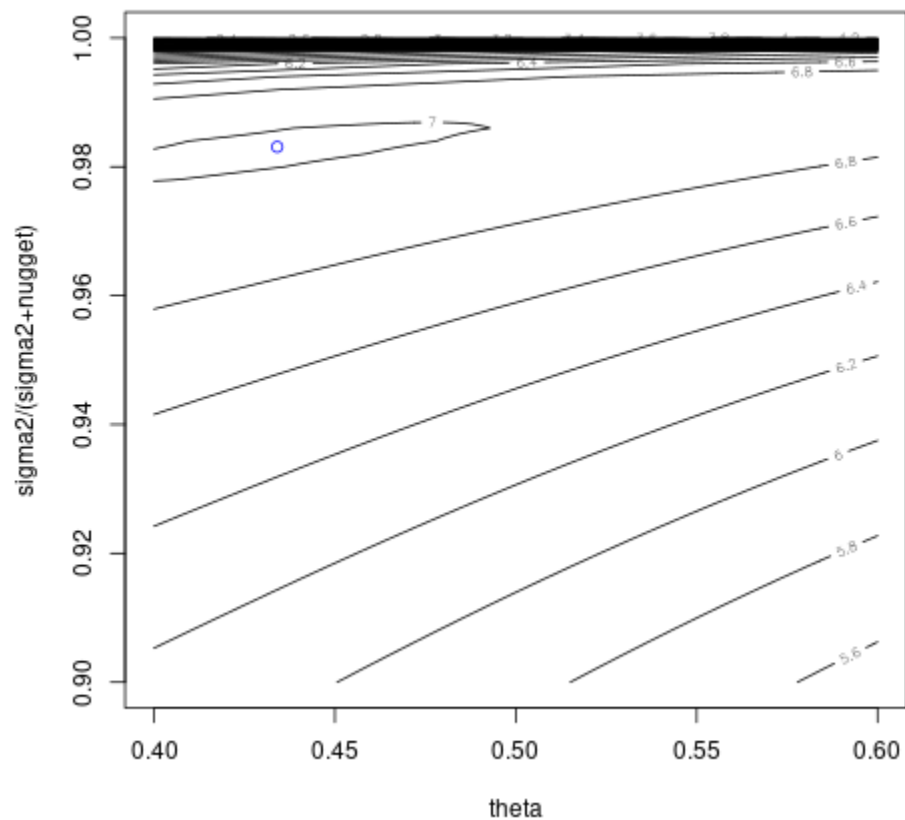
k <- NuggetKriging(y, X, "matern3_2", objective="LMP")
print(k)

# theta0 = k$theta()
# lmp_alpha <- function(alpha) k$logMargPostFun(cbind(theta0,alpha))$logMargPost
# a <- seq(from = 0.9, to = 1.0, length.out = 101)
# plot(a, Vectorize(lmp_alpha)(a), type = "l", xlim=c(0.9,1))
# abline(v = k$sigma2()/(k$sigma2()+k$nugget()), col = "blue")
#
# alpha0 = k$sigma2()/(k$sigma2()+k$nugget())
# lmp_theta <- function(theta) k$logMargPostFun(cbind(theta,alpha0))$logMargPost
# t <- seq(from = 0.001, to = 2, length.out = 101)
# plot(t, Vectorize(lmp_theta)(t), type = 'l')
# abline(v = k$theta(), col = "blue")

lmp <- function(theta_alpha) k$logMargPostFun(theta_alpha)$logMargPost
t <- seq(from = 0.4, to = 0.6, length.out = 51)
a <- seq(from = 0.9, to = 1, length.out = 51)
contour(t,a,matrix(ncol=length(t),lmp(expand.grid(t,a))),nlevels=50,xlab="theta",ylab=
  ↪ "sigma2/(sigma2+nugget)")
points(k$theta(),k$sigma2()/(k$sigma2()+k$nugget()),col='blue')
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.149491,0.940566]
* trend constant (est.): 0.389559
* variance (est.): 0.192207
* covariance:
  * kernel: matern3_2
  * range (est.): 0.434061
  * nugget (est.): 0.00330572
* fit:
  * objective: LMP
  * optim: BFGS
```



NoiseKriging::fit

Description

Fit a NoiseKriging Model Object with given Observations

Usage

- Python

```
# k = NoiseKriging(kernel=...)
k.fit(y, noise, X,
      regmodel = "constant",
      normalize = False,
      optim = "BFGS",
      objective = "LL",
      parameters = None)
```

- R

```
# k = NoiseKriging(kernel=...)
k$fit(y, noise, X,
      regmodel = "constant",
      normalize = FALSE,
      optim = "BFGS",
      objective = "LL",
      parameters = NULL)
```

- Matlab/Octave

```
% k = NoiseKriging(kernel=...)
k.fit(y, noise, X,
      regmodel = "constant",
      normalize = false,
      optim = "BFGS",
      objective = "LL",
      parameters = [])
```

Arguments

| Argument | Description |
|-------------------------|---|
| <code>y</code> | Numeric vector of response values. |
| <code>noise</code> | Numeric vector of response variances. |
| <code>X</code> | Numeric matrix of input design. |
| <code>regmodel</code> | Universal NoiseKriging linear trend. |
| <code>normalize</code> | Logical. If TRUE both the input matrix <code>X</code> and the response <code>y</code> in normalized to take values in the interval $[0, 1]$. |
| <code>optim</code> | Character giving the Optimization method used to fit hyper-parameters. Possible values are: "BFGS" and "none", the later simply keeping the values given in <code>parameters</code> . The method "BFGS" uses the gradient of the objective. |
| <code>objective</code> | Character giving the objective function to optimize. Possible values are: "LL" for the Log-Likelihood. |
| <code>parameters</code> | Numerical values for the hyper-parameters. When provided this must be named list with elements "sigma2" and "theta" containing the initial value(s) for the variance and for the range parameters. If <code>theta</code> is a matrix with more than one row, each row is used as a starting point for optimization. |
| <code>kernel</code> | Character defining the covariance model: "exp", "gauss", "matern3_2", "matern5_2". |

Details

The hyper-parameters (variance and vector of correlation ranges) are estimated thanks to the optimization of a criterion given by `objective`, using the method given in `optim`. For now only the maximum-likelihood estimation is allowed. See [this section](#) for more details on the maximum-likelihood estimation.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X)) # add noise dep. on X

k <- NoiseKriging("matern3_2")
print("before fit")
print(k)

k$fit(y,noise=(X/10)^2,X)
print("after fit")
print(k)
```

Results

```
[1] "before fit"
* covariance:
  * kernel: matern3_2
[1] "after fit"
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
  0.00278895, 0.00796412, 0.00304081, 0.00208497
* fit:
  * objective: LL
  * optim: BFGS
```

functions/examples/fit.NoiseKriging.md.png

NoiseKriging::logLikelihood

Description

Get the Maximized Log-Likelihood of a NoiseKriging Model Object

Usage

- Python

```
# k = NoiseKriging(...)
k.logLikelihood()
```

- R

```
# k = NoiseKriging(...)
k$logLikelihood()
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k.logLikelihood()
```

Details

See `logLikelihoodFun.NoiseKriging` for more details on the corresponding profile log-likelihood function.

Value

The value of the maximized profile log-likelihood $\ell_{\text{prof}}(\hat{\theta}, \hat{\sigma}^2)$. This is also the maximized value $\ell(\hat{\theta}, \hat{\sigma}^2, \hat{\beta})$ of the log-likelihood.

Examples

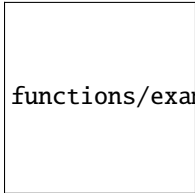
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X))

k <- NoiseKriging(y, (X/10)^2, X, kernel = "matern3_2", objective="LL")
print(k)

k$logLikelihood()
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
  00278895, 0.00796412, 0.00304081, 0.00208497
* fit:
  * objective: LL
  * optim: BFGS
[1] 5.200129
```



functions/examples/logLikelihood.NoiseKriging.md.png

Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NoiseKriging.cpp#L94>

NoiseKriging::logLikelihoodFun

Description

Compute the Profile Log-Likelihood of a NoiseKriging Model for a given Vector θ of Correlation Ranges and a given GP Variance σ^2

Usage

- Python

```
# k = NoiseKriging(...)
k.logLikelihoodFun(theta_sigma2, grad)
```

- R

```
# k = NoiseKriging(...)
k$logLikelihoodFun(theta_sigma2, grad)
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k.logLikelihoodFun(theta_sigma2, grad)
```

Arguments

| Argument | Description |
|--------------|---|
| theta_sigma2 | A numeric vector of (positive) range parameters and variance at which the log-likelihood will be evaluated. |
| grad | Logical. Should the function return the gradient? |

Details

The profile log-likelihood is obtained from the log-likelihood function $\ell(\theta, \sigma^2, \beta)$ by replacing the vector β of trend coefficients by its ML estimate $\hat{\beta}$ which is obtained by Generalized Least Squares. See [here](#) for more details.

Value

The value of the profile log-likelihood $\ell_{\text{prof}}(\theta, \sigma^2)$ for the given vector θ of correlation ranges and the given GP variance σ^2 .

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X))

k <- NoiseKriging(y, (X/10)^2, X, kernel = "matern3_2")
print(k)

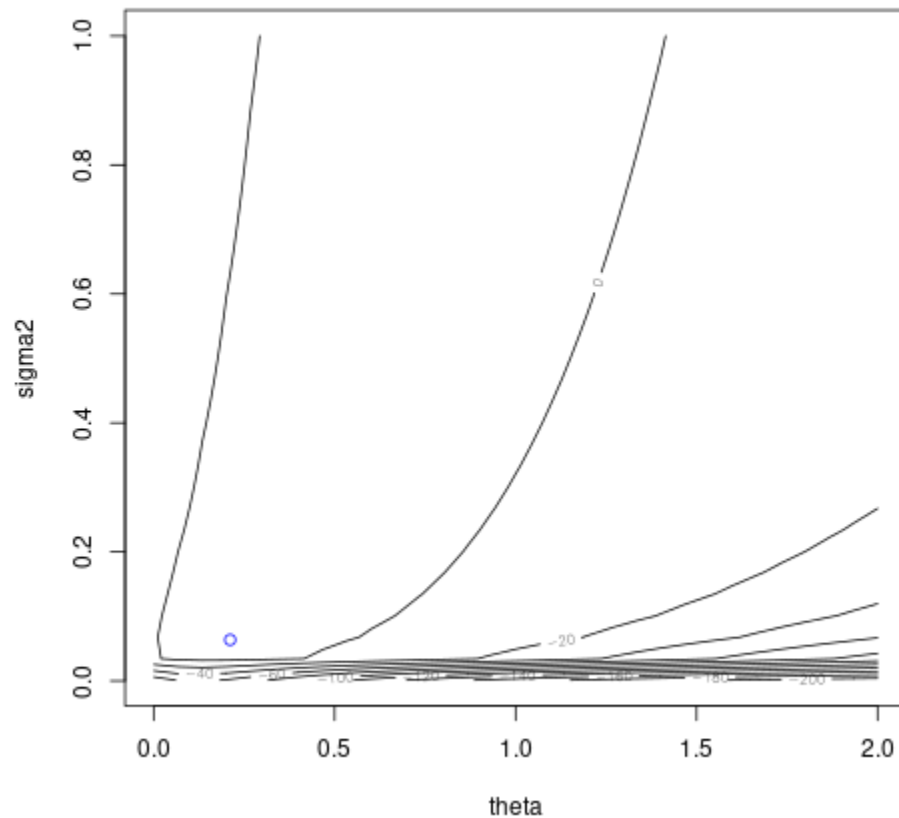
# theta0 = k$theta()
# ll_sigma2 <- function(sigma2) k$logLikelihoodFun(cbind(theta0,sigma2))$logLikelihood
# s2 <- seq(from = 0.001, to = 1, length.out = 101)
# plot(s2, Vectorize(ll_sigma2)(s2), type = 'l')
# abline(v = k$sigma2(), col = "blue")

# sigma20 = k$sigma2()
# ll_theta <- function(theta) k$logLikelihoodFun(cbind(theta,sigma20))$logLikelihood
# t <- seq(from = 0.001, to = 2, length.out = 101)
# plot(t, Vectorize(ll_theta)(t), type = 'l')
# abline(v = k$theta(), col = "blue")

ll <- function(theta_sigma2) k$logLikelihoodFun(theta_sigma2)$logLikelihood
s2 <- seq(from = 0.001, to = 1, length.out = 31)
t <- seq(from = 0.001, to = 2, length.out = 31)
contour(t,s2,matrix(ncol=length(s2),ll(expand.grid(t,s2))),xlab="theta",ylab="sigma2")
points(k$theta(),k$sigma2(),col='blue')
```

Results

```
* data: 10x[0.0455565,0.940467] -> 10x[0.152144,0.957381]
* trend constant (est.): 0.487335
* variance (est.): 0.0635381
* covariance:
  * kernel: matern3_2
  * range (est.): 0.211413
  * noise: 0.000827008, 0.00621425, 0.00167262, 0.0077972, 0.00884479, 2.07539e-05, 0.
  00278895, 0.00796412, 0.00304081, 0.00208497
* fit:
  * objective: LL
  * optim: BFGS
```



1.3.3 Prediction and simulation

Kriging::predict

Description

Predict from a Kriging Model Object

Usage

- Python

```
# k = Kriging(...)
k.predict(x, stdev = True, cov = False, deriv = False)
```

- R

```
# k = Kriging(...)
k$predict(x, stdev = TRUE, cov = FALSE, deriv = FALSE)
```

- Matlab/Octave

```
% k = Kriging(...)
k.predict(x, stdev = true, cov = false, deriv = false)
```

Arguments

| Argument | Description |
|--------------|---|
| x | Input points where the prediction must be computed. |
| stdev | Logical . If TRUE the standard deviation is returned. |
| cov | Logical . If TRUE the covariance matrix of the predictions is returned. |
| deriv | Logical . If TRUE the derivatives of mean and sd of the predictions are returned. |

Details

Given n^* “new” input points \mathbf{x}_j^* , the method compute the expectation, the standard deviation and (optionally) the covariance of the “new” observations $y(\mathbf{x}_j^*)$ of the stochastic process, conditional on the n values $y(\mathbf{x}_i)$ at the input points \mathbf{x}_i as used when fitting the model. The n^* input vectors (with length d) are given as the rows of a \mathbf{X}^* corresponding to \mathbf{x} .

The computation of these quantities is often called *Universal Kriging* see [here](#) for more details.

Value

A list containing the element mean and possibly stdev and cov.

- The expectation in `mean` is the estimate of the vector $E[\mathbf{y}^* | \mathbf{y}]$ with length n^* where \mathbf{y}^* and \mathbf{y} are the random vectors corresponding to the observation and the “new” input points. Similarly the conditional standard deviation in `stdev` is a vector with length n^* and the conditional covariance in `cov` is a $n^* \times n^*$ matrix.
- The (optional) derivatives are two $n^* \times d$ matrices `pred_mean_deriv` and `pred_sdtdev_deriv` with their row j containing the vector of derivatives w.r.t. to the new input point \mathbf{x}^* evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$. So the row j of `pred_mean_deriv` contains the derivative $\partial_{\mathbf{x}^*} E[y(\mathbf{x}^*) | \mathbf{y}]$. evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$.

Note that for a Kriging object the prediction is actually an interpolation.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)
points(X, y, col = "blue", pch = 16)

k <- Kriging(y, X, "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
p <- k$predict(x)

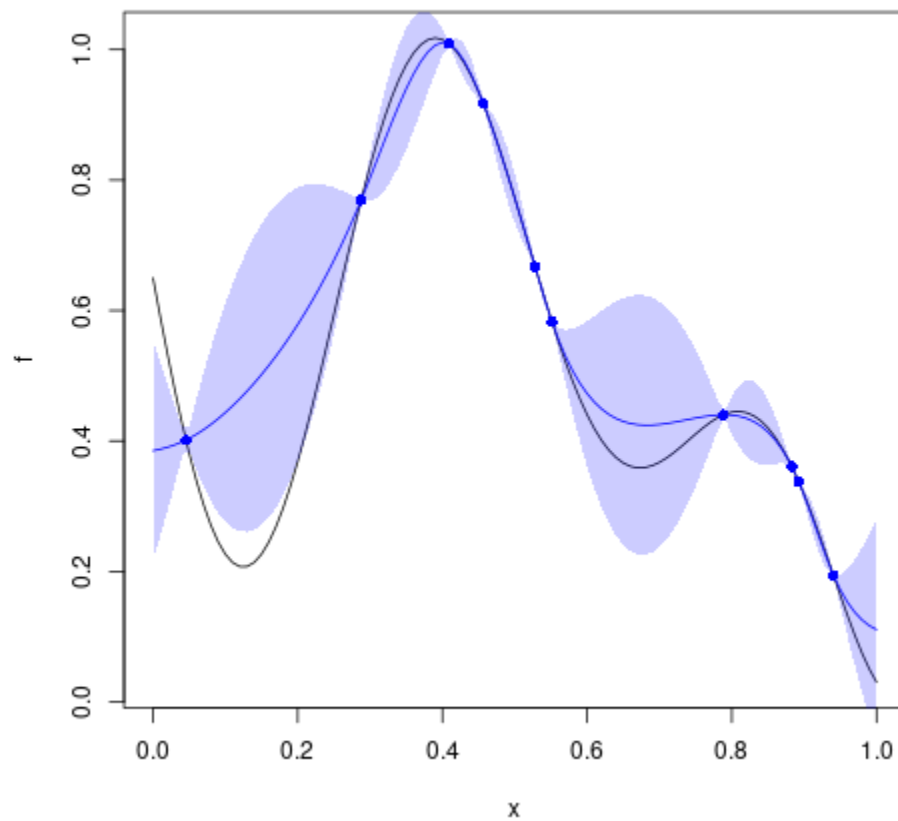
lines(x, p$mean, col = "blue")
```

(continues on next page)

(continued from previous page)

```
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)), border = NA, col = rgb(0, 0, 1, 0.2))
```

Results



Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/Kriging.cpp#L1326>

Kriging::simulate

Description

Simulate from a Kriging Model Object.

Usage

- Python

```
# k = Kriging(...)
k.predict(nsim = 1, seed = 123, x)
```

- R

```
# k = Kriging(...)
k$predict(nsim = 1, seed = 123, x)
```

- Matlab/Octave

```
% k = Kriging(...)
k.predict(nsim = 1, seed = 123, x)
```

Arguments

| Argument | Description |
|----------|--|
| nsim | Number of simulations to perform. |
| seed | Random seed used. |
| x | Points in model input space where to simulate. |

Details

This method draws n_{sim} paths of the stochastic process $y(\mathbf{x})$ at the n^* given new input points \mathbf{x}_j^* conditional on the values $y(\mathbf{x}_i)$ at the input points used in the fit.

Value

A matrix with `length(x)` rows and `nsim` columns containing the simulated paths at the inputs points given in `x`.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X)
points(X, y, col = "blue")

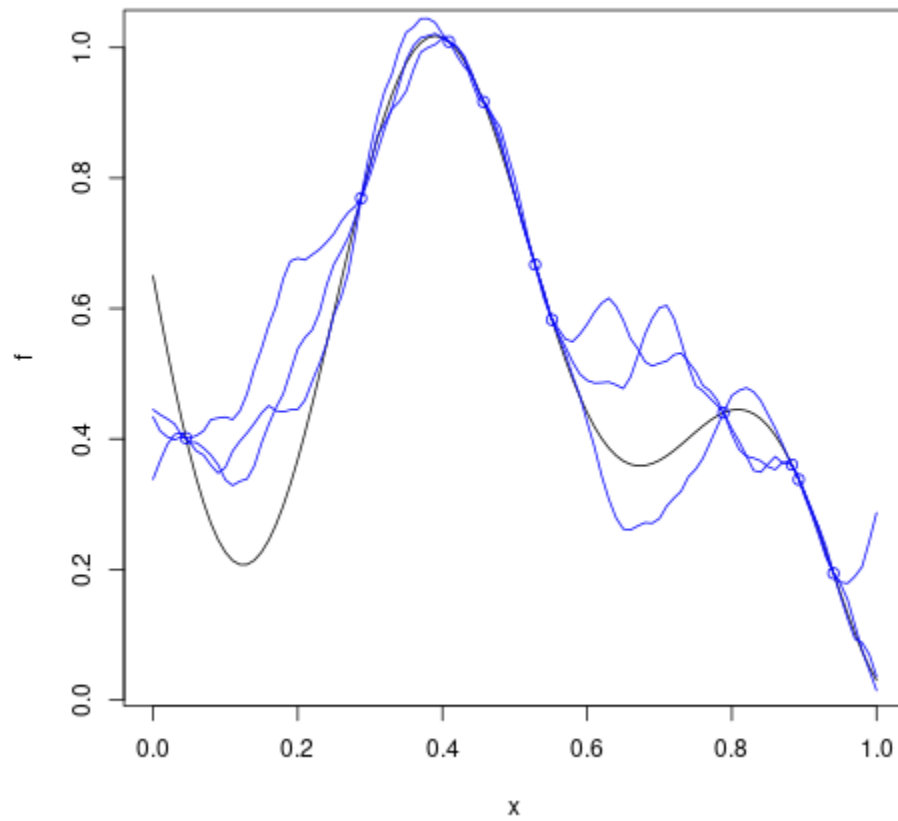
k <- Kriging(y, X, kernel = "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
s <- k$simulate(nsim = 3, x = x)

lines(x, s[, 1], col = "blue")
lines(x, s[, 2], col = "blue")
lines(x, s[, 3], col = "blue")
```

Results





Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/Kriging.cpp#L1501>

NuggetKriging::predict

Description

Predict from a NuggetKriging Model Object

Usage

- Python

```
# k = NuggetKriging(...)
k.predict(x, stdev = True, cov = False, deriv = False)
```

- R

```
# k = NuggetKriging(...)
k$predict(x, stdev = TRUE, cov = FALSE, deriv = FALSE)
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.predict(x, stdev = true, cov = false, deriv = false)
```

Arguments

| Argument | Description |
|--------------|---|
| x | Input points where the prediction must be computed. |
| stdev | Logical . If TRUE the standard deviation is returned. |
| cov | Logical . If TRUE the covariance matrix of the predictions is returned. |
| deriv | Logical . If TRUE the derivatives of mean and sd of the predictions are returned. |

Details

Given n^* “new” input points \mathbf{x}_j^* , the method compute the expectation, the standard deviation and (optionally) the covariance of the “new” observations $y(\mathbf{x}_j^*)$ of the stochastic process, conditional on the n values $y(\mathbf{x}_i)$ at the input points \mathbf{x}_i as used when fitting the model. The n^* input vectors (with length d) are given as the rows of a \mathbf{X}^* corresponding to \mathbf{x} .

The computation of these quantities is often called *Universal Kriging* see [here](#) for more details.

Value

A list containing the element **mean** and possibly **stdev** and **cov**.

- The expectation in **mean** is the estimate of the vector $\mathbb{E}[\mathbf{y}^* | \mathbf{y}]$ with length n^* where \mathbf{y}^* and \mathbf{y} are the random vectors corresponding to the observation and the “new” input points. Similarly the conditional standard deviation in **stdev** is a vector with length n^* and the conditional covariance in **cov** is a $n^* \times n^*$ matrix.
- The (optional) derivatives are two $n^* \times d$ matrices **pred_mean_deriv** and **pred_sdtdev_deriv** with their row j containing the vector of derivatives w.r.t. to the new input point \mathbf{x}^* evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$. So the row j of **pred_mean_deriv** contains the derivative $\partial_{\mathbf{x}^*} \mathbb{E}[y(\mathbf{x}^*) | \mathbf{y}]$. evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$.

Note that for a **NuggetKriging** object if it happens that the new input \mathbf{x}^* is exactly equal to one of the inputs \mathbf{x}_i then the corresponding prediction will be equal to the corresponding observed output y_i . So the prediction is discontinuous at the observations.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))
points(X, y, col = "blue", pch = 16)

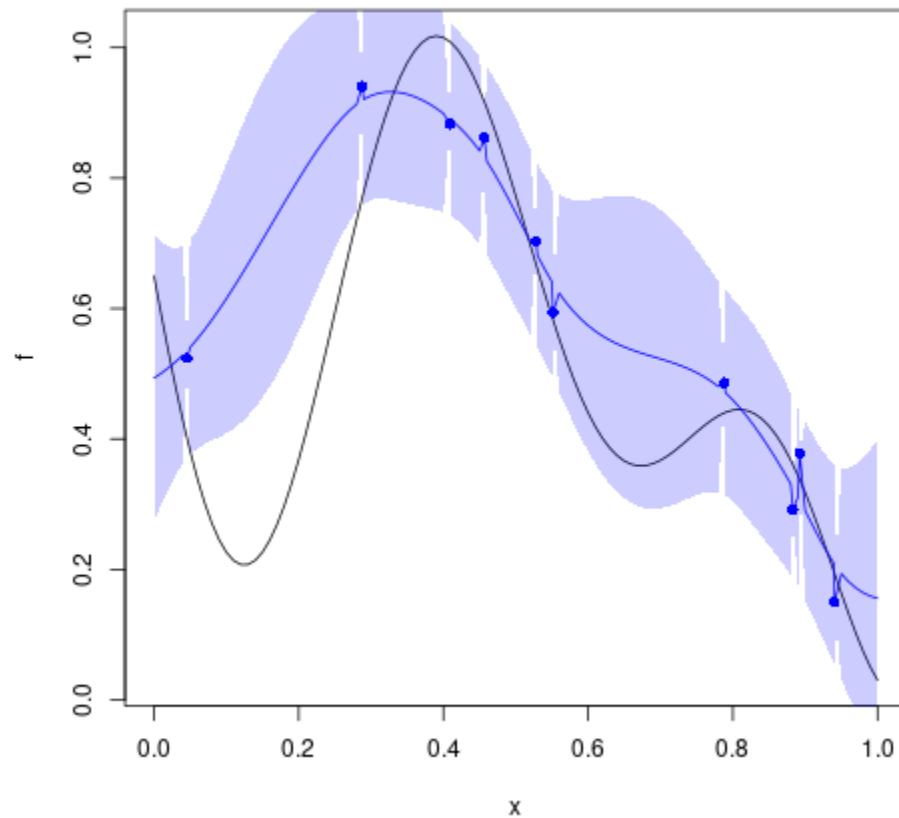
k <- NuggetKriging(y, X, "matern3_2")

x <- sort(c(X, seq(from = 0, to = 1, length.out = 101))) # include design points to see
↳ interpolation
p <- k$predict(x)

lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)), border = NA,
↳ col = rgb(0, 0, 1, 0.2))
```

Results





Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NuggetKriging.cpp#L1326>

NuggetKriging::simulate

Description

Simulation from a NuggetKriging model object.

Usage

- Python

```
# k = NuggetKriging(...)
k.predict(nsim = 1, seed = 123, x)
```

- R

```
# k = NuggetKriging(...)
k$predict(nsim = 1, seed = 123, x)
```

- Matlab/Octave

```
% k = NuggetKriging(...)
k.predict(nsim = 1, seed = 123, x)
```

Arguments

| Argument | Description |
|----------|--|
| nsim | Number of simulations to perform. |
| seed | Random seed used. |
| x | Points in model input space where to simulate. |

Details

This method draws paths of the stochastic process at new input points conditional on the values at the input points used in the fit.

Value

a matrix with `length(x)` rows and `nsim` columns containing the simulated paths at the inputs points given in `x`.

Examples

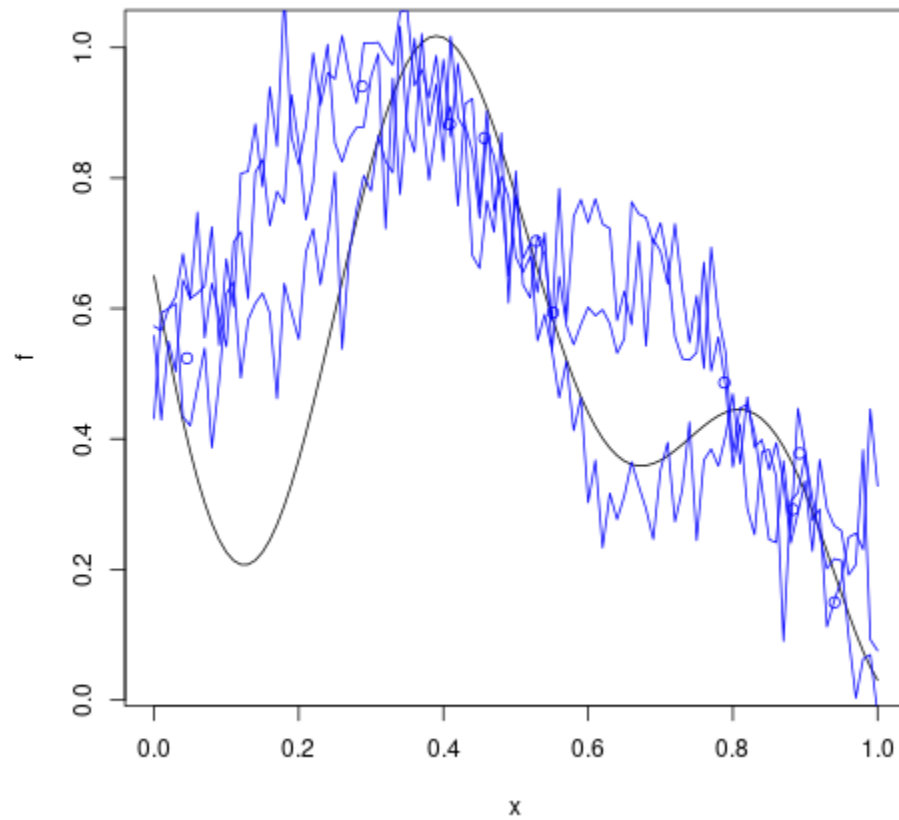
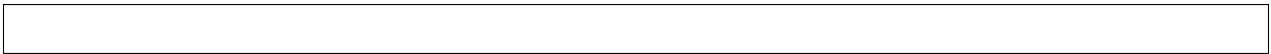
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + 0.1 * rnorm(nrow(X))
points(X, y, col = "blue")

k <- NuggetKriging(y, X, kernel = "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
s <- k$simulate(nsim = 3, x = x)

lines(x, s[, 1], col = "blue")
lines(x, s[, 2], col = "blue")
lines(x, s[, 3], col = "blue")
```


Results



Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NuggetKriging.cpp#L1501>

NoiseKriging::predict

Description

Predict from a NoiseKriging Model Object

Usage

- Python

```
# k = NoiseKriging(...)
k.predict(x, stdev = True, cov = False, deriv = False)
```

- R

```
# k = NoiseKriging(...)
k$predict(x, stdev = TRUE, cov = FALSE, deriv = FALSE)
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k.predict(x, stdev = true, cov = false, deriv = false)
```

Arguments

| Argument | Description |
|--------------|---|
| x | Input points where the prediction must be computed. |
| stdev | Logical . If TRUE the standard deviation is returned. |
| cov | Logical . If TRUE the covariance matrix of the predictions is returned. |
| deriv | Logical . If TRUE the derivatives of mean and sd of the predictions are returned. |

Details

Given n^* “new” input points \mathbf{x}_j^* , the method compute the expectation, the standard deviation and (optionally) the covariance of the estimated values of the “trend + GP” stochastic process $\mu(\mathbf{x}_j^*) + \zeta(\mathbf{x}_j^*)$ at the “new” observations. The estimation is based on the distribution conditional on the n noisy observations y_i made at the input points \mathbf{x}_i as used when fitting the model. The n^* input vectors (with length d) are given as the rows of a \mathbf{X}^* corresponding to \mathbf{x} .

The computation of these quantities is often called *Universal Kriging* see [here](#) for more details.

Value

A list containing the element **mean** and possibly **stdev** and **cov**.

- The expectation in **mean** is the estimate of the vector $\mathbf{E}[\boldsymbol{\mu}^* + \boldsymbol{\zeta}^* | \mathbf{y}]$ with length n^* where $\boldsymbol{\mu}^*$ and $\boldsymbol{\zeta}^*$ are for “new” points and \mathbf{y} corresponds to the observations. Similarly the conditional standard deviation in **stdev** is a vector with length n^* and the conditional covariance in **cov** is a $n^* \times n^*$ matrix.
- The (optional) derivatives are two $n^* \times d$ matrices **pred_mean_deriv** and **pred_sdtdev_deriv** with their row j containing the vector of derivatives w.r.t. to the new input point \mathbf{x}^* evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$. So the row j of **pred_mean_deriv** contains the derivative $\partial_{\mathbf{x}^*} \mathbb{E}[y(\mathbf{x}^*) | \mathbf{y}]$. evaluated at $\mathbf{x}^* = \mathbf{x}_j^*$.

Note that for a **NoiseKriging** object the prediction is actually a *smoothing*. The so-called *Kriging mean* function $\mathbf{x}^* \mapsto \mathbb{E}[y(\mathbf{x}^*) | \mathbf{y}]$ is a smooth function. Depending on the given noise variances σ_i^2 given in the *fit step*, the prediction at $\mathbf{x}^* \approx \mathbf{x}_i$ will be more or less close to the observed value y_i . As opposed to the *NuggetKriging* model case, duplicated inputs can be used in the design.

Examples

```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X))
points(X, y, col = "blue", pch = 16)

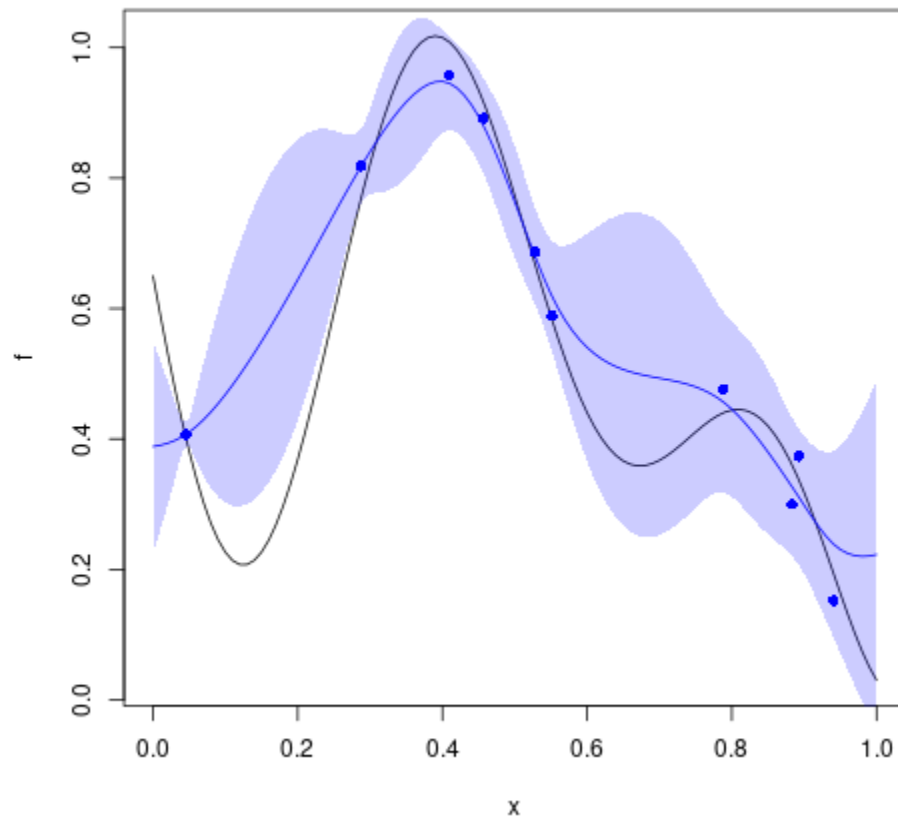
k <- NoiseKriging(y, (X/10)^2, X, "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
p <- k$predict(x)

lines(x, p$mean, col = "blue")
polygon(c(x, rev(x)), c(p$mean - 2 * p$stdev, rev(p$mean + 2 * p$stdev)),
border = NA, col = rgb(0, 0, 1, 0.2))
```

Results





Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NoiseKriging.cpp#L1326>

NoiseKriging::simulate

Description

Simulation from a NoiseKriging model object.

Usage

- Python

```
# k = NoiseKriging(...)
k.predict(nsim = 1, seed = 123, x)
```

- R

```
# k = NoiseKriging(...)
k$predict(nsim = 1, seed = 123, x)
```

- Matlab/Octave

```
% k = NoiseKriging(...)
k.predict(nsim = 1, seed = 123, x)
```

Arguments

| Argument | Description |
|----------|--|
| nsim | Number of simulations to perform. |
| seed | Random seed used. |
| x | Points in model input space where to simulate. |

Details

This method draws paths of the stochastic process at new input points conditional on the values at the input points used in the fit.

Value

a matrix with `length(x)` rows and `nsim` columns containing the simulated paths at the inputs points given in `x`.

Examples

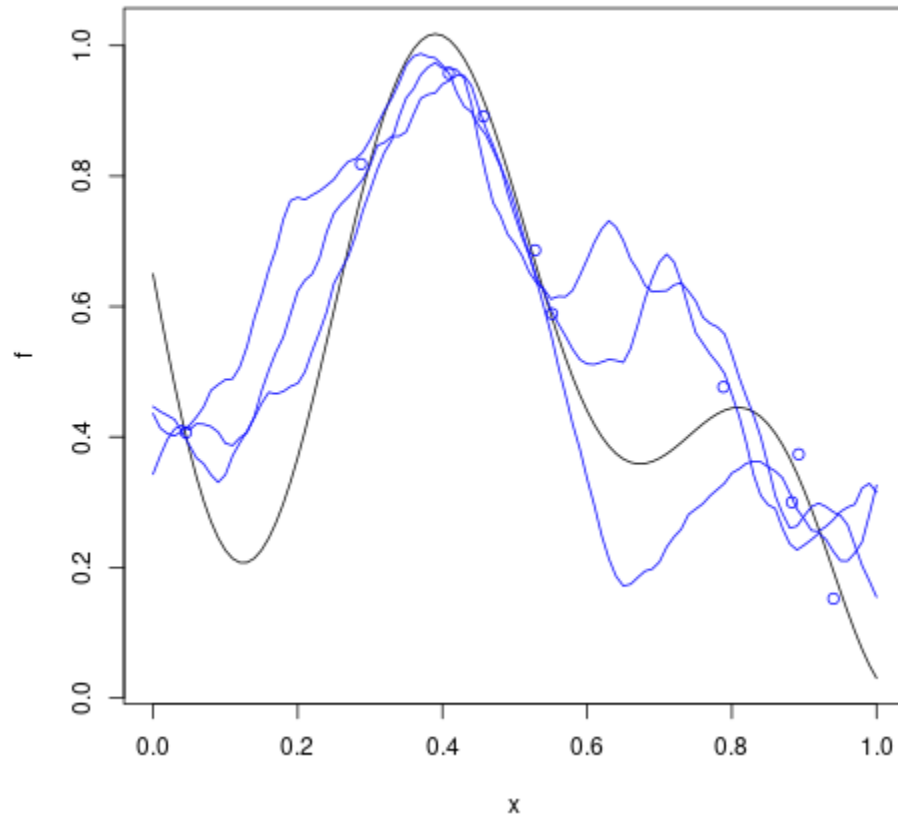
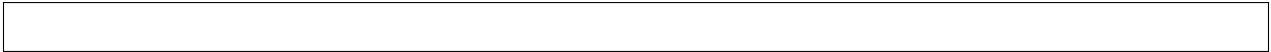
```
f <- function(x) 1 - 1 / 2 * (sin(12 * x) / (1 + x) + 2 * cos(7 * x) * x^5 + 0.7)
plot(f)
set.seed(123)
X <- as.matrix(runif(10))
y <- f(X) + X/10 * rnorm(nrow(X))
points(X, y, col = "blue")

k <- NoiseKriging(y, (X/10)^2, X, kernel = "matern3_2")

x <- seq(from = 0, to = 1, length.out = 101)
s <- k$simulate(nsim = 3, x = x)

lines(x, s[, 1], col = "blue")
lines(x, s[, 2], col = "blue")
lines(x, s[, 3], col = "blue")
```

Results



Reference

- Code: <https://github.com/libKriging/libKriging/blob/master/src/lib/NoiseKriging.cpp#L1501>

1.4 Models description

1.4.1 Kriging models

Components of Kriging models

libKriging makes available several kinds of Kriging models as commonly used in the field of computer experiments. All models involve a stochastic process $y(\mathbf{x})$ indexed by a vector $\mathbf{x} \in \mathbb{R}^d$ of d real inputs x_k , sometimes called the design vector. The response variable or output y is assumed to be observed for n values \mathbf{x}_i of the input vector with corresponding response values y_i for $i = 1, \dots, n$. The response values are considered as realizations of random variables.

The models involve the following elements or components.

- **Trend** A known vector-valued function $\mathbb{R}^d \rightarrow \mathbb{R}^p$ with value denoted by $\mathbf{f}(\mathbf{x})$. It is used in relation with an unknown vector $\boldsymbol{\beta}$ of trend parameters to provide the trend term $\mu(\mathbf{x}) = \mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta}$.
- **Smooth Gaussian Process (GP)** An unobserved GP $\zeta(\mathbf{x})$, at least continuous, with mean zero and known covariance kernel $C_\zeta(\mathbf{x}, \mathbf{x}')$.
- **Nugget** A White noise GP $\varepsilon(\mathbf{x})$ with variance τ^2 hence with covariance kernel $\tau^2 \delta(\mathbf{x}, \mathbf{x}')$ where δ is the Dirac function $\delta(\mathbf{x}, \mathbf{x}') := 1_{\{\mathbf{x}=\mathbf{x}'\}}$.
- **Noise** A collection of independent random variables ε_i with variances τ_i^2 .

Note that the words *nugget* and *noise* are sometimes considered as equivalent. Yet in **libKriging** *nugget* will be used only when a single path is considered for the stochastic process, in which case no duplicated value can exist for the vector of inputs.

When a nugget term is used, the process $y(\mathbf{x})$ is discontinuous, so the prediction at a new value \mathbf{x}^* will be identical to $y(\mathbf{x}_i)$ if it happens that $\mathbf{x}^* = \mathbf{x}_i$ for some i . We may say that the prediction is an interpolation, in relation with this feature. However, in the usual acceptance of this term, interpolation involves the use of a *smooth* function, say at least continuous.

Note The so-called *Gaussian-Process Regression* framework corresponds to the noisy case. However duplicated designs are generally allowed and the noise r.v.s are assumed to have either a common unknown variance τ^2 or a variance $\tau^2(\mathbf{x})$ depending on the design according to some specification.

libKriging implements the three classes "Kriging", "NoiseKriging" and "NuggetKriging" of objects corresponding to Kriging models. In each class we find the linear trend, the smooth GP. The difference relates to the presence of a nugget or noise term.

Classes of Kriging model objects

To describe the three classes of Kriging models, we assume that n observations are given corresponding to n input vectors \mathbf{x}_i .

- The **Kriging** class correspond to observations of the form

$$y(\mathbf{x}_i) = \underbrace{\mathbf{f}(\mathbf{x}_i)^\top \boldsymbol{\beta}}_{\text{trend}} + \underbrace{\zeta(\mathbf{x}_i)}_{\text{smooth GP}}, \quad i = 1, \dots, n.$$

- The **"NuggetKriging"** class corresponds to observations of the form

$$y(\mathbf{x}_i) = \underbrace{\mathbf{f}(\mathbf{x}_i)^\top \boldsymbol{\beta}}_{\text{trend}} + \underbrace{\zeta(\mathbf{x}_i)}_{\text{smooth GP}} + \underbrace{\varepsilon(\mathbf{x}_i)}_{\text{nugget}}, \quad i = 1, \dots, n.$$

The sum $\eta(\mathbf{x}) := \zeta(\mathbf{x}) + \varepsilon(\mathbf{x})$ defines a GP with discontinuous paths and covariance kernel $C(\mathbf{x}, \mathbf{x}') + \tau^2 \delta(\mathbf{x}, \mathbf{x}')$.

- The **"NoiseKriging"** class corresponds to observations of the form

$$y_i = \underbrace{\mathbf{f}(\mathbf{x}_i)^\top \boldsymbol{\beta}}_{\text{trend}} + \underbrace{\zeta(\mathbf{x}_i)}_{\text{smooth GP}} + \underbrace{\varepsilon_i}_{\text{noise}}, \quad i = 1, \dots, n$$

where the noise r.v.s ε_i are Gaussian with mean zero and known variances τ_i^2 . Although the response y_i corresponds to the input \mathbf{x}_i as for the classes "Kriging" and "NuggetKriging", there can be several observations made at the same input \mathbf{x}_i . We may then speak of *duplicated* inputs.

Matrix formalism and assumptions

The n input vectors \mathbf{x}_i are conveniently considered as the (transposed) rows of a matrix.

- The $n \times d$ design or input matrix \mathbf{X} having \mathbf{x}_i^\top as its row i .
- The $n \times p$ trend matrix $\mathbf{F}(\mathbf{X})$ or simply \mathbf{F} having $\mathbf{f}(\mathbf{x}_i)^\top$ as its row i .
- The $n \times n$ covariance matrix $\mathbf{C}(\mathbf{X}, \mathbf{X}) = [C(\mathbf{x}_i, \mathbf{x}_j)]_{i,j}$ is sometimes called the Gram matrix and is often simply denoted as \mathbf{C} .

The observations for a Kriging model write in matrix notations $\mathbf{y} = \mathbf{F}\boldsymbol{\beta} + \boldsymbol{\zeta}$, while those for `NuggetKriging` and `NoiseKriging` models write as $\mathbf{y} = \mathbf{F}\boldsymbol{\beta} + \boldsymbol{\zeta} + \boldsymbol{\varepsilon}$. Similar notations are used if a sequence of n^* “new” designs \mathbf{x}_i^* are considered, resulting in matrices with n^* rows \mathbf{X}^* and \mathbf{F}^* .

It must be kept in mind that unless explicitly stated otherwise, the covariance matrix \mathbf{C} is *that of the non-trend component* $\boldsymbol{\eta}$ including the smooth GP plus the nugget or noise. It will be assumed that the matrix \mathbf{F} has rank p (hence that $n \geq p$) and that the matrix \mathbf{C} is positive definite. Inasmuch a positive kernel $C_\zeta(\mathbf{x}, \mathbf{x}')$ is used the matrix $\mathbf{C}_\zeta(\mathbf{X}, \mathbf{X})$ is positive definite for every design \mathbf{X} corresponding to distinct inputs \mathbf{x}_i .

Note Berliner and Thomas-Agnan [BTA04] define Kriging models as the sum of a deterministic trend and a stochastic process with stationary increments, as is the case for splines. So the name *Kriging model* is understood here in a more restrictive way.

See the [Prediction and simulation](#) page.

1.4.2 Kriging steps

Kriging models can be used in different steps depending on the goal.

- **Trend estimation** If only the trend parameters β_k are unknown, these can be estimated by [Generalized Least Squares](#). This step separates the observed response y_i into a trend and component $\hat{\mu}(\mathbf{x}_i)$ a non-trend component. The non-trend component involves a smooth GP component $\hat{\zeta}(\mathbf{x}_i)$ and, optionally, a nugget or noise component $\hat{\varepsilon}(\mathbf{x}_i)$ or $\hat{\varepsilon}_i$.
- **Fit** Find estimates of the parameters, including the covariance parameters. Several methods are implemented, all relying on the *optimization* of a function of the [covariance parameters](#) called the *objective*. This objective can relates to frequentist estimation methods: [Maximum-Likelihood](#) (ML) and [Leave-One-Out](#) Cross-Validation. It can also be a Bayesian [Marginal Posterior Density](#), in relation with specific priors, in which case the estimate will be a Maximum A Posteriori (MAP). Mind that in **libKriging** only *point estimates* will be given for the correlation parameters.
- **Update** Update a model object by processing n^* new observations. Once this step is achieved, the predictions will be based on the full set of $n + n^*$ observations. The covariance parameters can optionally be updated by using the new observations when computing the fitting objective.
- **Predict** Given n^* “new” inputs \mathbf{x}_i^* forming the rows of a matrix \mathbf{X}^* , compute the Gaussian distribution of \mathbf{y}^* conditional on \mathbf{y} . As long as the covariance parameters are regarded as known, the conditional distribution is Gaussian, and is characterized by its expectation vector and its covariance matrix. These are often called the *Kriging mean* and the *Kriging covariance*.
- **Simulate** Given n^* “new” inputs \mathbf{x}_i^* forming the rows of a matrix \mathbf{X}^* , draw a sample of n_{sim} vectors $\mathbf{y}^* k = 1, \dots, n_{\text{sim}}$ from the distribution of $y(\mathbf{x})$ conditional on the observations.

By “Kriging” one often means the prediction step. The fit step is generally the most costly one in terms of computation because the fit objective has to be evaluated repeatedly (say dozens of times) and each evaluation involves $O(n^3)$ elementary operations.

1.4.3 Trend functions in Kriging models

The possible trend functions in **libKriging** are as follow, by increasing level of complexity.

- The **constant trend** involves $p = 1$ coefficient and $\mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta} = \beta$.
- The **linear trend** involves $p = d + 1$ coefficients

$$\mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta} = \beta_0 + \sum_{i=1}^d \beta_i x_i.$$

- The **interactive trend** involves $1 + d + d(d - 1)/2$ coefficients

$$\mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta} = \beta_0 + \sum_{i=1}^d \sum_{j=1}^{i-1} \beta_{ji} x_j x_i.$$

- The **quadratic trend** involves $p = 1 + d + d(d + 1)/2$ coefficients

$$\mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta} = \beta_0 + \sum_{i=1}^d \sum_{j=1}^i \beta_{ji} x_j x_i.$$

Starting from the constant trend, the other forms come by adding the d linear terms x_i , adding the $d \times (d - 1)/2$ interaction terms $x_i x_j$ with $j < i$, and finally adding the squared input terms x_i^2 .

For instance with $d = 3$ inputs the four possible trends are in order of complexity

| | |
|-------------|--|
| constant | $\mathbf{f}(\mathbf{x})^\top = [1]$ |
| linear | $\mathbf{f}(\mathbf{x})^\top = [1, x_1, x_2, x_3]$ |
| interaction | $\mathbf{f}(\mathbf{x})^\top = [1, x_1, x_2, x_1 x_2, x_3, x_1 x_3, x_2 x_3]$ |
| quadratic | $\mathbf{f}(\mathbf{x})^\top = [1, x_1, x_1^2, x_2, x_1 x_2, x_2^2, x_3, x_1 x_3, x_2 x_3, x_3^2]$ |

Mind that the coefficients relate to a specific order of the inputs.

Note The number of coefficients required in the interactive and quadratic trend increases quadratically with the dimension. For $d = 10$ the quadratic trend involves 66 coefficients.

1.4.4 The tensor product kernel

General form

The zero-mean smooth GP $\zeta(\mathbf{x})$ is characterized by its covariance kernel $C_\zeta(\mathbf{x}, \mathbf{x}') := \mathbb{E}[\zeta(\mathbf{x}), \zeta(\mathbf{x}')]$. **libKriging** uses a specific form of covariance kernel $C_\zeta(\mathbf{x}, \mathbf{x}')$ on the input space \mathbb{R}^d which can be called *tensor-product*. With $\mathbf{h} := \mathbf{x} - \mathbf{x}'$ the kernel value expresses as

$$C_\zeta(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}, \sigma^2) = C_\zeta(\mathbf{h}; \boldsymbol{\theta}, \sigma^2) = \sigma^2 \prod_{\ell=1}^d \kappa(h_\ell / \theta_\ell)$$

where $\kappa(h)$ is a stationary correlation kernel on \mathbb{R} and $\boldsymbol{\theta}$ is a vector of d parameters $\theta_\ell > 0$ called *correlation ranges*. See Stein [Ste12] for a discussion on the tensor product kernel a.k.a. *separable* kernel.

A further constraint used in **libKriging** is that $\kappa(h)$ takes only positive values: $\gamma(h) > 0$ for all h . With $\lambda(h) := -\log \gamma(h)$ the derivative w.r.t. the correlation range θ_ℓ can be computed as

$$\partial_{\theta_\ell} C_\zeta(\mathbf{h}; \boldsymbol{\theta}) = \theta_\ell^{-2} \lambda'(h_\ell / \theta_\ell) C_\zeta(\mathbf{h}; \boldsymbol{\theta}).$$

Available 1D correlation kernels

The 1D correlation kernels available are listed in the Table below. Remind that in this setting the smoothness of the paths of the GP $\zeta(\mathbf{x})$ is controlled by the smoothness of the kernel $C_\zeta(\mathbf{h})$ at $\mathbf{h} = \mathbf{0}$ hence by the smoothness of the correlation kernel $\kappa(h)$ for $h = 0$. Note that the 1D exponential kernel is not differentiable at $h = 0$ and the corresponding paths are continuous but nowhere differentiable. The kernels are given in the table by order of increasing smoothness.

Note The Gaussian kernel is a radial kernel in the sense that it depends on \mathbf{h} only through its square norm $\sum_\ell h_\ell^2/\theta_\ell^2$.

| kernel | Name | Expression |
|-------------|------------------------|---|
| "exp" | Exponential | $\kappa(h) = \exp\{- h \}$ |
| "matern3_2" | Matérn whith shape 3/2 | $\kappa(h) = [1 + z] \exp\{-z\}, z := \sqrt{3} h $ |
| "matern5_2" | Matérn whith shape 5/2 | $\kappa(h) = [1 + z + z^2/3] \exp\{-z\}, z := \sqrt{5} h $ |
| "gauss" | Gaussian | $\kappa(h) = \exp\{-h^2/2\}$ |

1.4.5 Parameters

The parameters of the models are given in the Table below. Note that the trend parameters in β are of a somewhat different nature than the other ones. The parameters β_k can best be compared to the values $\zeta(\mathbf{x}_i)$ of the unobserved GP. Indeed if no nugget or noise is used, the estimation of β is the same thing as the estimation of ζ .

The trend parameters β_j never appear in the objective function used to fit the models, be it of frequentist or Bayesian nature.

| | Trend | GP Cov | Noise/Nug. | Optim |
|-----------------|---------|----------------------|--------------|--|
| "Kriging" | β | $[\theta, \sigma^2]$ | | θ |
| "NuggetKriging" | β | $[\theta, \sigma^2]$ | τ^2 | $[\theta, \alpha], \alpha := \sigma^2/(\sigma^2 + \tau^2)$ |
| "NoiseKriging" | β | $[\theta, \sigma^2]$ | $[\tau_i^2]$ | $[\theta, \sigma^2]$ |

Parameters used for the trend, the smooth GP and the noise or nugget parts. The column **Optim** is for the parameters used in the optimization. The other parameters are either known as $[\tau_i^2]$ or marginalized out, or replaced by their MLE β .

1.4.6 Functional point of view

For the models used with **libKriging**, both the trend functions and the covariance kernel have an impact. While a GP model for $\zeta(\mathbf{x})$ relates to a covariance kernel and to the corresponding Reproducing Kernel Hilbert Space (RKHS), a Kriging model as described in *Kriging models* relates to a *semi-RKHS* Berlinet and Thomas-Agnan [BTA04]. This space \mathcal{H} is a semi-Hilbert space of functions in which the trend functions f_k generate a finite-dimensional linear subspace \mathcal{F} called the *nullspace* which contains so-called *unpenalized* functions i.e., functions with (semi) norm zero.

When the covariance parameters are known, Kriging provides as the *Kriging mean* the function $h \in \mathcal{H}$ which minimises the Penalized Sum of Squares (PSS) criterion

$$\text{PSS} := \frac{1}{\tau^2} \sum_{i=1}^n \{y_i - h(\mathbf{x}_i)\}^2 + \|h\|_{\mathcal{H}}^2.$$

In the case where no nugget is used (corresponding to $\tau^2 \rightarrow 0$), the discrete sum in the PSS criterion is actually zero at the optimum so that h interpolates the data and has minimal norm $\|\cdot\|_{\mathcal{H}}$ amongst the functions $h \in \mathcal{H}$ that interpolates the data. We may regard Kriging as using a prior on a functional space, with an implied non-informative prior for

the trend part. At the right-hand side of the equation above, the first term can be regarded as $-2 \log L$ where L is the likelihood while the square norm can *formally* be regarded as $-2 \log \pi(h)$ where $\pi(h)$ is a prior density, although this is not tenable from a theoretical point of view. By minimizing PSS, we get the function h with maximum posterior density which is also the posterior mean.

The Kriging framework is similar to the splines framework, but as opposed to the later one, the trend functions are chosen quite arbitrarily and may also belong to the RKHS of the kernel. This will indeed be the case when $d = 1$ and a constant trend is used with one of the kernels available in **libKriging**: the constant trend function is therefore unpenalized, which makes the Kriging smoothing and the Kriging prediction behaves well w.r.t. a translation of the observations $\mathbf{y} \rightarrow \mathbf{y} + \text{Cst}$: the predicted values are then translated similarly. The function $h \in \mathcal{H}$ minimizing the criterion PSS above can be written in a non-unique way as

$$h(\mathbf{x}) = \sum_{i=1}^n \alpha_i C(\mathbf{x}_i, \mathbf{x}) + \sum_{k=1}^p \beta_k f_k(\mathbf{x}),$$

and Kriging indeeds find suitable vectors α and β . The representation of h can be made unique by imposing orthogonality constraints, see [The Bending Energy Matrix](#). See Wahba [Wah78] for the use of an improper prior on the coefficients of the trend functions.

Note Allowing for a non-informative trend has an important implication in terms of implementation since *Universal Kriging* equations must be used. By contrast, an informative trend can be coped with by using only *Simple Kriging* equations and sum of kernels. Indeed the informative trend corresponds to a kernel of the form $\mathbf{f}(\mathbf{x})^\top \mathbf{A} \mathbf{f}(\mathbf{x})$ where \mathbf{A} is a $p \times p$ positive definite matrix. The informative approach is most often retained in the Machine Learning community.

1.4.7 Trend estimation

Generalized Least Squares

Using n given observations y_i , we can estimate the trend at the inputs \mathbf{x}_i . For that aim we must find an estimate $\hat{\beta}$ of the unknown vector β . When no nugget or noise is used, the GP part comes as the difference $\hat{\zeta} = \mathbf{y} - \mathbf{F}\hat{\beta}$. When instead a nugget or a noise is present a further step is needed to separate the smooth GP part from the nugget or noise in $\mathbf{y} - \mathbf{F}\hat{\beta}$.

If the covariance parameters are known, the estimate $\hat{\beta}$ can be obtained by using General Least Squares (GLS); this estimate is also the Maximum Likelihood estimate. The computations related to GLS can rely on the Cholesky and the QR decompositions of matrices as now detailed.

The "Kriging" case

In the "Kriging" case, we have $\mathbf{C} = \sigma^2 \mathbf{R}$ where \mathbf{R} is the correlation matrix depending on θ . If the correlation matrix \mathbf{R} is known, then the ML estimate of β and its covariance are given by

$$\hat{\beta} = [\mathbf{F}^\top \mathbf{R}^{-1} \mathbf{F}]^{-1} \mathbf{F}^\top \mathbf{R}^{-1} \mathbf{y}, \quad \text{Cov}(\hat{\beta}) = \sigma^2 [\mathbf{F}^\top \mathbf{R}^{-1} \mathbf{F}]^{-1}.$$

Moreover the ML estimate $\hat{\sigma}^2$ is available as well.

In practice we can use the Cholesky decomposition $\mathbf{R} = \mathbf{L}\mathbf{L}^\top$ where \mathbf{L} is a $n \times n$ lower triangular matrix with positive diagonal elements. By left-multiplying the relation $\mathbf{y} = \mathbf{F}\beta + \zeta$ by \mathbf{L}^{-1} , we get

$$\mathbf{y}^\dagger = \mathbf{F}^\dagger \beta + \zeta^\dagger$$

where the “dagged” symbols indicate a left multiplication by \mathbf{L}^{-1} e.g., $\mathbf{y}^\dagger = \mathbf{L}^{-1} \mathbf{y}$. We get a standard linear regression with i.i.d. Gaussian errors ζ_i^\dagger having zero mean and variance σ^2 . So the ML estimates $\hat{\beta}$ and $\hat{\sigma}^2$ come by Ordinary

Least Squares. Using $\hat{\zeta} = \mathbf{y} - \mathbf{F}\hat{\beta}$ and $\zeta^\dagger := \mathbf{L}^{-1}\hat{\zeta}$ we have

$$\hat{\sigma}_{\text{ML}}^2 = \frac{1}{n} S^2, \quad \text{with} \quad S^2 := \hat{\zeta}^{\dagger\top} \hat{\zeta}^\dagger = \hat{\zeta}^\top \mathbf{R}^{-1} \hat{\zeta}.$$

Note that $\hat{\sigma}_{\text{ML}}^2$ is a biased estimate of σ^2 . An alternative unbiased estimate can be obtained by using $n - p$ instead of n as the denominator: this is the so-called *Restricted Maximum Likelihood* (REML) estimate.

The computations rely on the so-called “thin” or “economical” QR decomposition of the transformed trend matrix \mathbf{F}^\dagger

$$\mathbf{F}^\dagger = \mathbf{Q}_{\mathbf{F}^\dagger} \mathbf{R}_{\mathbf{F}^\dagger}$$

where $\mathbf{Q}_{\mathbf{F}^\dagger}$ is a $n \times p$ orthogonal matrix and $\mathbf{R}_{\mathbf{F}^\dagger}$ is a $p \times p$ upper triangular matrix. The orthogonality means that $\mathbf{Q}_{\mathbf{F}^\dagger}^\top \mathbf{Q}_{\mathbf{F}^\dagger} = \mathbf{I}_p$. The estimate $\hat{\beta}$ comes by solving the triangular system $\mathbf{R}_{\mathbf{F}^\dagger} \beta = \mathbf{Q}_{\mathbf{F}^\dagger}^\top \mathbf{y}^\dagger$, and the covariance of the estimate is $\text{Cov}(\hat{\beta}) = \mathbf{R}_{\mathbf{F}^\dagger}^{-1} \mathbf{R}_{\mathbf{F}^\dagger}^{-\top}$

Following a popular linear regression trick, one can further use the QR decomposition of the matrix \mathbf{F}_+^\dagger obtained by adding a new column \mathbf{y}^\dagger to \mathbf{F}^\dagger

$$\mathbf{F}_+^\dagger := [\mathbf{F}^\dagger \mid \mathbf{y}^\dagger] = \mathbf{Q}_{\mathbf{F}_+^\dagger} \mathbf{R}_{\mathbf{F}_+^\dagger}.$$

Then the $p + 1$ column of $\mathbf{Q}_{\mathbf{F}_+^\dagger}$ contains the vector of residuals $\hat{\zeta}^\dagger = \mathbf{y}^\dagger - \mathbf{F}^\dagger \hat{\beta}$ in its first p elements and the residual sum of squares is given by the square of the element $R_{\mathbf{F}_+^\dagger}[p + 1, p + 1]$. See Lange [Lan10].

"NuggetKriging" and "NoiseKriging"

When a nugget or noise term is used, the estimate of β can be obtained as above provided that the covariance matrix is that of the non-trend component hence includes the nugget or noise variance in its diagonal. In the *NuggetKriging* case the GLS will provide an estimate of the variance $\nu^2 = \sigma^2 + \tau^2$ but the ML estimate of σ^2 can only be obtained by using a numerical optimization providing the ML estimate of α from which the estimate of σ^2 is found.

The Bending Energy Matrix

Since $\hat{\beta}$ is a linear function of \mathbf{y} we have

$$[\mathbf{y} - \mathbf{F}\hat{\beta}]^\top \mathbf{C}^{-1} [\mathbf{y} - \mathbf{F}\hat{\beta}] = \mathbf{y}^\top \mathbf{B} \mathbf{y}$$

where the $n \times n$ matrix \mathbf{B} called the *Bending Energy Matrix* (BEM) is given by

$$\mathbf{B} = \mathbf{C}^{-1} - \mathbf{C}^{-1} \mathbf{F} [\mathbf{F}^\top \mathbf{C}^{-1} \mathbf{F}]^{-1} \mathbf{F}^\top \mathbf{C}^{-1}.$$

The $n \times n$ matrix \mathbf{B} is such that $\mathbf{B}\mathbf{F} = \mathbf{0}$ which means that the columns of \mathbf{F} are eigenvectors of \mathbf{B} with eigenvalue 0. If \mathbf{C} is positive definite and \mathbf{F} has full column rank as assumed, then \mathbf{B} has rank $n - p$.

In the special case where no trend is used i.e., $p = 0$ the bending energy matrix can consistently be defined as $\mathbf{B} := \mathbf{C}^{-1}$, the trend matrix \mathbf{F} then being a matrix with zero columns and the vector β being of length zero.

The BEM matrix is closely related to smoothing since the trend and GP component of \mathbf{y} are given by

$$\mathbf{y} = \underbrace{\hat{\mu}}_{\text{trend}} + \underbrace{\hat{\eta}}_{\text{GP}} = [\mathbf{I}_n - \mathbf{C}\mathbf{B}] \mathbf{y} + \mathbf{C}\mathbf{B} \mathbf{y}.$$

The matrix $\mathbf{I}_n - \mathbf{C}\mathbf{B}$ is the matrix of the orthogonal projection on the linear space spanned by the columns of \mathbf{F} in \mathbb{R}^n equipped with the inner product $\langle \mathbf{z}, \mathbf{z}' \rangle_{\mathbf{C}^{-1}} := \mathbf{z}^\top \mathbf{C}^{-1} \mathbf{z}'$.

Note The BEM does not depend on the specific basis used to define the linear space of trend functions. It also depends on the kernel only through the *reduced kernel* related to the trend linear space, see Pronzato [Pro19]. So the eigen-decomposition of the BEM provides useful insights into the model used such as the so-called *Principal Kriging Functions*

The BEM \mathbf{B} can be related to the matrices \mathbf{C} and \mathbf{F} by a block inversion

$$\begin{bmatrix} \mathbf{C} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{0} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{B} & \mathbf{U} \\ \mathbf{U}^\top & \mathbf{V} \end{bmatrix} \quad \text{with} \quad \begin{cases} \mathbf{V} := -[\mathbf{F}^\top \mathbf{C}^{-1} \mathbf{F}]^{-1} \\ \mathbf{U} := -\mathbf{C}^{-1} \mathbf{F} \mathbf{V} \end{cases}$$

where the inverse exists provided that \mathbf{F} has full column rank, the kernel being assumed to be definite positive.

The relation can be derived by using the so-called *kernel shift* functions $\mathbf{x} \mapsto C(\mathbf{x}, \mathbf{x}_i)$ to represent the GP component of $y(\mathbf{x})$ in the Kriging mean function

$$h(\mathbf{x}) = \underbrace{\sum_{i=1}^n \alpha_i C(\mathbf{x}_i, \mathbf{x})}_{\text{GP}} + \underbrace{\sum_{k=1}^p \beta_k f_k(\mathbf{x})}_{\text{trend}}.$$

In the case where the model has no nugget or noise, using the n observations y_i we can find the $n + p$ unknown coefficients α_i and β_k by imposing the orthogonality constraints $\mathbf{F}^\top \boldsymbol{\alpha} = \mathbf{0}_p$, leading to the linear system

$$\begin{bmatrix} \mathbf{C} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix},$$

see Mardia *et al.* [MKGL96].

It turns out that the trend part of the solution is then identical to the GLS estimate $\hat{\boldsymbol{\beta}}$.

If n^* “new” inputs \mathbf{x}_j^* are given in a matrix \mathbf{X}^* , then with $\mathbf{C}^* := \mathbf{C}(\mathbf{X}^*, \mathbf{X})$ and $\mathbf{F}^* := \mathbf{F}(\mathbf{X}^*)$ the prediction writes in blocks form as

$$\hat{\mathbf{y}}^* = [\mathbf{C}^* \quad \mathbf{F}^*] \begin{bmatrix} \hat{\boldsymbol{\alpha}} \\ \hat{\boldsymbol{\beta}} \end{bmatrix} = [\mathbf{C}^* \quad \mathbf{F}^*] \begin{bmatrix} \mathbf{B} & \mathbf{U} \\ \mathbf{U}^\top & \mathbf{V} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix}.$$

1.4.8 Prediction and simulation

Framework

Consider first the cases where the observations y_i are from a stochastic process $y(\mathbf{x})$ namely the **Kriging** and the **NuggetKriging** cases. Consider n^* “new” inputs \mathbf{x}_j^* given as the rows of a $n^* \times d$ matrix \mathbf{X}^* and the random vector of “new” responses $\mathbf{y}^* := [y(\mathbf{x}_1^*), \dots, y(\mathbf{x}_{n^*}^*)]^\top$. The distribution of \mathbf{y}^* conditional on the observations \mathbf{y} is known: this is a Gaussian distribution, characterized by its mean vector and its covariance matrix

$$\mathbb{E}[\mathbf{y}^* | \mathbf{y}] \quad \text{and} \quad \text{Cov}[\mathbf{y}^* | \mathbf{y}].$$

The computation of this distribution is often called *Kriging*, and more precisely *Universal Kriging* when a linear trend $\mu(\mathbf{x}) = \mathbf{f}(\mathbf{x})^\top \boldsymbol{\beta}$ and a smooth unobserved GP $\zeta(\mathbf{x})$ are used, possibly with a nugget GP $\varepsilon(\mathbf{x})$. Interestingly, the computation can provide estimates $\hat{\mu}(\mathbf{x})$, $\hat{\zeta}(\mathbf{x})$ and $\hat{\varepsilon}(\mathbf{x})$ for the unobserved components: *trend*, *smooth GP* and *nugget*.

In the noisy case “**NoiseKriging**”, the observations y_i are noisy versions of the “trend + GP” process $\eta(\mathbf{x}) := \mu(\mathbf{x}) + \zeta(\mathbf{x})$. Under the assumption that the ε_i are Gaussian, the distribution of the random vector $\boldsymbol{\eta}^* := [\eta(\mathbf{x}_1^*), \dots, \eta(\mathbf{x}_{n^*}^*)]^\top$ conditional on the observations \mathbf{y} is a Gaussian distribution, characterized by its mean vector and its covariance matrix that can be computed by using the same Kriging equations as for the previous cases.

- **The predict method** will provide the conditional expectation $\mathbb{E}[\mathbf{y}^* | \mathbf{y}]$ or $\mathbb{E}[\boldsymbol{\eta}^* | \mathbf{y}]$ a.k.a the Kriging mean. The method can also provide the vector of conditional standard deviations or the conditional covariance matrix which can be called Kriging standard deviation or Kriging covariance.

- The **simulate method** generates partial observations from paths of the process $\eta(\mathbf{x})$ - or $y(\mathbf{x})$ in the non noisy-cases- conditional on the known observations. More precisely, the method returns the values $y^{[k]}(\mathbf{x}_j^*)$ at the new design points for n_{sim} independent drawings $k = 1, \dots, n_{\text{sim}}$ of the process conditional on the observations y_i for $i = 1, \dots, n$. So if n_{sim} is large the average $n_{\text{sim}}^{-1} \sum_{k=1}^{n_{\text{sim}}} y^{[k]}(\mathbf{x}_j^*)$ should be close to the conditional expectation given by the predict method.

In order to give more details on the prediction, the following notations will be used.

- $\mathbf{F}^* := \mathbf{F}(\mathbf{X}^*)$ is the “new” trend matrix with dimension $n^* \times p$.
- $\mathbf{C}^* := \mathbf{C}(\mathbf{X}^*, \mathbf{X})$ is the $n^* \times n$ covariance matrix between the new and the observation inputs. When $n^* = 1$ we have row matrix.
- $\mathbf{C}^{**} := \mathbf{C}(\mathbf{X}^*, \mathbf{X}^*)$ is the $n^* \times n^*$ covariance matrix for the new inputs.

We will assume that the design matrix \mathbf{F} used in the first step has rank p , implying that $n \geq p$ observations are used.

The Kriging prediction

Non-noisy cases Kriging and NuggetKriging

If the covariance kernel is known, the Kriging mean is given by

$$\mathbb{E}[\mathbf{y}^* | \mathbf{y}] = \underbrace{\mathbf{F}^{*\hat{\beta}}}_{\text{trend}} + \underbrace{\mathbf{C}^* \mathbf{C}^{-1} [\mathbf{y} - \mathbf{F}\hat{\beta}]}_{\text{GP}},$$

where $\hat{\beta}$ stands for the GLS estimate of β . At the right-hand side the first term is the prediction of the trend and the second term is the simple Kriging prediction for the GP part ζ^* where the estimation $\hat{\zeta} = \mathbf{y} - \mathbf{F}\hat{\beta}$ is used as if it was containing the unknown observations ζ . The Kriging covariance is given by

$$\text{Cov}[\mathbf{y}^* | \mathbf{y}] = \underbrace{[\mathbf{F}^* - \hat{\mathbf{F}}^*] \text{Cov}(\hat{\beta}) [\mathbf{F}^* - \hat{\mathbf{F}}^*]^\top}_{\text{trend}} + \underbrace{\mathbf{C}^{**} - \mathbf{C}^* \mathbf{C}^{-1} \mathbf{C}^{*\top}}_{\text{GP}},$$

where $\hat{\mathbf{F}}^* := \mathbf{C}^* \mathbf{C}^{-1} \mathbf{F}$ is the simple Kriging prediction of the trend matrix. At the right-hand side, the first term accounts for the uncertainty due to the trend. It disappears if the estimation of β is perfect or if the trend functions are perfectly predicted by Kriging. The second and third terms are the unconditional covariance of the GP part and the (co)variance reduction due to the correlation of the GP between the observations and the new inputs.

Note The conditional covariance can be expressed as

$$\text{Cov}[\mathbf{y}^* | \mathbf{y}] = \mathbf{C}^{**} - \begin{bmatrix} \mathbf{C}^* & \mathbf{F}^* \end{bmatrix} \begin{bmatrix} \mathbf{C} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{C}^{*\top} \\ \mathbf{F}^{*\top} \end{bmatrix}.$$

The block square matrix to be inverted is not positive hence its inverse is not positive either. So the prediction covariance can be larger than the conditional covariance \mathbf{C}^{**} of the GP. This is actually the case in the classical linear regression framework corresponding to the GP $\zeta(\mathbf{x})$ being a white noise.

Note Since a stationary GP $\zeta(\mathbf{x})$ is used in the model, the “Kriging prediction” *returns to the trend*: for a new input \mathbf{x}^* which is far away from the inputs used to fit the model, the prediction $\hat{y}(\mathbf{x}^*)$ tends to the estimated trend $\mathbf{f}(\mathbf{x}^*)^\top \hat{\beta}$.

Noisy case NoiseKriging

In the noisy case we compute the expectation and covariance of η^* conditional on the observations in \mathbf{y} . The formulas are identical to those used for \mathbf{y}^* above. The matrices \mathbf{C}^* and \mathbf{C}^{**} relate to the covariance kernel of the GP $\eta(\mathbf{x})$ yet for the matrix \mathbf{C} , the provided noise variances σ_i^2 must be added to the corresponding diagonal terms.

Plugging the covariance parameters into the prediction

In **libKriging** the prediction is computed by plugging the correlation parameters θ i.e., by replacing these by their estimate obtained by optimizing the chosen objective: *log-likelihood*, *Leave-One-Out Sum of Squared Errors*, or *marginal posterior density*. So the *ranges* θ_ℓ are regarded as *perfectly known*. Similarly the GP variance σ^2 and the nugget variance τ^2 are replaced by their estimates.

Note Mind that the expression *predictive distribution* used in Gu *et al.* [GWB18] is potentially misleading since the correlation parameters are simply plugged into the prediction instead of being marginalized out of the distribution of \mathbf{y}^* conditional on \mathbf{y} .

Confidence interval on the Kriging mean

Consistently with the non-parametric regression framework $y = h(\mathbf{x}) + \varepsilon$ where h is a function that must be estimated, we can speak of a *confidence interval* on the unknown mean at a “new” input point \mathbf{x}^* . It must be understood that the confidence interval is on the smooth part “trend + smooth GP” $h(\mathbf{x}^*) = \mu(\mathbf{x}^*) + \zeta(\mathbf{x}^*)$ of the stochastic process regarded as an unknown deterministic quantity. The “trend + smooth GP” model provides a prior for the unknown function h and the posterior distribution for $h(\mathbf{x}^*)$ is the Gaussian distribution provided by the Kriging prediction.

Some variants of the confidence interval can easily be implemented. In the **Kriging** case here no nugget or noise is used, the maximum likelihood estimate of σ^2 is biased but the *restricted maximum-likelihood estimate* $\hat{\sigma}_{\text{REML}}^2 = \hat{\sigma}_{\text{ML}}^2 \times n/(n-p)$ is unbiased. Also the quantiles of the Student distribution with $n-p$ degree of freedom can be used in place of those of the normal distribution to account for the uncertainty on σ^2 . The same ideas can be used for the “NuggetKriging” and “NoiseKriging” cases.

Derivative w.r.t. the input

The derivative (or gradient) of the prediction mean and of the standard deviation vector with respect to the input vector \mathbf{x}^* can be optionally provided. These derivatives are required in Bayesian Optimization. The derivatives are obtained by applying the chain rule to the expressions for the expectation and the variance.

1.4.9 Maximum likelihood

General form of the likelihood

The general form of the likelihood is

$$L(\boldsymbol{\psi}, \boldsymbol{\beta}; \mathbf{y}) = \frac{1}{[2\pi]^{n/2}} \frac{1}{|\mathbf{C}|^{1/2}} \exp \left\{ -\frac{1}{2} [\mathbf{y} - \mathbf{F}\boldsymbol{\beta}]^\top \mathbf{C}^{-1} [\mathbf{y} - \mathbf{F}\boldsymbol{\beta}] \right\}$$

where $\boldsymbol{\psi}$ is the vector of covariance parameters which depend on the specific Kriging model used, see the section *Parameters*. The notation $|\mathbf{C}|$ is for the determinant of the matrix \mathbf{C} .

Profile likelihood

In the ML framework it turns out that at least the ML estimate $\hat{\beta}$ of the trend coefficient vector can be computed by GLS as exposed in Section *Generalized Least Squares*. Moreover the GLS step can provide an estimate of the variance for the non-trend part component i.e., the difference between the response and the trend part. See Roustant *et al.* [RGD12].

This allows the maximization of a *profile likelihood* function L_{prof} depending on a smaller number of parameters. In practice the log-likelihood $\ell := \log L$ and the log-profile likelihood $\ell_{\text{prof}} := \log L_{\text{prof}}$ are used. The profile log-likelihood functions are detailed and summarized in the *Table below*.

Remind that if we replace β by its estimate $\hat{\beta}$ in the sum of squares used in the log-likelihood, we get a quadratic form of y

$$\left[y - F\hat{\beta} \right]^\top C^{-1} \left[y - F\hat{\beta} \right] = y^\top B y$$

where B is the *Bending Energy Matrix* (BEM).

"Kriging"

In the "Kriging" case where $C = \sigma^2 R(\theta)$, both the ML estimates $\hat{\beta}$ and $\hat{\sigma}^2$ are provided by GLS. So these parameters are "concentrated out of the likelihood" and we can use the profile likelihood function depending on θ only $L_{\text{prof}}(\theta) := L(\theta, \hat{\sigma}^2, \hat{\beta})$ where both $\hat{\sigma}^2$ and $\hat{\beta}$ depend on θ .

"NuggetKriging"

In the "NuggetKriging" case, beside the vector θ of correlation ranges and instead of the couple of parameters $[\sigma^2, \tau^2]$ or $[\sigma^2, \alpha]$ we can use the couple $[\nu^2, \alpha]$ defined by

$$\nu^2 := \sigma^2 + \tau^2, \quad \alpha := \sigma^2 / \nu^2$$

and which can be named the total variance and the variance ratio. The covariance matrix used in the likelihood is then

$$C = \sigma^2 R(\theta) + \tau^2 I = \nu^2 \{ \alpha R(\theta) + (1 - \alpha) I_n \} = \nu^2 R_\alpha(\theta),$$

where R_α is a correlation matrix. As for the Kriging case the ML estimate $\hat{\nu}^2$ can be obtained by GLS as $\hat{\nu}^2 = S^2/n$. Therefore we can use a profile likelihood function depending on the correlation ranges θ and the variance ratio α , namely $L_{\text{prof}}(\theta, \alpha) := L(\theta, \hat{\nu}^2, \hat{\beta})$.

"NoiseKriging"

The covariance matrix to be used in the likelihood is

$$C = \sigma^2 R(\theta) + \text{diag}([\tau_i^2])$$

where the noise variances τ_i^2 are known. In this case the parameter σ^2 can no longer be concentrated out and the profile likelihood to be maximized is a function of θ and σ^2 with only the trend parameter being concentrated out $L_{\text{prof}}(\theta, \sigma^2) := L(\theta, \hat{\beta})$.

Table

The following table gives the profile log-likelihood for the different forms of Kriging models. The sum of squares S^2 is given by $S^2 = \mathbf{e}^\top \hat{\mathbf{C}}^{-1} \mathbf{e}$ where $\mathbf{e} := \mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}$ is the estimated non-trend component and $\hat{\mathbf{C}}$ is the correlation matrix (equal to \mathbf{R} or \mathbf{R}_α).

| | |
|-----------------|---|
| | |
| "Kriging" | $-2\ell_{\text{prof}}(\boldsymbol{\theta}) = \log \mathbf{R} + n \log S^2$ |
| "NuggetKriging" | $-2\ell_{\text{prof}}(\boldsymbol{\theta}, \alpha) = \log \mathbf{R}_\alpha + n \log S^2$ |
| "NoiseKriging" | $-2\ell_{\text{prof}}(\boldsymbol{\theta}, \sigma^2) = \log \mathbf{C} + \mathbf{e}^\top \mathbf{C}^{-1} \mathbf{e}$ |

Note that $\hat{\boldsymbol{\beta}}$ and \mathbf{e} depend on the covariance parameters as do the correlation or covariance matrix. The profile log-likelihood are given up to additive constants. The sum of squares S^2 can be expressed as $S^2 = \mathbf{y}^\top \mathring{\mathbf{B}} \mathbf{y}$ where $\mathring{\mathbf{B}} := \sigma^2 \mathbf{B}$ is a scaled version of the *Bending Energy matrix* \mathbf{B} .

Derivatives w.r.t. the parameters

In the three cases, the symbolic derivatives of the log-profile likelihood w.r.t. the parameters can be obtained by chain rule hence be used in the optimization routine.

1.4.10 Leave-one-out

Consider n observations y_i from a Kriging model corresponding to the "Kriging" case with no nugget or noise. For $i = 1, \dots, n$ let $\hat{y}_{i|-i}$ be the prediction of y_i based on the vector \mathbf{y}_{-i} obtained by omitting the observation i in \mathbf{y} . The vector of *leave-one-out* (LOO) predictions is defined by

$$\hat{\mathbf{y}}_{\text{LOO}} := [\hat{y}_{1|-1}, \dots, \hat{y}_{n|-n}]^\top,$$

and the leave-one-out Sum of Square Errors criterion is defined by

$$\text{SSE}_{\text{LOO}} := \sum_{i=1}^n \{y_i - \hat{y}_{i|-i}\}^2 = \|\mathbf{y} - \hat{\mathbf{y}}_{\text{LOO}}\|^2.$$

It can be shown that

$$\mathbf{y} - \hat{\mathbf{y}}_{\text{LOO}} = \mathbf{D}_{\mathbf{B}}^{-1} \mathbf{B} \mathbf{y}$$

where \mathbf{B} is the *Bending Energy Matrix* (BEM) and $\mathbf{D}_{\mathbf{B}}$ is the diagonal matrix with the same diagonal as \mathbf{B} .

By minimizing SSE_{LOO} with respect to the covariance parameters θ_ℓ we get estimates of these. Note that similarly to the profile likelihood, the LOO MSE does not depend on the vector $\boldsymbol{\beta}$ of trend parameters.

An estimate of the GP variance σ^2 is given by

$$\hat{\sigma}_{\text{LOO}}^2 = \frac{1}{n} \mathbf{y}^\top \mathring{\mathbf{B}} \mathbf{D}_{\mathring{\mathbf{B}}}^{-1} \mathring{\mathbf{B}} \mathbf{y}$$

where $\mathring{\mathbf{B}} := \sigma^2 \mathbf{B}$ does not depend on σ^2 and $\mathbf{D}_{\mathring{\mathbf{B}}}$ is the diagonal matrix having the same diagonal as $\mathring{\mathbf{B}}$.

The LOO estimation can be preferable to the maximum-likelihood estimation when the covariance kernel is misspecified, see Bachoc [Bac12] who provides many details on the criterion SSE_{LOO} , including its derivatives.

1.4.11 Bayesian marginal analysis

Motivation and general form of prior

Berger, De Oliveira, and Sansó have shown that the ML estimation of Kriging models often gives estimated ranges $\hat{\theta}_k = 0$ or $\hat{\theta}_k = \infty$, leading to poor predictions. Although finite positive bounds can be imposed in the optimization to address this issue, the bounds are quite arbitrary. Berger, De Oliveira, and Sansó have shown that one can instead replace the ML estimates by the marginal posterior mode in a Bayesian analysis. Provided that suitable priors are used, it can be shown that the estimated ranges will be both finite and positive: $0 < \hat{\theta}_k < \infty$.

Note In **libKriging** the Bayesian approach will be used only to provide alternatives to the ML estimation of the range or correlation parameters. The Bayesian inference on these parameters will not be achieved. Rather than the profile likelihood, a so-called *marginal likelihood* will be used.

In this section we switch to a Bayesian style of notations. The vector of parameters is formed by three blocks: the vector $\boldsymbol{\theta}$ of correlation ranges, the GP variance σ^2 and the vector $\boldsymbol{\beta}$ of trend parameters. A major concern is the elicitation of the prior density $\pi(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta})$.

Objective priors of Gu et al

A natural idea is that the prior should not provide information about $\boldsymbol{\beta}$, implying the use of *improper* probability densities. With the factorization

$$\pi(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta}) = \pi(\boldsymbol{\beta} | \boldsymbol{\theta}, \sigma^2) \times \pi(\boldsymbol{\theta}, \sigma^2),$$

a further assumption is that the trend parameter vector $\boldsymbol{\beta}$ is a priori independent of the covariance parameters $\boldsymbol{\theta}$ and σ^2 , and that the prior for $\boldsymbol{\beta}$ is an improper prior with constant density

$$\pi(\boldsymbol{\beta} | \boldsymbol{\theta}, \sigma^2) = \pi(\boldsymbol{\beta}) \propto 1.$$

Then the problem boils down to the choice of the joint prior $\pi(\boldsymbol{\theta}, \sigma^2)$.

In the case where no nugget or noise is used, an interesting choice is

$$\pi(\boldsymbol{\theta}, \sigma^2) = \frac{\pi(\boldsymbol{\theta})}{(\sigma^2)^a}$$

with $a > 0$. With this specific form the result of the integration of the likelihood or of the posterior density with respect to σ^2 and $\boldsymbol{\beta}$ is then known in closed form.

Fit: Bayesian marginal analysis

In the **Kriging** case, the *marginal likelihood* a.k.a. *integrated likelihood* for $\boldsymbol{\theta}$ is obtained by marginalizing the GP variance σ^2 and the trend parameter vector $\boldsymbol{\beta}$ out of the likelihood according to

$$L_{\text{marg}}(\boldsymbol{\theta}; \mathbf{y}) := p(\mathbf{y} | \boldsymbol{\theta}) \propto \int p(\mathbf{y} | \boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta}) \frac{1}{\sigma^{2a}} d\sigma^2 d\boldsymbol{\beta},$$

where $p(\mathbf{y} | \boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta})$ is the likelihood $L(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta}; \mathbf{y})$. We get a closed expression given in the [table below](#). Now for a prior having the form (1), the marginal posterior factorizes as

$$p_{\text{marg}}(\boldsymbol{\theta} | \mathbf{y}) \propto \pi(\boldsymbol{\theta}) \times L_{\text{marg}}(\boldsymbol{\theta}; \mathbf{y}).$$

In the **NuggetKriging** case, the same approach can be used, but the parameter used for the nugget is not marginalized out so it remains an argument of the marginal likelihood. In **libKriging** the nugget parameter is taken as $\alpha := \sigma^2 / (\sigma^2 + \tau^2)$ where τ^2 is the nugget variance. We then have the factorization

$$p_{\text{marg}}(\boldsymbol{\theta}, \alpha | \mathbf{y}) \propto \pi(\boldsymbol{\theta}, \alpha) \times L_{\text{marg}}(\boldsymbol{\theta}, \alpha; \mathbf{y}).$$

Note The marginal likelihood differs from the frequentist notion attached to this name. But it also differs from the marginal likelihood as often used in the GP community e.g., in Rasmussen and Williams [RW06] where the marginalization is for the values ζ of the unobserved GP hence is nothing but the likelihood described in [this section](#).

Table of marginal likelihood functions

The following table gives the marginal log-likelihood for the different forms of Kriging models. The sum of squares S^2 is given by $S^2 := \mathbf{e}^\top \mathring{\mathbf{C}}^{-1} \mathbf{e}$ where $\mathbf{e} := \mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}$ and $\mathring{\mathbf{C}}$ is the correlation matrix (equal to \mathbf{R} or \mathbf{R}_α). The sum of squares S^2 can be expressed as $S^2 = \mathbf{y}^\top \mathring{\mathbf{B}} \mathbf{y}$ where $\mathring{\mathbf{B}} := \sigma^2 \mathbf{B}$ is a scaled version of the [Bending Energy matrix](#) \mathbf{B} .

| | |
|-----------------|---|
| | |
| "Kriging" | $-2\ell_{\text{marg}}(\boldsymbol{\theta}) = \log \mathbf{R} + \log \mathbf{F}^\top \mathbf{R}^{-1} \mathbf{F} + (n - p + 2a - 2) \log S^2$ |
| "NuggetKriging" | $-2\ell_{\text{marg}}(\boldsymbol{\theta}, \alpha) = \log \mathbf{R}_\alpha + \log \mathbf{F}^\top \mathbf{R}_\alpha^{-1} \mathbf{F} + (n - p + 2a - 2) \log S^2$ |
| "NoiseKriging" | <i>not used</i> |

It can be interesting to compare this table with the [table of profile log-likelihoods](#).

Reference prior for the correlation parameters [not implemented yet]

For the case when no nugget or noise is used, [Berger, De Oliveira, and Sansó](#) define the reference joint prior for $\boldsymbol{\theta}$ and σ^2 in relation to the integrated likelihood where only the trend parameter $\boldsymbol{\beta}$ is marginalized out, that is $p(\mathbf{y} | \boldsymbol{\theta}, \sigma^2) = \int p(\boldsymbol{\theta}, \sigma^2, \boldsymbol{\beta}) d\boldsymbol{\beta}$ and they show that it has the form

$$\pi_{\text{ref}}(\boldsymbol{\theta}, \sigma^2) = \frac{\pi_{\text{ref}}(\boldsymbol{\theta})}{\sigma^2}$$

where $\pi_{\text{ref}}(\boldsymbol{\theta})$ no longer depends on σ^2 .

We now give some hints on the derivation and the computation of the reference prior. Let $\mathbf{I}^*(\boldsymbol{\theta}, \sigma^2)$ be the $(d+1) \times (d+1)$ Fisher information matrix based on the marginal log-likelihood $\ell_{\text{marg}}(\boldsymbol{\theta}, \sigma^2) = \log L_{\text{marg}}(\boldsymbol{\theta}, \sigma^2)$

$$\mathbf{I}^*(\boldsymbol{\theta}, \sigma^2) := \begin{bmatrix} -\mathbb{E} \left\{ \frac{\partial^2}{\partial \boldsymbol{\theta} \partial \boldsymbol{\theta}^\top} \ell_{\text{marg}}(\boldsymbol{\theta}, \sigma^2) \right\} & -\mathbb{E} \left\{ \frac{\partial^2}{\partial \boldsymbol{\theta} \partial \sigma^2} \ell_{\text{marg}}(\boldsymbol{\theta}, \sigma^2) \right\} \\ -\mathbb{E} \left\{ \frac{\partial^2}{\partial \sigma^2 \partial \boldsymbol{\theta}} \ell_{\text{marg}}(\boldsymbol{\theta}, \sigma^2) \right\} & -\mathbb{E} \left\{ \frac{\partial^2}{\partial \sigma^2 \partial \sigma^2} \ell_{\text{marg}}(\boldsymbol{\theta}, \sigma^2) \right\} \end{bmatrix} =: \begin{bmatrix} \mathbf{H} & \mathbf{u}^\top \\ \mathbf{u} & b \end{bmatrix}.$$

One can show that this information matrix can be expressed by using the $n \times n$ symmetric matrices $\mathbf{N}_k := \mathbf{L}^{-1} [\partial_{\theta_k} \mathbf{R}] \mathbf{L}^{-\top}$ where \mathbf{L} is the lower Cholesky root of the correlation matrix according to

$$\mathbf{H} = \frac{1}{2} \begin{bmatrix} \text{tr}(\mathbf{N}_1 \mathbf{N}_1) & \text{tr}(\mathbf{N}_1 \mathbf{N}_2) & \dots & \text{tr}(\mathbf{N}_1 \mathbf{N}_p) \\ \text{tr}(\mathbf{N}_2 \mathbf{N}_1) & \text{tr}(\mathbf{N}_2 \mathbf{N}_2) & \dots & \text{tr}(\mathbf{N}_2 \mathbf{N}_p) \\ \vdots & \vdots & \ddots & \vdots \\ \text{tr}(\mathbf{N}_p \mathbf{N}_1) & \text{tr}(\mathbf{N}_p \mathbf{N}_2) & \dots & \text{tr}(\mathbf{N}_p \mathbf{N}_p) \end{bmatrix}, \quad \mathbf{u} = \frac{1}{2\sigma^2} \begin{bmatrix} \text{tr}(\mathbf{N}_1) \\ \text{tr}(\mathbf{N}_2) \\ \vdots \\ \text{tr}(\mathbf{N}_p) \end{bmatrix}, \quad b = \frac{n-p}{2\sigma^4}.$$

By multiplying by σ^2 both the last row and the last column of $\mathbf{I}^*(\boldsymbol{\theta}, \sigma^2)$ corresponding to σ^2 , we get a new $(d+1) \times (d+1)$ matrix say $\mathbf{I}^*(\boldsymbol{\theta})$ which no longer depends on σ^2 , the notation $\mathbf{I}^*(\boldsymbol{\theta})$ being consistent with Gu *et al.* [GWB18]. Then $\pi_{\text{ref}}(\boldsymbol{\theta}) = |\mathbf{I}^*(\boldsymbol{\theta})|^{1/2}$.

Note that the determinant expresses as

$$|\mathbf{I}^*(\boldsymbol{\theta})| = |\mathbf{H}| \times |n - p - \hat{\mathbf{u}}^\top \mathbf{H}^{-1} \hat{\mathbf{u}}|$$

where $\hat{\mathbf{u}} := \sigma^2 \mathbf{u}$. See Gu [Gu16] for details.

Note The information matrix takes the blocks in the order: “ $\boldsymbol{\theta}$ then σ^2 ”, while the opposite order is used in Gu [Gu16]. The reference prior suffers from its high computational cost. Indeed, in order to get the value of the prior density one needs the derivatives of the correlation matrix \mathbf{R} and in order to use the derivatives of the prior to find the posterior mode, the second order derivatives of \mathbf{R} are required. An alternative is the following so-called *Jointly robust* prior.

The “Jointly Robust” prior of Gu

Gu [Gu19] defines an easily computed prior called the *Jointly Robust* (JR) prior. This prior is implemented in the R package **RobustGaSP**. In the nugget case the prior is defined with some obvious abuse of notation by

$$\pi_{\text{JR}}(\boldsymbol{\theta}, \sigma^2, \alpha) \propto \frac{\pi_{\text{JR}}(\boldsymbol{\theta}, \alpha)}{\sigma^2}$$

where as above $\alpha := \sigma^2/(\sigma^2 + \tau^2)$ so that the nugget variance ratio $\eta := \tau^2/\sigma^2$ of Gu [Gu19] is $\eta = (1 - \alpha)/\alpha$. The JR prior corresponds to

$$\pi_{\text{JR}}(\boldsymbol{\theta}, \alpha) \propto t^{a_{\text{JR}}} \exp\{-b_{\text{JR}}t\} \quad t := \frac{1 - \alpha}{\alpha} + \sum_{\ell=1}^d \frac{C_{\ell}}{\theta_{\ell}},$$

where $a_{\text{JR}} > -(d + 1)$ and $b_{\text{JR}} > 0$ are two hyperparameters and C_{ℓ} is proportional to the range r_{ℓ} of the column ℓ in \mathbf{X}

$$C_{\ell} = n^{-1/d} \times r_{\ell}, \quad r_{\ell} := \max_i \{X_{i\ell}\} - \min_i \{X_{i\ell}\}.$$

The values of a_{JR} and b_{JR} are chosen as

$$a_{\text{JR}} := 0.2, \quad b_{\text{JR}} := n^{-1/d} \times (a_{\text{JR}} + d).$$

Note that as opposed to the objective prior described above, the JR prior does not depend on the specific kernel chosen for the GP. However the integration w.r.t. σ^2 and β is the same as for the reference prior, which means that the marginal likelihood is the same as for the reference prior above corresponding to $a = 1$ in the prior (1) above.

Caution The parameter a_{JR} is denoted by a in Gu [Gu19] and in the code of **libKriging**. It differs from the exponent a of σ^{-2} used above.

1.5 References

BIBLIOGRAPHY

- [Bac12] François Bachoc. *Parametric Estimation of Covariance Function in Gaussian-Process based Kriging Models. Application to Uncertainty Quantification for Computer Experiments*. PhD thesis, Université Paris Diderot, 2012.
- [BDOS01] James O. Berger, Victor De Oliveira, and Bruno Sansó. Objective Bayesian Analysis of Spatially Correlated Data. *Journal of the American Statistical Association*, 96(456):1361–1374, 2001. doi:10.1198/016214501753382282.
- [BTA04] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Space in Probability and Statistics*. Springer, 2004. doi:10.1007/978-1-4419-9096-9.
- [Gu16] Mengyang Gu. *Robust Uncertainty Quantification and Scalable Computation for Computer Models with Massive Output*. PhD thesis, Duke University, 2016. URL: <https://hdl.handle.net/10161/12882>.
- [Gu19] Mengyang Gu. Jointly Robust Prior for Gaussian Stochastic Process in Emulation, Calibration and Variable Selection. *Bayesian Analysis*, 14(3):857–885, 2019. doi:10.1214/18-BA1133.
- [GWB18] Mengyang Gu, Xiaojing Wand, and James O. Berger. Robust Gaussian Stochastic Process Emulation. *Annals of Statistics*, 46(6A):3038–3066, 2018. doi:10.1214/17-AOS1648.
- [Lan10] Kenneth Lange. *Numerical Analysis for Statisticians*. Statistics & Computing. Springer-Verlag, 2nd edition, 2010. doi:10.1007/978-1-4419-5945-4.
- [MKGL96] Kantilal V. Mardia, John T. Kent, Colin R. Goodall, and John A. Little. Kriging and Splines with Derivative Information. *Biometrika*, 83(1):207–221, 03 1996. doi:10.1093/biomet/83.1.207.
- [Pro19] Luc Pronzato. Sensitivity Analysis via Karhunen-Loève Expansion of a Random Field Model: Estimation of Sobol' Indices and Experimental Design. *Reliability Engineering and System Safety*, pages 93–109, 2019. doi:10.1016/j.ress.2018.01.010.
- [RW06] Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. doi:10.7551/mitpress/3206.001.0001.
- [RGD12] Olivier Roustant, David Ginsbourger, and Yves Deville. DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. *Journal of Statistical Software*, pages 1–55, 2012. doi:10.18637/jss.v051.i01.
- [Ste12] Michael L. Stein. *Interpolation of Spatial Data. Some Theory for Kriging*. Springer Series in Statistics. Springer-Verlag, 2012. doi:10.1007/978-1-4612-1494-6.
- [Wah78] Grace Wahba. Improper Priors, Spline Smoothing and the Problem of Guarding Against Model Errors in Regression. *Journal of the Royal Statistical Society: Series B*, 40(3):364–372, 1978. doi:10.1111/j.2517-6161.1978.tb01050.x.