Package: rkriging (via r-universe)

November 6, 2026

Type Package

Title Kriging Modeling

Version 1.1.0

Description An 'Eigen'-based computationally efficient 'C++' implementation for fitting various kriging models to data. This research is supported by U.S. National Science Foundation grant DMS-2310637.

License GPL (>= 2)

Depends R (>= 3.0.2)

Imports Rcpp, nloptr (>= 1.2.0), methods, stats

LinkingTo Rcpp, RcppEigen (>= 0.3.3.5.0), BH (>= 1.75.0.0), nloptr (>= 1.2.0)

Encoding UTF-8

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Chaofan Huang [aut], Difan Song [aut, cre], V. Roshan Joseph [aut]

Maintainer Difan Song <dfsong96@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-06 18:10:29 UTC

RemoteUrl https://github.com/cran/rkriging

RemoteRef HEAD

RemoteSha 7e0cc44352686e5d0b3ee5b9b01e81cd95b2edd4

Contents

Add.Next.Design 2
ALM.Sequential 4
EI.Sequential 6
Evaluate.Kernel 9

Fit.Kriging . . . 10
Gaussian.Kernel . . . 15
Generalized.Rational.Kriging . . . 16
Get.Best.Design . . . 18
Get.Data . . . 20
Get.Kernel . . . 21
Get.Kriging.Parameters . . . 23
Get.Next.Design . . . 25
Get.Sequential . . . 27
IM.Kernel . . . 29
Limit.Kriging . . . 31
Matern.Kernel . . . 33
Matern12.Kernel . . . 34
Matern32.Kernel . . . 36
Matern52.Kernel . . . 37
MultiplicativeIM.Kernel . . . 38
MultiplicativeMatern.Kernel . . . 40
MultiplicativeRQ.Kernel . . . 41
MultiplicativeUDF.Kernel . . . 42
Ordinary.Kriging . . . 44
PI.Sequential . . . 46
Predict.Kriging . . . 48
Predict.Sequential . . . 50
Rational.Kriging . . . 51
RQ.Kernel . . . 53
UCB.Sequential . . . 55
UDF.Kernel . . . 57
UDF.Sequential . . . 58
Universal.Kriging . . . 61

Index **64**

`Add.Next.Design` *Add Next Design to Sequential Object*

Description

This function adds a new design point and its corresponding output to the sequential design object and updates kriging (optionally refit the kriging hyperparameters).

Usage

```
Add.Next.Design(sequential, new.design, refit.kriging = TRUE)
```

Arguments

`sequential`	a sequential design object (e.g., from ALM.Sequential, EI.Sequential, etc.)
`new.design`	a list containing:
	`X` matrix or vector of new design point(s) to add
	`y` vector of corresponding output value(s)
`refit.kriging`	a logical indicating whether to refit the kriging hyperparameters

Details

The new design is provided as a list containing:

`X` a matrix or vector of new design point(s); if a vector, it will be converted to a 1×p matrix

`y` a vector of corresponding output value(s)

This function is typically used in a sequential design loop together with `Get.Next.Design`, allowing fine-grained control over each iteration. If the sequential object was created from a scaled kriging object, `new.design$X` should be provided on the original input scale and is scaled internally before updating the kriging model.

Value

The sequential object (modified in-place).

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

Get.Next.Design, Get.Sequential.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
```

```
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("EI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")
```

`ALM.Sequential` *ALM Sequential Design*

Description

This function creates an Active Learning MacKay (ALM) sequential design object for emulation (space-filling based on predictive uncertainty).

Usage

```
ALM.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`
: a list specifying the design space, containing:

  `design.lower.bound` a vector for the lower bound of the design space

  `design.upper.bound` a vector for the upper bound of the design space

`kriging`
: (optional if kriging.parameters provided) a fitted kriging class object (e.g., from Ordinary.Kriging or Fit.Kriging)

`kriging.parameters`
: (optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters

`nlopt.parameters`
: a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

ALM selects the next design point as the one that maximizes the predictive standard deviation:

$$x_{next} = \arg\max_x \sigma(x),$$

where $\sigma(x)$ is the kriging predictive standard deviation at x. This criterion is suitable for emulation tasks where the goal is to reduce overall prediction uncertainty.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via `design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion.

Value

An ALM Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

MacKay, D. J. C. (1992). Information-based objective functions for active data selection. *Neural computation*, 4(4), 590-604.

See Also

Predict.Sequential, Get.Next.Design, Add.Next.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- ALM.Sequential(design.parameters, kriging)

# sequential design loop with visualization
n.iterations <- 5
for (i in 1:n.iterations) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("ALM Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="ALM Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
```

`EI.Sequential` *EI Sequential Design*

Description

This function creates an Expected Improvement (EI) sequential design object for optimization (finding the minimum of the objective function).

Usage

```
EI.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`
: a list specifying the design space, containing:

  `design.lower.bound` a vector for the lower bound of the design space

  `design.upper.bound` a vector for the upper bound of the design space

`kriging`
: (optional if kriging.parameters provided) a fitted kriging class object (e.g., from Ordinary.Kriging or Fit.Kriging)

`kriging.parameters`
: (optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters

`nlopt.parameters`
: a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

EI selects the next design point as the one that maximizes the expected improvement:

$$x_{next} = \arg\max_x \mathrm{EI}(x) = \sigma(x)\left[u\Phi(u) + \phi(u)\right],$$

where $u = (y_{min} - \mu(x))/\sigma(x)$, $\mu(x)$ and $\sigma(x)$ are the kriging predictive mean and standard deviation, and Φ and ϕ are the standard normal CDF and PDF. This is the most popular acquisition function for Bayesian optimization.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via `design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion.

Value

An EI Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4), 455-492.

See Also

PI.Sequential, UCB.Sequential, UDF.Sequential, Get.Next.Design, Add.Next.Design, Get.Best.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("EI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
```

```
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")
```

`Evaluate.Kernel` *Evaluate Kernel*

Description

This function computes the kernel (correlation) matrix. Given the kernel class object and the input data X of size n, this function computes the corresponding $n \times n$ kernel (correlation) matrix.

Usage

```
Evaluate.Kernel(kernel, X)
```

Arguments

`kernel`	a kernel class object
`X`	input data

Value

The kernel (correlation) matrix of X evaluated by the kernel.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

Get.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

kernel <- Gaussian.Kernel(lengthscale)
Evaluate.Kernel(kernel, X)
```

`Fit.Kriging` *Fit or Create Kriging Models*

Description

This function provides a common interface to fit various kriging models from the data. OK (Ordinary.Kriging), UK (Universal.Kriging), LK (Limit.Kriging), RK (Rational.Kriging), and GRK (Generalized.Rational.Kriging) are supported in this package.

Usage

```
Fit.Kriging(
  X,
  y,
  interpolation = TRUE,
  fit = TRUE,
  nugget = NULL,
  model = "OK",
  model.parameters = list(),
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list(),
  scale = FALSE
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`interpolation`	whether to interpolate, for noisy data please set `interpolate=FALSE`
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`model`	choice of kriging model: OK, UK, LK, RK, GRK
`model.parameters`	a list of parameters required for the specific kriging model, e.g. basis functions for universal kriging
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation
`scale`	whether to scale each input dimension to `[0, 1]` before fitting. When `TRUE`, the training minima and maxima are stored in the returned object and reused by Predict.Kriging.

Details

Kriging gives the best linear unbiased predictor given the data. It can also be given a Bayesian interpretation by assuming a Gaussian process (GP) prior for the underlying function. Please see Santner et al. (2003), Rasmussen and Williams (2006), Joseph (2025), and Joseph (2026) for details.

For data from deterministic computer experiments, use `interpolation=TRUE` and will give an interpolator. For noisy data, use `interpolation=FALSE`, which will give an approximator of the underlying function. Currently, approximation is supported for OK (Ordinary.Kriging) and UK (Universal.Kriging).

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section for detail usages.

When the lengthscale / correlation parameters are known, simply provide them in (i) the kernel class object to `kernel` or (ii) in `kernel.parameters`, and set `fit=FALSE`. The kriging model will be fitted with the user provided parameters.

When the lengthscale / correlation parameters are unknown, they can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt, a open-source library for nonlinear optimization. All gradient-free optimization methods in NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section for detail usages.

Value

A Kriging Class Object. When `scale=TRUE`, the returned object stores the training input minima and maxima as `scale_min` and `scale_max` attributes.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Santner, T. J., Williams, B. J., & Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Springer.

Joseph, V. R. (2006). *Limit kriging*. Technometrics, 48(4), 458-466.

Joseph, V. R. (2025). Rational Kriging. *Journal of the American Statistical Association*. 120, 548-558.

Joseph, V. R. (2026). *Experimental Design for Data Science and Engineering*. Chapman & Hall/CRC, doi:10.1201/9781003661535.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Predict.Kriging, Get.Kriging.Parameters, Get.Kernel, Ordinary.Kriging, Universal.Kriging, Limit.Kriging, Rational.Kriging, Generalized.Rational.Kriging.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

##############################################################################
############### Minimal Example for Fitting a Kriging Model ################
##############################################################################
# Ordinary Kriging
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# Universal Kriging
basis.function <- function(x) {c(1,x[1],x[1]^2)}
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="UK",
                       model.parameters=list(basis.function=basis.function),
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# Limit Kriging
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="LK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
```

```
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# Rational Kriging
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="RK",
                       kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# Generalized Rational Kriging
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="GRK",
                       kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

##############################################################################
################ Fitting a Kriging Model with Kernel Object #################
##############################################################################
kernel <- Gaussian.Kernel(rep(1,p))
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK", kernel=kernel)
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

##############################################################################
############### Creating a Kriging Model with Kernel Object #################
##############################################################################
# set fit = FALSE to create Kriging model with user provided kernel
# no optimization for the length scale parameters
kernel <- Gaussian.Kernel(rep(1e-1,p))
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=FALSE, model="OK", kernel=kernel)
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
```

```
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

##############################################################################
############ Fitting a Kriging Model with Range for Lengthscale #############
##############################################################################
kernel.parameters <- list(
    type = 'Gaussian',
    lengthscale = rep(1,p), # initial value
    lengthscale.lower.bound = rep(1e-3,p), # lower bound
    lengthscale.upper.bound = rep(1e3,p) # upper bound
) # if not provided, a good estimate would be computed from data
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=kernel.parameters)
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

##############################################################################
########### Fitting a Kriging Model with Different Optimization #############
##############################################################################
nlopt.parameters <- list(
    algorithm = 'NLOPT_GN_DIRECT', # optimization method
    maxeval = 250 # maximum number of evaluation
)
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"),
                       nlopt.parameters=nlopt.parameters)
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# Global-Local Optimization
nlopt.parameters <- list(
    algorithm = 'NLOPT_GN_MLSL_LDS', # optimization method
    local.algorithm = 'NLOPT_LN_SBPLX', # local algorithm
    maxeval = 250 # maximum number of evaluation
)
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"),
                       nlopt.parameters=nlopt.parameters)
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
```

```
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

#############################################################################
################ Fitting a Kriging Model from Noisy Data ####################
#############################################################################
y <- y + 0.1 * rnorm(length(y))
kriging <- Fit.Kriging(X, y, interpolation=FALSE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

`Gaussian.Kernel` *Gaussian Kernel*

Description

This function specifies the Gaussian / Squared Exponential (SE) / Radial Basis Function (RBF) kernel.

Usage

```
Gaussian.Kernel(lengthscale)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters

Details

The Gaussian kernel is given by

$$k(r) = \exp(-r^2/2),$$

where

$$r(x, x') = \sqrt{\sum_{i=1}^{p} \left(\frac{x_i - x_i'}{l_i} \right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

A Gaussian Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- Gaussian.Kernel(lengthscale)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="Gaussian")
Evaluate.Kernel(kernel, X)
```

Generalized.Rational.Kriging

Generalized Rational Kriging

Description

This functions fits the generalized rational kriging model to the data.

Usage

```
Generalized.Rational.Kriging(
  X,
  y,
  fit = TRUE,
  nugget = NULL,
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation

Details

Ordinary kriging and rational kriging can be obtained as special cases of generalized rational kriging. Please see Joseph (2025) for details. The Spectra library is used for fast computation of the first eigenvalues/vectors. Only interpolation is available. Noisy output is not supported for generalized rational kriging.

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section of Fit.Kriging for detail usages.

When the lengthscale / correlation parameters are unknown, all parameters including the constant mean can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt library, a open-source library for nonlinear optimization. All gradient-free optimization methods in NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section of Fit.Kriging for detail usages.

Value

A Generalized Rational Kriging Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2025). Rational Kriging. *Journal of the American Statistical Association*. 120, 548-558.

Qiu, Y., Guennebaud, G., & Niesen, J. (2015). Spectra: C++ library for large scale eigenvalue problems.

See Also

Fit.Kriging, Predict.Kriging, Get.Kriging.Parameters.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

# approach 1
kriging <- Generalized.Rational.Kriging(X, y, fit=TRUE,
                                        kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# approach 2
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="GRK",
                       kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

`Get.Best.Design` *Get Best Design from Optimization Sequential*

Description

This function returns the best (minimum) objective value and its corresponding design from an optimization sequential design object (e.g., EI.Sequential, PI.Sequential, UCB.Sequential).

Usage

```
Get.Best.Design(sequential)
```

Arguments

sequential	an optimization sequential design object

Details

If the sequential object was created from a scaled kriging object, the returned best design is converted back to the original input scale.

Value

A list with components:

`design` the design (input) corresponding to the best objective

`objective` the best (minimum) objective value

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

EI.Sequential, PI.Sequential, UCB.Sequential.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# fit kriging
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)

# sequential design loop with visualization
for (i in 1:5) {
```

```
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("EI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")
```

`Get.Data` *Get Data from Sequential*

Description

This function retrieves the current training data (X and y) from a sequential design object.

Usage

```
Get.Data(sequential)
```

Arguments

`sequential`	a sequential design object (e.g., from ALM.Sequential, EI.Sequential, etc.)

Details

The sequential design object maintains the current training data internally. This function provides access to both the design matrix X and the corresponding output vector y. If the sequential object was created from a scaled kriging object, `X` is converted back to the original input scale.

Value

A list with components:

`X` the design matrix (training inputs)

`y` the output vector (training responses)

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

Get.Sequential.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)
Get.Data(sequential)
```

Get.Kernel *Kernel*

Description

This function provides a common interface to specify various kernels. See arguments section for the available kernels in this package.

Usage

```
Get.Kernel(lengthscale, type, parameters = list())
```

Arguments

lengthscale	a vector for the positive length scale parameters
type	kernel type: Gaussian, RQ, IM, Matern12, Matern32, Matern52, Matern, UDF, MultiplicativeRQ, MultiplicativeIM, MultiplicativeMatern, MultiplicativeUDF
parameters	a list of parameters required for the specific kernel

Value

A Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Evaluate.Kernel, Gaussian.Kernel, RQ.Kernel, IM.Kernel, Matern12.Kernel, Matern32.Kernel, Matern52.Kernel Matern.Kernel, UDF.Kernel, MultiplicativeRQ.Kernel, MultiplicativeIM.Kernel, MultiplicativeMatern.Kernel, MultiplicativeUDF.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# Gaussian
kernel <- Get.Kernel(lengthscale, type="Gaussian")
Evaluate.Kernel(kernel, X)

# Rational Quadratic (RQ)
kernel <- Get.Kernel(lengthscale, type="RQ", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)

# Inverse Multiquadric (IM)
kernel <- Get.Kernel(lengthscale, type="IM", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)

# Matern(1/2)
kernel <- Get.Kernel(lengthscale, type="Matern12")
Evaluate.Kernel(kernel, X)

# Matern(3/2)
```

```
kernel <- Get.Kernel(lengthscale, type="Matern32")
Evaluate.Kernel(kernel, X)

# Matern(5/2)
kernel <- Get.Kernel(lengthscale, type="Matern52")
Evaluate.Kernel(kernel, X)

# Generalized Matern
kernel <- Get.Kernel(lengthscale, type="Matern", parameters=list(nu=2.01))
Evaluate.Kernel(kernel, X)

# User Defined Function (UDF) Kernel
kernel.function <- function(sqdist) {return (exp(-sqrt(sqdist)))}
kernel <- Get.Kernel(lengthscale, type="UDF",
                     parameters=list(kernel.function=kernel.function))
Evaluate.Kernel(kernel, X)

# Multiplicative Rational Quadratic (RQ)
kernel <- Get.Kernel(lengthscale, type="MultiplicativeRQ", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)

# Multiplicative Inverse Multiquadric (IM)
kernel <- Get.Kernel(lengthscale, type="MultiplicativeIM", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)

# Multiplicative Generalized Matern
kernel <- Get.Kernel(lengthscale, type="MultiplicativeMatern", parameters=list(nu=2.01))
Evaluate.Kernel(kernel, X)

# Multiplicative User Defined Function (UDF)
kernel.function <- function(sqdist) {return (exp(-sqrt(sqdist)))}
kernel <- Get.Kernel(lengthscale, type="MultiplicativeUDF",
                     parameters=list(kernel.function=kernel.function))
Evaluate.Kernel(kernel, X)
```

`Get.Kriging.Parameters` *Get Kriging Parameters*

Description

This function can be used for extracting the estimates of the kriging parameters.

Usage

```
Get.Kriging.Parameters(kriging)
```

Arguments

`kriging`	a kriging class object

Value

`loss`	the loss value of the fitted kriging model
`loss.type`	the loss type used for fitting (NLLH)
`lengthscale`	the lengthscale of the kernel
`mu`	mean of the kriging model
`nu2`	variance of the kriging model
`sigma2`	variance of the random noise when `interpolation=FALSE`
`beta`	coefficients of the basis functions for universal kriging
`c`	c for the rational / generalized rational kriging, see Joseph (2025)
`c0`	c0 for the generalized rational kriging, see Joseph (2025)
`scale`	whether the model was fitted on scaled inputs
`scale_min`	training input minima used for scaling, present when `scale=TRUE`
`scale_max`	training input maxima used for scaling, present when `scale=TRUE`

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2006). *Limit kriging*. Technometrics, 48(4), 458-466.

Joseph, V. R. (2025). Rational Kriging. *Journal of the American Statistical Association*. 120, 548-558.

Joseph, V. R. (2026). *Experimental Design for Data Science and Engineering*. Chapman & Hall/CRC, doi:10.1201/9781003661535.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Fit.Kriging.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
```

```
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"))
Get.Kriging.Parameters(kriging)
```

`Get.Next.Design` *Get Next Design from Sequential*

Description

This function selects the next design point using the sequential criterion.

Usage

```
Get.Next.Design(sequential, design.candidates = NULL)
```

Arguments

`sequential`	a sequential design object (e.g., from ALM.Sequential, EI.Sequential, etc.)
`design.candidates`	(optional) a matrix of candidate design points; if provided, the next design is selected from this set instead of NLopt optimization

Details

If `design.candidates` is provided, the next design is selected from the candidate set by evaluating the criterion at each candidate point. Otherwise, NLopt optimization is used within the design space bounds specified in `design.parameters`. If the sequential object was created from a scaled kriging object, `design.candidates` should be provided on the original input scale and the returned next design is also on the original input scale.

Value

A vector representing the next design point.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

Get.Sequential, Add.Next.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# fit kriging
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("EI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")

# alternatively can pass a set of candidate points for finding the next design
design.candidates <- matrix(seq(0,1,length=1001), ncol=p)
Get.Next.Design(sequential, design.candidates=design.candidates)
```

`Get.Sequential` *Get Sequential Design*

Description

This function provides a common interface to create various sequential design objects. ALM (ALM.Sequential), EI (EI.Sequential), PI (PI.Sequential), UCB (UCB.Sequential), and UDF (UDF.Sequential) are supported.

Usage

```
Get.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  method = "ALM",
  method.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`
: a list specifying the design space, containing:

  `design.lower.bound` a vector for the lower bound of the design space

  `design.upper.bound` a vector for the upper bound of the design space

`kriging`
: (optional if kriging.parameters provided) a fitted kriging class object

`kriging.parameters`
: (optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters

`method`
: choice of sequential design method: ALM, EI, PI, UCB, UDF

`method.parameters`
: a list of parameters required for the specific method, e.g. `lambda` for UCB, `criterion.function` and `criterion.parameters` for UDF

`nlopt.parameters`
: a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

Sequential design methods select the next design point iteratively based on an acquisition criterion. For emulation tasks (space-filling), use `method="ALM"` which maximizes predictive uncertainty. For optimization tasks (finding the minimum), use `method="EI"`, `"PI"`, or `"UCB"`. A custom criterion can be specified via `method="UDF"`.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via

`design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion. For `method="UDF"`, the user-defined criterion function receives the current training `X` on the original input scale.

The kriging model can be provided either as:

An existing kriging object e.g., from Ordinary.Kriging or Fit.Kriging

Kriging parameters as a list to pass to Fit.Kriging, must contain at least X and y

Value

A Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

ALM.Sequential, EI.Sequential, PI.Sequential, UCB.Sequential, UDF.Sequential, Predict.Sequential, Get.Next.Design, Add.Next.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- Get.Sequential(design.parameters, kriging, method="EI")

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("EI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
```

```
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")

# alternatively can create sequential by passing kriging parameters
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
kriging.parameters <- list(X=X, y=y, interpolation=TRUE, fit=TRUE,
                           kernel.parameters=list(type="Gaussian"))
sequential <- Get.Sequential(design.parameters, kriging.parameters=kriging.parameters, method="EI")
```

`IM.Kernel` *Inverse Multiquadric (IM) Kernel*

Description

This function specifies the Inverse Multiquadric (IM) kernel.

Usage

```
IM.Kernel(lengthscale, alpha = 1)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`alpha`	a positive scalar shape parameter that controls the kernel decay

Details

The Inverse Multiquadric (IM) kernel is given by

$$k(r;\alpha) = \left(1 + \frac{r^2}{2}\right)^{-\alpha},$$

where α is a positive shape parameter and

$$r(x, x') = \sqrt{\sum_{i=1}^{p} \left(\frac{x_i - x_i'}{l_i} \right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

An Inverse Multiquadric (IM) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Hardy, R. L. & Göpfert, W. M. (1975). Least squares prediction of gravity anomalies, geoidal undulations, and deflections of the vertical with multiquadric harmonic functions. *Geophysical Research Letters*, 2(10), 423–426. doi:10.1029/GL002i010p00423

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

MultiplicativeIM.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- IM.Kernel(lengthscale, alpha=1)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="IM", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)
```

`Limit.Kriging` *Limit Kriging*

Description

This functions fits the limit kriging model to the data.

Usage

```
Limit.Kriging(
  X,
  y,
  fit = TRUE,
  nugget = NULL,
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation

Details

Limit kriging avoids the mean reversion issue of ordinary kriging. Please see Joseph (2006) for details. Only interpolation is available. Noisy output is not supported for limit kriging.

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section of Fit.Kriging for detail usages.

When the lengthscale / correlation parameters are unknown, all parameters including the constant mean can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt, a open-source library for nonlinear optimization. All gradient-free optimization methods in

NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section of Fit.Kriging for detail usages.

Value

A Limit Kriging Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2006). *Limit kriging*. Technometrics, 48(4), 458-466.

See Also

Fit.Kriging, Predict.Kriging, Get.Kriging.Parameters.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

# approach 1
kriging <- Limit.Kriging(X, y, fit=TRUE, kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# approach 2
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="LK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
```

```
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

`Matern.Kernel` *Generalized Matern Kernel*

Description

This function specifies the (Generalized) Matern kernel with any smoothness parameter ν.

Usage

```
Matern.Kernel(lengthscale, nu = 2.01)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`nu`	a positive scalar parameter that controls the smoothness

Details

The Generalized Matern kernel is given by

$$k(r;\nu) = \frac{2^{1-\nu}}{\Gamma(\nu)}(\sqrt{2\nu}r)^{\nu}K_{\nu}(\sqrt{2\nu}r),$$

where ν is the smoothness parameter, K_{ν} is the modified Bessel function, Γ is the gamma function, and

$$r(x,x') = \sqrt{\sum_{i=1}^{p}\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's. As $\nu \to \infty$, it converges to the Gaussian.Kernel.

Value

A Generalized Matern Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Matern12.Kernel, Matern32.Kernel, Matern52.Kernel, MultiplicativeMatern.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- Matern.Kernel(lengthscale, nu=2.01)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="Matern", parameters=list(nu=2.01))
Evaluate.Kernel(kernel, X)
```

Matern12.Kernel *Matern(1/2) Kernel*

Description

This function specifies the Matern kernel with smoothness parameter ν=1/2. It is also known as the Exponential kernel.

Usage

```
Matern12.Kernel(lengthscale)
```

Arguments

lengthscale	a vector for the positive length scale parameters

Details

The Matern(1/2) kernel is given by

$$k(r) = \exp(-r),$$

where

$$r(x, x') = \sqrt{\sum_{i=1}^{p} \left(\frac{x_i - x_i'}{l_i} \right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

A Matern(1/2) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Matern.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- Matern12.Kernel(lengthscale)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="Matern12")
Evaluate.Kernel(kernel, X)
```

`Matern32.Kernel` *Matern(3/2) Kernel*

Description

This function specifies the Matern kernel with smoothness parameter ν=3/2.

Usage

```
Matern32.Kernel(lengthscale)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters

Details

The Matern(3/2) kernel is given by

$$k(r) = (1 + \sqrt{3}r)\exp(-\sqrt{3}r),$$

where

$$r(x, x') = \sqrt{\sum_{i=1}^{p}\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

A Matern(3/2) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Matern.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- Matern32.Kernel(lengthscale)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="Matern32")
Evaluate.Kernel(kernel, X)
```

Matern52.Kernel *Matern(5/2) Kernel*

Description

This function specifies the Matern kernel with smoothness parameter ν=5/2.

Usage

```
Matern52.Kernel(lengthscale)
```

Arguments

lengthscale	a vector for the positive length scale parameters

Details

The Matern(5/2) kernel is given by

$$k(r) = (1 + \sqrt{5}r + 5r^2/3)\exp(-\sqrt{5}r),$$

where

$$r(x, x') = \sqrt{\sum_{i=1}^{p}\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

A Matern(5/2) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Matern.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- Matern52.Kernel(lengthscale)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="Matern52")
Evaluate.Kernel(kernel, X)
```

MultiplicativeIM.Kernel

Multiplicative Inverse Multiquadric (IM) Kernel

Description

This function specifies the Multiplicative Inverse Multiquadric (IM) kernel.

Usage

```
MultiplicativeIM.Kernel(lengthscale, alpha = 1)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`alpha`	a positive scalar shape parameter that controls the kernel decay

Details

The Multiplicative Inverse Multiquadric (IM) kernel is given by

$$k(r;\alpha) = \prod_{i=1}^{p} \left(1 + \frac{r_i^2}{2}\right)^{-\alpha},$$

where α is a positive shape parameter and

$$r_i(x, x') = \sqrt{\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the dimension-wise euclidean distances between x and x' weighted by the length scale parameters l_i's.

Value

A Multiplicative Inverse Multiquadric (IM) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Song, D. & Joseph, V. R. (2026). Efficient Active Learning Strategies for Computer Experiments. *Technometrics*, 68(1), 65–78. doi:10.1080/00401706.2025.2546366

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

IM.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- MultiplicativeIM.Kernel(lengthscale, alpha=1)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="MultiplicativeIM", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)
```

`MultiplicativeMatern.Kernel` *Multiplicative Generalized Matern Kernel*

Description

This function specifies the Multiplicative Generalized Matern kernel.

Usage

```
MultiplicativeMatern.Kernel(lengthscale, nu = 2.01)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`nu`	a positive scalar parameter that controls the smoothness

Details

The Multiplicative Generalized Matern kernel is given by

$$k(r;\nu) = \prod_{i=1}^{p} \frac{2^{1-\nu}}{\Gamma(\nu)} (\sqrt{2\nu} r_i)^{\nu} K_{\nu}(\sqrt{2\nu} r_i),$$

where ν is the smoothness parameter, K_ν is the modified Bessel function, Γ is the gamma function, and

$$r_i(x, x') = \sqrt{\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the dimension-wise euclidean distances between x and x' weighted by the length scale parameters l_i's.

Value

A Multiplicative Generalized Matern Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Matern.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- MultiplicativeMatern.Kernel(lengthscale, nu=2.01)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="MultiplicativeMatern", parameters=list(nu=2.01))
Evaluate.Kernel(kernel, X)
```

`MultiplicativeRQ.Kernel`

Multiplicative Rational Quadratic (RQ) Kernel

Description

This function specifies the Multiplicative Rational Quadratic (RQ) kernel.

Usage

```
MultiplicativeRQ.Kernel(lengthscale, alpha = 1)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`alpha`	a positive scalar for the scale mixture parameter that controls the relative weighting of large-scale and small-scale variations

Details

The Multiplicative Rational Quadratic (RQ) kernel is given by

$$k(r;\alpha) = \prod_{i=1}^{p} \left(1 + \frac{r_i^2}{2\alpha}\right)^{-\alpha},$$

where α is the scale mixture parameter and

$$r_i(x, x') = \sqrt{\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the dimension-wise euclidean distances between x and x' weighted by the length scale parameters l_i's.

Value

A Multiplicative Rational Quadratic (RQ) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Song, D. and Joseph, V. R. (2026). "Efficient Active Learning Strategies for Computer Experiments." *Technometrics*, 68, 65-78.

See Also

RQ.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- MultiplicativeRQ.Kernel(lengthscale, alpha=1)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="MultiplicativeRQ", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)
```

MultiplicativeUDF.Kernel

Multiplicative User Defined Function (UDF) Kernel

Description

This function specifies the Multiplicative kernel with the user defined R function.

Usage

```
MultiplicativeUDF.Kernel(lengthscale, kernel.function)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`kernel.function`	user defined kernel function

Details

The Multiplicative User Defined Function (UDF) kernel is given by

$$k(r) = \prod_{i=1}^{p} f(r_i^2),$$

where f is the user defined kernel function that takes r_i^2 as input, where

$$r_i(x, x') = \sqrt{\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the dimension-wise euclidean distances between x and x' weighted by the length scale parameters l_i's.

Value

A Multiplicative User Defined Function (UDF) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

UDF.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

kernel.function <- function(sqdist) {return (exp(-sqrt(sqdist)))}

# approach 1
kernel <- MultiplicativeUDF.Kernel(lengthscale, kernel.function=kernel.function)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="MultiplicativeUDF",
                     parameters=list(kernel.function=kernel.function))
Evaluate.Kernel(kernel, X)
```

`Ordinary.Kriging` *Ordinary Kriging*

Description

This functions fits the ordinary kriging model to the data.

Usage

```
Ordinary.Kriging(
  X,
  y,
  interpolation = TRUE,
  fit = TRUE,
  nugget = NULL,
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`interpolation`	interpolation whether to interpolate, for noisy data please set `interpolate=FALSE`
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation

Details

Ordinary kriging assumes a constant mean. Please see Joseph (2026) for details.

For data from deterministic computer experiments, use `interpolation=TRUE` and will give an interpolator. For noisy data, use `interpolation=FALSE`, which will give an approximator of the underlying function.

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section of Fit.Kriging for detail usages.

When the lengthscale / correlation parameters are unknown, all parameters including the constant mean can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt, a open-source library for nonlinear optimization. All gradient-free optimization methods in NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section of Fit.Kriging for detail usages.

Value

A Ordinary Kriging Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2026). *Experimental Design for Data Science and Engineering*. Chapman & Hall/CRC, doi:10.1201/9781003661535.

See Also

Fit.Kriging, Predict.Kriging, Get.Kriging.Parameters.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

# approach 1
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
```

```
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# approach 2
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

`PI.Sequential` *PI Sequential Design*

Description

This function creates a Probability of Improvement (PI) sequential design object for optimization (finding the minimum of the objective function).

Usage

```
PI.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`
: a list specifying the design space, containing:

  `design.lower.bound` a vector for the lower bound of the design space

  `design.upper.bound` a vector for the upper bound of the design space

`kriging`
: (optional if kriging.parameters provided) a fitted kriging class object (e.g., from Ordinary.Kriging or Fit.Kriging)

`kriging.parameters`
: (optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters

`nlopt.parameters`
: a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

PI selects the next design point as the one that maximizes the probability of improvement:

$$x_{next} = \arg\max_x \text{PI}(x) = \Phi(u),$$

where $u = (y_{min} - \mu(x))/\sigma(x)$, $\mu(x)$ and $\sigma(x)$ are the kriging predictive mean and standard deviation, and Φ is the standard normal CDF.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via `design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion.

Value

A PI Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1), 97-106.

See Also

EI.Sequential, UCB.Sequential, UDF.Sequential, Get.Next.Design, Add.Next.Design, Get.Best.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
```

```
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- PI.Sequential(design.parameters, kriging)

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("PI Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization and best design
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="PI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")
```

`Predict.Kriging` *Kriging Prediction*

Description

This function gives prediction and uncertainty quantification of the kriging model on a new input.

Usage

```
Predict.Kriging(kriging, X, Neumann = FALSE)
```

Arguments

`kriging`	a kriging class object
`X`	a matrix for the new input (features) to perform predictions
`Neumann`	whether to use the Neumann series approximation for prediction when the fitted model interpolates; default is `FALSE`, which uses the current fast implementation. This approximation is only recommended when the nugget is larger than 10^{-6}.

Details

If `kriging` was fitted with `scale=TRUE` in Fit.Kriging, `X` is scaled to `[0, 1]` using the training minima and maxima stored in the fitted object before prediction.

Value

`mean`	kriging mean computed at the new input
`sd`	kriging standard computed at the new input

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2006). *Limit kriging*. Technometrics, 48(4), 458-466.

Joseph, V. R. (2025). Rational Kriging. *Journal of the American Statistical Association*. 120, 548-558.

Joseph, V. R. (2026). *Experimental Design for Data Science and Engineering*. Chapman & Hall/CRC, doi:10.1201/9781003661535.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

Fit.Kriging.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="OK",
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
```

```
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
```

`Predict.Sequential` *Predict from Sequential Design*

Description

This function computes kriging predictions from the sequential design object at new input locations.

Usage

```
Predict.Sequential(sequential, newX)
```

Arguments

`sequential`	a sequential design object (e.g., from ALM.Sequential, EI.Sequential, etc.)
`newX`	a matrix of new input locations for prediction

Details

If the sequential object was created from a scaled kriging object, `newX` should be provided on the original input scale and is scaled internally before prediction.

Value

A list with components:

`mean` the predictive mean

`sd` the predictive standard deviation

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

Get.Sequential.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- EI.Sequential(design.parameters, kriging)
# get prediction and visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="EI Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
```

`Rational.Kriging` *Rational Kriging*

Description

This functions fits the rational kriging model to the data.

Usage

```
Rational.Kriging(
  X,
  y,
  fit = TRUE,
  nugget = NULL,
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation

Details

Rational kriging gives a rational predictor in terms of the inputs, which gives a more stable estimate of the mean. Please see Joseph (2025) for details. Only interpolation is available. Noisy output is not supported for rational kriging.

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section of Fit.Kriging for detail usages.

When the lengthscale / correlation parameters are unknown, all parameters including the constant mean can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt, a open-source library for nonlinear optimization. All gradient-free optimization methods in NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section of Fit.Kriging for detail usages.

Value

A Rational Kriging Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2025). Rational Kriging. *Journal of the American Statistical Association*. 120, 548-558.

See Also

Fit.Kriging, Predict.Kriging, Get.Kriging.Parameters.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

# approach 1
kriging <- Rational.Kriging(X, y, fit=TRUE, kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# approach 2
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="RK",
                       kernel.parameters=list(type="RQ",alpha=1))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

`RQ.Kernel` *Rational Quadratic (RQ) Kernel*

Description

This function specifies the Rational Quadratic (RQ) kernel.

Usage

```
RQ.Kernel(lengthscale, alpha = 1)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`alpha`	a positive scalar for the scale mixture parameter that controls the relative weighting of large-scale and small-scale variations

Details

The Rational Quadratic (RQ) kernel is given by

$$k(r;\alpha) = \left(1 + \frac{r^2}{2\alpha}\right)^{-\alpha},$$

where α is the scale mixture parameter and

$$r(x,x') = \sqrt{\sum_{i=1}^{p}\left(\frac{x_i - x_i'}{l_i}\right)^2}$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's. As $\alpha \to \infty$, it converges to the Gaussian.Kernel.

Value

A Rational Quadratic (RQ) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

MultiplicativeRQ.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

# approach 1
kernel <- RQ.Kernel(lengthscale, alpha=1)
Evaluate.Kernel(kernel, X)
```

```
# approach 2
kernel <- Get.Kernel(lengthscale, type="RQ", parameters=list(alpha=1))
Evaluate.Kernel(kernel, X)
```

`UCB.Sequential` *UCB Sequential Design*

Description

This function creates an Upper Confidence Bound (UCB) sequential design object for optimization (finding the minimum of the objective function).

Usage

```
UCB.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  lambda = 2,
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`
: a list specifying the design space, containing:

  `design.lower.bound` a vector for the lower bound of the design space

  `design.upper.bound` a vector for the upper bound of the design space

`kriging`
: (optional if kriging.parameters provided) a fitted kriging class object (e.g., from Ordinary.Kriging or Fit.Kriging)

`kriging.parameters`
: (optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters

`lambda`
: a positive scalar controlling the exploration-exploitation tradeoff (default: 2)

`nlopt.parameters`
: a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

UCB selects the next design point as the one that minimizes the lower confidence bound:

$$x_{next} = \arg\min_x \mu(x) - \lambda\sigma(x),$$

where $\mu(x)$ and $\sigma(x)$ are the kriging predictive mean and standard deviation, and λ controls the exploration-exploitation tradeoff. Larger λ encourages more exploration.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via `design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion.

Value

A UCB Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Srinivas, N., Krause, A., Kakade, S. M., & Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

See Also

EI.Sequential, PI.Sequential, UDF.Sequential, Get.Next.Design, Add.Next.Design, Get.Best.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))
sequential <- UCB.Sequential(design.parameters, kriging, lambda=2)

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("UCB Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
```

```
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}

# final visualization and best design
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="UCB Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
best.design <- Get.Best.Design(sequential)
points(best.design$design, best.design$objective, pch=16, col="green")
```

`UDF.Kernel` *User Defined Function (UDF) Kernel*

Description

This function specifies a kernel with the user defined R function.

Usage

```
UDF.Kernel(lengthscale, kernel.function)
```

Arguments

`lengthscale`	a vector for the positive length scale parameters
`kernel.function`	user defined kernel function

Details

The User Defined Function (UDF) kernel is given by

$$k(r) = f(r^2)$$

where f is the user defined kernel function that takes r^2 as input, where

$$r(x, x') = \sqrt{\sum_{i=1}^{p} \left(\frac{x_i - x_i'}{l_i} \right)^2},$$

is the euclidean distance between x and x' weighted by the length scale parameters l_i's.

Value

A User Defined Function (UDF) Kernel Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Duvenaud, D. (2014). *The kernel cookbook: Advice on covariance functions*.

Rasmussen, C. E. & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

See Also

MultiplicativeUDF.Kernel, Get.Kernel, Evaluate.Kernel.

Examples

```
n <- 5
p <- 3
X <- matrix(rnorm(n*p), ncol=p)
lengthscale <- c(1:p)

kernel.function <- function(sqdist) {return (exp(-sqrt(sqdist)))}

# approach 1
kernel <- UDF.Kernel(lengthscale, kernel.function=kernel.function)
Evaluate.Kernel(kernel, X)

# approach 2
kernel <- Get.Kernel(lengthscale, type="UDF",
                     parameters=list(kernel.function=kernel.function))
Evaluate.Kernel(kernel, X)
```

UDF.Sequential *UDF Sequential Design*

Description

This function creates a User Defined Function (UDF) sequential design object with a custom acquisition function.

Usage

```
UDF.Sequential(
  design.parameters,
  kriging = NULL,
  kriging.parameters = NULL,
  criterion.function,
  criterion.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`design.parameters`	a list specifying the design space, containing: `design.lower.bound` a vector for the lower bound of the design space `design.upper.bound` a vector for the upper bound of the design space
`kriging`	(optional if kriging.parameters provided) a fitted kriging class object (e.g., from Ordinary.Kriging or Fit.Kriging)
`kriging.parameters`	(optional if kriging provided) a list of parameters to create kriging via Fit.Kriging, must contain X, y and optionally other Fit.Kriging parameters
`criterion.function`	an R function that takes (mean, sd, X, y, criterion.parameters) and returns a scalar criterion value to be maximized
`criterion.parameters`	a list of additional parameters to pass to the criterion function
`nlopt.parameters`	a list of parameters for NLopt optimization (used when `design.candidates` is not provided)

Details

The UDF sequential design allows users to specify a custom criterion function for selecting the next design point. The criterion function receives the predictive mean, predictive standard deviation, the current training data, and any additional user-defined parameters. If the kriging object was fitted with `scale=TRUE`, the criterion function receives the current training `X` on the original input scale.

The design space is specified via `design.parameters`, which must contain `design.lower.bound` and `design.upper.bound` vectors. During optimization, candidate points can be provided via `design.candidates` in `Get.Next.Design`. If the kriging object was fitted with `scale=TRUE`, design bounds, candidates, new design points, predictions, returned designs, and returned data are supplied and returned on the original input scale; scaling is handled internally for the sequential criterion.

Value

A UDF Sequential Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

See Also

ALM.Sequential, EI.Sequential, PI.Sequential, UCB.Sequential, Predict.Sequential, Get.Next.Design, Add.Next.Design.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
n <- 10
p <- 1
X <- matrix(runif(n), ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0, 1, length=1001), ncol=p)

# create sequential
kriging <- Ordinary.Kriging(X, y, interpolation=TRUE, fit=TRUE,
                            kernel.parameters=list(type="Gaussian"))
design.parameters <- list(design.lower.bound=c(0), design.upper.bound=c(1))

# define custom criterion function (e.g., ALM-like using predictive SD)
my.criterion <- function(mean, sd, X, y, clist) { return(sd) }
sequential <- UDF.Sequential(design.parameters, kriging,
                             criterion.function=my.criterion,
                             criterion.parameters=list())

# sequential design loop with visualization
for (i in 1:5) {
  # visualize current kriging
  pred <- Predict.Sequential(sequential, newX)
  plot(newX, f(newX), "l", main=sprintf("UDF Iteration %d", i))
  points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
  lines(newX, pred$mean, col="red", lty=2)
  lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
  lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
  # get next design and evaluate
  Xn <- Get.Next.Design(sequential)
  new.design <- list(X=Xn, y=f(Xn))
  points(new.design$X, new.design$y, pch=16, col="green")
  # add to kriging and update
  sequential <- Add.Next.Design(sequential, new.design)
}
```

```
# final visualization
pred <- Predict.Sequential(sequential, newX)
plot(newX, f(newX), "l", main="UDF Final")
points(Get.Data(sequential)$X, Get.Data(sequential)$y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
```

`Universal.Kriging` *Universal Kriging*

Description

This functions fits the universal kriging model to the data.

Usage

```
Universal.Kriging(
  X,
  y,
  basis.function,
  interpolation = TRUE,
  fit = TRUE,
  nugget = NULL,
  kernel = NULL,
  kernel.parameters = list(),
  nlopt.parameters = list()
)
```

Arguments

`X`	a matrix for input (feature)
`y`	a vector for output (target), only one-dimensional output is supported
`basis.function`	the basis functions for specifying the prior mean
`interpolation`	interpolation whether to interpolate, for noisy data please set `interpolate=FALSE`
`fit`	whether to fit the length scale parameters from data
`nugget`	an optional nugget value to add to the diagonal of the correlation matrix for numerical stability or noise modeling
`kernel`	a kernel class object
`kernel.parameters`	a list of parameters required for the kernel, if no kernel class object is provided
`nlopt.parameters`	a list of parameters required for NLopt, including choice of optimization algorithm and maximum number of evaluation

Details

Universal kriging permits a more general function of mean, which can be specified using `basis.function`. Please see Joseph (2006) for details.

For data from deterministic computer experiments, use `interpolation=TRUE` and will give an interpolator. For noisy data, use `interpolation=FALSE`, which will give an approximator of the underlying function.

The kernel choices are required and can be specified by (i) providing the kernel class object to `kernel` or (ii) specifying the kernel type and other parameters in `kernel.parameters`. Please see examples section of Fit.Kriging for detail usages.

When the lengthscale / correlation parameters are unknown, all parameters including the constant mean can be estimated via Maximum Likelihood method by setting `fit=TRUE`. The initial / lower bound / upper bound of the lengthscale parameters can be provided in `kernel.parameters`, otherwise a good initial and range would be estimated from the data. The optimization is performed via NLopt, a open-source library for nonlinear optimization. All gradient-free optimization methods in NLopt are supported and can be specified in `nlopt.parameters`. See `nloptr::nloptr.print.options()` for the list of available derivative-free algorithms (prefix with NLOPT_GN or NLOPT_LN). The maximum number of optimization steps can also be defined in `nlopt.parameters`. Please see examples section of Fit.Kriging for detail usages.

Value

A Universal Kriging Class Object.

Author(s)

Chaofan Huang, Difan Song, and V. Roshan Joseph

References

Joseph, V. R. (2026). *Experimental Design for Data Science and Engineering*. Chapman & Hall/CRC, doi:10.1201/9781003661535.

See Also

Fit.Kriging, Predict.Kriging, Get.Kriging.Parameters.

Examples

```
# one dimensional example
f <- function(x) {
  x <- 0.5 + 2*x
  y <- sin(10*pi*x)/(2*x) + (x-1)^4
  return (y)
}

set.seed(1234)
# train set
n <- 30
p <- 1
```

```
X <- matrix(runif(n),ncol=p)
y <- apply(X, 1, f)
newX <- matrix(seq(0,1,length=1001), ncol=p)

basis.function <- function(x) {c(1,x[1],x[1]^2)}

# approach 1
kriging <- Universal.Kriging(X, y, basis.function=basis.function,
                             interpolation=TRUE, fit=TRUE,
                             kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)

# approach 2
kriging <- Fit.Kriging(X, y, interpolation=TRUE, fit=TRUE, model="UK",
                       model.parameters=list(basis.function=basis.function),
                       kernel.parameters=list(type="Gaussian"))
pred <- Predict.Kriging(kriging, newX)
plot(newX, f(newX), "l")
points(X, y, pch=16, col="blue")
lines(newX, pred$mean, col="red", lty=2)
lines(newX, pred$mean-2*pred$sd, col="red", lty=3)
lines(newX, pred$mean+2*pred$sd, col="red", lty=3)
Get.Kriging.Parameters(kriging)
```

Index

Add.Next.Design, 2, *5*, *8*, *25*, *28*, *47*, *56*, *60*
ALM.Sequential, *3*, 4, *20*, *25*, *27, 28*, *50*, *60*

EI.Sequential, *3*, 6, *18–20*, *25*, *27, 28*, *47*, *50*, *56*, *60*
Evaluate.Kernel, 9, *16*, *22*, *30*, *34–36*, *38, 39*, *41–43*, *54*, *58*

Fit.Kriging, *5*, *7*, 10, *17, 18*, *24*, *27, 28*, *31, 32*, *44–46*, *49*, *52*, *55*, *59*, *62*

Gaussian.Kernel, 15, *22*, *33*, *54*
Generalized.Rational.Kriging, *10*, *12*, 16
Get.Best.Design, *8*, 18, *47*, *56*
Get.Data, 20
Get.Kernel, *9*, *12*, *16*, 21, *30*, *34–36*, *38, 39*, *41–43*, *54*, *58*
Get.Kriging.Parameters, *12*, *18*, 23, *32*, *45*, *52*, *62*
Get.Next.Design, *3*, *5*, *7, 8*, 25, *28*, *47*, *56*, *59, 60*
Get.Sequential, *3*, *21*, *25*, 27, *50*

IM.Kernel, *22*, 29, *39*

Limit.Kriging, *10*, *12*, 31

Matern.Kernel, *22*, 33, *35, 36*, *38*, *41*
Matern12.Kernel, *22*, 34, *34*
Matern32.Kernel, *22*, *34*, 36
Matern52.Kernel, *22*, *34*, 37
MultiplicativeIM.Kernel, *22*, *30*, 38
MultiplicativeMatern.Kernel, *22*, *34*, 40
MultiplicativeRQ.Kernel, *22*, 41, *54*
MultiplicativeUDF.Kernel, *22*, 42, *58*

Ordinary.Kriging, *5*, *7*, *10–12*, *28*, 44, *46*, *55*, *59*

PI.Sequential, *8*, *18, 19*, *27, 28*, 46, *56*, *60*
Predict.Kriging, *10*, *12*, *18*, *32*, *45*, 48, *52*, *62*
Predict.Sequential, *5*, *28*, 50, *60*

Rational.Kriging, *10*, *12*, 51
RQ.Kernel, *22*, *42*, 53

UCB.Sequential, *8*, *18, 19*, *27, 28*, *47*, 55, *60*
UDF.Kernel, *22*, *43*, 57
UDF.Sequential, *8*, *27, 28*, *47*, *56*, 58
Universal.Kriging, *10–12*, 61