Package: KRMM (via r-universe)

August 30, 2024

Type Package

Title Kernel Ridge Mixed Model

Version 1.1

Author Laval Jacquin [aut, cre]

Maintainer Laval Jacquin <jacquin.julien@gmail.com>

Description Solves kernel ridge regression, within the the mixed model framework, for the linear, polynomial, Gaussian, Laplacian and ANOVA kernels. The model components (i.e. fixed and random effects) and variance parameters are estimated using the expectation-maximization (EM) algorithm. All the estimated components and parameters, e.g. BLUP of dual variables and BLUP of random predictor effects for the linear kernel (also known as RR-BLUP), are available. The kernel ridge mixed model (KRMM) is described in Jacquin L, Cao T-V and Ahmadi N (2016) A Unified and Comprehensible View of Parametric and Kernel Methods for Genomic Prediction with Application to Rice. Front. Genet. 7:145. <doi:10.3389/fgene.2016.00145>.

Depends R (>= 3.3.0)

Imports stats,MASS,kernlab,cvTools,robustbase,foreach,parallel,doParallel,Matrix

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 5.0.1

NeedsCompilation no

Re-packaged 2024-04-14 10:00:00 UTC; laval_jacquin

Date/Publication 2017-06-03 17:46:04 UTC

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

RemoteUrl https://github.com/ljacquin/krmm

RemoteRef HEAD

RemoteSha 9e3138f8e81403d29049ee9a4e1a7dcb98a61e28

Contents

KRMM-package *Kernel Ridge Mixed Model (KRMM) Package*

Description

The KRMM package provides tools for solving kernel ridge regression within the mixed model framework. It supports various kernels including linear, polynomial, Gaussian, Laplacian, and ANOVA. The package uses the expectation-maximization (EM) algorithm to estimate model components (fixed and random effects) and variance parameters. Estimated components and parameters such as BLUP of dual variables and BLUP of random predictor effects for the linear kernel (also known as RR-BLUP) are available.

Details

The KRMM package implements kernel ridge regression for various kernels in the mixed model framework defined by $Y = X\beta + Zu + \epsilon$, where X and Z are the design matrices of predictors with fixed and random effects, respectively. The package provides the following functions: `krmm`, `predict_krmm`, `tune_krmm`, and `em_reml_mm`.

Author(s)

Laval Jacquin

Maintainer: Laval Jacquin <jacquin.julien@gmail.com>

References

Jacquin et al. (2016). A Unified and Comprehensible View of Parametric and Kernel Methods for Genomic Prediction with Application to Rice. Front. Genet. 7:145 (in peer review)

Robinson, G. K. (1991). That blup is a good thing: the estimation of random effects. Statistical Science, 534 15-32

Foulley, J.-L. (2002). Algorithme em: théorie et application au modèle mixte. Journal de la Société française de Statistique 143, 57-109

Examples

```
# load libraries
library(KRMM)

# simulate data
set.seed(123)
p <- 500
n <- 300
gamma <- rnorm(p, mean = 0, sd = 0.5)
M <- matrix(runif(p * n, min = 0, max = 1), ncol = p, byrow = T)  # matrix of covariates
f <- tcrossprod(gamma, M)                                        # data generating process
eps <- rnorm(n, mean = 0, sd = 0.1)                                 # add residuals
Y <- f + eps                                                 # data generating process (DGP)

# split data into training and test set
n_train <- floor(n * 0.67)
idx_train <- sample(1:n, size = n_train, replace = F)

# train
M_train <- M[idx_train, ]
y_train <- Y[idx_train]

# train krmm with linear kernel (i.e. dot product)
linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RR-BLUP")

summary(linear_krmm_model)
print(linear_krmm_model$beta_hat)
hist(linear_krmm_model$gamma_hat)
hist(linear_krmm_model$vect_alpha)

# train krmm with non linear gaussian kernel (gaussian is the default kernel for RKHS method)
non_linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RKHS")

summary(non_linear_krmm_model)
print(non_linear_krmm_model$beta_hat)
hist(non_linear_krmm_model$vect_alpha)

# get test data from matrix of covariates for prediction
M_test <- M[-idx_train, ]

# get unknown true value generated by DGP we want to predict using predict_krmm
f_test <- f[-idx_train]

# -- prediction with linear kernel

# without fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression without fixed effects")
cor(f_hat_test, f_test)

# with added fixed effects
```

```
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression with fixed effects added")
cor(f_hat_test, f_test)

# -- prediction with non linear gaussian kernel

# without fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression without fixed effects,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with fixed effects added,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# -- tune krmm model with a gaussian kernel and make predictions for test data
non_linear_opt_krmm_obj <- tune_krmm(
  Y = y_train, Matrix_covariates = M_train,
  rate_decay_grid = seq(0.01, 0.1, length.out = 5), nb_folds = 3,
  method = "RKHS", kernel = "Gaussian",
)
print(non_linear_opt_krmm_obj$optimal_h)
plot(non_linear_opt_krmm_obj$rate_decay_grid,
     non_linear_opt_krmm_obj$mean_loss_grid, type = "l")

# get the optimized krmm model
non_linear_opt_krmm_model <- non_linear_opt_krmm_obj$optimized_model

# without fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = F)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)
```

`em_reml_mm` *Expectation-Maximization (EM) Algorithm for Restricted Maximum Likelihood (REML) in Mixed Models*

Description

em_reml_mm estimates the components and variance parameters of the mixed model $Y = X\beta + Zu + \epsilon$ using the EM-REML algorithm.

Usage

```
em_reml_mm(Mat_K_inv, Y, X, Z, init_sigma2K, init_sigma2E,
           convergence_precision, nb_iter, display)
```

Arguments

Mat_K_inv	numeric matrix; inverse of the kernel matrix
Y	numeric vector; response vector
X	numeric matrix; design matrix of fixed effects
Z	numeric matrix; design matrix of random effects
init_sigma2K, init_sigma2E	numeric scalars; initial guess values for variance parameters in the EM-REML algorithm
convergence_precision, nb_iter	numeric scalars; convergence precision and maximum iterations for the EM-REML algorithm
display	boolean; display estimated components at each iteration

Value

beta_hat	estimated fixed effect(s)
u_hat	estimated random effect(s)
sigma2K_hat, sigma2E_hat	estimated variance components

Author(s)

Laval Jacquin <jacquin.julien@gmail.com>

References

Foulley, J.-L. (2002). Algorithme em: théorie et application au modèle mixte. Journal de la Société française de Statistique 143, 57-109

krmm *Kernel ridge regression in the mixed model framework*

Description

krmm solves kernel ridge regression for various kernels within the mixed model framework: $Y = X\beta + Zu + \epsilon$, where X and Z are design matrices of predictors with fixed and random effects, respectively.

Usage

```
krmm(Y, X = rep(1, length(Y)), Z = diag(1, length(Y)),
     Matrix_covariates, method = "RKHS", kernel = "Gaussian",
     center_covariates = T, scale_covariates = F,
     rate_decay_kernel = 0.1, degree_poly = 2, scale_poly = 1,
     offset_poly = 1, degree_anova = 3,
     init_sigma2K = 2, init_sigma2E = 3, convergence_precision = 1e-08,
     nb_iter = 1000, display = FALSE)
```

Arguments

Y	numeric vector; response vector for training data
X	numeric matrix; design matrix of predictors with fixed effects for training data (default is a vector of ones)
Z	numeric matrix; design matrix of predictors with random effects for training data (default is identity matrix)
Matrix_covariates	numeric matrix; entries for training data used to build the kernel matrix
center_covariates	boolean; center the matrix of covariates
scale_covariates	boolean; scale the matrix of covariates
method	character string; RKHS, GBLUP, or RR-BLUP
kernel	character string; "Gaussian", "Laplacian" or "ANOVA"" (kernels for RKHS regression ONLY, linear kernel for GBLUP and RR-BLUP)
rate_decay_kernel	numeric scalar; hyperparameter of the kernel (default is 0.1)
degree_poly, scale_poly, offset_poly	numeric scalars; parameters for polynomial kernel (defaults are 2, 1, and 1)
degree_anova	numeric scalar; parameter for ANOVA kernel (default is 3)
init_sigma2K, init_sigma2E	numeric scalars; initial guess values for the EM-REML algorithm (defaults are 2 and 3)

`convergence_precision`, `nb_iter`
: numeric scalars; convergence precision and maximum iterations for the EM-REML algorithm (defaults are 1e-8 and 1000)

`display`
: boolean; display estimated components at each iteration

Details

The matrix `Matrix_covariates` is mandatory to build the kernel matrix for model estimation and prediction.

Value

`beta_hat`
: estimated fixed effect(s)

`sigma2K_hat`, `sigma2E_hat`
: estimated variance components

`vect_alpha`
: estimated dual variables

`gamma_hat`
: RR-BLUP of covariate effects (available for RR-BLUP method only)

Author(s)

Laval Jacquin

Maintainer: Laval Jacquin <jacquin.julien@gmail.com>

References

Jacquin et al. (2016); Robinson (1991); Foulley (2002)

Examples

```
# load libraries
library(KRMM)

# simulate data
set.seed(123)
p <- 500
n <- 300
gamma <- rnorm(p, mean = 0, sd = 0.5)
M <- matrix(runif(p * n, min = 0, max = 1), ncol = p, byrow = T)  # matrix of covariates
f <- tcrossprod(gamma, M)                                        # data generating process
eps <- rnorm(n, mean = 0, sd = 0.1)                                 # add residuals
Y <- f + eps                                              # data generating process (DGP)

# split data into training and test set
n_train <- floor(n * 0.67)
idx_train <- sample(1:n, size = n_train, replace = F)

# train
M_train <- M[idx_train, ]
y_train <- Y[idx_train]
```

```
# train krmm with linear kernel (i.e. dot product)
linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RR-BLUP")

summary(linear_krmm_model)
print(linear_krmm_model$beta_hat)
hist(linear_krmm_model$gamma_hat)
hist(linear_krmm_model$vect_alpha)

# train krmm with non linear gaussian kernel (gaussian is the default kernel for RKHS method)
non_linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RKHS")

summary(non_linear_krmm_model)
print(non_linear_krmm_model$beta_hat)
hist(non_linear_krmm_model$vect_alpha)

# get test data from matrix of covariates for prediction
M_test <- M[-idx_train, ]

# get unknown true value generated by DGP we want to predict using predict_krmm
f_test <- f[-idx_train]

# -- prediction with linear kernel

# without fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression without fixed effects")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression with fixed effects added")
cor(f_hat_test, f_test)

# -- prediction with non linear gaussian kernel

# without fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression without fixed effects,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with fixed effects added,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# -- tune krmm model with a gaussian kernel and make predictions for test data
non_linear_opt_krmm_obj <- tune_krmm(
```

```
  Y = y_train, Matrix_covariates = M_train,
  rate_decay_grid = seq(0.01, 0.1, length.out = 5), nb_folds = 3,
  method = "RKHS", kernel = "Gaussian",
)
print(non_linear_opt_krmm_obj$optimal_h)
plot(non_linear_opt_krmm_obj$rate_decay_grid,
     non_linear_opt_krmm_obj$mean_loss_grid, type = "l")

# get the optimized krmm model
non_linear_opt_krmm_model <- non_linear_opt_krmm_obj$optimized_model

# without fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = F)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)
```

`predict_krmm` *Predict Function for KRMM Object*

Description

The `predict_krmm` function computes predicted values for a given vector or design matrix of covariates using a `krmm` model object.

Usage

```
predict_krmm(krmm_model, Matrix_covariates,
             X = rep(1, nrow(Matrix_covariates)),
             Z = diag(1, nrow(Matrix_covariates)), add_fixed_effects = FALSE)
```

Arguments

`krmm_model`	a `krmm` object
`Matrix_covariates`	numeric matrix; design matrix of covariates for target data
`X`	numeric matrix; design matrix of predictors with fixed effects for target data (default is a vector of ones)
`Z`	numeric matrix; design matrix of predictors with random effects for target data (default is identity matrix)
`add_fixed_effects`	logical; should fixed effects be added to the prediction (default is `FALSE`)

Details

The `Matrix_covariates` argument is mandatory for building the kernel matrix required for prediction, using the covariates from the `krmm_model`.

Value

`f_hat` or `u_hat` only

Predicted values for target data, calculated as $\hat{f} = X\hat{\beta} + Z\hat{u}$.

Author(s)

Laval Jacquin

Maintainer: Laval Jacquin <jacquin.julien@gmail.com>

Examples

```
# load libraries
library(KRMM)

# simulate data
set.seed(123)
p <- 500
n <- 300
gamma <- rnorm(p, mean = 0, sd = 0.5)
M <- matrix(runif(p * n, min = 0, max = 1), ncol = p, byrow = T)  # matrix of covariates
f <- tcrossprod(gamma, M)                                        # data generating process
eps <- rnorm(n, mean = 0, sd = 0.1)                                 # add residuals
Y <- f + eps                                                 # data generating process (DGP)

# split data into training and test set
n_train <- floor(n * 0.67)
idx_train <- sample(1:n, size = n_train, replace = F)

# train
M_train <- M[idx_train, ]
y_train <- Y[idx_train]

# train krmm with linear kernel (i.e. dot product)
linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RR-BLUP")

summary(linear_krmm_model)
print(linear_krmm_model$beta_hat)
hist(linear_krmm_model$gamma_hat)
hist(linear_krmm_model$vect_alpha)

# train krmm with non linear gaussian kernel (gaussian is the default kernel for RKHS method)
non_linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RKHS")

summary(non_linear_krmm_model)
print(non_linear_krmm_model$beta_hat)
hist(non_linear_krmm_model$vect_alpha)
```

```
# get test data from matrix of covariates for prediction
M_test <- M[-idx_train, ]

# get unknown true value generated by DGP we want to predict using predict_krmm
f_test <- f[-idx_train]

# -- prediction with linear kernel

# without fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression without fixed effects")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression with fixed effects added")
cor(f_hat_test, f_test)

# -- prediction with non linear gaussian kernel

# without fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression without fixed effects,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with fixed effects added,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# -- tune krmm model with a gaussian kernel and make predictions for test data
non_linear_opt_krmm_obj <- tune_krmm(
  Y = y_train, Matrix_covariates = M_train,
  rate_decay_grid = seq(0.01, 0.1, length.out = 5), nb_folds = 3,
  method = "RKHS", kernel = "Gaussian",
)
print(non_linear_opt_krmm_obj$optimal_h)
plot(non_linear_opt_krmm_obj$rate_decay_grid,
     non_linear_opt_krmm_obj$mean_loss_grid, type = "l")

# get the optimized krmm model
non_linear_opt_krmm_model <- non_linear_opt_krmm_obj$optimized_model

# without fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = F)
dev.new()
```

```
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)
```

`tune_krmm` *Tune Kernel Ridge Regression in the Mixed Model Framework*

Description

The `tune_krmm` function tunes the rate of decay parameter of kernels for kernel ridge regression using K-folds cross-validation.

Usage

```
tune_krmm(Y, X = rep(1, length(Y)), Z = diag(1, length(Y)),
       Matrix_covariates, method = "RKHS", kernel = "Gaussian", rate_decay_kernel = 0.1,
           degree_poly = 2, scale_poly = 1, offset_poly = 1, degree_anova = 3,
            init_sigma2K = 2, init_sigma2E = 3, convergence_precision = 1e-8,
            nb_iter = 1000, display = FALSE,
       rate_decay_grid = seq(0.1, 1.0, length.out = 10), nb_folds = 5, loss = "mse")
```

Arguments

`Y`	numeric vector; response vector
`X`	numeric matrix; design matrix of predictors with fixed effects (default is a vector of ones)
`Z`	numeric matrix; design matrix of predictors with random effects (default is identity matrix)
`Matrix_covariates`	numeric matrix; entries used to build the kernel matrix
`method`	character string; RKHS, GBLUP, or RR-BLUP
`kernel`	character string; Gaussian, Laplacian, or ANOVA
`rate_decay_grid`	grid over which the rate of decay is tuned by K-folds cross-validation
`nb_folds`	number of folds for cross-validation (default is 5)
`loss`	loss function to optimize; "mse" for mean square error or "cor" for correlation (default is "mse")
`rate_decay_kernel`	numeric scalar; hyperparameter of the kernel (default is 0.1)

`degree_poly`, `scale_poly`, `offset_poly`
: numeric scalars; parameters for polynomial kernel (defaults are 2, 1, and 1)

`degree_anova`
: numeric scalar; parameter for ANOVA kernel (default is 3)

`init_sigma2K`, `init_sigma2E`
: numeric scalars; initial guess values for variance parameters in the EM-REML algorithm

`convergence_precision`, `nb_iter`
: numeric scalars; convergence precision and maximum iterations for the EM-REML algorithm

`display`
: boolean; display estimated components at each iteration

Value

`optimized_model`
: the tuned model (a `krmm` object)

`mean_loss_grid`
: average loss for each rate of decay tested over the grid

`optimal_h`
: rate of decay minimizing the average loss

Author(s)

Laval Jacquin

Maintainer: Laval Jacquin <jacquin.julien@gmail.com>

Examples

```
# load libraries
library(KRMM)

# simulate data
set.seed(123)
p <- 500
n <- 300
gamma <- rnorm(p, mean = 0, sd = 0.5)
M <- matrix(runif(p * n, min = 0, max = 1), ncol = p, byrow = T)  # matrix of covariates
f <- tcrossprod(gamma, M)                                        # data generating process
eps <- rnorm(n, mean = 0, sd = 0.1)                                 # add residuals
Y <- f + eps                                              # data generating process (DGP)

# split data into training and test set
n_train <- floor(n * 0.67)
idx_train <- sample(1:n, size = n_train, replace = F)

# train
M_train <- M[idx_train, ]
y_train <- Y[idx_train]

# train krmm with linear kernel (i.e. dot product)
linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RR-BLUP")

summary(linear_krmm_model)
```

```
print(linear_krmm_model$beta_hat)
hist(linear_krmm_model$gamma_hat)
hist(linear_krmm_model$vect_alpha)

# train krmm with non linear gaussian kernel (gaussian is the default kernel for RKHS method)
non_linear_krmm_model <- krmm(Y = y_train, Matrix_covariates = M_train, method = "RKHS")

summary(non_linear_krmm_model)
print(non_linear_krmm_model$beta_hat)
hist(non_linear_krmm_model$vect_alpha)

# get test data from matrix of covariates for prediction
M_test <- M[-idx_train, ]

# get unknown true value generated by DGP we want to predict using predict_krmm
f_test <- f[-idx_train]

# -- prediction with linear kernel

# without fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression without fixed effects")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Linear RKHS regression with fixed effects added")
cor(f_hat_test, f_test)

# -- prediction with non linear gaussian kernel

# without fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression without fixed effects,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with fixed effects added,
     and default rate of decay (not optimized)")
cor(f_hat_test, f_test)

# -- tune krmm model with a gaussian kernel and make predictions for test data
non_linear_opt_krmm_obj <- tune_krmm(
  Y = y_train, Matrix_covariates = M_train,
  rate_decay_grid = seq(0.01, 0.1, length.out = 5), nb_folds = 3,
  method = "RKHS", kernel = "Gaussian",
)
```

```
print(non_linear_opt_krmm_obj$optimal_h)
plot(non_linear_opt_krmm_obj$rate_decay_grid,
     non_linear_opt_krmm_obj$mean_loss_grid, type = "l")

# get the optimized krmm model
non_linear_opt_krmm_model <- non_linear_opt_krmm_obj$optimized_model

# without fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = F)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)

# with added fixed effects
f_hat_test <- predict_krmm(non_linear_opt_krmm_model, Matrix_covariates = M_test, add_flxed_effects = T)
dev.new()
plot(f_hat_test, f_test, main = "Gaussian RKHS regression with optimized rate of decay")
cor(f_hat_test, f_test)
```

Index