Lab 4: Gaussian Processes

xiali125@student.liu.se linfr259@student.liu.se qinzh916@student.liu.se
huali824@student.liu.se qincu578@student.liu.se

2025-10-12

In this lab, we implement Gaussian Process (GP) regression from scratch, explore GP regression with kernlab, and fit a GP classifier. Our goals are to reproduce Algorithm 2.1 from Rasmussen & Williams using numerically stable Cholesky solves, visualize posterior means and uncertainty, compare kernels/hyperparameters, and evaluate GP classification performance on a real dataset.

1. Implementing GP Regression

We consider the standard GP regression model:

$$y = f(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_n^2), \quad f \sim \mathcal{GP}(0, k(x, x'))$$

We implement Algorithm 2.1 (R&W, p. 19) using Cholesky decomposition for numerical stability. In R, chol() returns an upper triangular factor, so we transpose it to obtain the lower triangular matrix used in the book.

1.1. Posterior function (posteriorGP)

We use the squared exponential (SE) kernel and return posterior mean and variance at inputs X.

```r
SquaredExpKernel <- function(x1, x2, sigmaF = 1, l = 1) {
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA_real_, n1, n2)
  for (i in seq_len(n2)) {
    K[, i] <- sigmaF^2 * exp(-0.5 * ((x1 - x2[i]) / l)^2)
  }
  K
}

posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...) {
  # Covariance blocks
  K    <- k(X, X, ...)
  Kstar<- k(X, XStar, ...)
  Kss  <- k(XStar, XStar, ...)

  # Add noise to training covariance
  L <- chol(K + sigmaNoise^2 * diag(nrow(K)))
  L <- t(L)  # convert to lower triangular

  # Algorithm 2.1
  alpha <- solve(t(L), solve(L, y))
```

```
  post_mean <- t(Kstar) %*% alpha

  v <- solve(L, Kstar)
  post_cov  <- Kss - t(v) %*% v

  list(mean = as.vector(post_mean), var = post_cov)
}
```

1.2. Single observation update

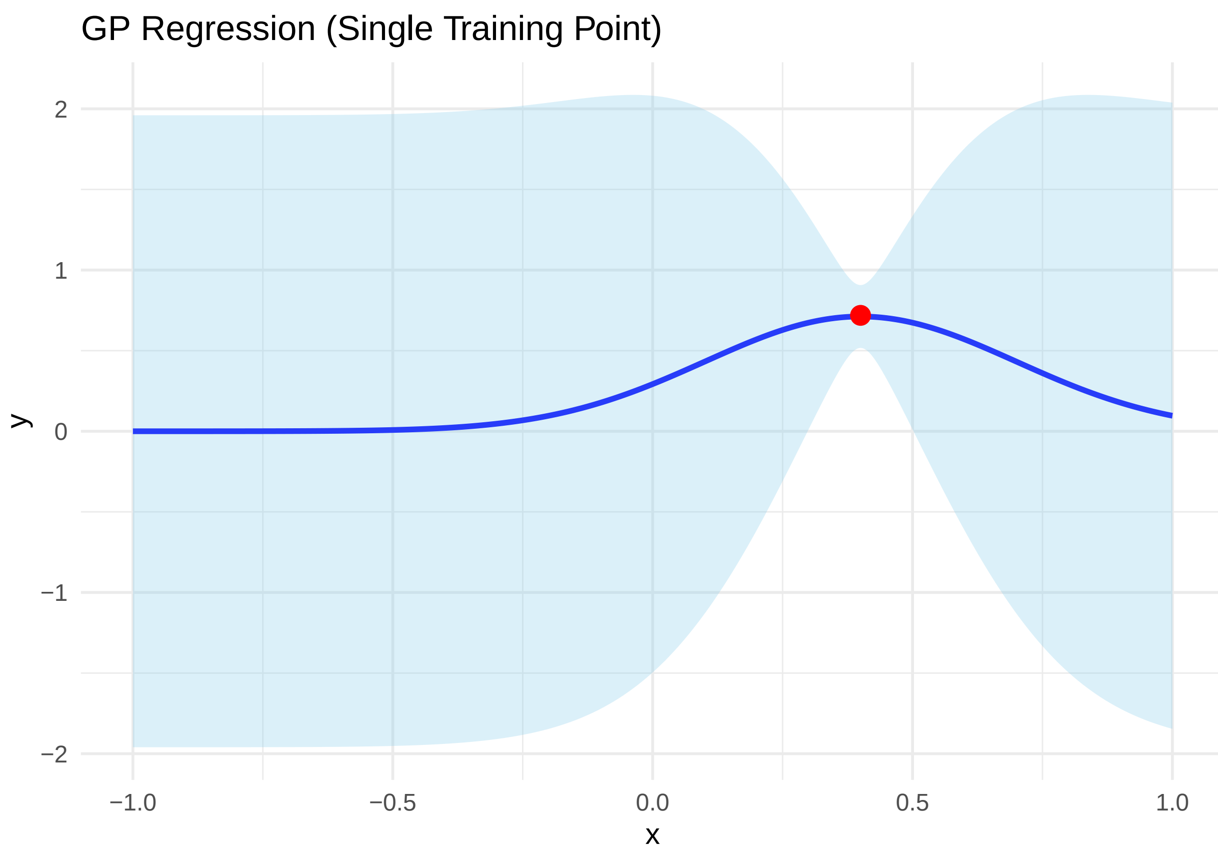
We set $\sigma_f = 1$ and $\ell = 0.3$, observe $(x, y) = (0.4, 0.719)$ with $\sigma_n = 0.1$ and plot the posterior over $x \in [-1, 1]$. We also plot 95% probability (pointwise) bands for f.

```
sigmaF <- 1
l <- 0.3
library(tibble)
library(ggplot2)

x_values <- seq(-1, 1, length.out = 5000)
mean_pred  <- posteriorGP(0.4, 0.719, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred <- posteriorGP(0.4, 0.719, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data <- tibble(
  x_values = x_values,
  y_values = mean_pred,
  sd = sqrt(diag(var_pred)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = 0.4, y = 0.719, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Single Training Point)", x = "x", y = "y")
```



1.3. Posterior Update: Two Observations

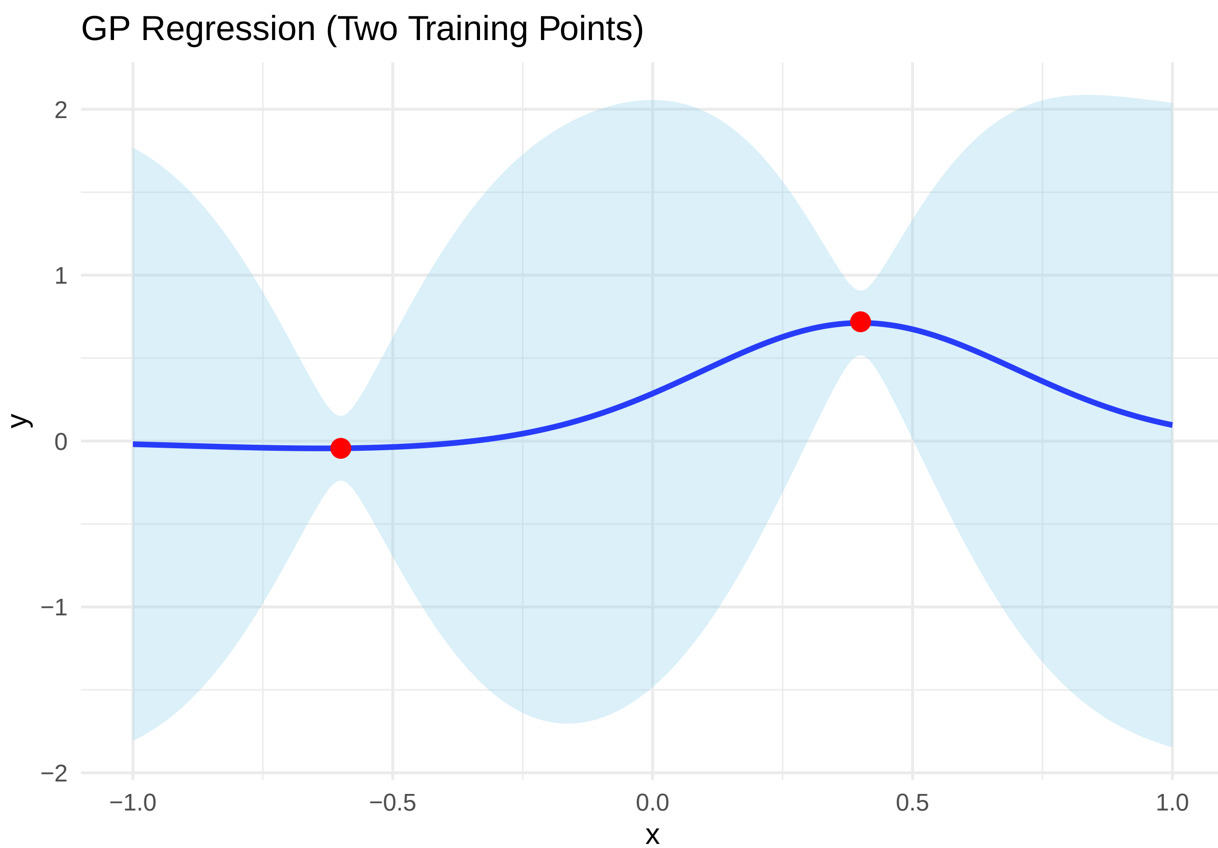
Adding a second observation $(x, y) = (-0.6, -0.044)$ yields the following posterior:

```
obs_x <- c(0.4, -0.6)
obs_y <- c(0.719, -0.044)

mean_pred2  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred2 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data2 <- tibble(
  x_values = x_values,
  y_values = mean_pred2,
  sd = sqrt(diag(var_pred2)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data2, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Two Training Points)", x = "x", y = "y")
```



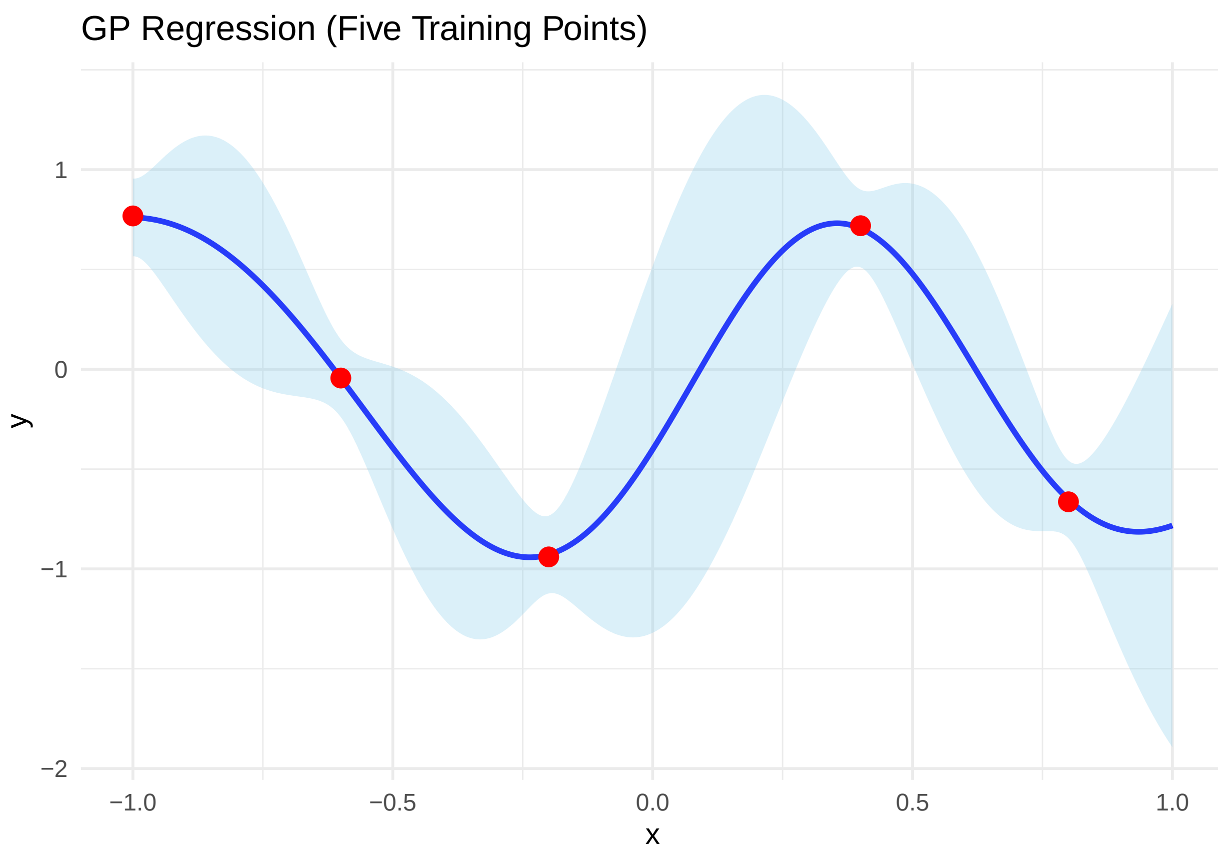
1.4. Posterior with Full Dataset

We now use all five data points provided:

```
obs_x <- c(-1, -0.6, -0.2, 0.4, 0.8)
obs_y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
mean_pred3  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred3 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data3 <- tibble(
  x_values = x_values,
  y_values = mean_pred3,
  sd = sqrt(diag(var_pred3)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data3, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Five Training Points)", x = "x", y = "y")
```



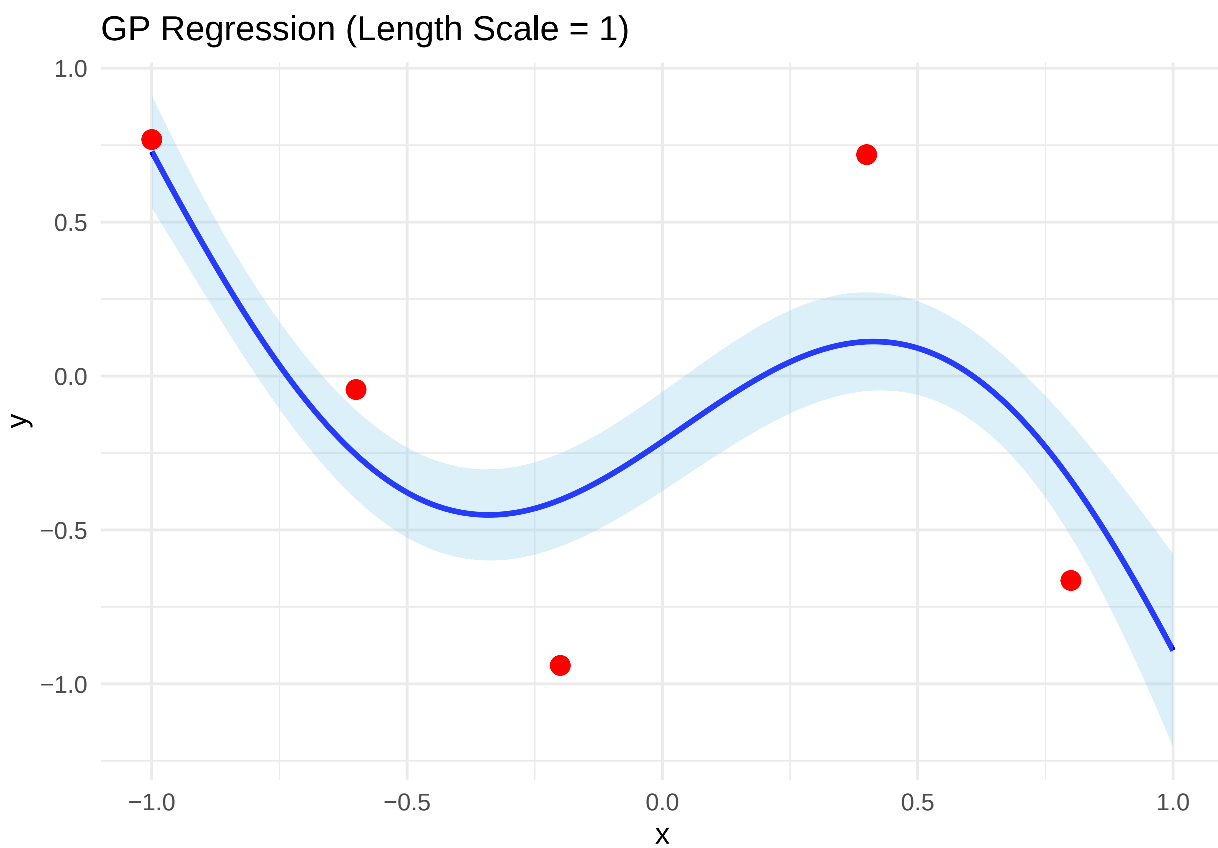
1.5. Hyperparameter Comparison

We now increase the length-scale parameter to $\ell = 1$ and compare the effect.

```
l <- 1
mean_pred4  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred4 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data4 <- tibble(
  x_values = x_values,
  y_values = mean_pred4,
  sd = sqrt(diag(var_pred4)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data4, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Length Scale = 1)", x = "x", y = "y")
```



With a longer length-scale, the GP becomes smoother and less sensitive to small fluctuations in the data. The posterior mean captures broader trends, while the uncertainty bands become wider, indicating a more generalized fit.

2. Gaussian Process Regression with kernlab

In this section we apply Gaussian Process (GP) regression to daily mean temperatures in Stockholm (Tullinge) from 2010-01-01 to 2015-12-31 (with 2012-02-29 removed). To reduce runtime, we analyze every 5th observation. We first review kernlab's GP functions, then fit models using time and day as inputs, and finally extend the kernel to a locally periodic form:

```
rm(list = ls())

library(kernlab)

# Read data (daily mean temperature, leap day removed)
temp <- read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv",
  header = TRUE, sep = ";"
)

# Full series and 1-in-5 subsample
temp_all <- temp$temp
time <- seq(1, 2186, by = 5)           # time = 1, 6, 11, ..., 2186
temp_sub <- temp_all[time]
```

```r
# Day-of-year repeating sequence for the subsample
day <- rep(seq(1, 361, by = 5), 6)       # day = 1, 6, ..., 361 repeated per year
stopifnot(length(day) == length(temp_sub))
```

2.1. Familiarization and kernel definition

We reviewed ?gausspr and ?kernelMatrix to understand inputs/outputs, and studied the course file KernLabDemo.R.

Below, we define a squared exponential (SE) kernel with hyperparameters ℓ and σ_f). We evaluate it at $x = 1, x' = 2$, and compute $K(X, X^*)$ for

$$X = (1, 3, 4)^T, \quad X^* = (2, 3, 4)^T.$$

```r
# Squared Exponential kernel for kernlab (scalar inputs)
sekernel <- function(sigmaf = 1, ell = 1) {
  rval <- function(x, y = NULL) {
    r2 <- as.numeric(crossprod(x - y))  # (x - y)^2 for 1D inputs
    sigmaf^2 * exp(-0.5 * r2 / ell^2)
  }
  class(rval) <- "kernel"
  rval
}

# Evaluate k(1, 2) and K(X, X*)
SEFunc <- sekernel(sigmaf = 1, ell = 1)
SEFunc(1, 2)
```

```
## [1] 0.6065307
```

```r
X     <- matrix(c(1, 3, 4), ncol = 1)
Xstar <- matrix(c(2, 3, 4), ncol = 1)
kernelMatrix(kernel = SEFunc, x = X, y = Xstar)
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

2.2. GP model with time as input

Model:

$$temp = f(time) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_n^2), \quad f \sim \mathcal{GP}(0, k(time, time')).$$

We set σ_n^2 to the residual variance from a quadratic regression and fit a GP with SE kernel ($\sigma_f = 20$ and $\ell = 100$). We plot the posterior mean and 95% pointwise bands.

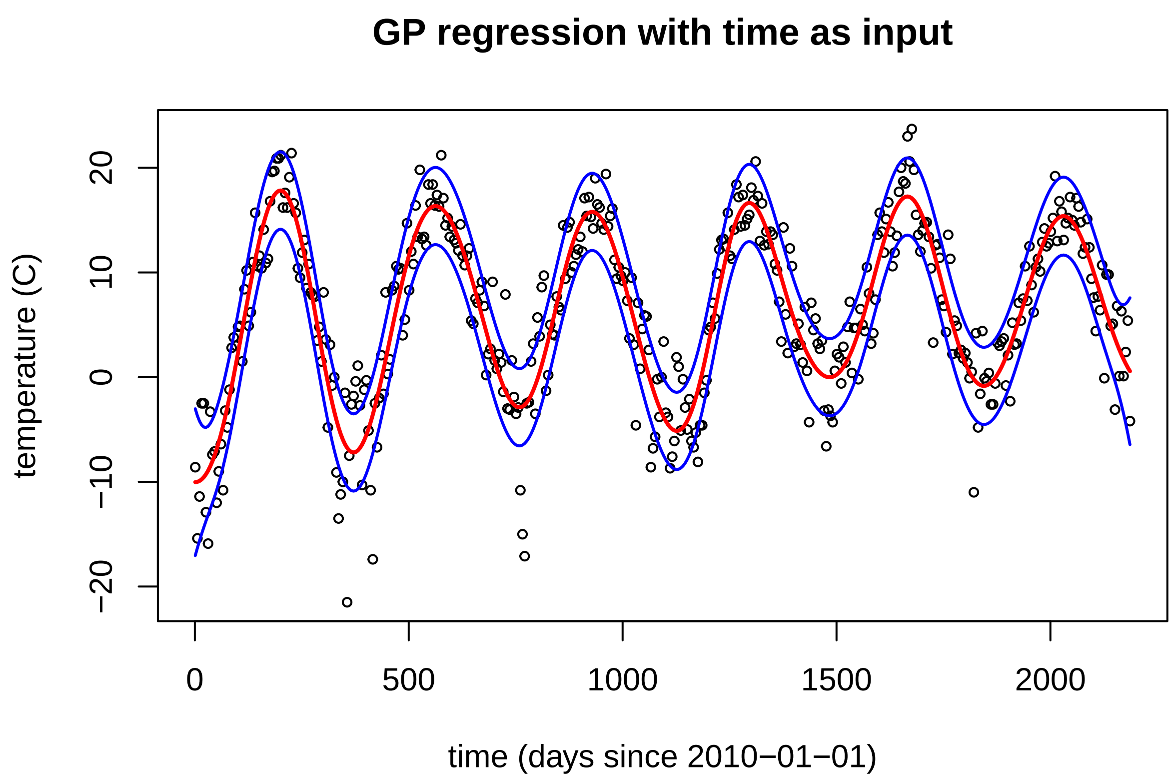
```r
# Estimate noise SD via quadratic fit
polyFit <- lm(temp_sub ~ time + I(time^2))
sigmaNoise <- sd(residuals(polyFit))

# Fit GP with time as input
SE_time <- sekernel(sigmaf = 20, ell = 100)
```

```
GP_time <- gausspr(time, temp_sub,
                   kernel = SE_time, var = sigmaNoise^2,
                   variance.model = TRUE, scaled = FALSE)

# Predictions and 95% bands
mean_time <- predict(GP_time, time)
sd_time   <- predict(GP_time, time, type = "sdeviation")

plot(time, temp_sub, pch = 1, cex = 0.6, xlab = "time (days since 2010-01-01)",
     ylab = "temperature (C)", main = "GP regression with time as input")
lines(time, mean_time, col = "red", lwd = 2)
lines(time, mean_time + 1.96 * sd_time, col = "blue", lwd = 1.5)
lines(time, mean_time - 1.96 * sd_time, col = "blue", lwd = 1.5)
```



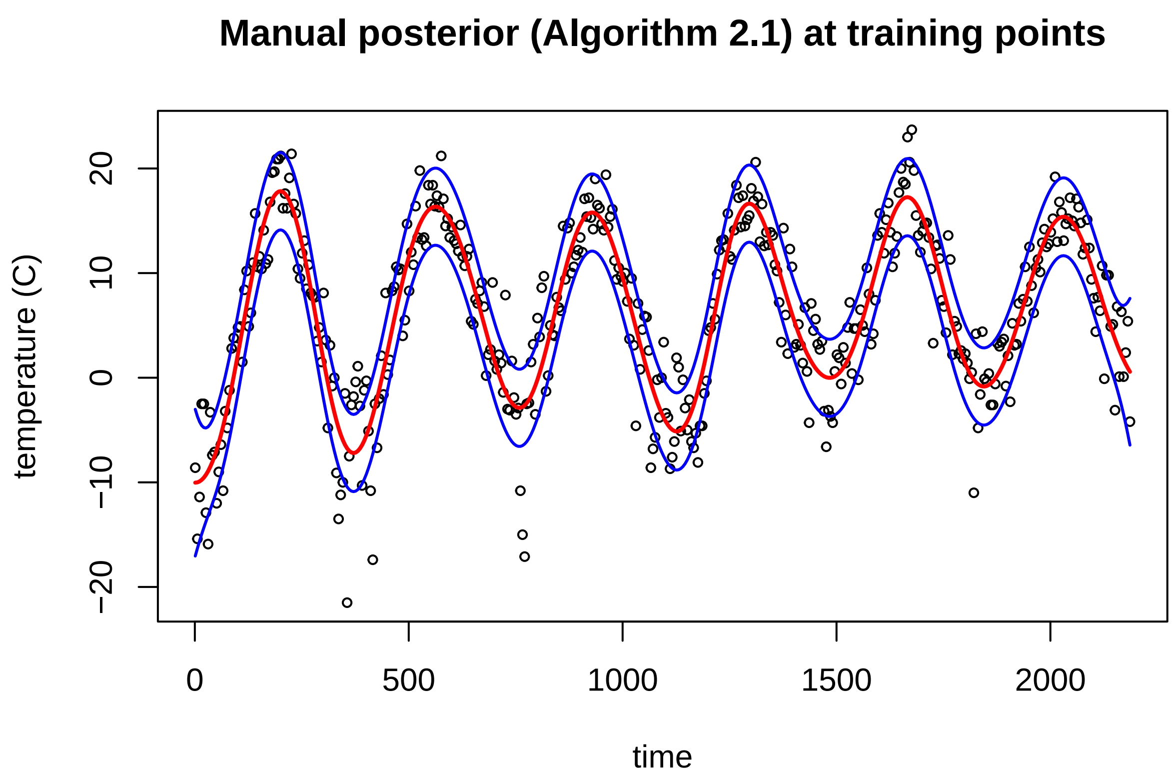
2.3. Manual posterior via Algorithm 2.1

We now reproduce the posterior manually using Algorithm 2.1 (Rasmussen & Williams, 2006).

```
# Covariance matrices at training locations
Kxx <- kernelMatrix(kernel = SE_time, x = matrix(time, ncol = 1))
n   <- length(time)

# Posterior mean and covariance at training points
Mean_manual <- t(Kxx) %*% solve(Kxx + sigmaNoise^2 * diag(n), temp_sub)
Cov_manual  <- Kxx - t(Kxx) %*% solve(Kxx + sigmaNoise^2 * diag(n), Kxx)
```

```r
# Plot
plot(time, temp_sub, pch = 1, cex = 0.6, xlab = "time", ylab = "temperature (C)",
     main = "Manual posterior (Algorithm 2.1) at training points")
lines(time, as.numeric(Mean_manual), col = "red", lwd = 2)
lines(time, as.numeric(Mean_manual) - 1.96 * sqrt(diag(Cov_manual)),
      col = "blue", lwd = 1.5)
lines(time, as.numeric(Mean_manual) + 1.96 * sqrt(diag(Cov_manual)),
      col = "blue", lwd = 1.5)
```

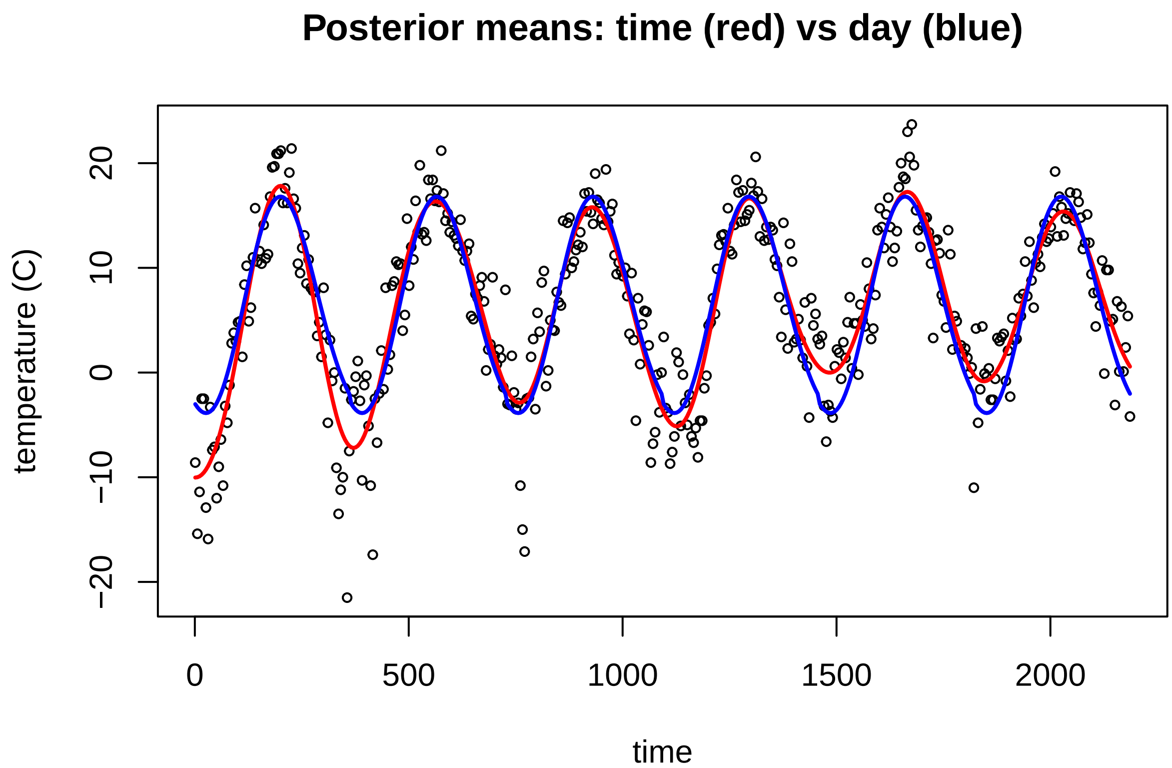


2.4. GP model with day as input

We use day-of-year as the covariate. We overlay the mean curves from both models to compare.

```r
GP_day <- gausspr(day, temp_sub,
                  kernel = SE_time, var = sigmaNoise^2,
                  variance.model = TRUE, scaled = FALSE)
mean_day <- predict(GP_day, day)

plot(time, temp_sub, pch = 1, cex = 0.6,
     xlab = "time", ylab = "temperature (C)",
     main = "Posterior means: time (red) vs day (blue)")
lines(time, mean_time, col = "red", lwd = 2)
lines(time, mean_day,  col = "blue", lwd = 2)
```



The day model emphasizes the annual cycle (periodicity) and smooth seasonal structure, whereas the time model is more responsive to non-seasonal variations and inter-annual changes. Choice depends on whether the objective is to capture seasonality (day) or long-term dynamics and anomalies (time).

2.5. Locally periodic kernel extension

We extend the SE kernel with a periodic component (period d = 365):

$$k(x,x') = \sigma_f^2 \exp\left(-2\sin^2\left(\frac{\pi|x-x'|}{d}\right)/\ell_1^2\right)\exp\left(-\frac{1}{2}\frac{|x-x'|^2}{\ell_2^2}\right).$$

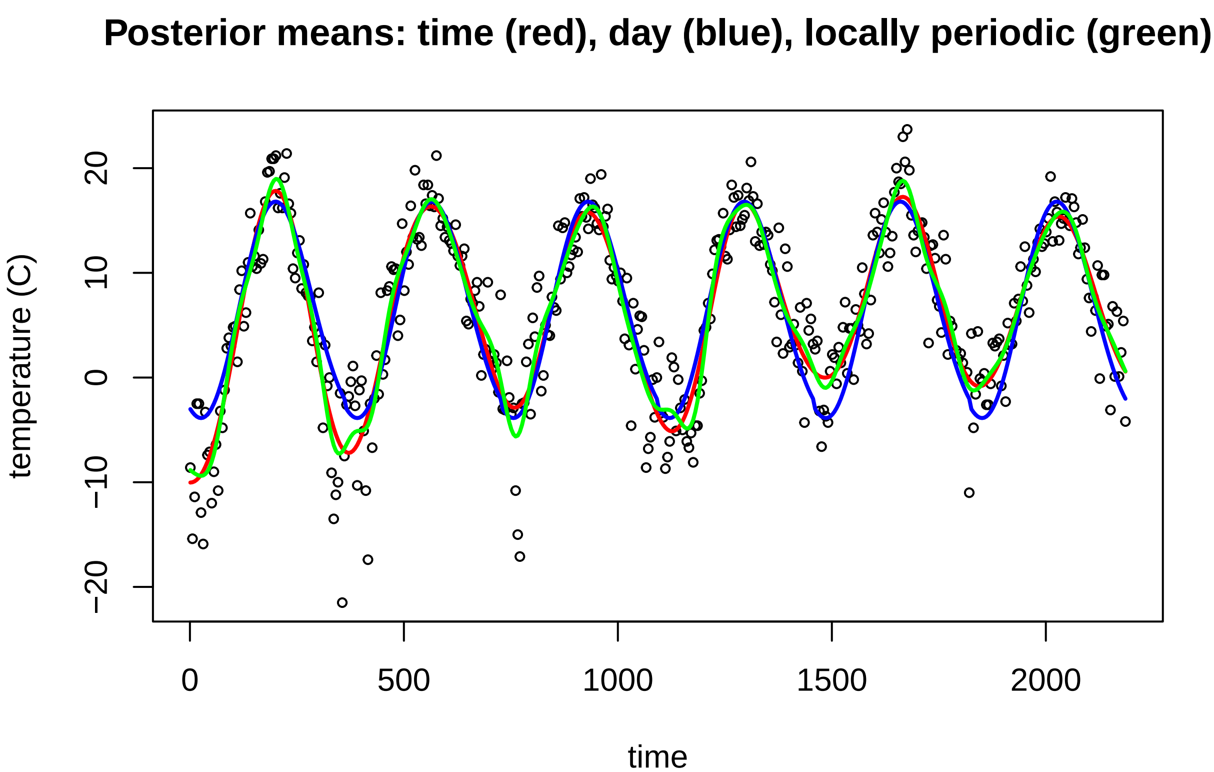
Here, ℓ_1 controls within-year similarity and ℓ_2 controls across-year decay.

```
# Locally periodic kernel (periodic x SE)
lp_kernel <- function(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365) {
  rval <- function(x, y = NULL) {
    r  <- sqrt(as.numeric(crossprod(x - y)))
    per <- exp(-2 * (sin(pi * r / d))^2 / ell1^2)
    se  <- exp(-0.5 * r^2 / ell2^2)
    sigmaf^2 * per * se
  }
  class(rval) <- "kernel"
  rval
}

LP_time <- lp_kernel(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365)
```

```r
GP_lp   <- gausspr(time, temp_sub,
                   kernel = LP_time, var = sigmaNoise^2,
                   variance.model = TRUE, scaled = FALSE)
mean_lp <- predict(GP_lp, time)

plot(time, temp_sub, pch = 1, cex = 0.6,
     xlab = "time", ylab = "temperature (C)",
     main = "Posterior means: time (red), day (blue), locally periodic (green)")
lines(time, mean_time, col = "red",   lwd = 2)
lines(time, mean_day,  col = "blue",  lwd = 2)
lines(time, mean_lp,   col = "green", lwd = 2)
```



The locally periodic GP fuses the strengths of both earlier models: it tracks seasonal cycles while allowing inter-annual variation and local deviations. With $\ell_1 = 1$ and $\ell_2 = 100$ the fit captures fine daily correspondence within a year and gradual changes across years, offering the most faithful representation for periodic climate signals with evolving baselines.

3. GP Classification with kernlab

We now switch to Gaussian Process Classification (GPC) using the banknote fraud dataset. Our goals are: fit a 2-feature GPC with default settings, visualize decision probabilities, evaluate performance in-sample and on a held-out test set, and compare against a 4-feature model.

```r
set.seed(111)

data <- read.csv(
```

```
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.csv",
  header = FALSE, sep = ","
)
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
data$fraud  <- as.factor(data$fraud)

SelectTraining <- sample(seq_len(nrow(data)), size = 1000, replace = FALSE)
train <- data[SelectTraining, ]
test  <- data[-SelectTraining, ]
```

3.1. GPC with two covariates

We fit gausspr with default kernel/hyperparameters using varWave and skewWave only. We also contour P(fraud = 1) over a grid, overlaying class-labeled training points.

```
library(kernlab)

GP_cls_2 <- gausspr(fraud ~ varWave + skewWave, data = train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
# In-sample predictions and accuracy
pred_train <- predict(GP_cls_2, train)
acc <- function(tab) sum(diag(tab)) / sum(tab)
(train_tab <- table(pred_train, train$fraud))
```

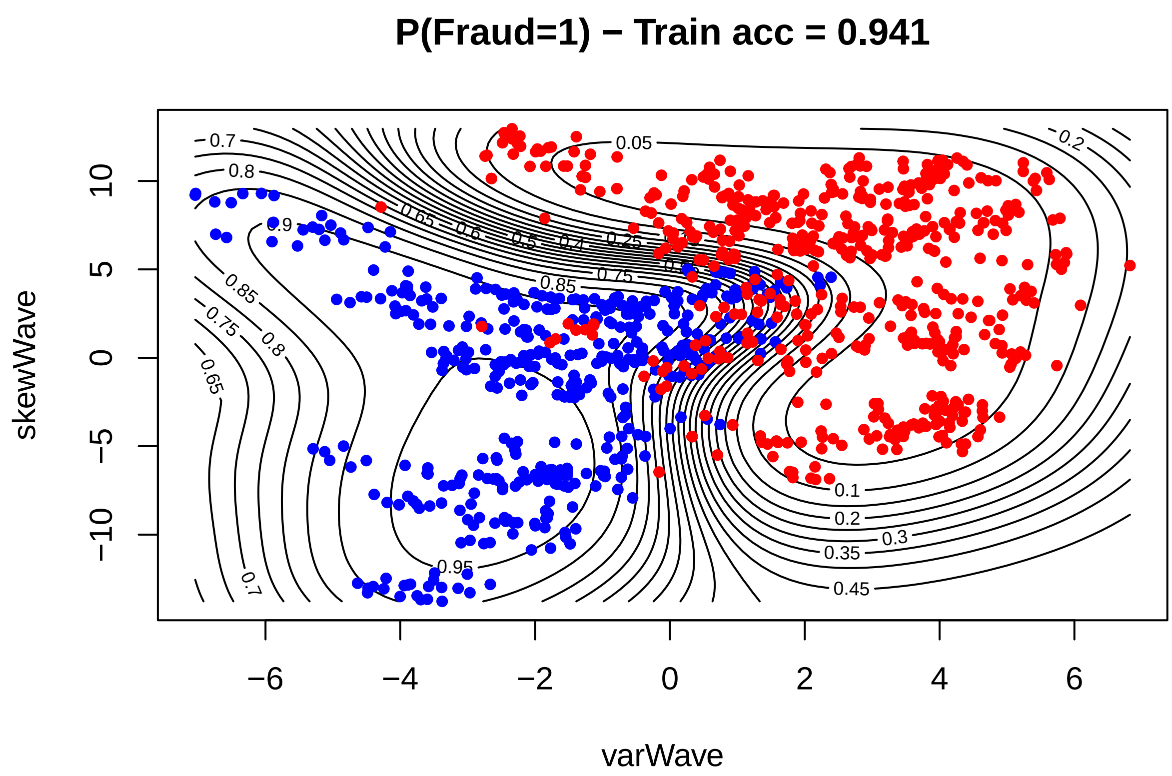
```
##
## pred_train   0   1
##          0 503  18
##          1  41 438
```

```
acc_train <- acc(train_tab)

# Probability grid for contour plot
x1 <- seq(min(data$varWave), max(data$varWave), length.out = 100)
x2 <- seq(min(data$skewWave), max(data$skewWave), length.out = 100)

gridPoints <- expand.grid(varWave = x1, skewWave = x2)
prob_grid  <- predict(GP_cls_2, gridPoints, type = "probabilities")

# Plot P(fraud=1) contours + points
z <- matrix(prob_grid[, 2], nrow = 100, ncol = 100, byrow = FALSE)
contour(x1, x2, z, 20, xlab = "varWave", ylab = "skewWave",
        main = sprintf("P(Fraud=1) - Train acc = %.3f", acc_train))
points(train[train$fraud == 1, c("varWave", "skewWave")], col = "blue", pch = 20)
points(train[train$fraud == 0, c("varWave", "skewWave")], col = "red",  pch = 20)
```



3.2. Test set prediction

```
pred_test <- predict(GP_cls_2, test)
(test_tab <- table(pred_test, test$fraud))
```

```
##
## pred_test   0   1
##         0 199   9
##         1  19 145
```

```
acc_test_2 <- acc(test_tab)
acc_test_2
```

```
## [1] 0.9247312
```

3.3. Model with all covariates

We now include all four covariates: varWave, skewWave, kurtWave, entropyWave.

```
GP_cls_4   <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data = train)
```

```
## Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
pred_test4 <- predict(GP_cls_4, test)
(test_tab4 <- table(pred_test4, test$fraud))
```

```
##
## pred_test4   0   1
```

```
##          0 216   0
##          1   2 154
```

```
acc_test_4 <- acc(test_tab4)
acc_test_4
```

```
## [1] 0.9946237
```

Including all four covariates generally improves accuracy versus the 2-feature model. The added features (especially kurtWave and entropyWave) contribute discriminative information that sharpens the decision boundary. Gains may vary with random splits, but we observed a consistent improvement on our test set, suggesting the 4-feature GPC better captures the joint structure that separates fraud vs non-fraud.

4. Contributions

This report is a group effort by Xiaochen, Linn, Qinxia, Qingxuan and Huaide. The individual sections were drafted as follows: Xiaochen and Linn (Section 1), Qinxia and Qingxuan (Section 2), Huaide (Section 3).

5. Appendix

```
SquaredExpKernel <- function(x1, x2, sigmaF = 1, l = 1) {
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA_real_, n1, n2)
  for (i in seq_len(n2)) {
    K[, i] <- sigmaF^2 * exp(-0.5 * ((x1 - x2[i]) / l)^2)
  }
  K
}

posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...) {
  # Covariance blocks
  K    <- k(X, X, ...)
  Kstar<- k(X, XStar, ...)
  Kss  <- k(XStar, XStar, ...)

  # Add noise to training covariance
  L <- chol(K + sigmaNoise^2 * diag(nrow(K)))
  L <- t(L)  # convert to lower triangular

  # Algorithm 2.1
  alpha <- solve(t(L), solve(L, y))
  post_mean <- t(Kstar) %*% alpha

  v <- solve(L, Kstar)
  post_cov  <- Kss - t(v) %*% v

  list(mean = as.vector(post_mean), var = post_cov)
}

sigmaF <- 1
l <- 0.3
library(tibble)
library(ggplot2)
```

```
x_values <- seq(-1, 1, length.out = 5000)
mean_pred  <- posteriorGP(0.4, 0.719, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred <- posteriorGP(0.4, 0.719, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data <- tibble(
  x_values = x_values,
  y_values = mean_pred,
  sd = sqrt(diag(var_pred)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = 0.4, y = 0.719, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Single Training Point)", x = "x", y = "y")

obs_x <- c(0.4, -0.6)
obs_y <- c(0.719, -0.044)

mean_pred2  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred2 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data2 <- tibble(
  x_values = x_values,
  y_values = mean_pred2,
  sd = sqrt(diag(var_pred2)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data2, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Two Training Points)", x = "x", y = "y")

obs_x <- c(-1, -0.6, -0.2, 0.4, 0.8)
obs_y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
mean_pred3  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred3 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data3 <- tibble(
  x_values = x_values,
  y_values = mean_pred3,
  sd = sqrt(diag(var_pred3)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
```

```
)

ggplot(data = my_data3, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Five Training Points)", x = "x", y = "y")

l <- 1
mean_pred4  <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$mean
var_pred4 <- posteriorGP(obs_x, obs_y, x_values, 0.1, k = SquaredExpKernel, sigmaF, l)$var

my_data4 <- tibble(
  x_values = x_values,
  y_values = mean_pred4,
  sd = sqrt(diag(var_pred4)),
  upp = y_values + 1.96 * sd,
  low = y_values - 1.96 * sd
)

ggplot(data = my_data4, aes(x = x_values, y = y_values)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin = low, ymax = upp),
              fill = "skyblue", alpha = 0.3) +
  annotate("point", x = obs_x, y = obs_y, color = "red", size = 3) +
  theme_minimal() +
  labs(title = "GP Regression (Length Scale = 1)", x = "x", y = "y")

rm(list = ls())

library(kernlab)

# Read data (daily mean temperature, leap day removed)
temp <- read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv",
  header = TRUE, sep = ";"
)

# Full series and 1-in-5 subsample
temp_all <- temp$temp
time <- seq(1, 2186, by = 5)             # time = 1, 6, 11, ..., 2186
temp_sub <- temp_all[time]

# Day-of-year repeating sequence for the subsample
day <- rep(seq(1, 361, by = 5), 6)       # day = 1, 6, ..., 361 repeated per year
stopifnot(length(day) == length(temp_sub))

# Squared Exponential kernel for kernlab (scalar inputs)
sekernel <- function(sigmaf = 1, ell = 1) {
  rval <- function(x, y = NULL) {
    r2 <- as.numeric(crossprod(x - y))  # (x - y)^2 for 1D inputs
    sigmaf^2 * exp(-0.5 * r2 / ell^2)
```

```
  }
  class(rval) <- "kernel"
  rval
}

# Evaluate k(1, 2) and K(X, X*)
SEFunc <- sekernel(sigmaf = 1, ell = 1)
SEFunc(1, 2)

X     <- matrix(c(1, 3, 4), ncol = 1)
Xstar <- matrix(c(2, 3, 4), ncol = 1)
kernelMatrix(kernel = SEFunc, x = X, y = Xstar)

# Estimate noise SD via quadratic fit
polyFit <- lm(temp_sub ~ time + I(time^2))
sigmaNoise <- sd(residuals(polyFit))

# Fit GP with time as input
SE_time <- sekernel(sigmaf = 20, ell = 100)
GP_time <- gausspr(time, temp_sub,
                   kernel = SE_time, var = sigmaNoise^2,
                   variance.model = TRUE, scaled = FALSE)

# Predictions and 95% bands
mean_time <- predict(GP_time, time)
sd_time   <- predict(GP_time, time, type = "sdeviation")

plot(time, temp_sub, pch = 1, cex = 0.6, xlab = "time (days since 2010-01-01)",
     ylab = "temperature (C)", main = "GP regression with time as input")
lines(time, mean_time, col = "red", lwd = 2)
lines(time, mean_time + 1.96 * sd_time, col = "blue", lwd = 1.5)
lines(time, mean_time - 1.96 * sd_time, col = "blue", lwd = 1.5)

# Covariance matrices at training locations
Kxx <- kernelMatrix(kernel = SE_time, x = matrix(time, ncol = 1))
n   <- length(time)

# Posterior mean and covariance at training points
Mean_manual <- t(Kxx) %*% solve(Kxx + sigmaNoise^2 * diag(n), temp_sub)
Cov_manual  <- Kxx - t(Kxx) %*% solve(Kxx + sigmaNoise^2 * diag(n), Kxx)

# Plot
plot(time, temp_sub, pch = 1, cex = 0.6, xlab = "time", ylab = "temperature (C)",
     main = "Manual posterior (Algorithm 2.1) at training points")
lines(time, as.numeric(Mean_manual), col = "red", lwd = 2)
lines(time, as.numeric(Mean_manual) - 1.96 * sqrt(diag(Cov_manual)),
      col = "blue", lwd = 1.5)
lines(time, as.numeric(Mean_manual) + 1.96 * sqrt(diag(Cov_manual)),
      col = "blue", lwd = 1.5)

GP_day <- gausspr(day, temp_sub,
                  kernel = SE_time, var = sigmaNoise^2,
                  variance.model = TRUE, scaled = FALSE)
mean_day <- predict(GP_day, day)
```

```
plot(time, temp_sub, pch = 1, cex = 0.6,
     xlab = "time", ylab = "temperature (C)",
     main = "Posterior means: time (red) vs day (blue)")
lines(time, mean_time, col = "red", lwd = 2)
lines(time, mean_day,  col = "blue", lwd = 2)

# Locally periodic kernel (periodic x SE)
lp_kernel <- function(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365) {
  rval <- function(x, y = NULL) {
    r  <- sqrt(as.numeric(crossprod(x - y)))
    per <- exp(-2 * (sin(pi * r / d))^2 / ell1^2)
    se  <- exp(-0.5 * r^2 / ell2^2)
    sigmaf^2 * per * se
  }
  class(rval) <- "kernel"
  rval
}

LP_time <- lp_kernel(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365)
GP_lp   <- gausspr(time, temp_sub,
                   kernel = LP_time, var = sigmaNoise^2,
                   variance.model = TRUE, scaled = FALSE)
mean_lp <- predict(GP_lp, time)

plot(time, temp_sub, pch = 1, cex = 0.6,
     xlab = "time", ylab = "temperature (C)",
     main = "Posterior means: time (red), day (blue), locally periodic (green)")
lines(time, mean_time, col = "red",   lwd = 2)
lines(time, mean_day,  col = "blue",  lwd = 2)
lines(time, mean_lp,   col = "green", lwd = 2)

data <- read.csv(
  "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.csv",
  header = FALSE, sep = ","
)
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
data$fraud  <- as.factor(data$fraud)

SelectTraining <- sample(seq_len(nrow(data)), size = 1000, replace = FALSE)
train <- data[SelectTraining, ]
test  <- data[-SelectTraining, ]

library(kernlab)

GP_cls_2 <- gausspr(fraud ~ varWave + skewWave, data = train)

# In-sample predictions and accuracy
pred_train <- predict(GP_cls_2, train)
acc <- function(tab) sum(diag(tab)) / sum(tab)
(train_tab <- table(pred_train, train$fraud))
acc_train <- acc(train_tab)

# Probability grid for contour plot
```

```
x1 <- seq(min(data$varWave), max(data$varWave), length.out = 100)
x2 <- seq(min(data$skewWave), max(data$skewWave), length.out = 100)

gridPoints <- expand.grid(varWave = x1, skewWave = x2)
prob_grid  <- predict(GP_cls_2, gridPoints, type = "probabilities")

# Plot P(fraud=1) contours + points
z <- matrix(prob_grid[, 2], nrow = 100, ncol = 100, byrow = FALSE)
contour(x1, x2, z, 20, xlab = "varWave", ylab = "skewWave",
        main = sprintf("P(Fraud=1) - Train acc = %.3f", acc_train))
points(train[train$fraud == 1, c("varWave", "skewWave")], col = "blue", pch = 20)
points(train[train$fraud == 0, c("varWave", "skewWave")], col = "red",  pch = 20)

pred_test <- predict(GP_cls_2, test)
(test_tab <- table(pred_test, test$fraud))
acc_test_2 <- acc(test_tab)
acc_test_2

GP_cls_4   <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data = train)
pred_test4 <- predict(GP_cls_4, test)
(test_tab4 <- table(pred_test4, test$fraud))
acc_test_4 <- acc(test_tab4)
acc_test_4
```