732A96 Grouplab 3

Kevin Neville, Gustav Sternelöv, Vuong Tran

October 3, 2016

Assignment 1

a)

The implementation of a function that simulates from the posterior distribution *f(x)* by using the squared exponential kernel is done in two steps. In the first step a function that computes the squared exponential kernel is created. The formula for the squared exponential kernel can be seen below:

$$K(x, x') = \sigma_f^2 \times exp(-0.5 \times (\frac{x - x'}{\iota})^2)$$

The second step is to build the function *PosteriorGP*. The aim with this function is to calculate the posterior mean and variance of f over a grid of x-values. This is done by implementing algorithm 2.1 from the book *Gaussian Processes for Machine Learning* (Rasmussen and Williams).

The code that has been used to implement the functions can be seen in the appendix *R-code*.

The prior mean of f is assumed to be zero for all x, which gives the following prior distribution for *f(x)*:

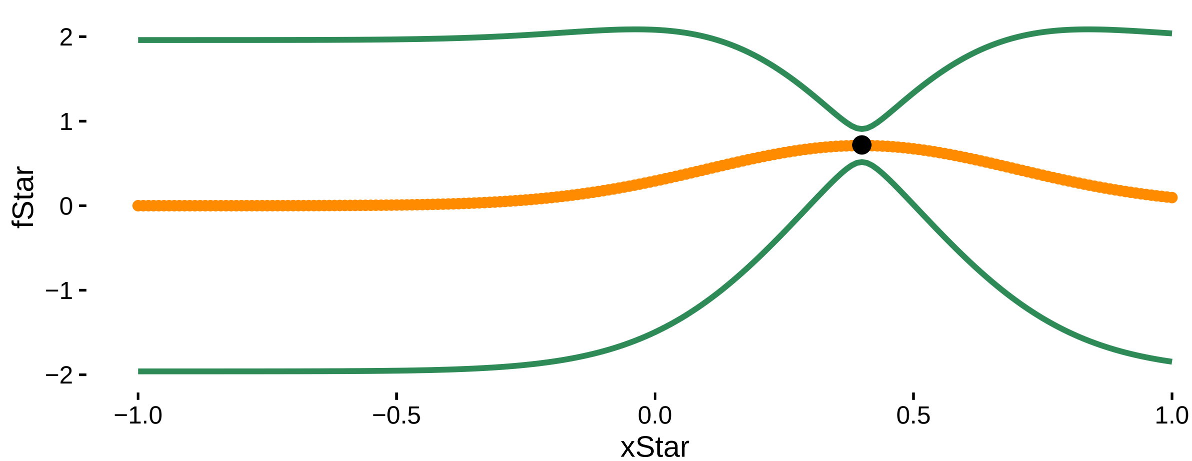
$$f(x) \sim GP(0, K(x, x'))$$

Then, the posterior guassian distribution looks as following:

$$f_* \mid x, y, x_* \sim N(\bar{f_*}, cov(\bar{f_*}))$$

b)

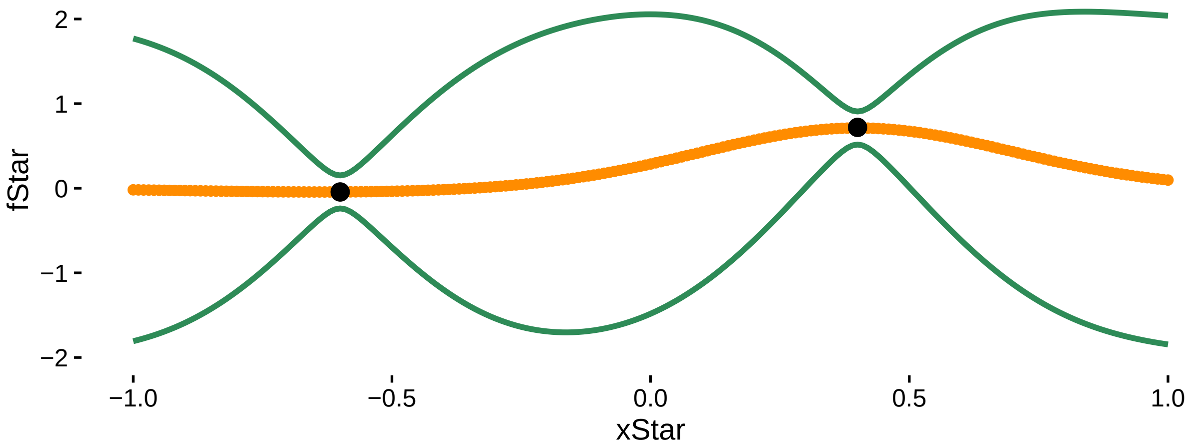
Since the noise standard deviation is assumed to be known the parameter σ_n is set to 0.1. The prior hyperparameter σ_f is set to 1 and the second prior hyperparameter ι is set to 0.3. Furthermore the prior is updated with one observation, *(x,y)*=(0.4, 0.719). A plot over the posterior mean of f over the interval $x \in$ [-1,1] with 95 % probability bands for f can be seen below.



The black dot is the observed value and it can be seen that the probability bands are more narrow around this value.

c)

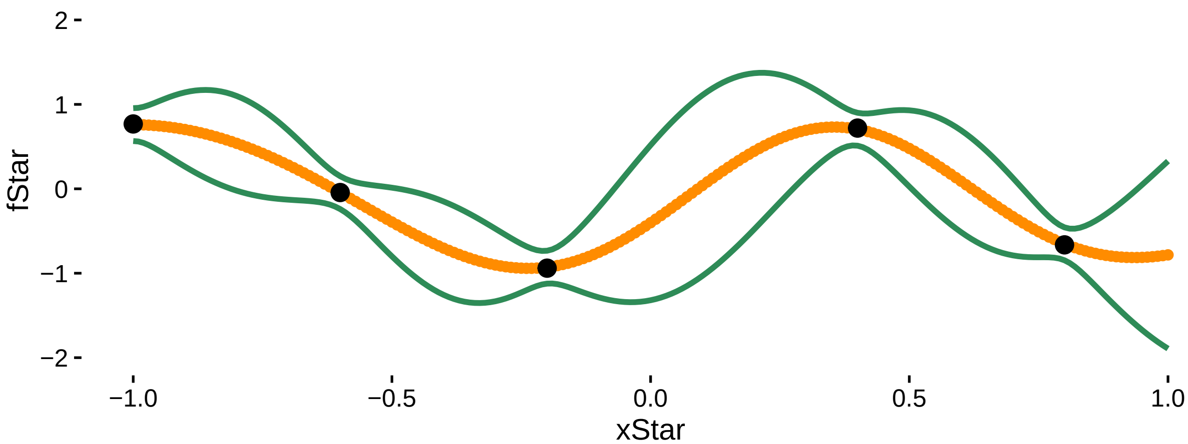
The posterior from *b)* is updated with another observation, *(x,y)*=(-0.6, -0.044).



Again it can be seen that the probability bands are more narrow around the observed values, and that they are quite wide for the other values.

d)

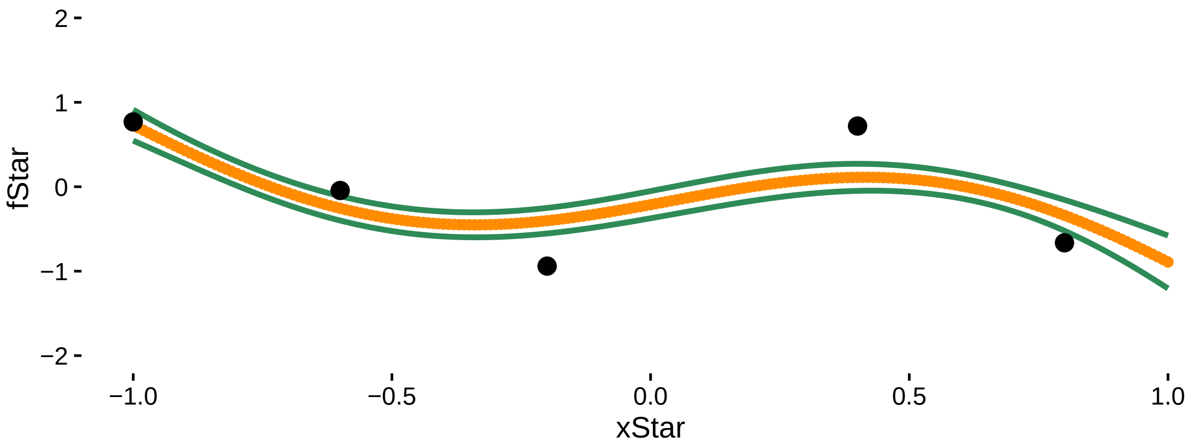
In *d)* the number of observations rises to five, resulting in the following plot over the posterior mean of f and its 95 % probability intervals.



Compared to the plots in *b)* and *c)*, the curve for the posterior mean of f is less straight/ more curvaceous than before. The probability bands has also changed and are thanks to the rise from two to five observed values more narrow, but also quite wiggly.

e)

The hyperparameter ι is now set to 1. The other parameters are unchanged and the same observations as in *d)* are used.



Compared to the plot in *d)*, the probability bands obtained with ι set to 1 looks much smoother and lies much closer to the curve for the posterior mean of *f*. Another change is that curve for the posterior mean of *f* no longer goes through all of the observed values. Instead the fitted curve appears to be an average between the observed values that lies closest to each other. With ι set to 1 the curve becomes all too smooth.

Summarize - Assignment 1

As the number of observations increases we become more sure about the values of the posterior mean. How sure we become depends on the prior hyperparameters. σ_f controls the uncertainty for the prior and length scale controls how many observations that affects the estimated posterior mean.

Assignment 2

a)

The code where we define our own square exponential kernel function can be seen below.

```
sqExpK <- function(sigmaf = 1, ell = 1) {
  rval <- function(x, xStar = NULL) {
    return(sigmaf * exp (-1/2 * ((x-xStar)/ell)^2 ))
  }
  class(rval) <- "kernel"
  return(rval)
}
```

When evaluated in the point $x = 1$ and $x' = 2$ the following value is given.

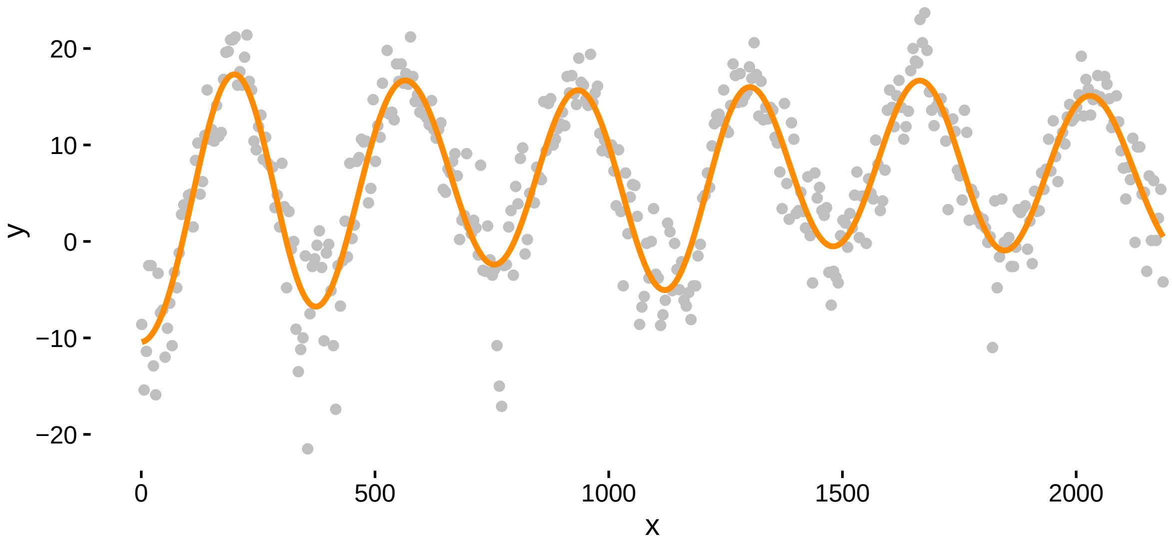
```
## [1] 0.8824969
```

The *kernelMatrix* is tested for the vectors $\mathbf{x} = (1, 3, 4)^T$ and $x_* = (2, 3, 4)^T$. Returned by this function is the covariance matrix $\mathbf{K}(x, x_*)$. This matrix can be interpreted as following:
The covariance between (x=1, x'=4) is 0.32.

```
## An object of class "kernelMatrix"
##           2         3         4
## 1 0.8824969 0.6065307 0.3246525
## 3 0.8824969 1.0000000 0.8824969
## 4 0.6065307 0.8824969 1.0000000
```

b)

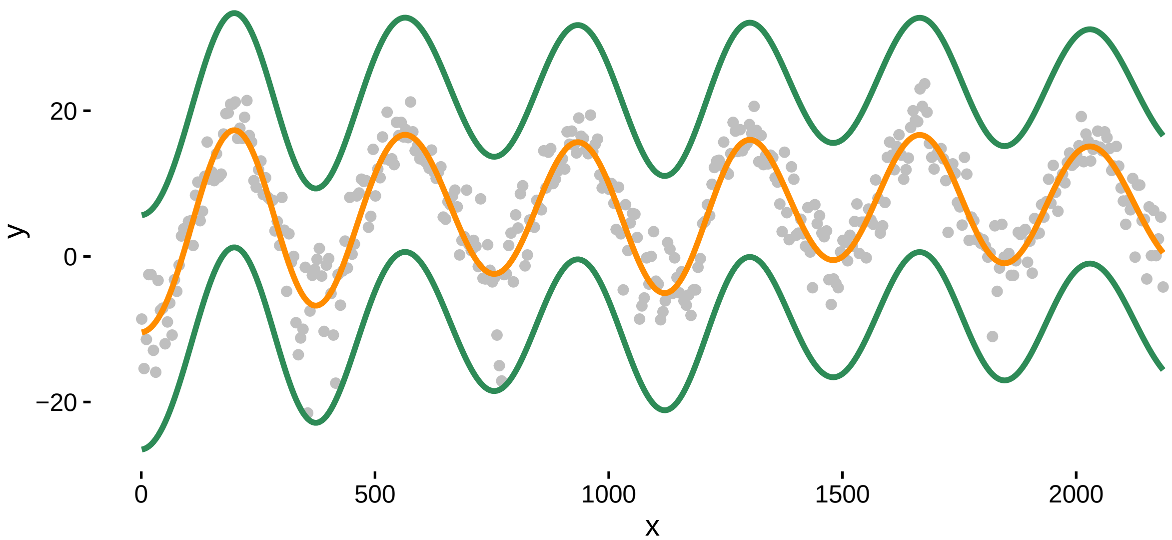
The gaussian process regression model where the temperature is a function of time is estimated. At every data point in the training set is the posterior mean computed. In the graph below are the points in the data plotted together with the posterior mean. The prior for σ_n^2 is 66.85 which is the variance obtained when a simple quadratic regression is estimated. The prior Hyperparameters σ_f and ι are set to 20 and 0.2.



The orange curve are the predicted values and the grey dots the observed temperature. Overall, the fitted model seem to be rather good. There is some yearly variation which is modeled quite well.

c)

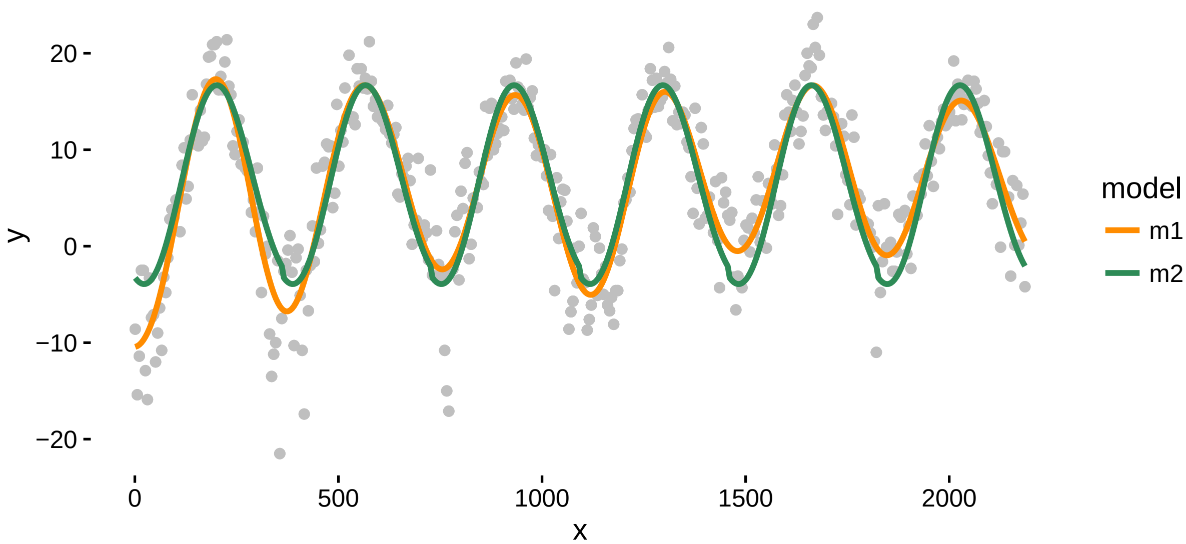
The algorithm implemented in *1.a)* that was used for computing the posterior distribution of f is now used for computing the posterior variance of the results presented in *2.b)*. The result is presented by adding 95 % posterior probability bands to the plot of the posterior means presented in the previous exercise.



The 95 % posterior probability bands looks reasonable as all the observed temperatures lies inside the bands. However, the bands are perhaps a bit too wide. For example, the 95 % interval for the temperature during summer goes from around 0 to 35.

d)

A model with temperature as a function of day is estimated. The posterior mean of this model is compared to the posterior mean of the model which had temperature as a function of time.



A comparsion of the curves gives that the model with day as input is constant and shows the exactly same pattern every year. Positive with this model is that it takes historical data into account when estimating the posterior mean. A negative aspect with this model is that it not allows any variation for the estimates given for specific day.

The model that has time as input is more flexible. It allows the posterior mean for a day to differ between years as the estimate mainly is affected by the temperatures observed the weeks or days before during that year. The observed values the years before does not have any larger impact on the estimate.

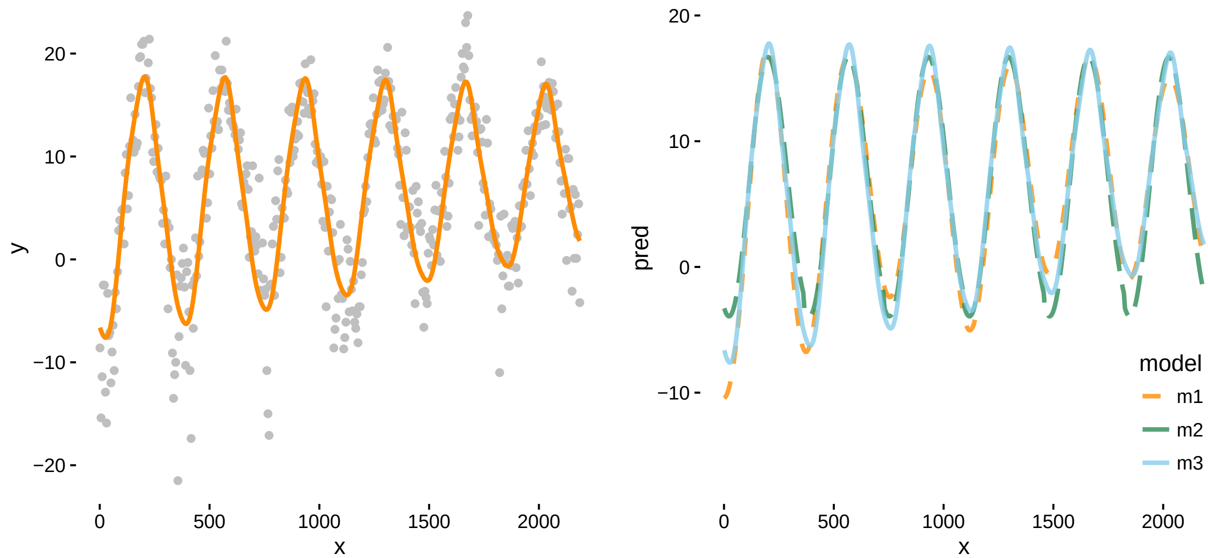
You could say that the pro for one model is a con for the second, and vice versa.

e)

A periodic kernel is implemented and the formula for the kernel is given below.

$$K(x, x') = \sigma_f^2 \times exp(\frac{2sin^2(\pi|x - x'|/d)}{\iota_1^2}) \times exp(-0.5 \times (\frac{x - x'}{\iota_1^2})^2)$$

Two new parameters are then introduced. The correlation between the same day in different years is controlled by the parameter ι_2^2 and the period is specified with the parameter d. A Gaussian process model is estimated with time as explanatory variable.



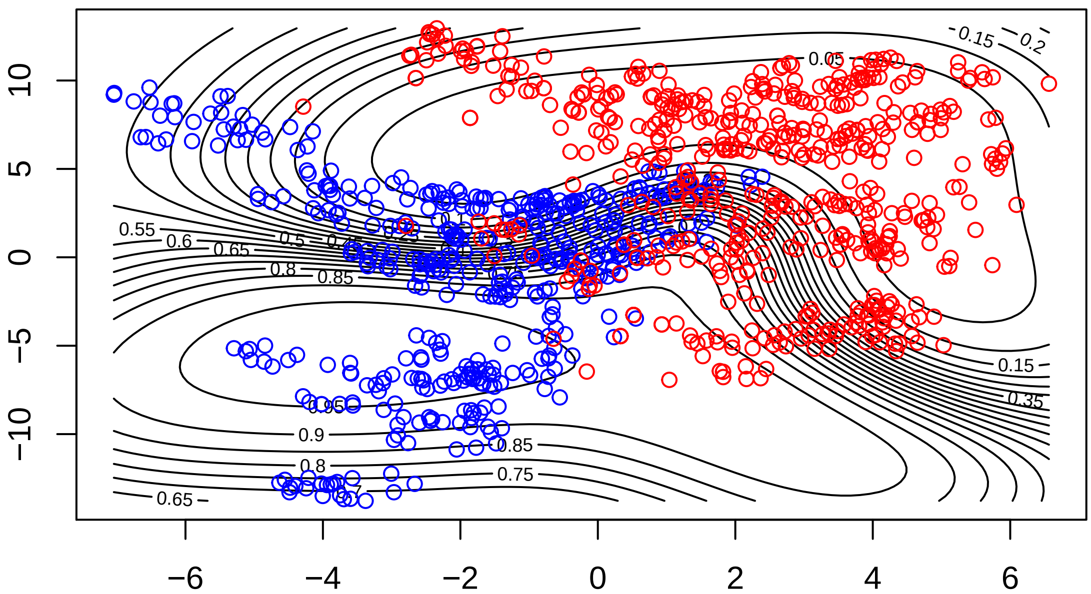
The periodic kernel combines both data from earlier years and data from days closer to the actual day. In practice it works as a combination of the models discussed in *2.d)*. The fitted curve in the plot to the left is similar to the earlier ones, with some exceptions. The posterior mean for the temperature during the winter increases for every year and the temperature for the summer looks to be constant.

This result very well shows the ability of the periodic kernel. For some parts of the year the posterior mean is constant and for some it is more flexible.

Assignment 3

a)

A model with the variables *varWave* and *skewWave* as input variables and fraud as target variable is estimated. The contours of the prediction probability for an observation to get classified as fraud or not is plotted below. In the contour plot is also training data added where fraud=1 are coloured blue and observations with fraud=0 coloured red.



It can be seen that the model relatively well separates the training data correct with the prediction probabilities obtained with the model. This conclusion is also confirmed by the confusion matrix for the classifier on the training data and the accuracy.

```
##
##         0   1
##   0 512  24
##   1  44 420
```

```
## Accuracy: 0.932
```

b)

The estimated model is then used for making predictions on the test set and the results are presented in a confusion matrix. The accuracy for the test data is almost the same as for the training data. In order words does the model seem to be a rather good fit.

```
##
##         0   1
##   0 191   9
##   1  15 157
```

```
## Accuracy: 0.9354839
```

c)

A new model with two more input variables, *kurtWave* and *entropyWave*, is fitted. The confusion matrix and the accuracy of this model when the observations in the test data are predicted is presented below. A higher accuracy is recieved as all but one of the observations are classified correctly with this model.

```
## Accuracy: 0.9973118
```

Group report

Each member of the group has contributed with their own graphs, written conclusions and participated in discussions about the lab.

Appendix - R-code

```r
library(ggplot2)
# Lab 3 Assignment 1
SqExpKernel <- function(x1, x2, hyperParam) {
    K <- matrix(nrow = length(x1), ncol = length(x2))
    for (i in 1:length(x2)) {
        K[, i] <- hyperParam[1]^2 * exp(-0.5 * ((x1 - x2[i])/hyperParam[2])^2)
    }
    return(K)
}
PosteriorGP <- function(x, y, xStar, hyperParam, sigmaNoise) {
    # Calculates f star bar
    K <- SqExpKernel(x, x, hyperParam)
    L <- t(chol(K + sigmaNoise * diag(dim(K)[1])))
    alpha <- solve(t(L), solve(L, y))
    # Posterior mean
    fStar <- (SqExpKernel(xStar, x, hyperParam)) %*% alpha
    # Posterior variance
    v_f <- solve(L, t(SqExpKernel(xStar, x, hyperParam)))
    cov_fStar <- SqExpKernel(xStar, xStar, hyperParam) - (t(v_f) %*% v_f)
    # Store all values in a list
    val_list <- list(fStar = fStar, xStar = xStar, cov_fStar = cov_fStar, xStar = xStar)
    return(val_list)
}
assign1B <- PosteriorGP(x = 0.4, y = 0.719, xStar = seq(-1, 1, 0.01), hyperParam = c(1,
    0.3), sigmaNoise = 0.1^2)
Upper1B <- assign1B$fStar + 1.96 * sqrt(diag(assign1B$cov_fStar))
Lower1B <- assign1B$fStar - 1.96 * sqrt(diag(assign1B$cov_fStar))
plotD <- data.frame(fStar = assign1B$fStar, xStar = assign1B$xStar, Lower = Lower1B,
    Upper = Upper1B)
xy <- data.frame(x = 0.4, y = 0.719)
ggplot(plotD, aes(y = fStar, x = xStar)) + geom_point(col = "darkorange") +
    ylim(-2, 2.2) + geom_line(data = plotD, aes(xStar, Lower), col = "seagreen",
    size = 1.1) + geom_line(data = plotD, aes(xStar, Upper), col = "seagreen",
    size = 1.1) + geom_point(data = xy, aes(x, y), size = 3) + theme_classic()
```

```r
assign1C <- PosteriorGP(x = c(0.4, -0.6), y = c(0.719, -0.044), xStar = seq(-1,
    1, 0.01), hyperParam = c(1, 0.3), sigmaNoise = 0.1^2)
Upper1C <- assign1C$fStar + 1.96 * sqrt(diag(assign1C$cov_fStar))
Lower1C <- assign1C$fStar - 1.96 * sqrt(diag(assign1C$cov_fStar))

plotD <- data.frame(fStar = assign1C$fStar, xStar = assign1C$xStar, Lower = Lower1C,
    Upper = Upper1C)
xy <- data.frame(x = c(0.4, -0.6), y = c(0.719, -0.044))
ggplot(plotD, aes(y = fStar, x = xStar)) + geom_point(col = "darkorange") +
    ylim(-2, 2.2) + geom_line(data = plotD, aes(xStar, Lower), col = "seagreen",
    size = 1.1) + geom_line(data = plotD, aes(xStar, Upper), col = "seagreen",
    size = 1.1) + geom_point(data = xy, aes(x, y), size = 3) + theme_classic()
assign1D <- PosteriorGP(x = c(0.8, 0.4, -0.2, -0.6, -1), y = c(-0.664, 0.719,
    -0.94, -0.044, 0.768), xStar = seq(-1, 1, 0.01), hyperParam = c(1, 0.3),
    sigmaNoise = 0.1^2)
Upper1D <- assign1D$fStar + 1.96 * sqrt(diag(assign1D$cov_fStar))
Lower1D <- assign1D$fStar - 1.96 * sqrt(diag(assign1D$cov_fStar))

plotD <- data.frame(fStar = assign1D$fStar, xStar = assign1D$xStar, Lower = Lower1D,
    Upper = Upper1D)
xy <- data.frame(x = c(0.8, 0.4, -0.2, -0.6, -1), y = c(-0.664, 0.719, -0.94,
    -0.044, 0.768))
ggplot(plotD, aes(y = fStar, x = xStar)) + geom_point(col = "darkorange") +
    ylim(-2, 2.2) + geom_line(data = plotD, aes(xStar, Lower), col = "seagreen",
    size = 1.1) + geom_line(data = plotD, aes(xStar, Upper), col = "seagreen",
    size = 1.1) + geom_point(data = xy, aes(x, y), size = 3) + theme_classic()
assign1E <- PosteriorGP(x = c(0.8, 0.4, -0.2, -0.6, -1), y = c(-0.664, 0.719,
    -0.94, -0.044, 0.768), xStar = seq(-1, 1, 0.01), hyperParam = c(1, 1), sigmaNoise = 0.1^2)
Upper1E <- assign1E$fStar + 1.96 * sqrt(diag(assign1E$cov_fStar))
Lower1E <- assign1E$fStar - 1.96 * sqrt(diag(assign1E$cov_fStar))

plotD <- data.frame(fStar = assign1E$fStar, xStar = assign1E$xStar, Lower = Lower1E,
    Upper = Upper1E)
xy <- data.frame(x = c(0.8, 0.4, -0.2, -0.6, -1), y = c(-0.664, 0.719, -0.94,
    -0.044, 0.768))
ggplot(plotD, aes(y = fStar, x = xStar)) + geom_point(col = "darkorange") +
    ylim(-2, 2.2) + geom_line(data = plotD, aes(xStar, Lower), col = "seagreen",
    size = 1.1) + geom_line(data = plotD, aes(xStar, Upper), col = "seagreen",
    size = 1.1) + geom_point(data = xy, aes(x, y), size = 3) + theme_classic()
# Assignment 2
library(kernlab)
temps <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge
    header = TRUE, sep = ";")
temps$time <- 1:2190
temps$day <- rep(1:365, 6)
temps$index <- rep(1:5, 438)
thinTemps <- subset(temps, temps$index == 1)[, 1:4]
sqExpK <- function(sigmaf = 1, ell = 1) {
    rval <- function(x, xStar = NULL) {
        return(sigmaf * exp(-1/2 * ((x - xStar)/ell)^2))
    }
    class(rval) <- "kernel"
    return(rval)
```

```r
}
sqExpKFunc = sqExpK(sigmaf = 1, ell = 2)  # sqExpKFunc is a kernel FUNCTION
sqExpKFunc(1, 2)  # Evaluating the kernel in x=1, x'=2
# Computing the whole covariance matrix K from the kernel.
x <- as.matrix(c(1, 3, 4))
xStar <- as.matrix(c(2, 3, 4))
K <- kernelMatrix(kernel = sqExpKFunc, x = x, y = xStar)  # So this is K(X,Xstar)
row.names(K) <- c(1, 3, 4)
colnames(K) <- c(2, 3, 4)
K
quadLM <- lm(temp ~ time + time^2, data = temps)
sigma_n <- var(quadLM$residuals)


sqExpKFunc = sqExpK(sigmaf = 20^2, ell = 0.2)
reg <- gausspr(thinTemps$time, thinTemps$temp, kernel = sqExpKFunc, var = sigma_n)
postMean <- data.frame(pred = predict(reg), y = thinTemps$temp, x = thinTemps$time)
plotB <- ggplot(postMean, aes(x = x, y = y)) + geom_point(col = "grey75") +
    geom_line(data = postMean, aes(x = x, y = pred), col = "darkorange", size = 1.1) +
    theme_classic()
plotB
covar_f <- PosteriorGP(x = thinTemps$time, y = thinTemps$temp, xStar = thinTemps$time,
    hyperParam = c(20^2, 0.2), sigmaNoise = sigma_n)
probBands <- data.frame(Upper = postMean[, 1] + 1.96 * sqrt(diag(covar_f$cov_fStar)),
    Lower = postMean[, 1] - 1.96 * sqrt(diag(covar_f$cov_fStar)), x = thinTemps$time)
plotB + geom_line(data = probBands, aes(x = x, y = Upper), size = 1.1, col = "seagreen") +
    geom_line(data = probBands, aes(x = x, y = Lower), size = 1.1, col = "seagreen")
quadLM2 <- lm(temp ~ day + day^2, data = temps)
sigma_n2 <- var(quadLM2$residuals)
# Small difference, 67.3641 vs 64.10528
sqExpKFuncD = sqExpK(sigmaf = 20^2, ell = 1.2)
regD <- gausspr(thinTemps$day, thinTemps$temp, kernel = sqExpKFuncD, var = sigma_n2)
postMeanD <- data.frame(pred = predict(regD), y = thinTemps$temp, x = thinTemps$time)
postMT <- data.frame(rbind(postMean, postMeanD), model = c(rep("m1", 438), rep("m2",
    438)))
ggplot(postMT, aes(y = y, x = x)) + geom_point(col = "grey75") + geom_line(aes(x = x,
    y = pred, col = model), size = 1.1) + theme_classic() + scale_color_manual(values = c("darkorange",
    "seagreen"))
PeriodicK <- function(sigmaf = 1, ell_one = 1, ell_two = 1, d = 1) {
    rval <- function(x, xStar = NULL) {
        return(sigmaf * exp(-(2 * sin(pi * abs(x - xStar)/d)^2)/ell_one^2) *
            exp(-0.5 * (abs(x - xStar)^2/ell_two^2)))
    }
    class(rval) <- "kernel"
    return(rval)
}
PeriodicKFunc = PeriodicK(sigmaf = 20^2, ell_one = 1, ell_two = 10, d = 365/sd(thinTemps$time))
# PeriodicKFunc is a kernel FUNCTION
regE <- gausspr(thinTemps$time, thinTemps$temp, kernel = PeriodicKFunc, var = sigma_n)
postMeanE <- data.frame(pred = predict(regE), y = thinTemps$temp, x = thinTemps$time)
plotE <- ggplot(postMeanE, aes(x = x, y = y)) + geom_point(col = "grey75") +
    geom_line(data = postMeanE, aes(x = x, y = pred), col = "darkorange", size = 1.1) +
    theme_classic()
```

```r
postMT <- rbind(postMT, data.frame(postMeanE, model = "m3"))
plotAll <- ggplot(postMT, aes(x = x, y = pred)) + geom_line(aes(col = model,
    linetype = model), size = 1.1, alpha = 0.8) + theme_classic() + scale_color_manual(values = c("darke
    "seagreen", "skyblue")) + scale_linetype_manual(values = c("dashed", "longdash",
    "solid")) + theme(legend.justification = c(1, 0), legend.position = c(1,
    0)) + scale_y_continuous(limits = c(-17, 20))
library(gridExtra)
grid.arrange(plotE, plotAll, ncol = 2)
# Assignment 3
library(AtmRay)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
    header = FALSE, sep = ",")
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
data[, 5] <- as.factor(data[, 5])
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
Train <- data[SelectTraining, ]
Test <- data[-SelectTraining, ]
# a)
cfA <- gausspr(fraud ~ varWave + skewWave, data = Train)
gridX <- seq(min(Train$varWave), max(Train$varWave), length = 100)
gridY <- seq(min(Train$skewWave), max(Train$skewWave), length = 100)
gridP <- meshgrid(gridX, gridY)
gridP <- cbind(c(gridP$x), c(gridP$y))
gridP <- data.frame(gridP)
names(gridP) <- names(Train)[1:2]
probPredsA <- predict(cfA, gridP, type = "probabilities")
contour(x = gridX, y = gridY, z = matrix(probPredsA[, 2], 100), 20)
points(Train[Train[, 5] == 1, 1], Train[Train[, 5] == 1, 2], col = "blue")
points(Train[Train[, 5] == 0, 1], Train[Train[, 5] == 0, 2], col = "red")
# predict on the training set
confMatA <- table(predict(cfA, Train[, 1:2]), Train[, 5])  # confusion matrix
confMatA
cat("Accuracy:", sum(diag(confMatA))/sum(confMatA))
confMatB <- table(predict(cfA, Test[, 1:2]), Test[, 5])  # confusion matrix
confMatB
cat("Accuracy:", sum(diag(confMatB))/sum(confMatB))
cfC <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data = Train)
confMatC <- table(predict(cfC, Test[, 1:4]), Test[, 5])  # confusion matrix
cat("Accuracy:", sum(diag(confMatC))/sum(confMatC))
## NA
```