Advanced Machine Learning Lab04

Naveen Gabriel (navga709), Sridhar Adhikarla (sriad858), Mariano Mariani (marma330), Yusur Al-Mter (yusal621)

2019-10-17

Contents

1. Implementing GP Regression.

This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon \quad with \quad N(0, \sigma^2) \quad and \quad f \sim GP(0, k(x, x^{'}))$$

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algorithm uses the Cholesky decomposition (chol in R) to attain numerical stability. Note that L in the algorithm is a lower triangular matrix, whereas the R function returns an upper triangular matrix. So, you need to transpose the output of the R function. In the algorithm, the notation $A\ b$ means the vector x that solves the equation Ax = b (see p. xvii in the book). This is implemented in R with the help of the function solve.

Here is what you need to do:

1. Simulating from the posterior distribution

Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it posteriorGP) should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values X_*. You can assume that the prior mean of f is zero for all x. The function should have the following inputs: - **X**: Vector of training inputs. - **y**: Vector of training targets/outputs. - **XStar**: Vector of inputs where the posterior distribution is evaluated, i.e. X_*. - **hyperParam**: Vector with two elements,$\sigma_f \quad and \quad l$. - **sigmaNoise**: Noise standard deviation σ_n.

Solution:

```
# Setting up the kernel
sq_kernel = function(x1,x2,sigmaF,l){
  n1 = length(x1)
  n2 = length(x2)
  kern = matrix(NA,n1,n2)
  for (i in 1:n2){
    kern[,i] = sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
    }
  return(kern)
}
```

```
# Posterior distribution for GP
posteriorGP = function(x,y,xstar,hyperParam,sigmaNoise){

  # distance (covariance) between training points K(x,x)
  K = sq_kernel(x,x,hyperParam[1],hyperParam[2])
  I =length(x)
  L = t(chol(K + diag(sigmaNoise^2,I)))

  # to compute the predictive mean
  temp = solve(L,y)
  alpha = solve(t(L),temp)

  # distance (covariance) between training and test points K(x,x*)
  K_star = sq_kernel(x,xstar,hyperParam[1],hyperParam[2])

  # predictive mean
  f_bar_star = t(K_star) %*% alpha

  # to compute the predictive variance
```

```
  v = solve(L,K_star)

  # distance (covariance) between test points K(x*,x*)
  K_test = sq_kernel(xstar,xstar,hyperParam[1],hyperParam[2])

  # predictive variance
  V_f_star = K_test - t(v) %*% v

  return(list("mean" = f_bar_star, "var" = V_f_star))
}
```

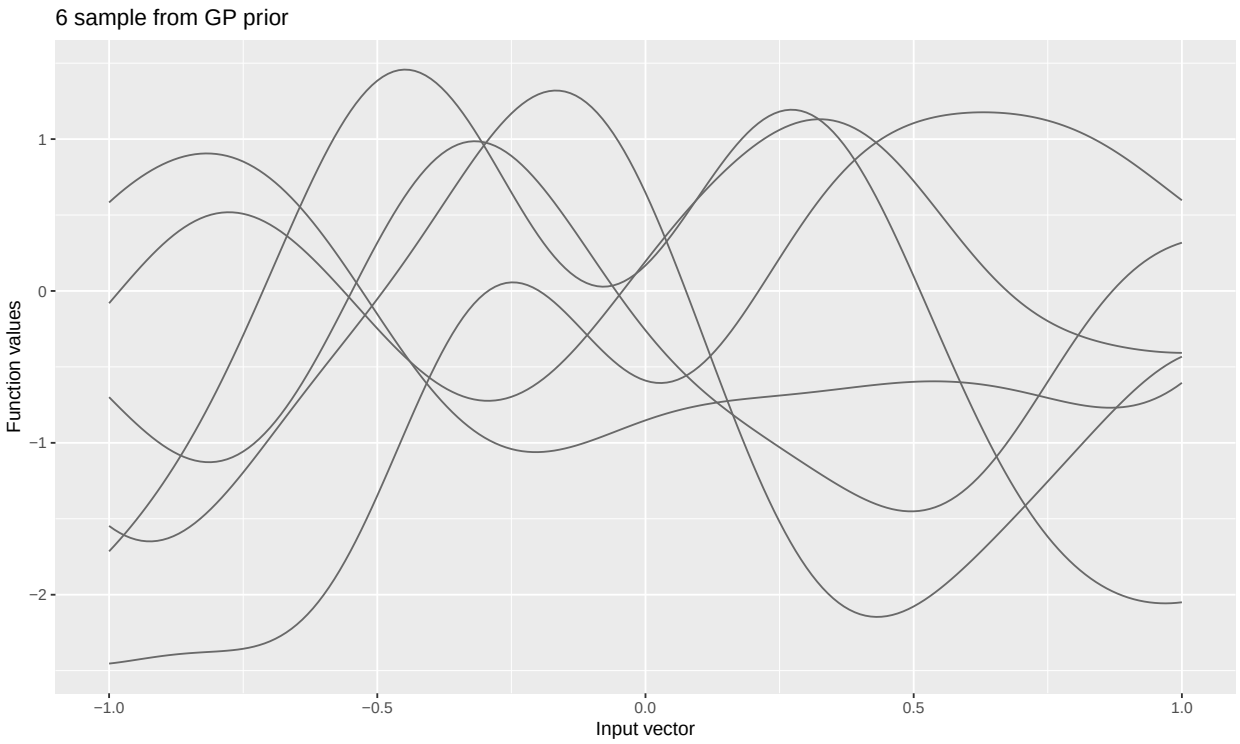
2. Update prior with single observation

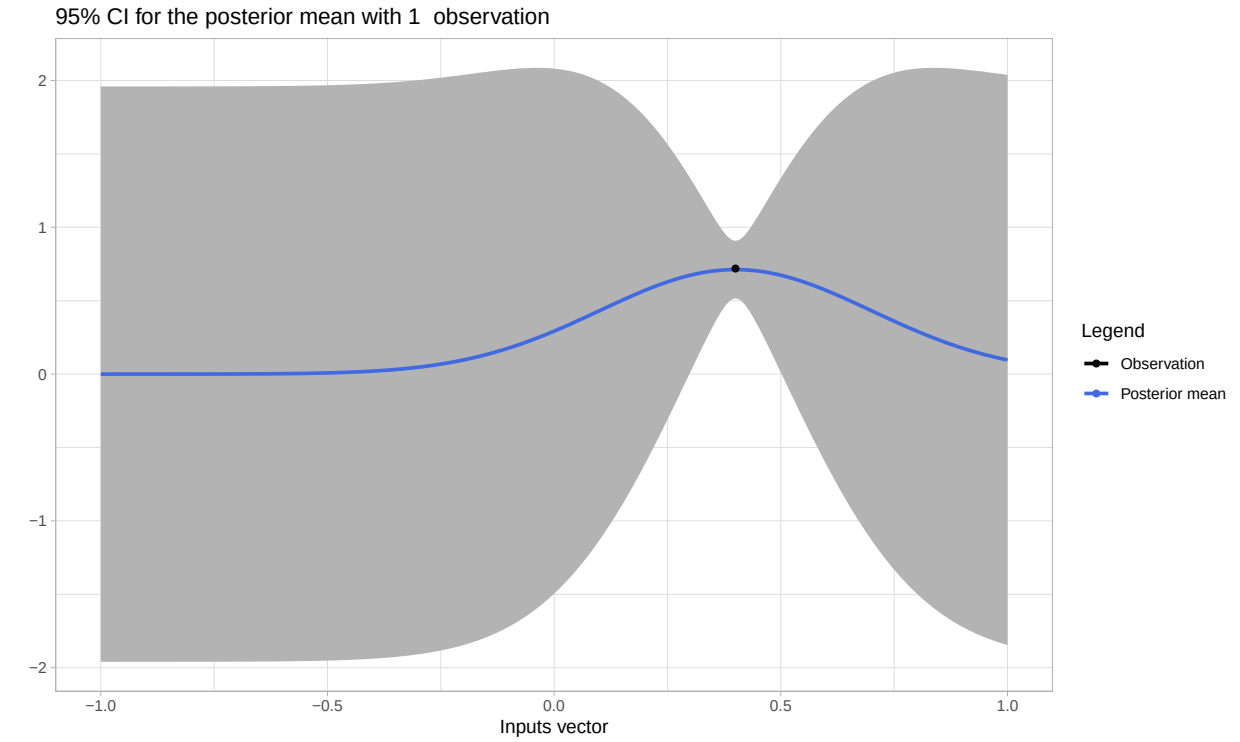
Now, let the prior hyperparameters be $\sigma_f = 1$ *and* $l = 0.3$. Update this prior with a single observation: *(x, y)* $= (0.4, 0.719)$. Assume that $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also the 95% (pointwise) bands for f.

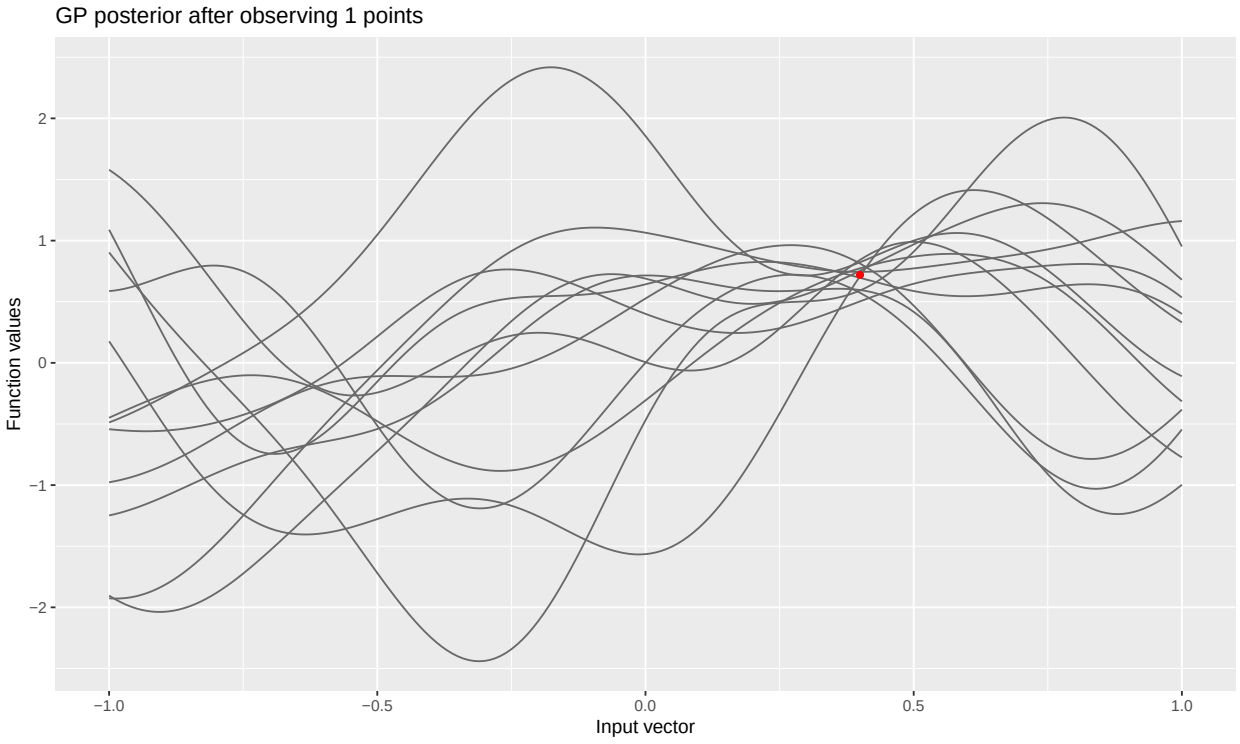
Solution:

```
# Utility function for the Plot

post_plot = function(x,y,xstar,hyperParam,noise){

  post = posteriorGP(x,y,xstar,hyperParam,noise)
  m = post$mean

  # Plot also 95 % probability (pointwise) bands for f.
  # for pred Y_star band add the noise(i.e the sigma2_n)
  lower = m - 1.96 * sqrt(diag(post$var))
  upper = m + 1.96 * sqrt(diag(post$var))

  cols <- c("Posterior mean"="#4169E1","Observation"="#000000")
  p = ggplot() + geom_ribbon(aes(ymin = lower, ymax = upper, xstar), fill = "grey70") +
                 geom_line(aes(x = xstar, y = m, col = "Posterior mean"),
                 alpha = 1,size = 1) +
                 geom_point(aes(x = x, y = y,  col = "Observation"), alpha = 1) +
                 scale_colour_manual(name = "Legend", values = cols) +
                 labs(title = paste("95% CI for the posterior mean with",
                                    length(x)," observation"),x = "Inputs vector", y = "")+
                 theme_light()

    points <- data.frame(x=x, y=y)
    func_val <- t(rmvnorm(12, post$mean, post$var))
    func_val <- as.data.frame(func_val)
    func_val$xstar <- xstar
    func_val <- melt(func_val,id="xstar")

    plt <- ggplot(func_val,aes(x=xstar,y=value)) +
                geom_line(aes(group=variable), colour="#696969",alpha = 1) +
                geom_point(data=points,aes(x=x,y=y),col="red") +
                ylab("Function values") + xlab("Input vector") +
                ggtitle(paste("GP posterior after observing",length(x),"points"))
```

```
  return(list(p,plt))
}



hyperParam = c(sigmaF = 1, l = 0.3)
xstar = seq(-1, 1, 0.01) # X star
noise = 0.1
```





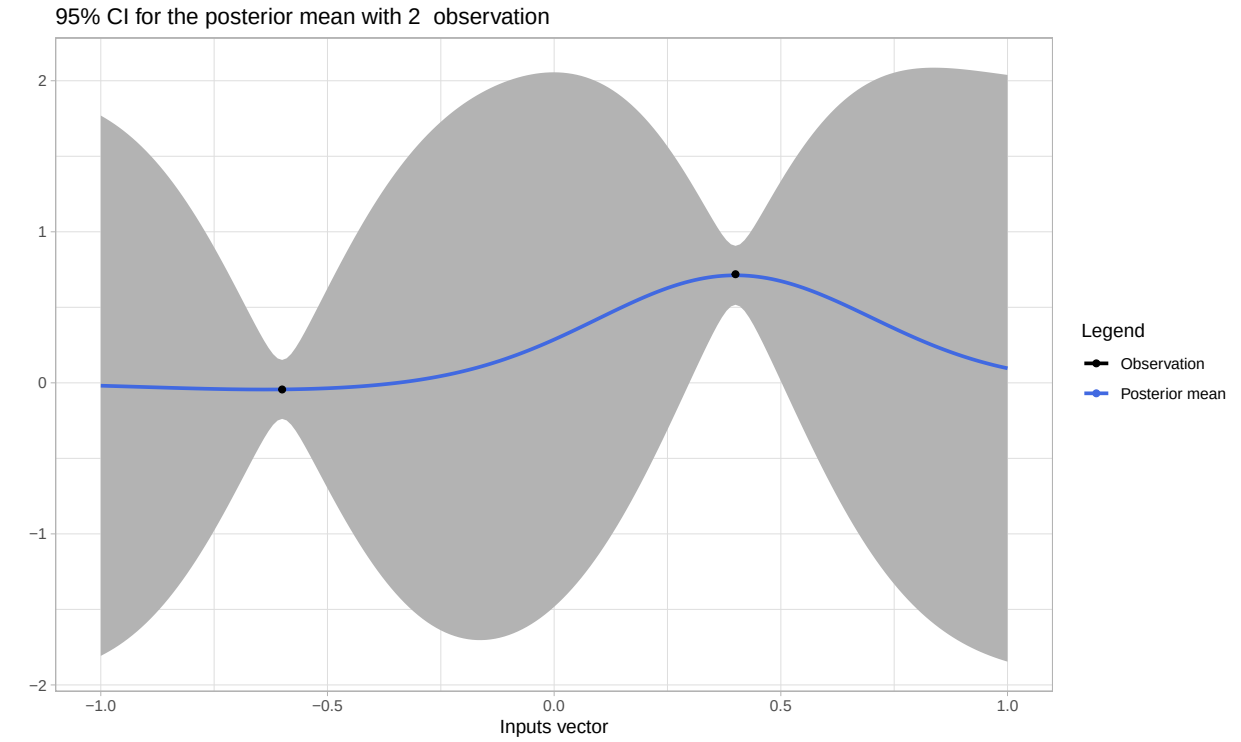


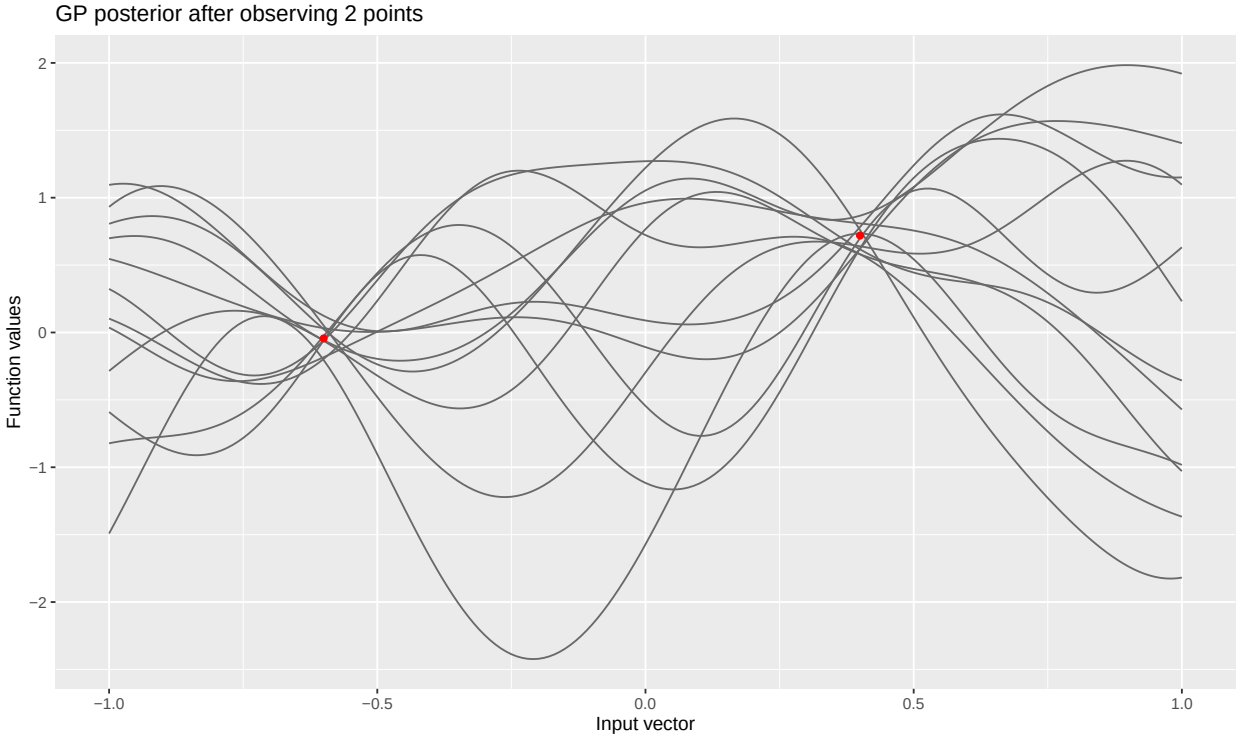
After seeing the first data, the uncertainity around that point decreases and it depends on the noise in the data. Since the noise is 0.1, the data is not so noisy. So all the uncertainty around that point squishes down close to the data.

3. Update prior with second observation

Update your posterior from (2) with another observation: (x,y) = (-0.6,-0.044). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.

Solution:



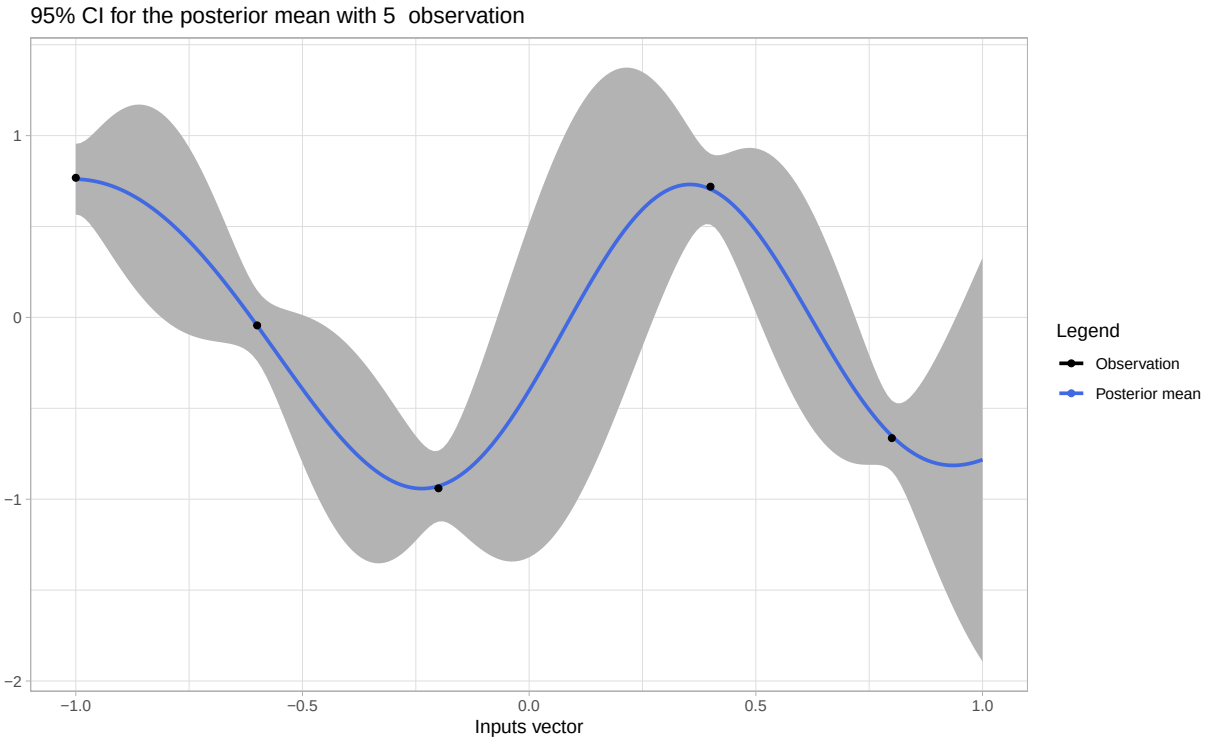


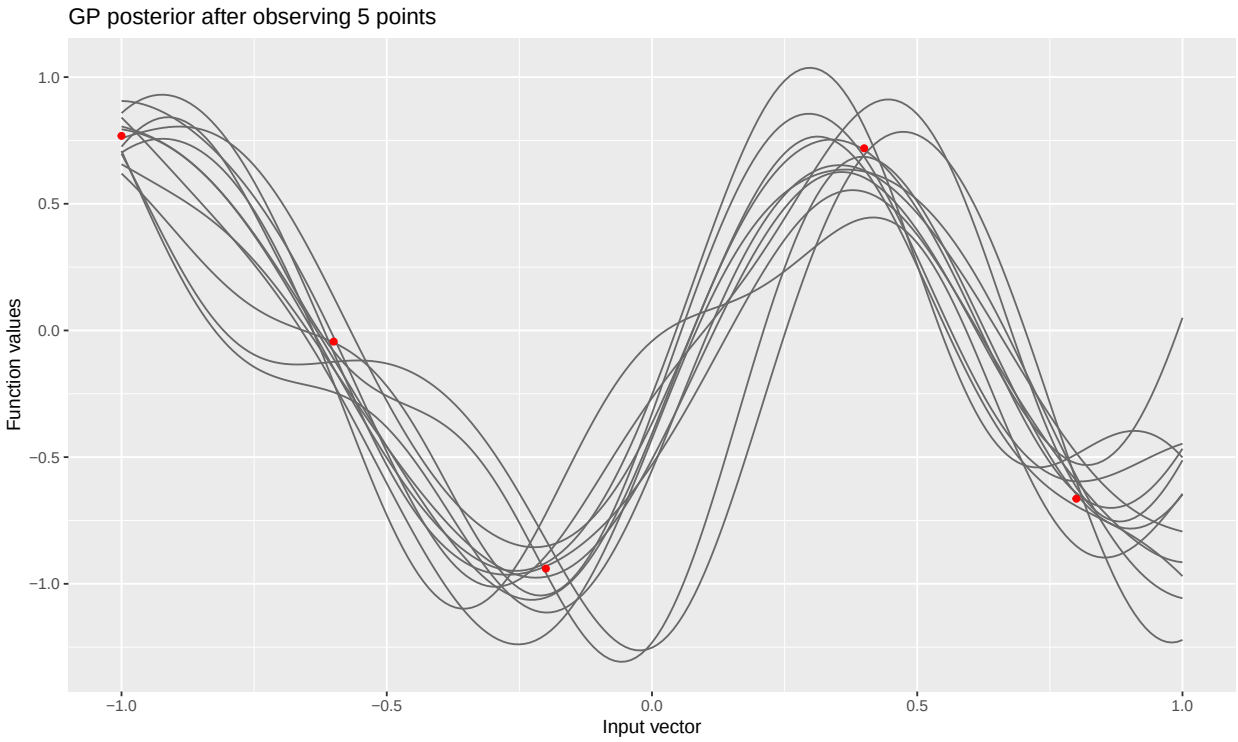
x	-1.0	-0.6	-0.2	0.4	0.4
y	0.768	-0.044	-0.940	0.719	-0.664

4. Compute posterior using 5 observation

Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.

Solution:

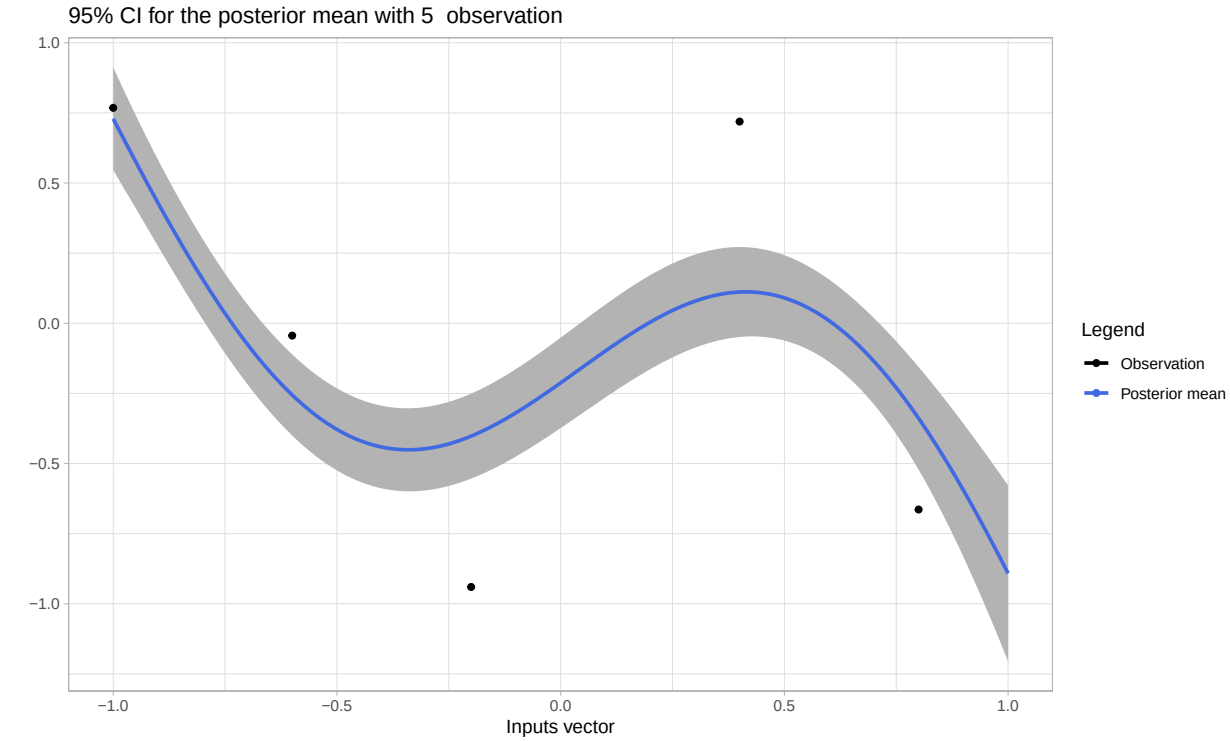


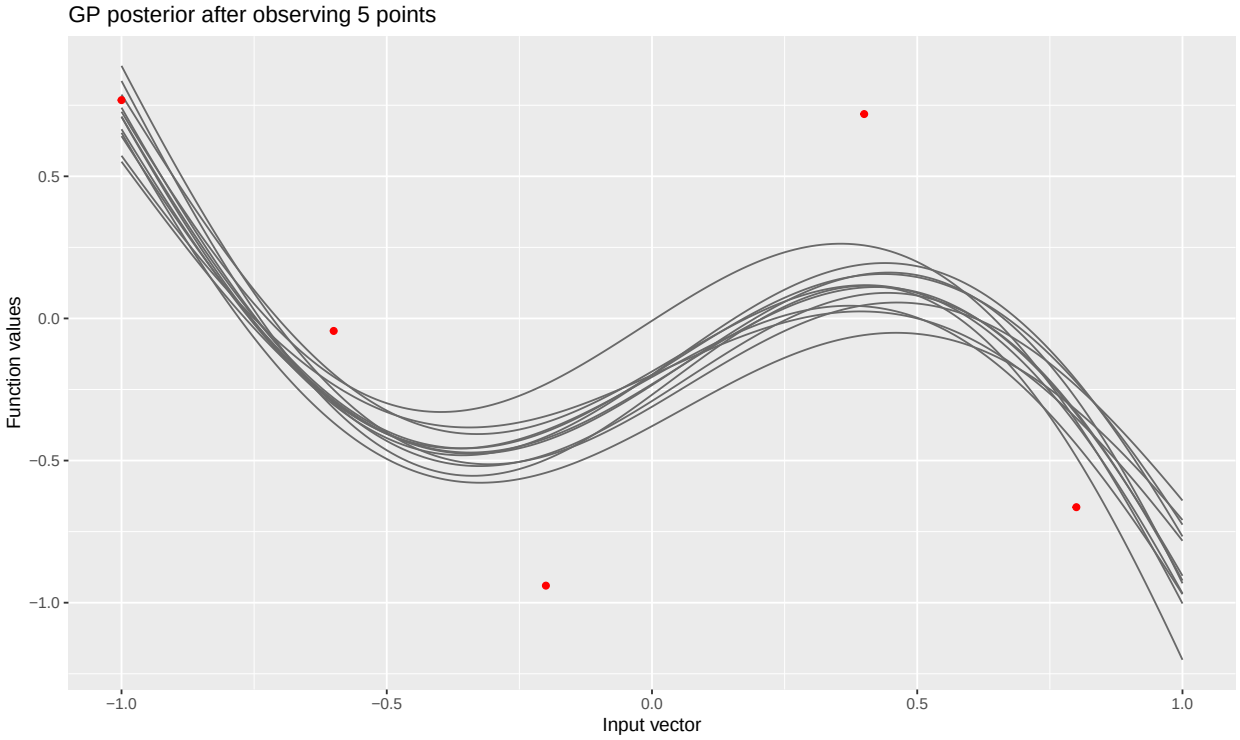


5. Repeat 4. with new hyperparameters.

Repeat (4), this time with hyperparameters $\sigma_f = 1$ *and* $l = 1$. Compare the results.

Solution:





It is obvious that with l=1, the predictive mean of f is much smoother that misses most of the data points. When we set l=1 what we are essentially saying is that the input values variance does not change much even if the points are farther away. That is the reason we see smooth curve. The effect of variance is seen beyond 1 unit. That is why the the posterior mean follows the data but essentially not passes through them.

In previous cases, l was 0.3 so that variance between closer points was more than when $l = 1$. As a result, our function values were effected more when points were closer than when $l = 1$

It would be interesting to see how posterior mean behaves when we have more data .

2. GP Regression with kernlab

In this exercise, you will work with the daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. We have removed the leap year day February 29, 2012 to make things simpler. You can read the dataset with the command:

read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/ Code/TempTullinge.csv", header=TRUE, sep=";")

Create the variable time which records the day number since the start of the dataset (i.e., *time* = 1, 2,..., $365 \times 6 = 2190$). Also, create the variable day that records the day number since the start of each year (i.e., *day* = 1, 2,..., 365, 1, 2,..., 365). Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the data and use only every *fifth* observation. This means that your time and day variables are now *time* = 1, 6,11,..., 2186 and *day* = 1, 6, 11,..., 361, 1, 6, 11,..., 361.

1. Familiarize with the functions *gausspr* and *kernelMatrix* in *kernlab*

Familiarize yourself with the functions *gausspr* and *kernelMatrix* in *kernlab*. Do *?gausspr* and read the input arguments and the output. Also, go through the file *KernLabDemo.R* available on the course website. You will need to understand it. Now, define your own square exponential kernel function (with parameters l(ell) and σ_f (sigmaf)), evaluate it in the point $x = 1, x^{'} = 2$, and use the *kernelMatrix* function to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

Solution:

```
SquaredExpKernel <- function(sigmaF,l){

    rval <- function(x, y=NULL) {
        return(sigmaF^2*exp(-0.5*( (x-y)/l)^2 ))
    }

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

l <- 1
sigmaf <- 2

sq_exp <- SquaredExpKernel(1,0.3)

#Evaluating the kernel in the points x=1 and x'=2
x <- matrix(c(1,3,4),byrow=FALSE)
x_ <- matrix(c(2,3,4),byrow = FALSE)
K <- kernelMatrix(kernel = sq_exp, x = x, y = x_)

cat("Square exp Kernel at points x=1 and x'=2 :", sq_exp(1,2) )
```

```
Square exp Kernel at points x=1 and x'=2 : 0.00386592
```

Covariance between $[1, 3, 4]^T$ and $[2, 3, 4]^T$:

```
             [,1]         [,2]        [,3]
[1,] 3.865920e-03 2.233631e-10 1.92875e-22
[2,] 3.865920e-03 1.000000e+00 3.86592e-03
[3,] 2.233631e-10 3.865920e-03 1.00000e+00
```

2. Gaussian process regression model

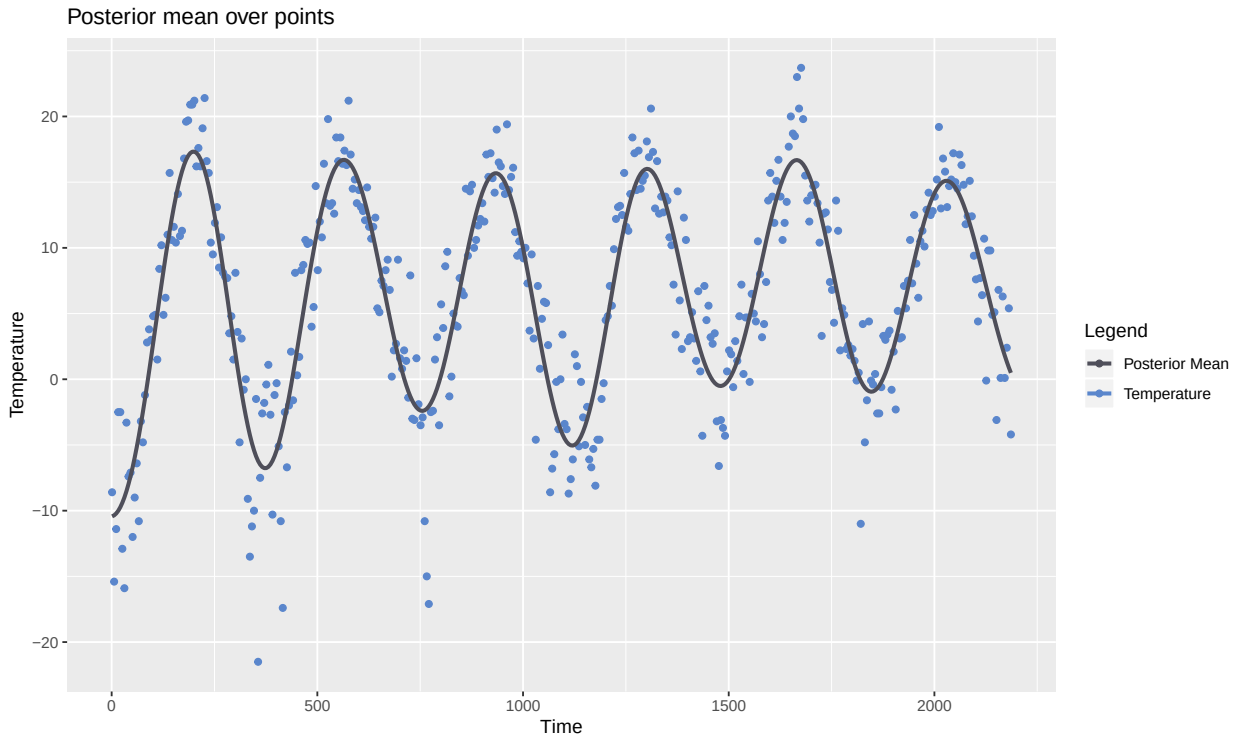
Consider first the following model:

$$temp = f(time) + \epsilon \;\; with \;\; \epsilon \sim N(0, \sigma_n^2) \;\; and \;\; f \sim GP(0, k(time, time^{'}))$$

Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the *lm* function in R). Estimate the above Gaussian process regression model using the squared exponential function from (1) with $\sigma_f = 20$ *and* $l = 0.2$ Use the *predict* function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use *type="l"* in the plot function). Play around with different values on σ_f *and* l and (no need to write this in the report though).

Solution:

```
polyFit <- lm(data_sam$temp ~ data_sam$time + I(data_sam$time^2))
sigmaNoise = sd(polyFit$residuals)

sig_n <- sd(scale(polyFit$residuals))
ell <- 0.2
sigmaF <- 20

GPfit <- gausspr(data_sam$time,data_sam$temp, kernel = SquaredExpKernel,
                         kpar =list(sigmaF,ell),var = sigmaNoise^2)

meanPred <- predict(GPfit, data_sam$time)


data_sam$posmean <- meanPred

cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                    geom_line(aes(y=posmean,colour="Posterior Mean"),size=1.1) +
                    scale_colour_manual(name = "Legend", values = cols)+
                    labs(title = "Posterior mean over points",
                                  x = "Time", y = "Temperature")
```

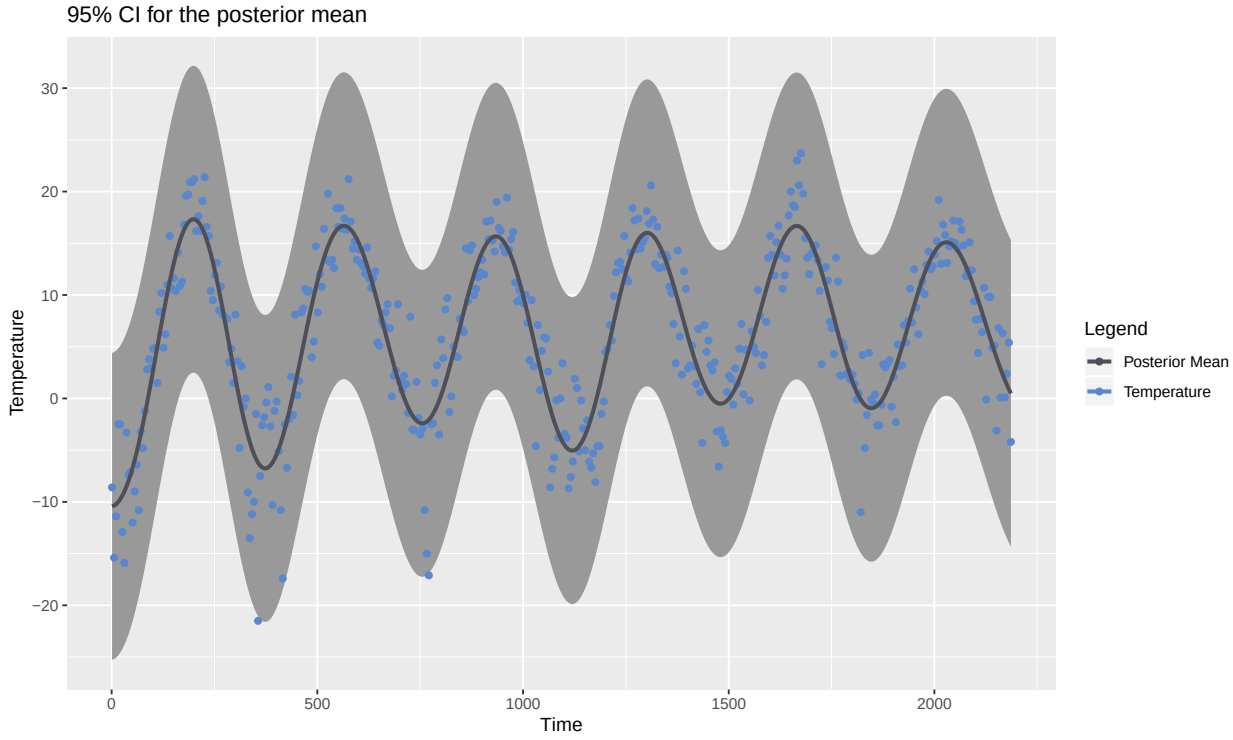


3. Compute the posterior variance

kernlab can compute the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95 % probability (pointwise) bands for f. Superimpose these bands on the figure with the posterior mean that you obtained in (2).

Solution:

```
scale_data <- data_sam[,c(2,3)]
scale_data[,1] <- scale(scale_data[,1])

dat <- posteriorGP(scale_data$time,scale_data$temp,scale_data$time, c(20,0.3),sigmaNoise)


data_sam$low <- data_sam$posmean- 1.96*sqrt(diag(dat[[2]]))
data_sam$high <- data_sam$posmean + 1.96*sqrt(diag(dat[[2]]))


cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_ribbon(aes(ymin = low,
                            ymax = high, time),fill = "grey60") +
                            geom_point(aes(y=temp,colour="Temperature")) +
                            geom_line(aes(y=posmean,colour="Posterior Mean"),size=1.1) +
                            scale_colour_manual(name = "Legend", values = cols) +
                            labs(title = "95% CI for the posterior mean",
                                 x = "Time", y = "Temperature")
```



4. Estimate temperature using day variable

Consider now the following model:

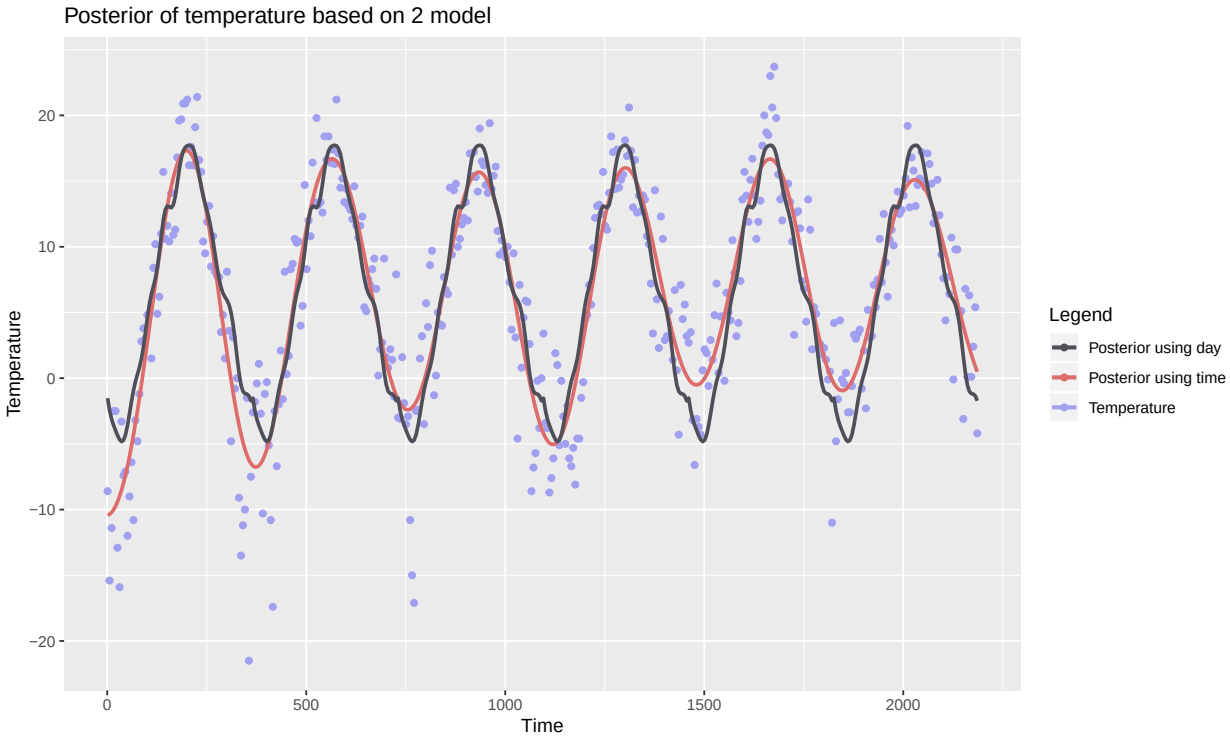
$$temp = f(day) + \epsilon \quad with \quad \epsilon \sim N(0, \sigma_n^2) \quad and \quad f \sim GP(0, k(day, day^{'}))$$

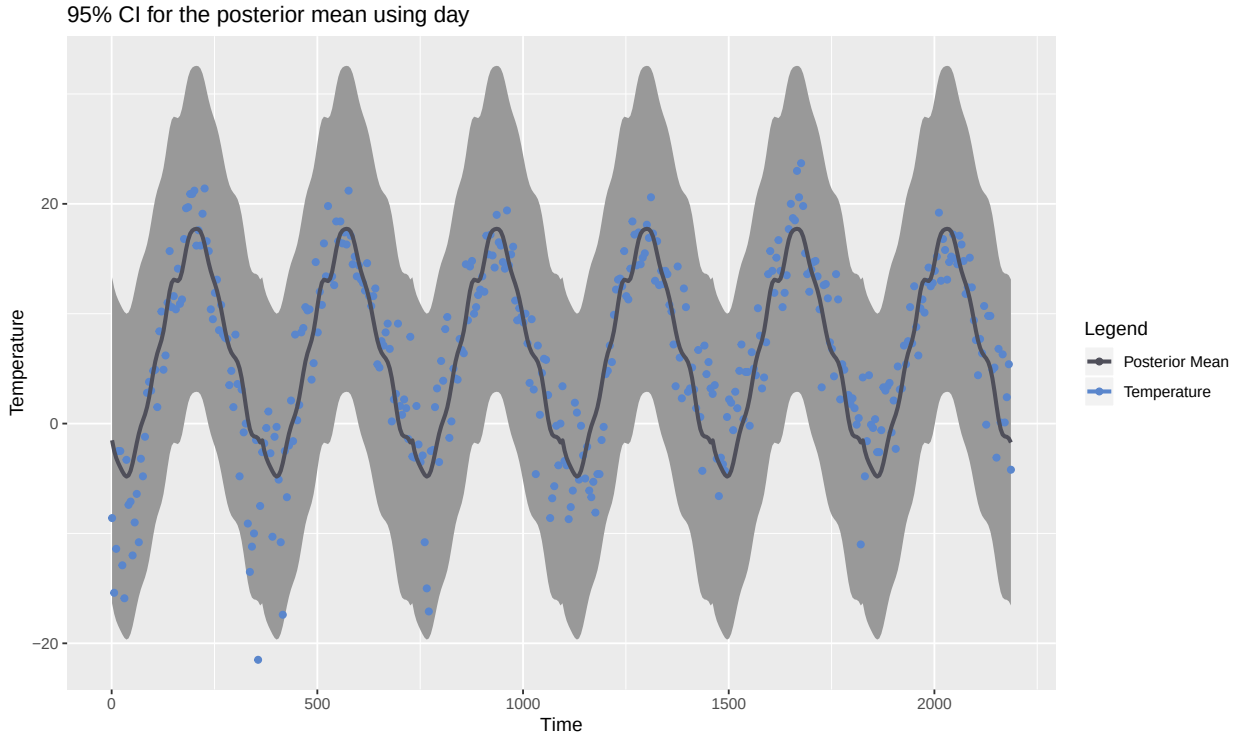
Estimate the model using the squared exponential function with $\sigma_f = 20$ *and* $l = 0.2$ Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis. Compare the results of both models. What are the pros and cons of each model?

Solution:

```
polyFit2 <- lm(data_sam$temp ~ data_sam$day + I(data_sam$day^2))
sigmaNoise = sd(polyFit2$residuals)

ell <- 0.2
sigmaF <- 20

GPfit2 <- gausspr(data_sam$day,data_sam$temp, kernel = SquaredExpKernel,
                        kpar =list(sigmaF,ell),var = sigmaNoise^2)

#calculating the posterior mean
meanPred2 <- predict(GPfit2, data_sam$day)
data_sam$posmean2 <- meanPred2


cols <- c("Temperature"="#A19FF0","Posterior using day"="#4F4F5A",
          "Posterior using time" = "#DD6C6A" )
```

```
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                  geom_line(aes(y=posmean,colour="Posterior using time"),size=1) +
                  geom_line(aes(y=posmean2,colour="Posterior using day"),size=1) +
                  labs(title = "Posterior of temperature based on 2 model ",
                                  x = "Time", y = "Temperature")+
                  scale_colour_manual(name = "Legend", values = cols)
```





The *Time* model process has an advantage in the sense of it can capture a trend isolated to a specific time since the model uses the closest observations in time, whereas the *Day* model assumes that the closest related temperature point is the one on the same day but it belongs for the previous year. The time model is a smoother model so this model try not to capture much possible noise. But does it qualify to be called as underfit ?

Moreover, camparing the 95% CI of temp predicted using day and time, it seems day variable captures data better than time variable.

One advantage of day model over time that we see is that the posterior mean passes through or atleast tries to cover more data trend than time model.

5. Periodic kernel

Finally, implement a generalization of the periodic kernel given in the lectures:

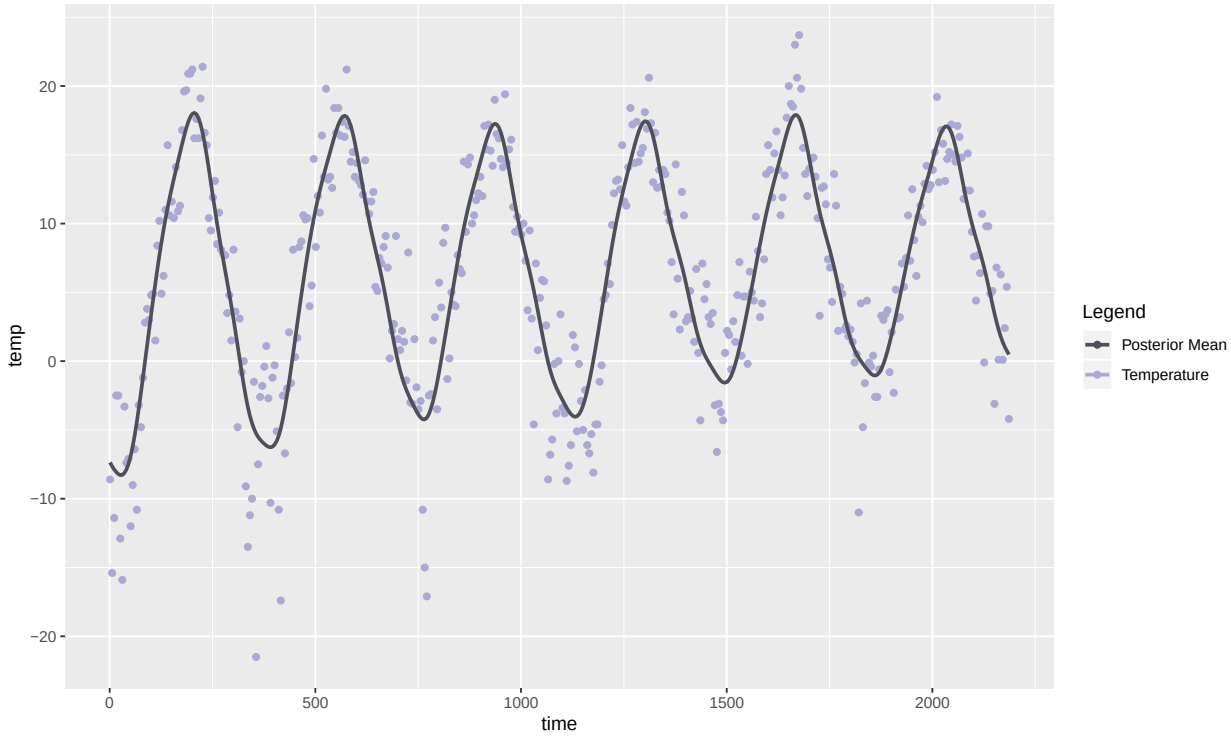
$$k(x, x^{'}) = \sigma_f^2 \; exp\bigg(- \frac{2 * sin^2(\pi|x - x^{'}|/d)}{l_1^2} \bigg) exp\bigg(- \frac{1}{2} \frac{|x - x^{'}|^2}{l_2^2} \bigg)$$

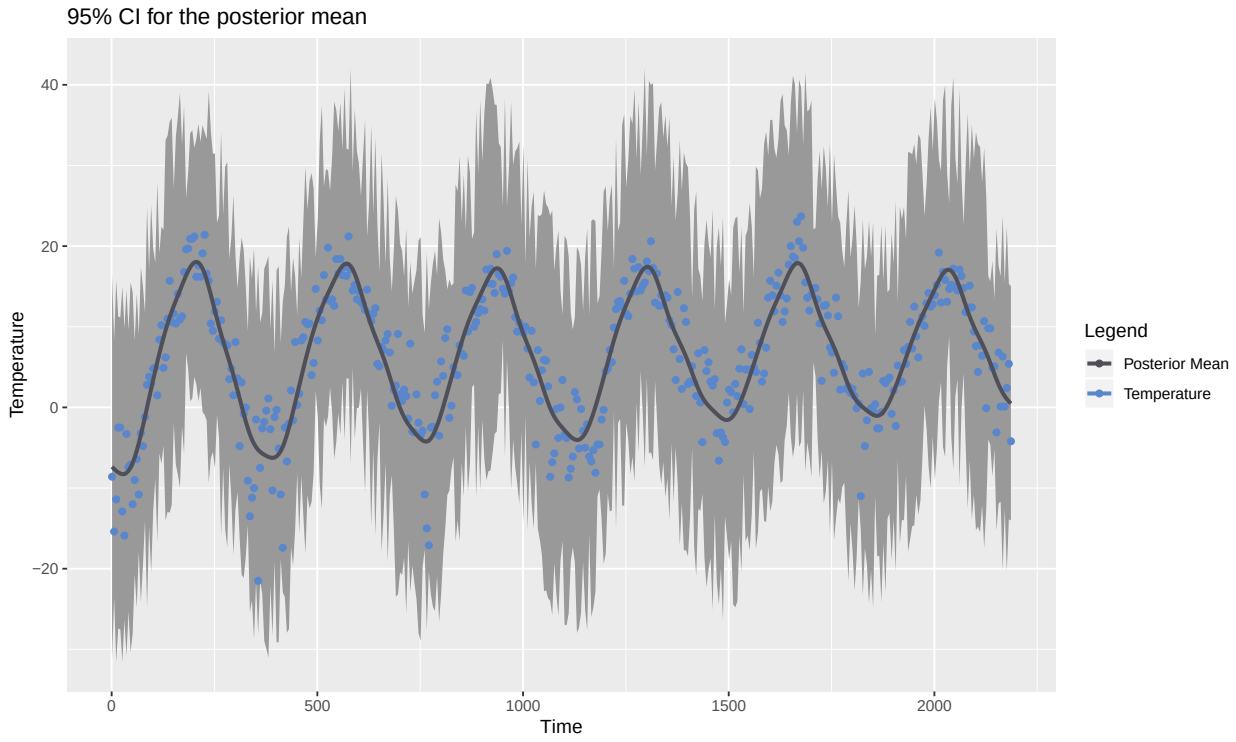
Note that we have two different length scales here, and l_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20, l_1 = 1, l_2 = 10 \; and \; d = 365/sd(time)$. The reason for the rather strange period here is that *kernlab* standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20 \; and \; l = 0.2$. Discuss the results.

Solution:

```
Periodic_kernel <- function(l1,l2,sig_f,d){

    rval <-   function(x, y) {
```

```
            frst_term <- sigmaF^2*exp(-2*(sin(pi*abs(x-y)/d)^2)/(l1^2))
            secnd_term <- exp(-0.5*abs(x-y)/(l2^2))
            return(frst_term*secnd_term)
    }

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

l1 <- 1
l2 <- 10
sig_f <- 20
d <- 365/sd(data_sam$time)


GPfit <- gausspr(data_sam$time,data_sam$temp, kernel = Periodic_kernel,
                         kpar =list(l1,l2,sig_f,d),var = sigmaNoise^2)

meanPred_period <- predict(GPfit, data_sam$time)

data_sam$period_mean <- meanPred_period

cols <- c("Temperature"="#AAA9D5","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                    geom_line(aes(y=period_mean,colour="Posterior Mean"),size=1) +
                    scale_colour_manual(name = "Legend", values = cols)
```





The Periodic kernel seems to catch both variation in day and over time. Periodic kernel probably should be better in capturing the data but it requires more parameter and hence more complex. Getting the parameter right is the key.

But from plotting the 95% CI , it seems it does not capture much of the data as it is done by using squared exponential kernel with time and day as variable.

3. GP Classification with kernlab.

Choose 1000 observations as training data using the following command (i.e., use the vector SelectTraining to subset the training observations):

Solution:

```
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[,5] <- as.factor(data[,5])

set.seed(111);
SelectTraining <- sample(1:dim(data)[1], size = 1000,replace = FALSE)

data_train <- data[SelectTraining,]
```

1. Gaussian process classification model

Use the R package kernlab to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates *varWave* and *skewWave* in the model. Plot contours of the prediction probabilities over a suitable grid of values for *varWave* and *skewWave*. Overlay

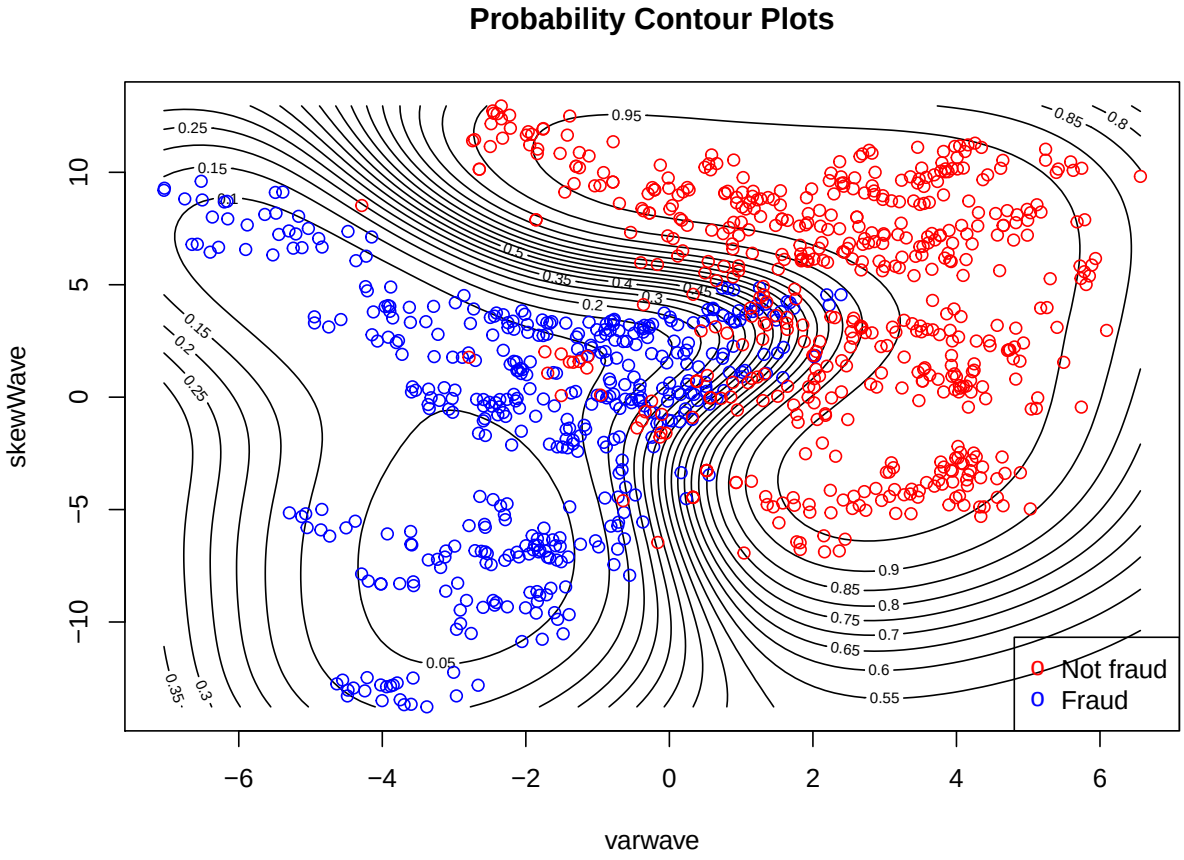
the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file *KernLabDemo.R* available on the course website. Compute the confusion matrix for the classifier and its accuracy.

Solution:

```
GPfitfraud <- gausspr(fraud ~ varWave + skewWave,data=data_train)
```

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

```
predict_fraud <- predict(GPfitfraud,data_train[,1:2],type="probabilities")

x1 <- seq(min(data_train[,1]),max(data_train[,1]),length=100)
x2 <- seq(min(data_train[,2]),max(data_train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data_train)[1:2]
probPreds <- predict(GPfitfraud, gridPoints, type="probabilities")

{contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varwave",
          ylab = "skewWave", main = 'Probability Contour Plots')
points(data_train[data_train[,5]==1,1],data_train[data_train[,5]==1,2],col="blue")
points(data_train[data_train[,5]==0,1],data_train[data_train[,5]==0,2],col="red")
legend("bottomright", pch = c("o","o"),
legend = c("Not fraud", "Fraud"),
col = c("red", "blue"))
}
```




```
Confusion matrix for train data:

          Expected
Prediction   0   1
         0 512  24
         1  44 420


Accuracy for train data : 93.2 %
```

2. Confusion matrix on test data

Using the estimated model from (1), make predictions for the test set. Compute the accuracy.

Solution:

```
Confusion matrix for test data:

          Expected
Prediction   0   1
         0 191   9
         1  15 157
```

```
Accuracy for test data : 93.55 %
```

3. Test on all covariates

Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

Solution:

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

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

```
predict_fraud_new <- predict(GPfitfraud_new,data_test[,1:4])
data_comp <- data.frame("Prediction"=predict_fraud_new, "Expected"= data_test[,5])
conf_mat <- table(data_comp)
```

```
Confusion matrix:

          Expected
Prediction   0   1
         0 205   0
         1   1 166


Accuracy : 99.73 %
```

Appendix

```
knitr::opts_chunk$set(
    message = FALSE,
    warning = FALSE,
    comment = NA,
    fig.width=10,
    fig.height=6
)
library(ggplot2)
library(kableExtra)
library(kernlab)
library(AtmRay)
library(mvtnorm)
library(reshape2)
library(gridExtra)
# Setting up the kernel
sq_kernel = function(x1,x2,sigmaF,l){
  n1 = length(x1)
  n2 = length(x2)
  kern = matrix(NA,n1,n2)
  for (i in 1:n2){
    kern[,i] = sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
    }
  return(kern)
}
# Posterior distribution for GP
posteriorGP = function(x,y,xstar,hyperParam,sigmaNoise){

  # distance (covariance) between training points K(x,x)
  K = sq_kernel(x,x,hyperParam[1],hyperParam[2])
  I =length(x)
  L = t(chol(K + diag(sigmaNoise^2,I)))

  # to compute the predictive mean
  temp = solve(L,y)
  alpha = solve(t(L),temp)

  # distance (covariance) between training and test points K(x,x*)
  K_star = sq_kernel(x,xstar,hyperParam[1],hyperParam[2])

  # predictive mean
  f_bar_star = t(K_star) %*% alpha

  # to compute the predictive variance
  v = solve(L,K_star)

  # distance (covariance) between test points K(x*,x*)
  K_test = sq_kernel(xstar,xstar,hyperParam[1],hyperParam[2])

  # predictive variance
  V_f_star = K_test - t(v) %*% v
```

```
  return(list("mean" = f_bar_star, "var" = V_f_star))
}

# Utility function for the Plot

post_plot = function(x,y,xstar,hyperParam,noise){

  post = posteriorGP(x,y,xstar,hyperParam,noise)
  m = post$mean

  # Plot also 95 % probability (pointwise) bands for f.
  # for pred Y_star band add the noise(i.e the sigma2_n)
  lower = m - 1.96 * sqrt(diag(post$var))
  upper = m + 1.96 * sqrt(diag(post$var))

  cols <- c("Posterior mean"="#4169E1","Observation"="#000000")
  p = ggplot() + geom_ribbon(aes(ymin = lower, ymax = upper, xstar), fill = "grey70") +
                 geom_line(aes(x = xstar, y = m, col = "Posterior mean"),
                 alpha = 1,size = 1) +
                 geom_point(aes(x = x, y = y,  col = "Observation"), alpha = 1) +
                 scale_colour_manual(name = "Legend", values = cols) +
                 labs(title = paste("95% CI for the posterior mean with",
                                    length(x)," observation"),x = "Inputs vector", y = "")+
                 theme_light()

    points <- data.frame(x=x, y=y)
    func_val <- t(rmvnorm(12, post$mean, post$var))
    func_val <- as.data.frame(func_val)
    func_val$xstar <- xstar
    func_val <- melt(func_val,id="xstar")

    plt <- ggplot(func_val,aes(x=xstar,y=value)) +
                geom_line(aes(group=variable), colour="#696969",alpha = 1) +
                geom_point(data=points,aes(x=x,y=y),col="red") +
                ylab("Function values") + xlab("Input vector") +
                ggtitle(paste("GP posterior after observing",length(x),"points"))

  return(list(p,plt))
}



hyperParam = c(sigmaF = 1, l = 0.3)
xstar = seq(-1, 1, 0.01) # X star
noise = 0.1
sigma <- sq_kernel(xstar,xstar,1,0.3)

# Simulate samples
nsim <- 6
func_val <- t(rmvnorm(nsim, rep(0, length(xstar)), sigma))
func_val <- as.data.frame(func_val)
func_val$x <- xstar
func_val <- melt(func_val,id="x")
```

```
ggplot(func_val,aes(x=x,y=value)) + geom_line(aes(group=variable),
                                    colour="#696969",alpha = 1) +
                  ylab("Function values") + xlab("Input vector") +
                  ggtitle("6 sample from GP prior")

# Single Observation
x = c(0.4)
y = c(0.719)

plt <- post_plot(x,y,xstar,hyperParam,noise)
plt[[1]]
plt[[2]]

# 2 Observations
x = c(0.4, -0.6)
y = c(0.719, -0.044)

plt <- post_plot(x,y,xstar,hyperParam,noise)
plt[[1]]
plt[[2]]
# Five data points
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)

plt <- post_plot(x,y,xstar,hyperParam,noise)
plt[[1]]
plt[[2]]
# Five data points with updated l= 1
hyperParam <- c(sigmaF = 1, l = 1)

x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)

plt <- post_plot(x,y,xstar,hyperParam,noise)
plt[[1]]
plt[[2]]

data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")

data$time <- seq(1:nrow(data))
data$day <- ifelse(data$time %% 365 !=0, data$time %% 365,365)

data_sam <- data[seq(1,nrow(data),by=5),]

SquaredExpKernel <- function(sigmaF,l){

    rval <- function(x, y=NULL) {
        return(sigmaF^2*exp(-0.5*( (x-y)/l)^2 ))
    }

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

```
}

l <- 1
sigmaf <- 2

sq_exp <- SquaredExpKernel(1,0.3)

#Evaluating the kernel in the points x=1 and x'=2
x <- matrix(c(1,3,4),byrow=FALSE)
x_ <- matrix(c(2,3,4),byrow = FALSE)
K <- kernelMatrix(kernel = sq_exp, x = x, y = x_)

cat("Square exp Kernel at points x=1 and x'=2 :", sq_exp(1,2) )
K@.Data
polyFit <- lm(data_sam$temp ~ data_sam$time + I(data_sam$time^2))
sigmaNoise = sd(polyFit$residuals)

sig_n <- sd(scale(polyFit$residuals))
ell <- 0.2
sigmaF <- 20

GPfit <- gausspr(data_sam$time,data_sam$temp, kernel = SquaredExpKernel,
                         kpar =list(sigmaF,ell),var = sigmaNoise^2)

meanPred <- predict(GPfit, data_sam$time)


data_sam$posmean <- meanPred

cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                    geom_line(aes(y=posmean,colour="Posterior Mean"),size=1.1) +
                    scale_colour_manual(name = "Legend", values = cols)+
                    labs(title = "Posterior mean over points",
                                 x = "Time", y = "Temperature")

scale_data <- data_sam[,c(2,3)]
scale_data[,1] <- scale(scale_data[,1])

dat <- posteriorGP(scale_data$time,scale_data$temp,scale_data$time, c(20,0.3),sigmaNoise)


data_sam$low <- data_sam$posmean- 1.96*sqrt(diag(dat[[2]]))
data_sam$high <- data_sam$posmean + 1.96*sqrt(diag(dat[[2]]))


cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_ribbon(aes(ymin = low,
                             ymax = high, time),fill = "grey60") +
                             geom_point(aes(y=temp,colour="Temperature")) +
                             geom_line(aes(y=posmean,colour="Posterior Mean"),size=1.1) +
                             scale_colour_manual(name = "Legend", values = cols) +
                             labs(title = "95% CI for the posterior mean",
```

```
                                x = "Time", y = "Temperature")


polyFit2 <- lm(data_sam$temp ~ data_sam$day + I(data_sam$day^2))
sigmaNoise = sd(polyFit2$residuals)

ell <- 0.2
sigmaF <- 20

GPfit2 <- gausspr(data_sam$day,data_sam$temp, kernel = SquaredExpKernel,
                        kpar =list(sigmaF,ell),var = sigmaNoise^2)

#calculating the posterior mean
meanPred2 <- predict(GPfit2, data_sam$day)
data_sam$posmean2 <- meanPred2


cols <- c("Temperature"="#A19FF0","Posterior using day"="#4F4F5A",
          "Posterior using time" = "#DD6C6A" )
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                   geom_line(aes(y=posmean,colour="Posterior using time"),size=1) +
                   geom_line(aes(y=posmean2,colour="Posterior using day"),size=1) +
                   labs(title = "Posterior of temperature based on 2 model ",
                                x = "Time", y = "Temperature")+
                   scale_colour_manual(name = "Legend", values = cols)

scale_data_day <- data_sam[,c(2,4)]
scale_data_day[,1] <- scale(scale_data_day[,1])


pos_day <- posteriorGP(scale_data_day$day,scale_data_day$temp,scale_data_day$day, c(20,0.3),sigmaNoise)


data_sam$low_day <- data_sam$posmean2 - 1.96*sqrt(diag(dat[[2]]))
data_sam$high_day <- data_sam$posmean2 + 1.96*sqrt(diag(dat[[2]]))


cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_ribbon(aes(ymin = low_day,
                            ymax = high_day, time),fill = "grey60") +
                            geom_point(aes(y=temp,colour="Temperature")) +
                            geom_line(aes(y=posmean2,colour="Posterior Mean"),size=1.1) +
                            scale_colour_manual(name = "Legend", values = cols) +
                            labs(title = "95% CI for the posterior mean using day",
                                 x = "Time", y = "Temperature")
Periodic_kernel <- function(l1,l2,sig_f,d){

    rval <-   function(x, y) {
              frst_term <- sigmaF^2*exp(-2*(sin(pi*abs(x-y)/d)^2)/(l1^2))
              secnd_term <- exp(-0.5*abs(x-y)/(l2^2))
              return(frst_term*secnd_term)
    }
```

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

l1 <- 1
l2 <- 10
sig_f <- 20
d <- 365/sd(data_sam$time)


GPfit <- gausspr(data_sam$time,data_sam$temp, kernel = Periodic_kernel,
                         kpar =list(l1,l2,sig_f,d),var = sigmaNoise^2)

meanPred_period <- predict(GPfit, data_sam$time)

data_sam$period_mean <- meanPred_period

cols <- c("Temperature"="#AAA9D5","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_point(aes(y=temp,colour="Temperature")) +
                   geom_line(aes(y=period_mean,colour="Posterior Mean"),size=1) +
                   scale_colour_manual(name = "Legend", values = cols)

scale_data_time <- data_sam[,c(2,3)]
scale_data_time[,1] <- scale(scale_data_time[,1])
#scale_data_time[,2] <- scale(scale_data_time[,2])


Periodic_kernel <- function(l1,l2,sig_f,d,x,y){
      n1 = length(x)
      n2 = length(y)
      kern = matrix(NA,n1,n2)

     for (i in 1:n2){
         frst_term <- sigmaF^2*exp(-2*(sin(pi*abs(x-y[i])/d)^2)/(l1^2))
         secnd_term <- exp(-0.5*abs(x-y[i])/(l2^2))
         kern[,i]<- frst_term*secnd_term
     }

    return(kern)
}

posteriorGP = function(x,y,xstar,sigmaNoise){

   l1 <- 1
   l2 <- 10
   sig_f <- 20
   d <- 365/sd(x)   #not sure if this has to be there ??

  # distance (covariance) between training points K(x,x)
  K <- Periodic_kernel(l1,l2,sig_f,d,x,xstar)
  I =length(x)
  L = t(chol(K + (sigmaNoise^2)*diag(length(x))))
```

```
  # to compute the predictive mean
  temp = solve(L,y)
  alpha = solve(t(L),temp)

  # distance (covariance) between training and test points K(x,x*)
  K_star = Periodic_kernel(l1,l2,sig_f,d,x,y)

  # predictive mean
  f_bar_star = t(K_star) %*% alpha

  # to compute the predictive variance
  v = solve(L,K_star)

  # distance (covariance) between test points K(x*,x*)
  K_test = Periodic_kernel(l1,l2,sig_f,d,xstar,xstar)

  # predictive variance
  V_f_star = K_test - t(v) %*% v

  return(list("mean" = f_bar_star, "var" = V_f_star))
}


pos_day <- posteriorGP(scale_data_time$time,scale_data_time$temp,scale_data_time$time,sigmaNoise)


data_sam$low_day <- meanPred_period - 1.96*sqrt(diag(pos_day[[2]]))
data_sam$high_day <- meanPred_period + 1.96*sqrt(diag(pos_day[[2]]))

cols <- c("Temperature"="#5a86cc","Posterior Mean"="#4F4F5A")
ggplot(data_sam,aes(x=time)) + geom_ribbon(aes(ymin = low_day,
                            ymax = high_day, time),fill = "grey60") +
                            geom_point(aes(y=temp,colour="Temperature")) +
                            geom_line(aes(y=period_mean,colour="Posterior Mean"),size=1.1) +
                            scale_colour_manual(name = "Legend", values = cols) +
                            labs(title = "95% CI for the posterior mean",
                                 x = "Time", y = "Temperature")
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[,5] <- as.factor(data[,5])

set.seed(111);
SelectTraining <- sample(1:dim(data)[1], size = 1000,replace = FALSE)

data_train <- data[SelectTraining,]
GPfitfraud <- gausspr(fraud ~ varWave + skewWave,data=data_train)
predict_fraud <- predict(GPfitfraud,data_train[,1:2],type="probabilities")

x1 <- seq(min(data_train[,1]),max(data_train[,1]),length=100)
x2 <- seq(min(data_train[,2]),max(data_train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
```

```
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(data_train)[1:2]
probPreds <- predict(GPfitfraud, gridPoints, type="probabilities")

{contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varwave",
          ylab = "skewWave", main = 'Probability Contour Plots')
points(data_train[data_train[,5]==1,1],data_train[data_train[,5]==1,2],col="blue")
points(data_train[data_train[,5]==0,1],data_train[data_train[,5]==0,2],col="red")
legend("bottomright", pch = c("o","o"),
legend = c("Not fraud", "Fraud"),
col = c("red", "blue"))
}

predict_fraud <- predict(GPfitfraud,data_train[,1:2])
data_comp <- data.frame("Prediction"=predict_fraud, "Expected"= data_train[,5])
conf_mat <- table(data_comp)

cat("Confusion matrix for train data:\n\n")
conf_mat

cat("\n\nAccuracy for train data :",paste(round(100*(conf_mat[1,1]+
                                  conf_mat[2,2])/nrow(data_train),2)),"%")
data_test <- data[-SelectTraining,]
predict_fraud <- predict(GPfitfraud,data_test[,1:2])
data_comp <- data.frame("Prediction"=predict_fraud, "Expected"= data_test[,5])
conf_mat <- table(data_comp)
cat("Confusion matrix for test data:\n\n")
conf_mat

cat("\n\nAccuracy for test data :",paste(round(100*(conf_mat[1,1]+conf_mat[2,2])/nrow(data_test),2)),"%
GPfitfraud_new <- gausspr(fraud ~ varWave + skewWave + kurtWave+ entropyWave,data=data_train)

predict_fraud_new <- predict(GPfitfraud_new,data_test[,1:4])
data_comp <- data.frame("Prediction"=predict_fraud_new, "Expected"= data_test[,5])
conf_mat <- table(data_comp)
cat("Confusion matrix:\n\n")
conf_mat
cat("\n\nAccuracy :",paste(round(100*(conf_mat[1,1]+conf_mat[2,2])/nrow(data_test),2)),"%")
```