# GP_Lab_GroupReport

*Jasleen(jasma846),Maria(marse306),Tejashree(tejma768),Vinay(vinbe289)*

*October 16, 2019*

## Implementing GP Regression

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

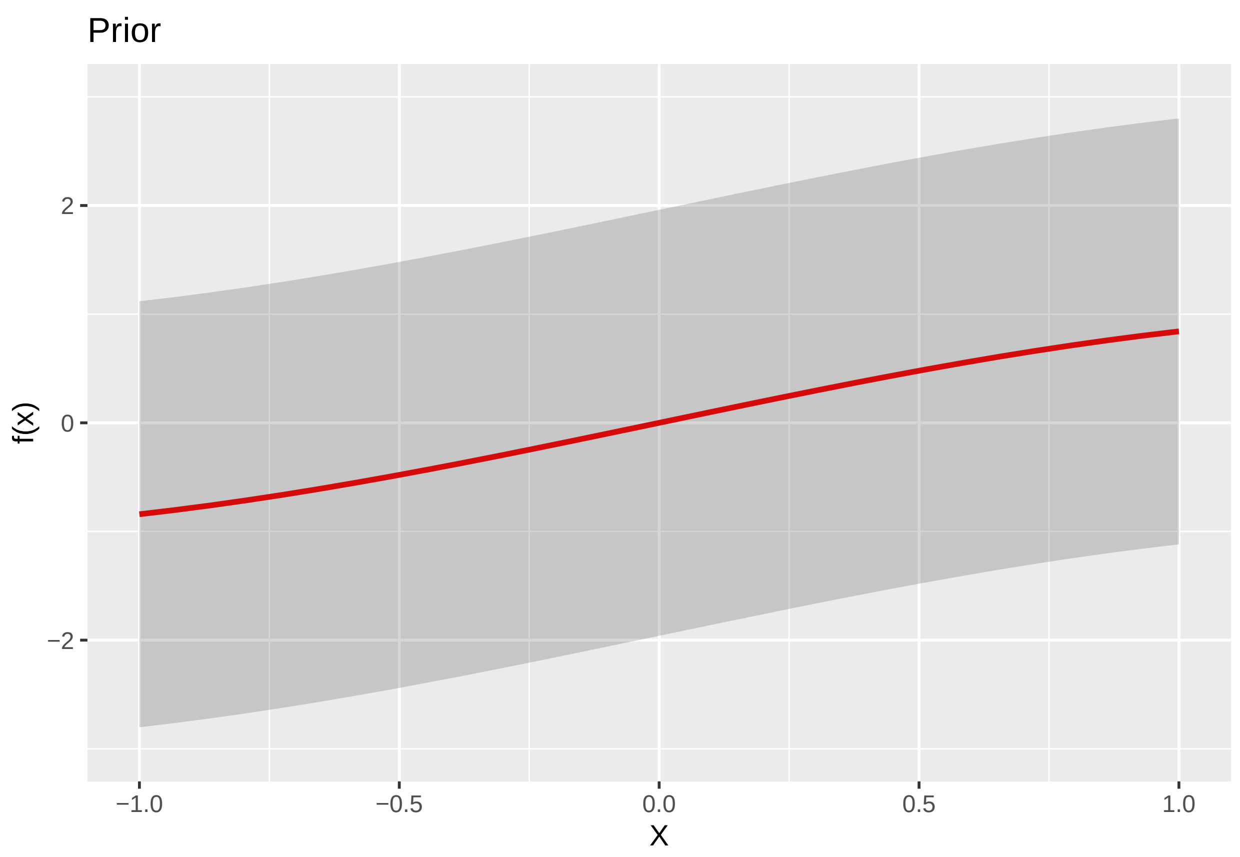
$$y = f(x) + \epsilon; \epsilon = \mathcal{N}(0, \sigma_n^2); f \sim GP(0, k(x, x'))$$

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

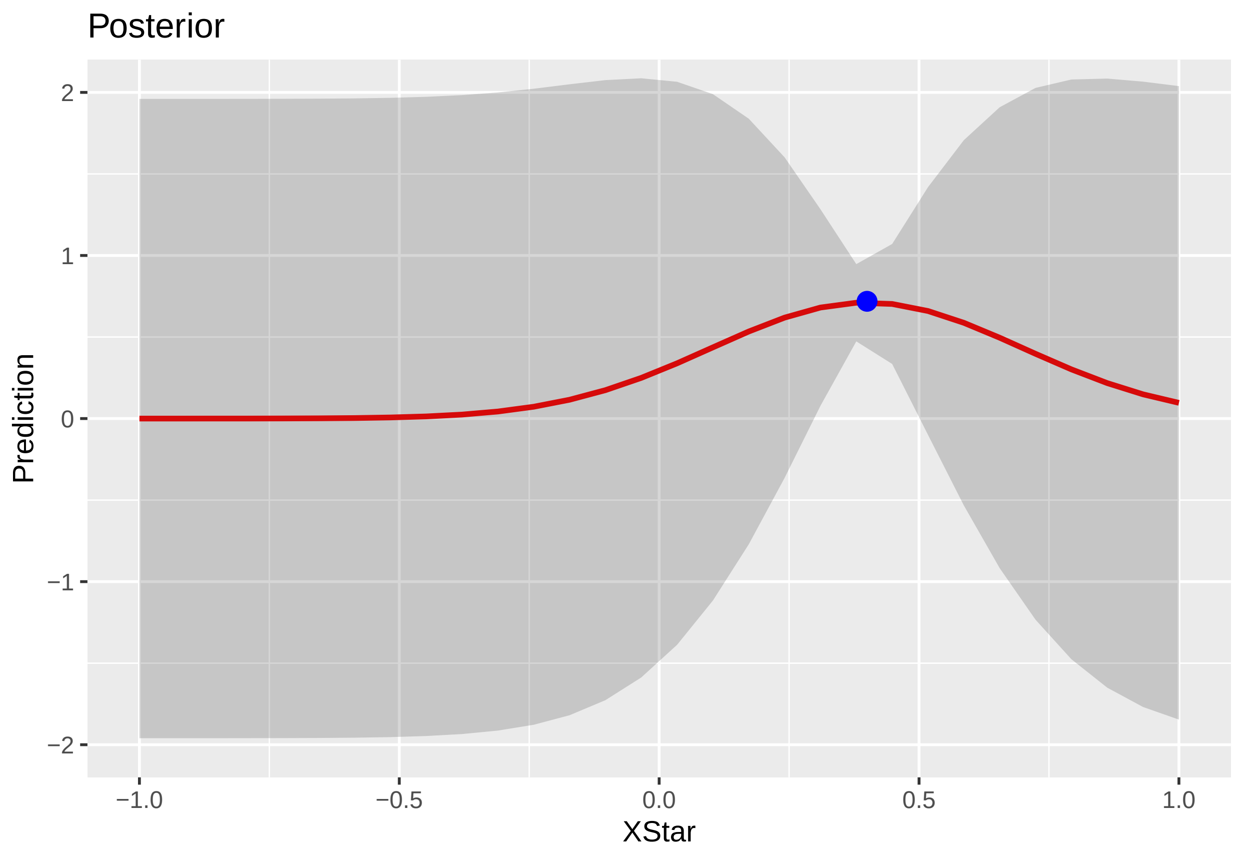
Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it posteriorGP) should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values ($X_*$). You can assume that the prior mean of f is zero for all x. The function should have the following inputs: a. X: Vector of training inputs. b. y: Vector of training targets/outputs. c. XStar: Vector of inputs where the posterior distribution is evaluated, i.e. ($X_*$). d. sigmaNoise: Noise standard deviation $\sigma_n$. e. k: Covariance function or kernel. That is, the kernel should be a separate function (see the file GaussianProcesses.R on the course web page).



```r
posteriorGP = function(X,y,XStar,sigmaNoise,k,...){
  K <- k(X,X,...)
  # Get cholesky decomposition (square root) of the
  # covariance matrix
  L = t(chol(K+sigmaNoise^2*diag(length(X))))
  alpha = solve(t(L),solve(L,y))
  Ks = k(X,XStar,...)
  f_bar= t(Ks)%*%alpha
  v = solve(L,Ks)
  Var = k(XStar,XStar,...) - (t(v)%*%v)
  return(list("mean" = f_bar, "var" = Var))
}
```

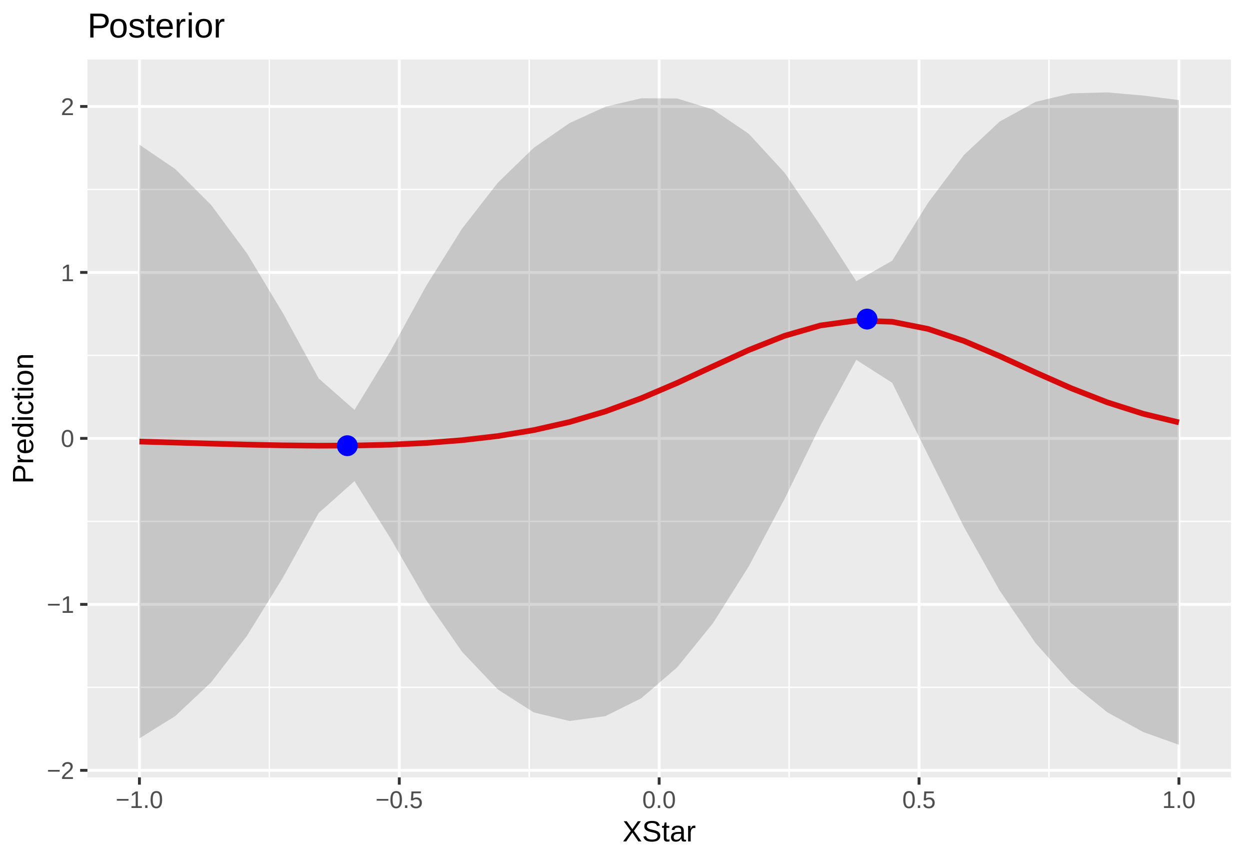
## 2

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 $x \epsilon [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.



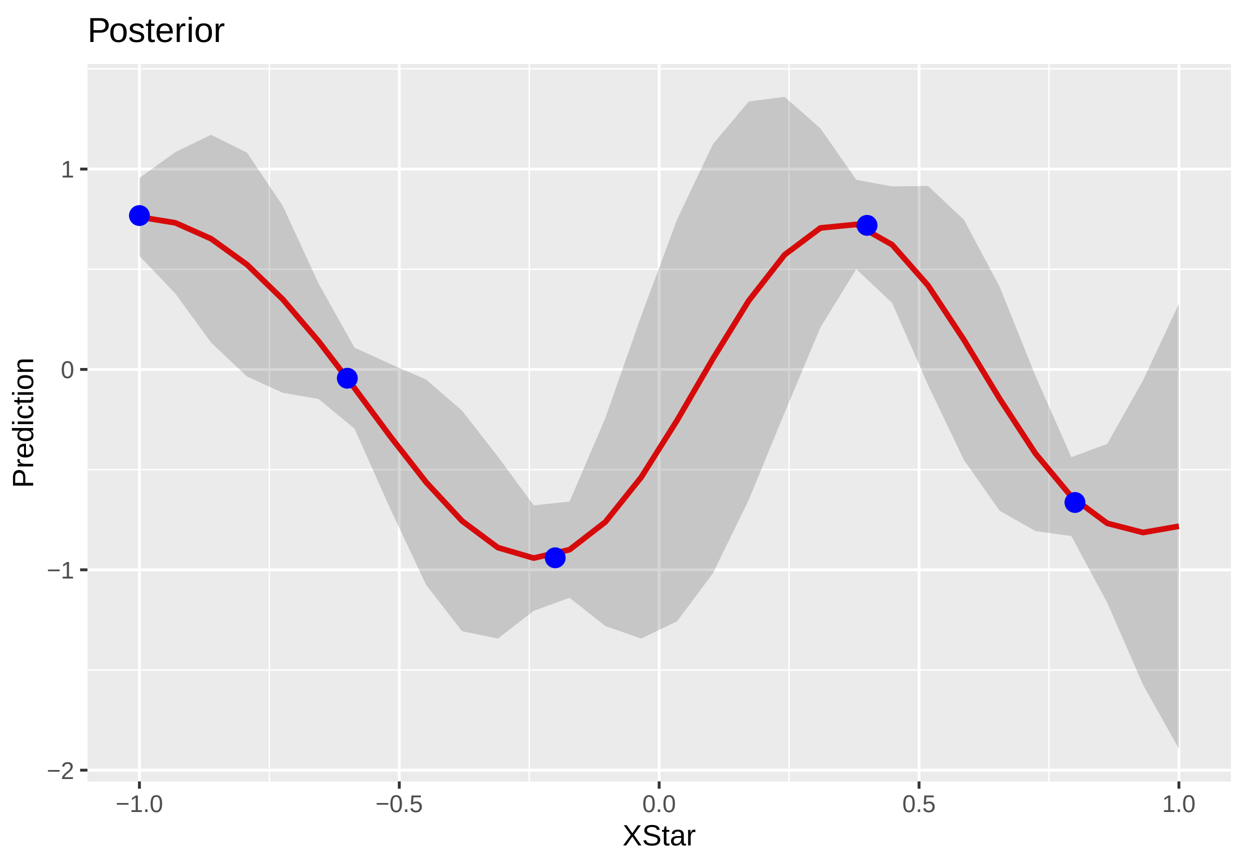
**3**

Update your posterior from (2) with another observation: (x,y) = (-0.6,-0.044). Plot the posterior mean of f over the interval $x\epsilon[-1, 1]$. Plot also 95 % probability (pointwise) bands for f. Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.



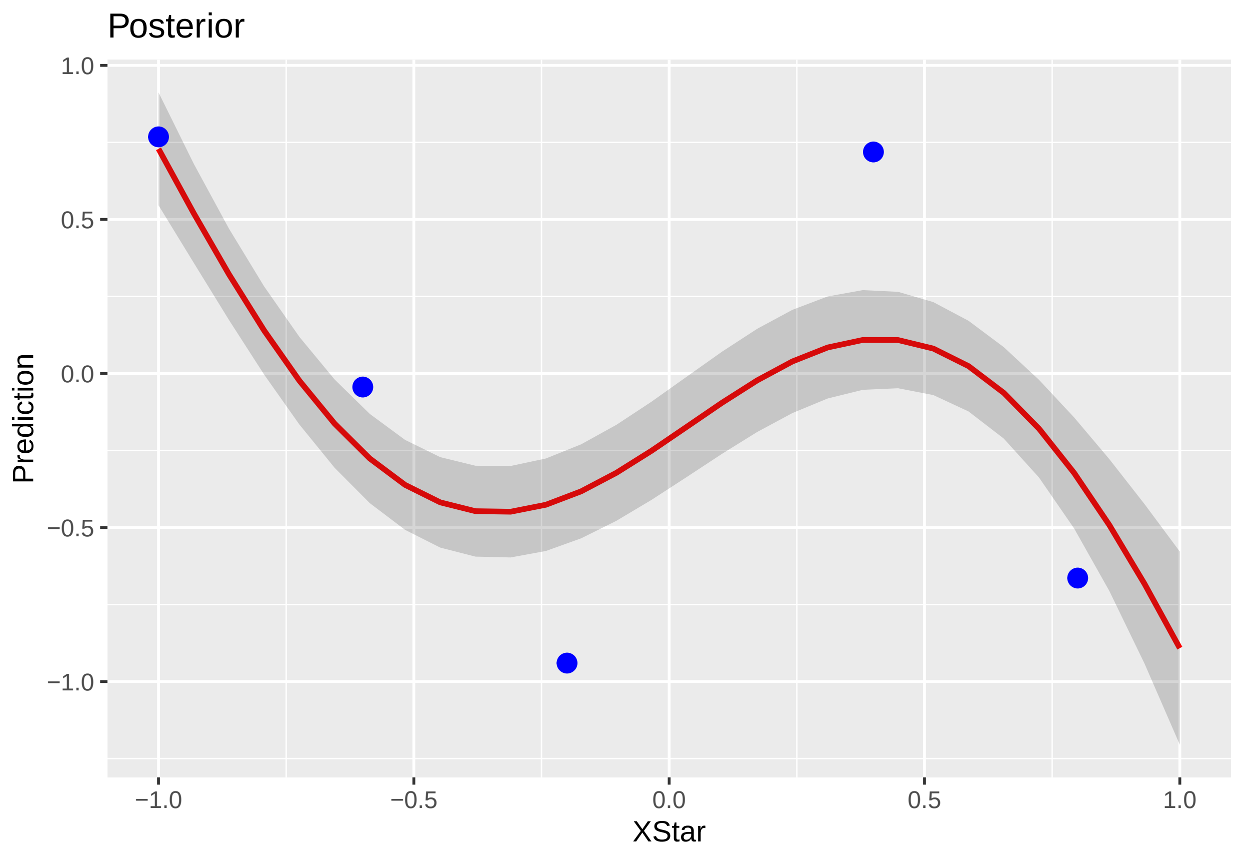
**4**

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). $x \epsilon [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.



**5**

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



Here we see that the mean does not pass through the training points and they are even outside the confidence bands.

The change that we made here compared to (4) is that we change the value of variable l from 0.3 to 1. The variable l in squared exponential kernal is the lengthscale which descibes how smooth the function is or it can be thought of as the distance you have to move before the function value can change significantly. So he as we change the length scale to 1, the function becomes smoother and is not able to find the patterns in the training data

## 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.

Create the variable time which records the day number since the start of the dataset (i.e., time= 1, 2, . . . , 365 ?? 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.

```
##        date  temp time day
## 1  01/01/10  -8.6    1   1
## 6  06/01/10 -15.4    6   6
## 11 11/01/10 -11.4   11  11
## 16 16/01/10  -2.5   16  16
## 21 21/01/10  -2.5   21  21
## 26 26/01/10 -12.9   26  26
```

### 1

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 $\sigma_f$ (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 $X_* = (2, 3, 4)^T$.

```
SquaredExpKernel <- function(sigmaF=1,ell=3)
  {
    rval = function(x1,x2){
      n1 <- length(x1)
      n2 <- length(x2)
      K <- matrix(NA,n1,n2)
      for (i in 1:n2){
        K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
      }
      return(K)
    }
   class(rval) <- "kernel"
   return(rval)
}
```

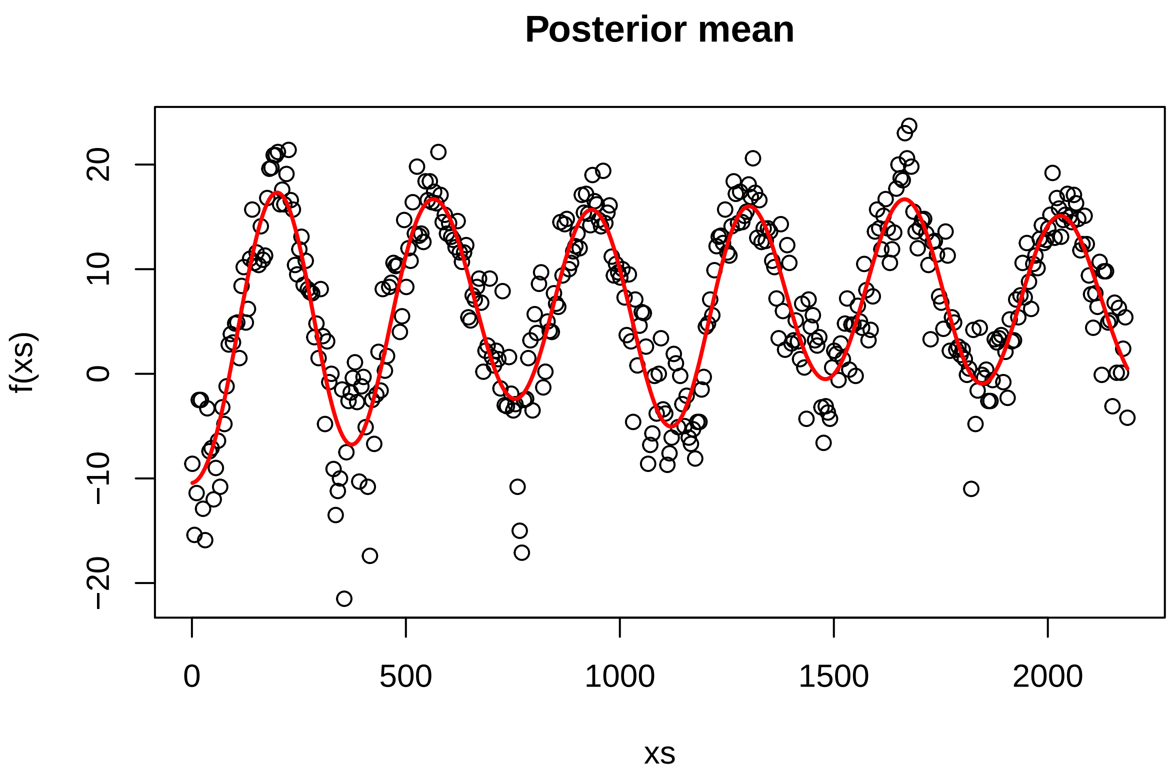
```
##
## Evaluating the kernel in x=1, x'=2 with sigmaF = 20,ell = 0.2

##             [,1]
## [1,] 0.001490661

##
## Covariance Matrix with sigmaF = 20,ell = 0.2

## An object of class "kernelMatrix"
##              [,1]         [,2]         [,3]
## [1,] 1.490661e-03 7.714999e-20 5.545373e-47
## [2,] 1.490661e-03 4.000000e+02 1.490661e-03
## [3,] 7.714999e-20 1.490661e-03 4.000000e+02
```

### 2

Consider first the following model:

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

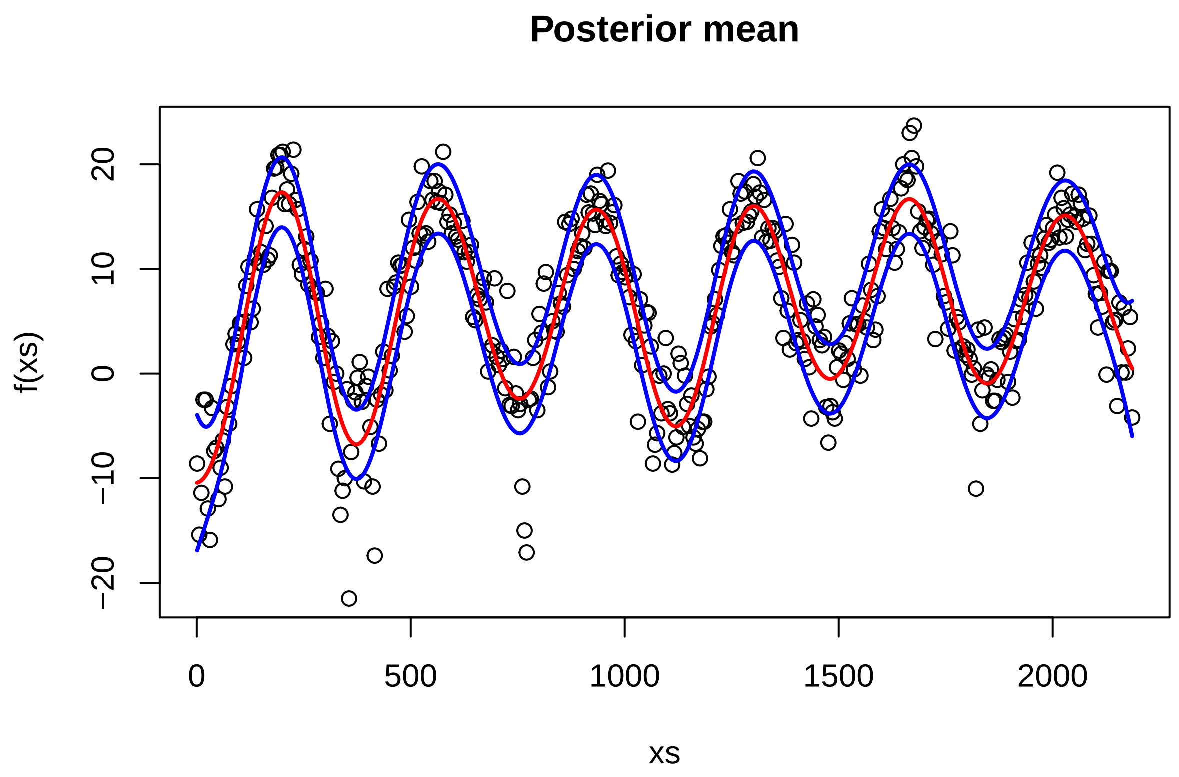
Let $\sigma_n^2$ be the residual variance from a simple quadratic regression fit (using the lm function in R). Estimate the above Gaussian process regression model using the squared exponential function from (1) with $\sigma_f = 20, l = 0.2$. Use the predict function in R to compute the posterior mean 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 $\sigma_f, l$ (no need to write this in the report though).



**3**

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).

**Hint**: Note that Algorithm 2.1 on page 19 of Rasmussen and Willams??? book already does the calculations required. Note also that kernlab scales the data by default to have zero mean and standard deviation one. So, the output of your implementation of Algorithm 2.1 will not coincide with the output of kernlab unless you scale the data first. For this, you may want to use the R function scale. When plotting the results, do not forget to unscale by multiplying with the standard deviation and adding the mean.

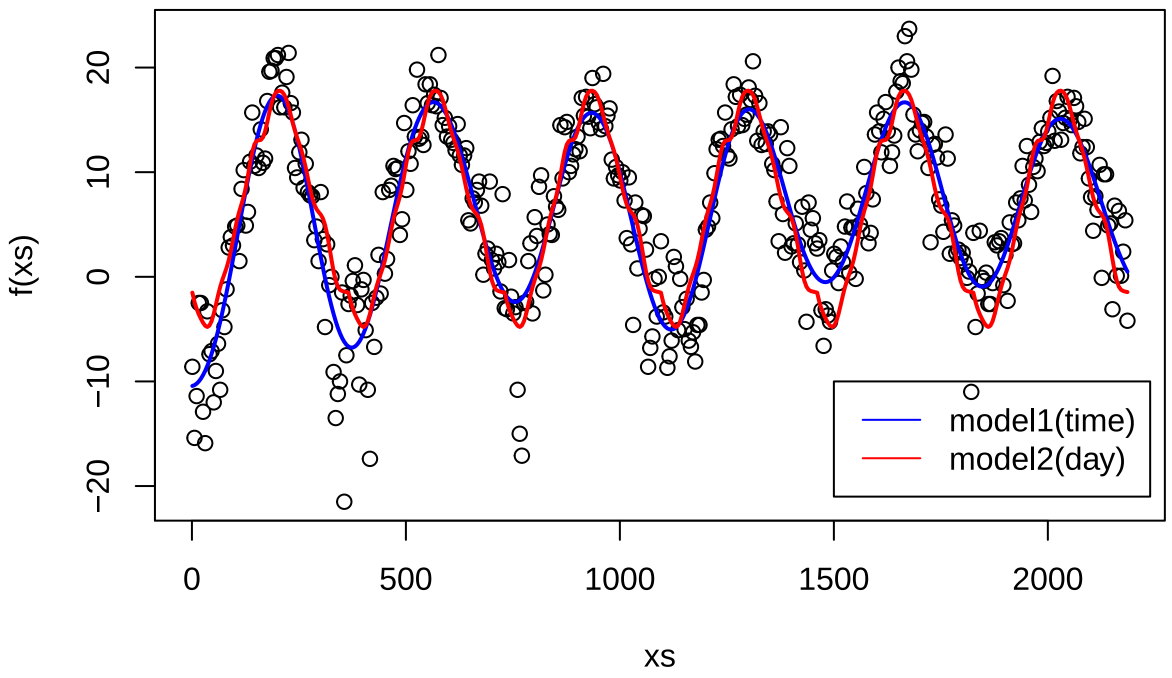


**4**

Consider now the following model:

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

Estimate the model using the squared exponential function with $\sigma_f = 20, l = 0.2$. 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?



**Pros and Cons**

The time model is a smoother so this model try to capture less noise. But one disadvantage of this kind of models is that it could even underfit the data.

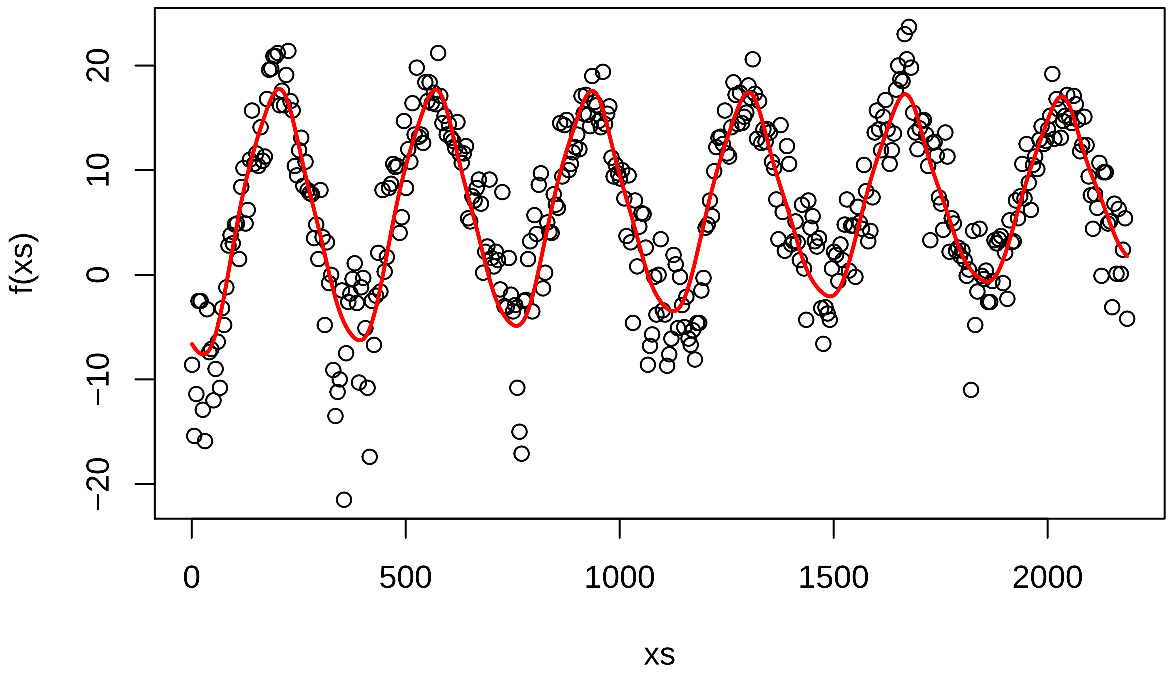
The day model gives a slightly better fit which could be explained by the fact that day gives us some periodic information and temperature is a periodically changing variable. One drawback of this compared to the other model is that the curve is more wiggly and hence this has chance of capturing more noise.

**5**

Finally, implement a generalization of the periodic kernel given in the lectures: $k(x,x') = \sigma_f^2 exp\{-\frac{2sin^2(\pi|x-x'|/d)}{l_1^2}\}.\{-\frac{1}{2}\frac{|x-x'|^2}{l_2^2}\}$

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 $\sigma_f = 20, l = 0.2$). Discuss the results.

```
periodicKernal = function(sigmaF, l1, l2, d){
  rval = function(x1,x2){
      K <- matrix(NA,length(x1),length(x2))
      for (i in 1:length(x2)){

        n = exp(-2*((sin(pi*(abs(x1 - x2[i]))/d))/l1)^2)*exp(-0.5*((x1-x2[i])/l2)^2)
        K[,i] <- sigmaF^2*n
      }
      return(K)
    }
   class(rval) <- "kernel"
   return(rval)

}
```



The output from periodic model here is almost similar to the other two models but looks slightly better. Since temperature is a periodically changing variable, using periodic kernal in gaussian process can help in capturing the periodic pattern and thus building better model.

## GP Classification with Kernlab

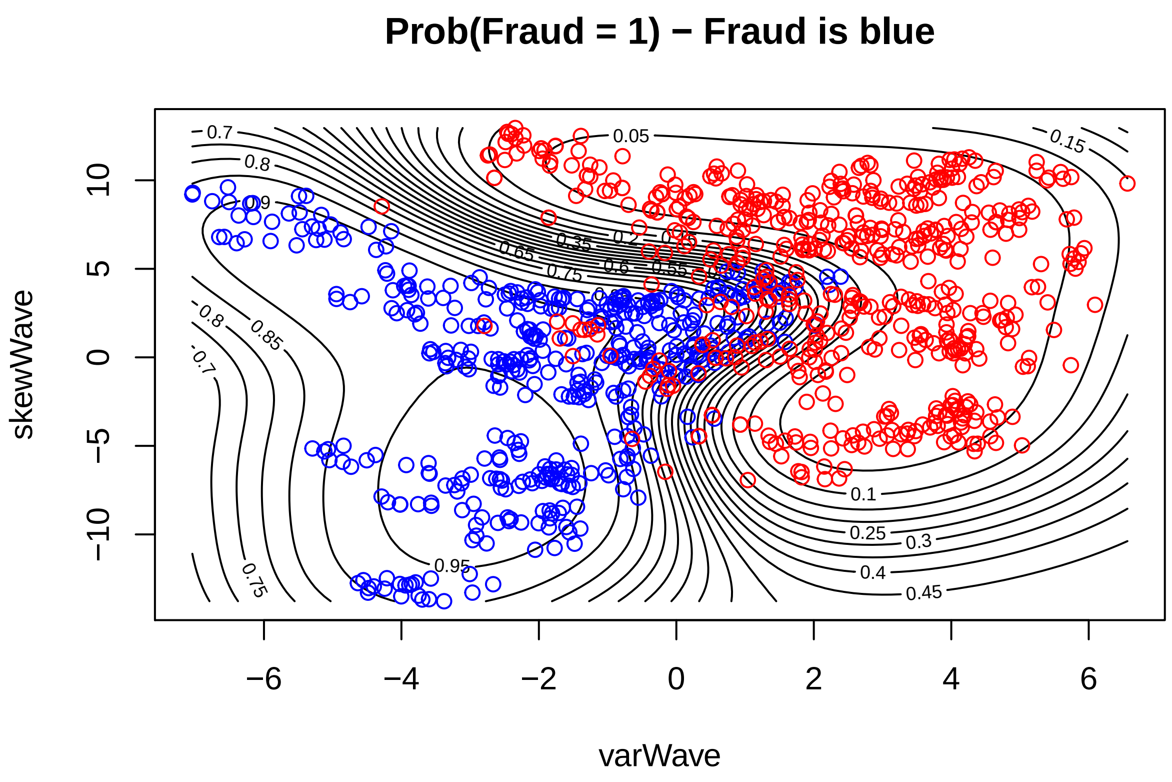
Download the banknote fraud data. Choose 1000 observations as training data using the following command (i.e., use the vector SelectTraining to subset the training observations):

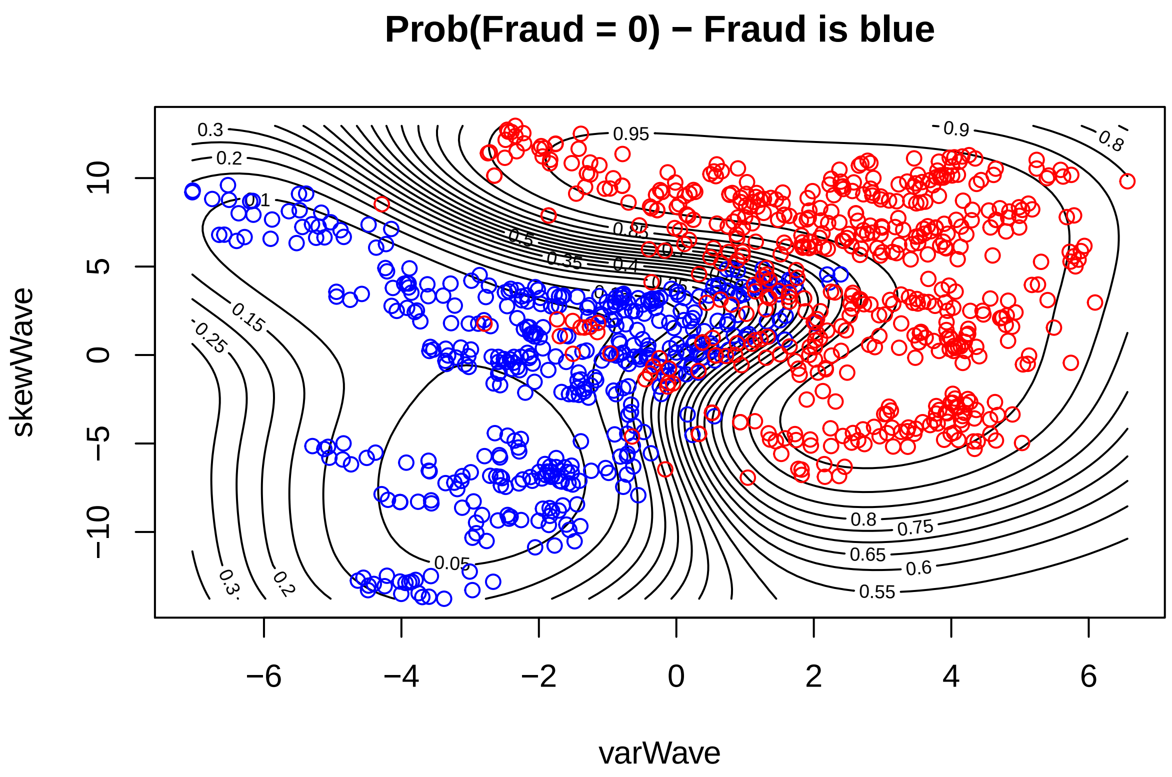
### 1

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.

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

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





```
##
## confusion matrix
##
##       0   1
##   0 512  24
##   1  44 420
##
##
##  Accuracy =  93.2 %
```

**2**

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

```
##
## confusion matrix - Test set
##
##       0   1
##   0 191   9
##   1  15 157
##
## Accuracy of test data=  93.54839 %
```

### 3

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

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

```
##
## confusion matrix - Train data
```

```
##
##       0   1
##   0 552   0
##   1   4 444
```

```
##
## Accuracy Train data=  99.6 %
```

```
##
##
## confusion matrix - Test data
```

```
##
##       0   1
##   0 205   0
##   1   1 166
```

```
##
## Accuracy Test data =  99.73118 %
```

The model accuracy has increased to 99.6% from 93.2 in previous model and test accuracy to 99.7% from 93.5 in previous model. This means that all the covariates are correlated to variable fraud.

## Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(mvtnorm)
library(ggplot2)
#Credit : GaussianProcess.R

SqExpKernel<- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

# Mean function
MeanFunc <- function(x){
  m <- sin(x)
  return(m)
}
```

```r
xGrid <- seq(-1,1,length=30)

# Plotting one draw
sigmaF <- 1
l <- 1
lower = MeanFunc(xGrid) - 1.96*sqrt(diag(SqExpKernel(xGrid,xGrid,sigmaF,l)))
upper = MeanFunc(xGrid) + 1.96*sqrt(diag(SqExpKernel(xGrid,xGrid,sigmaF,l)) )
ggplot() +
  geom_line(aes(x = xGrid, y=MeanFunc(xGrid)), colour="red" , size = 1) +
  geom_ribbon(aes(x = xGrid,ymin=lower, ymax=upper), alpha=0.2)+
  coord_cartesian(ylim = c(-3,3), xlim = c(-1, 1))+
  ggtitle("Prior") + ylab("f(x)") + xlab("X")



posteriorGP = function(X,y,XStar,sigmaNoise,k,...){
  K <- k(X,X,...)
  # Get cholesky decomposition (square root) of the
  # covariance matrix
  L = t(chol(K+sigmaNoise^2*diag(length(X))))
  alpha = solve(t(L),solve(L,y))
  Ks = k(X,XStar,...)
  f_bar= t(Ks)%*%alpha
  v = solve(L,Ks)
  Var = k(XStar,XStar,...) - (t(v)%*%v)
  return(list("mean" = f_bar, "var" = Var))
}

sigman = 0.1
XStar = seq(-1,1,length = 30)

X = c(0.4)
y = c(0.719)

post = posteriorGP(X,y,XStar,sigman,SqExpKernel,sigmaF = 1, l = 0.3)

upper = post$mean + 1.96*sqrt(diag(post$var))
lower = post$mean - 1.96*sqrt(diag(post$var))

ggplot() +
  geom_line(aes(x = XStar, y=post$mean), colour="red" , size = 1) +
  geom_ribbon(aes(x = XStar,ymin=lower, ymax=upper), alpha=0.2)+
  geom_point(aes(x = X ,y = y),shape=19, fill="blue", color="blue", size=3)+
  coord_cartesian(ylim = c(-2,2), xlim = c(-1, 1))+
  ggtitle("Posterior") + ylab("Prediction")



X = c(0.4 , -0.6)
y= c(0.719 , -0.044)


post = posteriorGP(X,y,XStar,sigman,SqExpKernel,sigmaF = 1, l = 0.3)
```

```r
upper = post$mean + 1.96* sqrt(diag(post$var))
lower = post$mean - 1.96* sqrt(diag(post$var))

ggplot() +
  geom_line(aes(x = XStar, y=post$mean), colour="red" , size = 1) +
  geom_ribbon(aes(x = XStar,ymin=lower, ymax=upper), alpha=0.2)+
  geom_point(aes(x = X ,y = y),shape=19, fill="blue", color="blue", size=3)+
  ggtitle("Posterior") + ylab("Prediction")

X = c(-1.0,-0.6,-0.2,0.4,0.8)
y = c(0.768,-0.044,-0.940,0.719,-0.664)

post = posteriorGP(X,y,XStar,sigman,SqExpKernel,sigmaF = 1, l = 0.3)

upper = post$mean + 1.96* sqrt(diag(post$var))
lower = post$mean - 1.96* sqrt(diag(post$var))


ggplot() +
  geom_line(aes(x = XStar, y=post$mean), colour="red" , size = 1) +
  geom_ribbon(aes(x = XStar,ymin=lower, ymax=upper), alpha=0.2)+
  geom_point(aes(x = X ,y = y),shape=19, fill="blue", color="blue", size=3)+
  ggtitle("Posterior") + ylab("Prediction")
post = posteriorGP(X,y,XStar,sigman,SqExpKernel,sigmaF = 1, l = 1)

upper = post$mean + 1.96* sqrt(diag(post$var))
lower = post$mean - 1.96* sqrt(diag(post$var))


ggplot() +
  geom_line(aes(x = XStar, y=post$mean), colour="red" , size = 1) +
  geom_ribbon(aes(x = XStar,ymin=lower, ymax=upper), alpha=0.2)+
  geom_point(aes(x = X ,y = y),shape=19, fill="blue", color="blue", size=3)+
  ggtitle("Posterior") + ylab("Prediction")
library(kernlab)
tempdata0 = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/
Code/TempTullinge.csv", header=TRUE, sep=";")
tempdata0$time = seq(1,2190)
tempdata0$day = rep(seq(1,365),times = 6)
tempdata = tempdata0[seq(1, length(tempdata0$day), 5),]

time = tempdata$time
day = tempdata$day
temp = tempdata$temp
head(tempdata)
SquaredExpKernel <- function(sigmaF=1,ell=3)
  {
    rval = function(x1,x2){
      n1 <- length(x1)
      n2 <- length(x2)
      K <- matrix(NA,n1,n2)
      for (i in 1:n2){
        K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
      }
      return(K)
    }
   class(rval) <- "kernel"
   return(rval)
}


X = matrix(c(1,3,4))
Xstar = matrix(c(2,3,4))

SquaredExpKernelFn = SquaredExpKernel(sigmaF = 20,ell = 0.2)

cat("\nEvaluating the kernel in x=1, x'=2 with sigmaF = 20,ell = 0.2\n")
SquaredExpKernelFn(1,2)# Evaluating the kernel in x=1, x'=2

# Computing the whole covariance matrix K from the kernel.
K <- kernelMatrix(kernel = SquaredExpKernelFn, x = X, y = Xstar) # So this is K(X,Xstar)

cat("\nCovariance Matrix with sigmaF = 20,ell = 0.2\n\n")

print(K)
#residual variance
polyFit <- lm(temp ~  time+I(time^2))
sigman = sd(polyFit$residuals)

# Fit the model with squared exponential kernal from (1)
sigmaF <- 20
ell <- 0.2

GPfit <- gausspr(time, temp, kernel = SquaredExpKernel,
                 kpar = list(sigmaF = sigmaF, ell=ell), var = sigman^2)
meanPred <- predict(GPfit, time)
plot(time,temp, xlab = "xs" , ylab = "f(xs)", main = "Posterior mean")
lines(time, meanPred, col="red", lwd = 2 )
#Implementation using PosteriorGP function

post = posteriorGP(scale(time),scale(temp),scale(time),
                   sigman,SqExpKernel,sigmaF = 20, l = 0.2)
xs = time

plot(xs,temp ,
      xlab = "xs" , ylab = "f(xs)", main = "Posterior mean")
lines(xs, meanPred, col="red", lwd = 2)


varP = sqrt(diag(post$var))

# Probability intervals for fStar
lines(xs, meanPred - 1.96*varP, col = "blue", lwd = 2)
lines(xs, meanPred + 1.96*varP, col = "blue", lwd = 2)

#residual variance
```

```r
polyFit <- lm(temp ~  day)
sigman = sd(polyFit$residuals)

GPfit <- gausspr(day, temp, kernel = SquaredExpKernel,
                 kpar = list(sigmaF = 20, ell=0.2), var = sigman^2)
meanPred_1 <- predict(GPfit, day)
plot(time,temp , xlab = "xs" , ylab = "f(xs)")
lines(time, meanPred, col="blue", lwd = 2)
lines(time,meanPred_1, col = "red" , lwd = 2)
legend(1500,-10 , legend = c("model1(time)" , "model2(day)") ,
       col = c("blue" , "red") , lty = c(1,1) )


periodicKernal = function(sigmaF, l1, l2, d){
  rval = function(x1,x2){
      K <- matrix(NA,length(x1),length(x2))
      for (i in 1:length(x2)){

        n = exp(-2*((sin(pi*(abs(x1 - x2[i]))/d))/l1)^2)*exp(-0.5*((x1-x2[i])/l2)^2)
        K[,i] <- sigmaF^2*n
      }
      return(K)
    }
   class(rval) <- "kernel"
   return(rval)

}
#residual variance
polyFit <- lm(temp ~  time + I(time^2))
sigman = sd(polyFit$residuals)
d = 365/sd(time)
GPfit <- gausspr(time, temp, kernel = periodicKernal,
        kpar = list(sigmaF = 20, l1 = 1, l2 = 10, d =d ), var = sigman^2)
meanPred <- predict(GPfit, time)
plot(time,temp , xlab = "xs" , ylab = "f(xs)")
lines(time, meanPred, col="red", lwd = 2)
library(AtmRay)

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)
train = data[SelectTraining,]
test = data[-c(SelectTraining),]
GPfitClass <- gausspr(fraud ~  varWave + skewWave, data=train)

x1 <- seq(min(train[,1]),max(train[,1]),length=100)
x2 <- seq(min(train[,2]),max(train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]

probPreds <- predict(GPfitClass, gridPoints, type="probabilities")

# Plotting for Prob(Fraud = 1)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 1) - Fraud is blue')
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")


# Plotting for Prob(Fraud = 0)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 0) - Fraud is blue')
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")



cm = table(predict(GPfitClass,train[,1:2]), train[,5])

cat("\nconfusion matrix\n\n")
cm

accuracy = sum(diag(cm))/sum(cm)
cat("\n\n Accuracy = ",accuracy*100,"%")
cm = table(predict(GPfitClass,test[,1:2]), test[,5])
cat("\nconfusion matrix - Test set\n")
cm

accuracy = sum(diag(cm))/sum(cm)
cat("\nAccuracy of test data= ",accuracy*100,"%")
GPfitClass <- gausspr(fraud ~., data=train)

cm = table(predict(GPfitClass,train[,1:4]), train[,5])
cat("\nconfusion matrix - Train data\n\n")

cm

accuracy = sum(diag(cm))/sum(cm)
cat("\nAccuracy Train data= ",accuracy*100,"%")

cm = table(predict(GPfitClass,test[,1:4]), test[,5])
cat("\n\nconfusion matrix - Test data\n\n")

cm

accuracy = sum(diag(cm))/sum(cm)
cat("\nAccuracy Test data = ",accuracy*100,"%")
```