Advanced Machine Learning Lab4

Omkar Bhutra (omkbh878), Obaid Ur Rehman (obaur539) ,Ruben Jared Muñoz Roquet (rubmu773)

October 16, 2019

Q1. GP Regression

2.1 Implement GP Regression

1)

```
library("mvtnorm")
```

```
## Warning: package 'mvtnorm' was built under R version 3.5.3
```

```
library(kernlab)
```

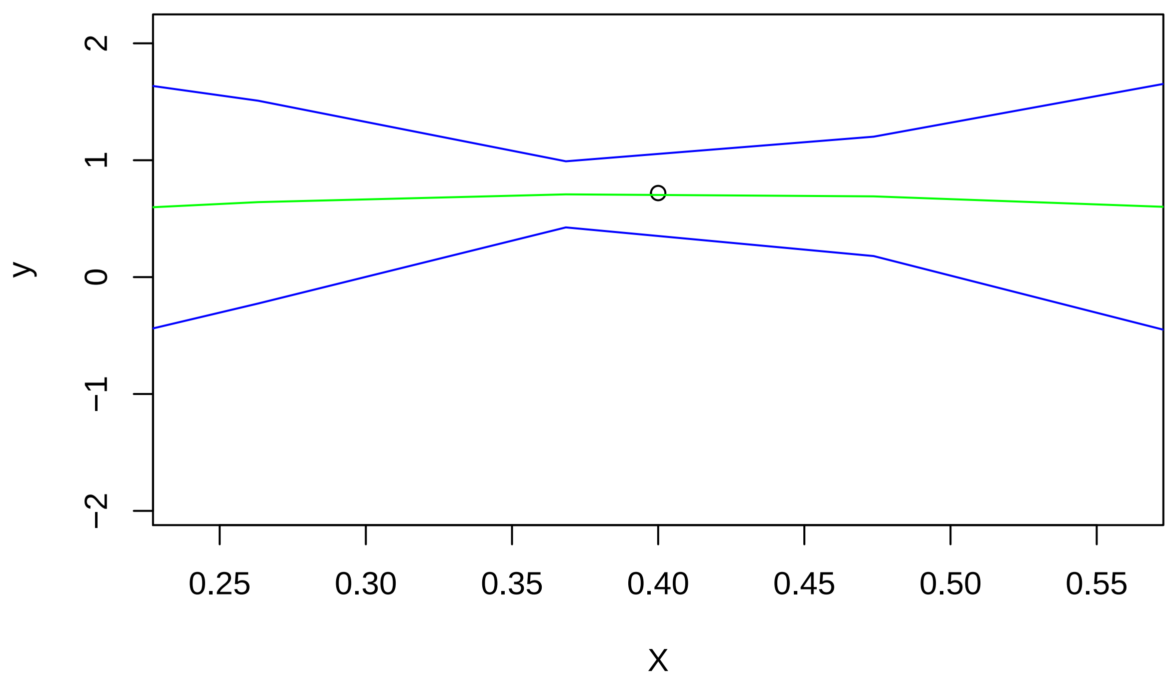
```
## Warning: package 'kernlab' was built under R version 3.5.2
```

```
mySqKernel <- function(x1,x2,sigf=1,l=3){
  K <- matrix(0,nrow = length(x1),ncol = length(x2))
  for (i in 1:ncol(K)){
    K[,i] <- sigf^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

posteriorGP <- function(X,y,Xstar,hyperParam,sigmaNoise){
  # K(x,x)
  k <- mySqKernel(X,X,hyperParam[1],hyperParam[2])
  L <- t(chol(k + diag(sigmaNoise^2,length(X))))
  alpha_ <- solve(L,y)
  alpha <- solve(t(L),alpha_)

  # K(x,xstar)
  kstar <- mySqKernel(X,Xstar,hyperParam[1],hyperParam[2])
  meanVector<- t(kstar)%*%alpha
  v <- solve(L,kstar)
  kstarstar <- mySqKernel(Xstar,Xstar,hyperParam[1],hyperParam[2])
  var <- kstarstar - t(v)%*%v
  return(list("mean" = meanVector, "var" = var))
}
```

2)

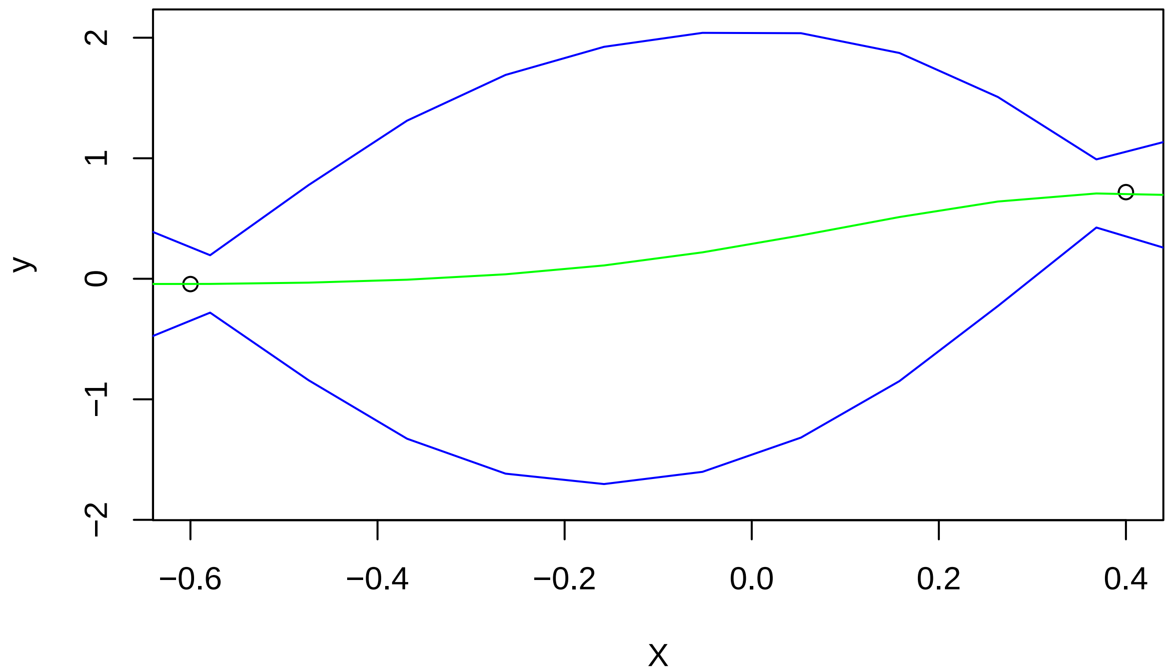
```
x <- c(0.4)
y <- c(0.719)
Xstar <- seq(-1,1,length=20)
hyperParam <- c(sigf=1,l=0.3)
noise <- 0.1

getBounds <- function(mean,var){
  lower <- mean - 1.96 * sqrt(diag(var))
  upper <-  mean + 1.96 * sqrt(diag(var))
  return(list(lower=lower,upper=upper))
}
```

```r
plotGP <- function(X,y,Xstar,hyperParam,sigNoise){
  posteriror <- posteriorGP(X,y,Xstar,hyperParam,sigNoise)
  meanVal <- posteriror$mean
  varVal <- posteriror$var
  bounds <- getBounds(meanVal,varVal)

  plot(X,y, type='p' ,ylim=c(min(bounds$lower), max(bounds$upper)))
  lines(Xstar,meanVal,col='green')
  lines(Xstar,bounds$lower,col='blue')
  lines(Xstar,bounds$upper,col='blue')
}
plotGP(x,y,Xstar,hyperParam ,noise)
```



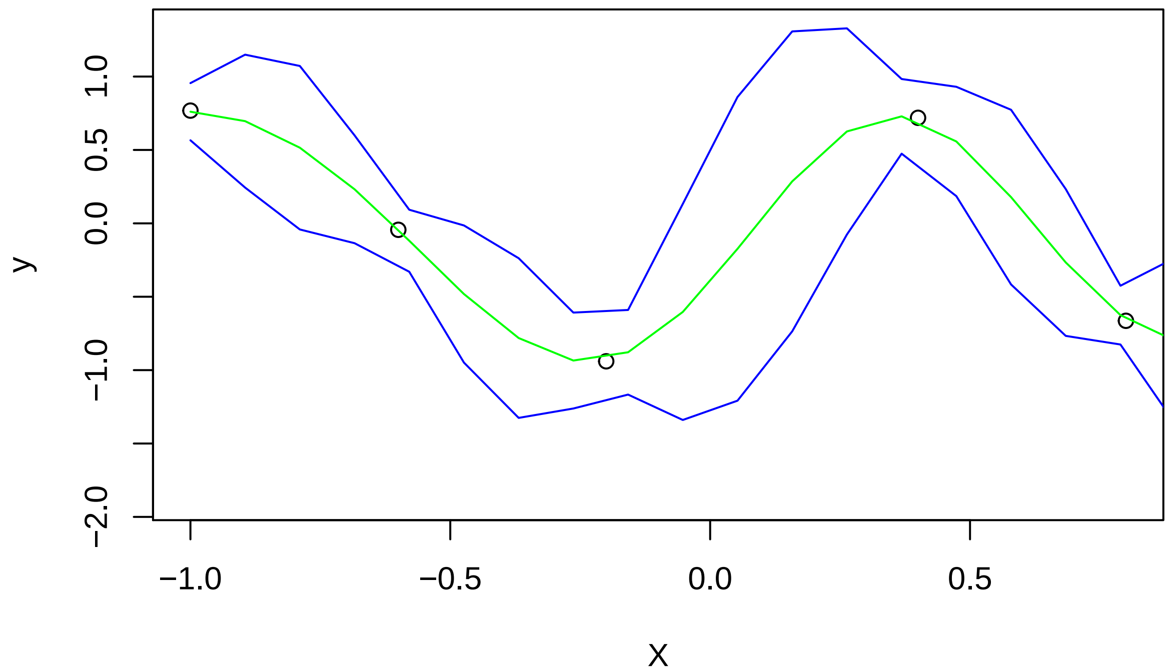
3)

```r
# Dont know how to update, following hint
x <- c(0.4,-0.6)
y <- c(0.719,-0.044)
plotGP(x,y,Xstar,hyperParam,noise)
```



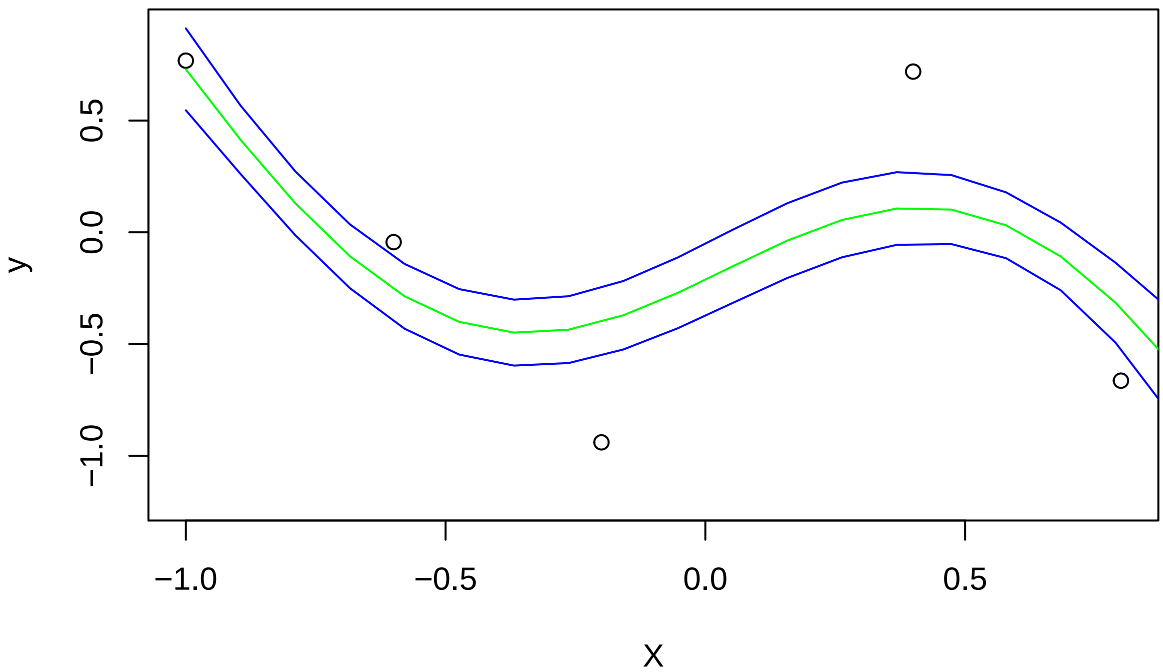
4)

```
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)
plotGP(x,y,Xstar,hyperParam,noise)
```



5)

```
hyperParam <- c(sigf = 1, l = 1)
plotGP(x,y,Xstar,hyperParam,noise)
```



The probability bands are smmoother now as compared to that obtained previosuly, but the bands does not include all the data points.

2.2) GP Regression with kernlab

1)

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
data$time <- c(1:nrow(data))
data$day <- c(1:365)

dataSampled = data[seq(1, nrow(data), 5), ]

mykern <- function(sigf = 1, l = 1)
{
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
  return(sigf^2 * exp(-0.5 * ( r / l )^2 ))
  }
  class(rval) <- "kernel"
  return(rval)
}
kernFunc <- mykern(sigf = 1, l = 1)
kernFunc(1,2) # Evaluate Kernel at (1,2)
```

```
##           [,1]
```

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

```r
x <- matrix(c(1,3,4), 3, 1)
xtran <- matrix(c(2,3,4), 3, 1)
K <- kernelMatrix(kernel = kernFunc, x = x, y = xtran)
K
```

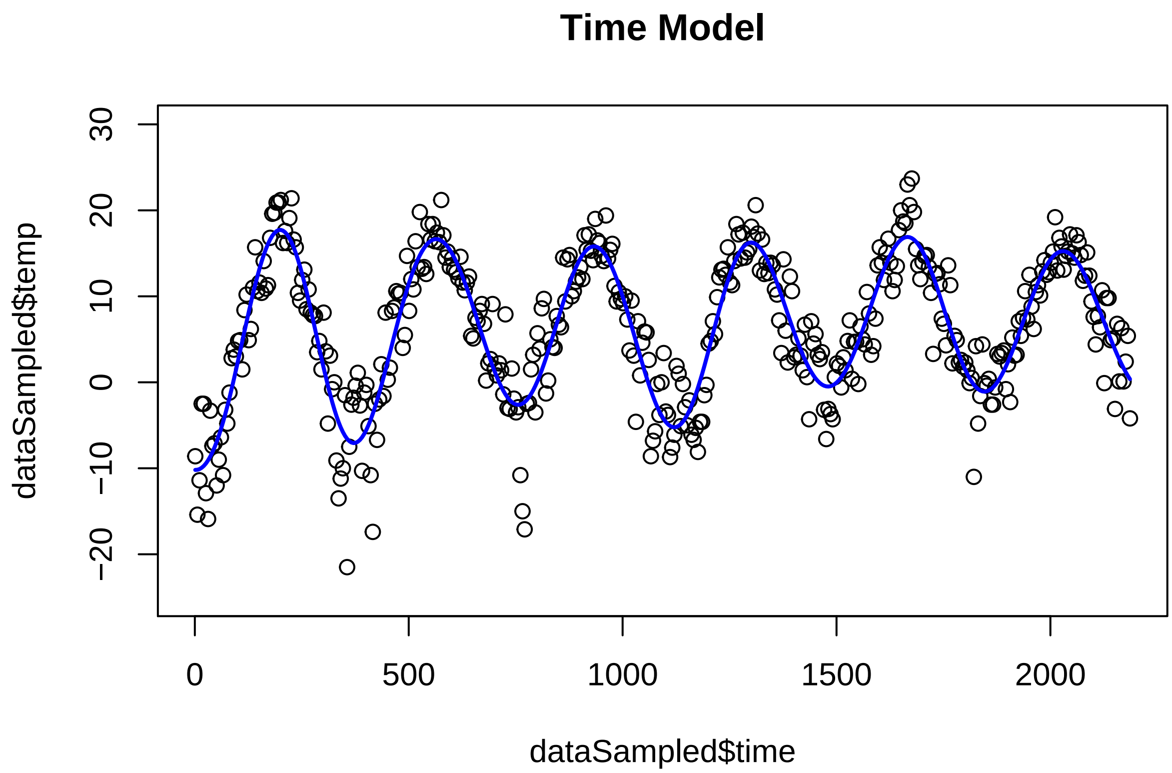
```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

2)

```r
fit <- lm(temp ~ I(time)+I(time^2),data = dataSampled) # According to equation given
sigmaNoise <- sd(resid(fit))
sigmaNoise
```

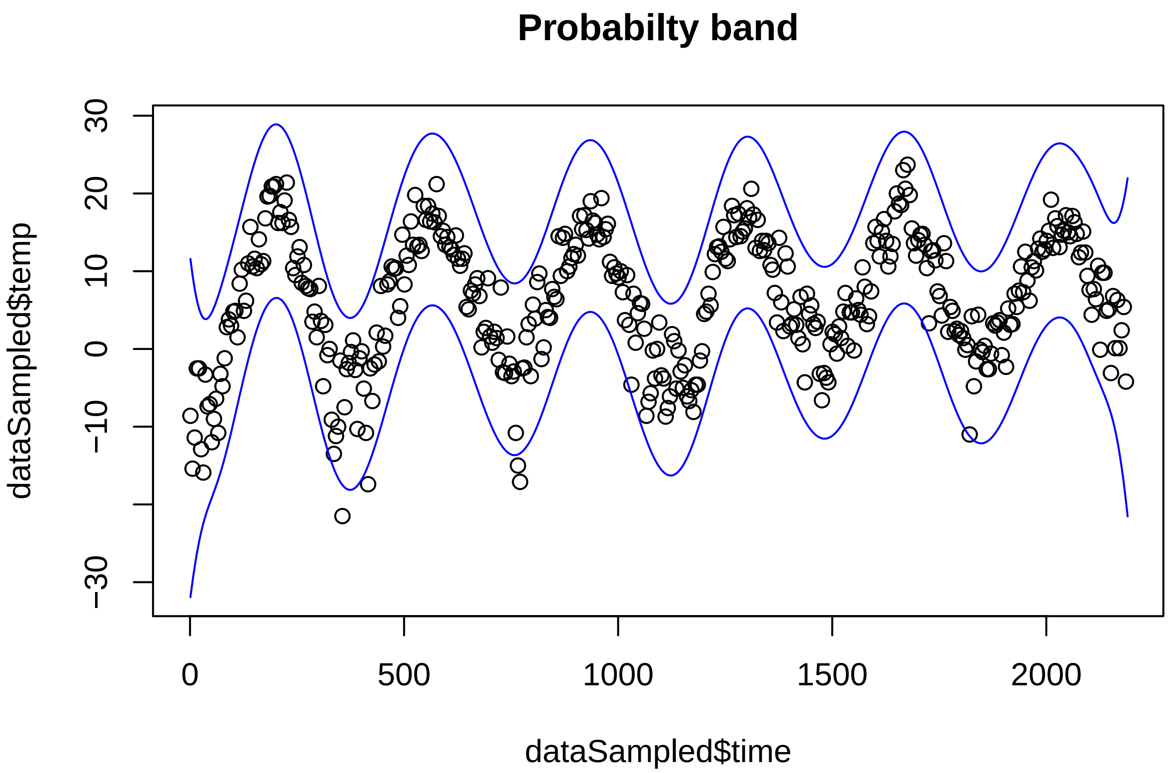
```
## [1] 8.176288
```

```r
# gp with square exp kernel
sigf <- 30
l <- 0.2
hyperParam <- c(sigf,l)
kern = mykern(sigf,l)
gpFit <- gausspr(dataSampled$time,dataSampled$temp,kernel=kern,var = sigmaNoise^2)
pred <- predict(gpFit,dataSampled$time)
{plot(dataSampled$time, dataSampled$temp, ylim=c(-25,30), main="Time Model")
lines(dataSampled$time,pred, col="blue", lwd = 2)}
```



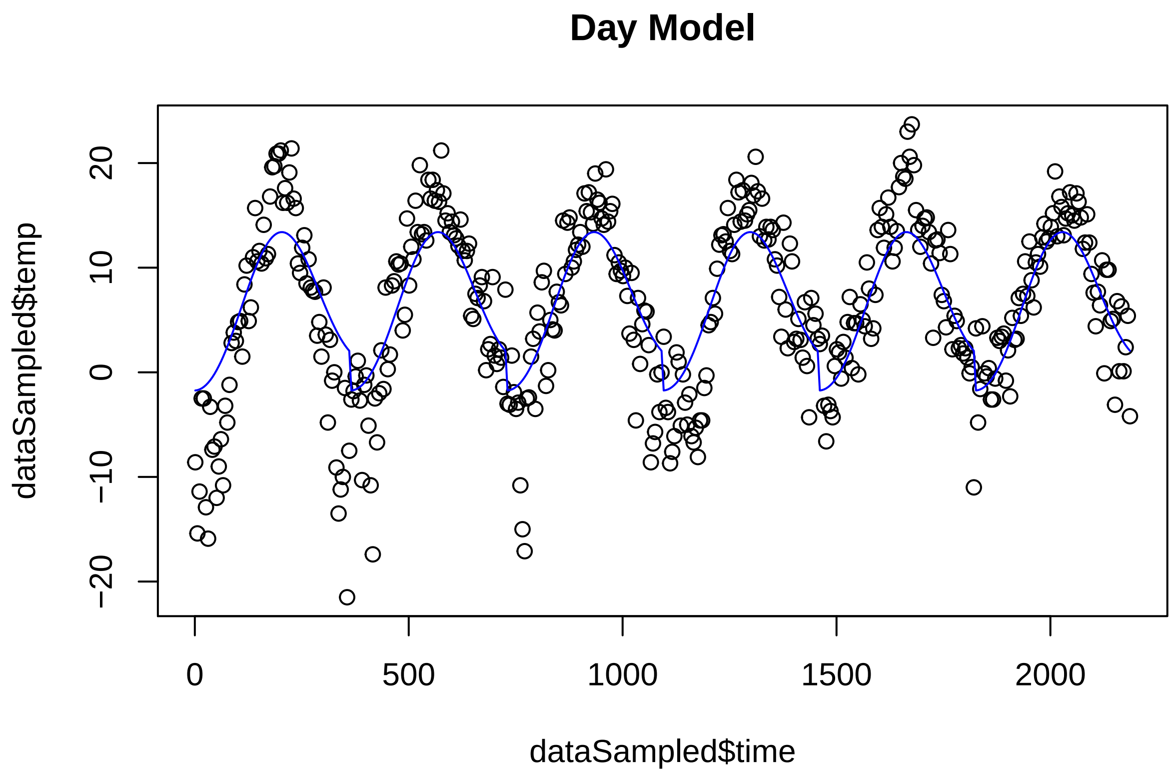
3)

```
tempS <- scale(dataSampled$temp)
timeS <- scale(dataSampled$time)

x <- seq(1,2190,length=1000)
xS <- scale(x)

post <- posteriorGP(timeS,tempS,xS,hyperParam,sigmaNoise)
meanPost <- post$mean
varPost <- post$var

meanPost <- meanPost *attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')

upper <- matrix(NA,length(meanPost),1)
lower <- matrix(nrow = length(meanPost), ncol = 1 ,0)
for (i in 1:length(post$mean)){
  var <- post$var[i,i]
  var <- var * attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')
  upper[i] <- meanPost[i] - 1.96*sqrt(var)
  lower[i] <- meanPost[i] + 1.96*sqrt(var)
}
{plot(dataSampled$time, dataSampled$temp, type="p", ylim=c(min(upper), max(lower)), main="Probabilty bar
lines(x, upper, col="blue")
lines(x, lower, col="blue")}
```



4)

```
fit2 <- lm(dataSampled$temp ~ I(dataSampled$day) + I(dataSampled$day^2) )
sigmaNoise = sd(fit2$residuals)

gpfit2 <- gausspr(dataSampled$day, dataSampled$temp, kernel = kernFunc, var = sigmaNoise^2)
postMean2 <- predict(gpfit2,dataSampled$day)
plot(dataSampled$time, dataSampled$temp, main="Day Model")
lines(dataSampled$time,postMean2, col="blue")
```



5)

```
sigf <- 20;l1<-1;l2<-10;

fit <- lm(dataSampled$temp ~ dataSampled$time + I(dataSampled$time^2))

timeSd <- sqrt(var(dataSampled$time)); d <- 365/timeSd;
sigNoise = sd(resid(fit))

preodicKern <- function(sigf,l1,l2){
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
    return( sigf^2 * exp(-0.5*( (r)/l2)^2 ) * exp(-2*( (sin(pi*r/d))/l1)^2) )
  }
  class(rval) <- "kernel"
  return(rval)
}

preodKern <- preodicKern(sigf,l1,l2)
gpPrediodic <- gausspr(dataSampled$time, dataSampled$temp, kernel = preodKern, var = sigmaNoise^2)
preodicMeanPred <- predict(gpPrediodic, dataSampled$time)
cat('Time Model Error')
```

```
## Time Model Error
```

```r
(timeSE = gpFit@error) #time fit
```

```
## [1] 0.1795078
```

```r
cat('Day Model error')
```

```
## Day Model error
```

```r
(datSE = gpfit2@error) # day fit
```

```
## [1] 0.3216524
```

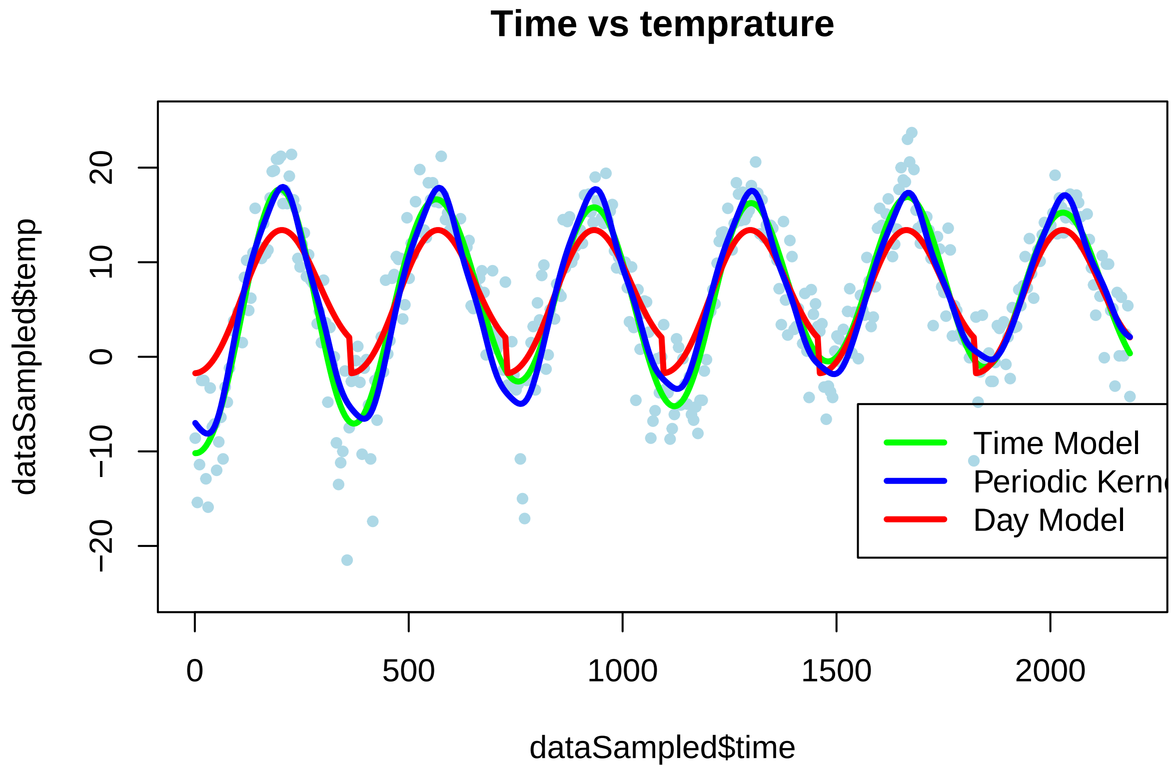
```r
cat('Periodic Model error')
```

```
## Periodic Model error
```

```r
(errorPrTime = gpPrediodic@error) # preodic time fit
```

```
## [1] 0.1929214
```

```r
{plot(dataSampled$time, dataSampled$temp, pch=20, col="lightblue", ylim=c(-25,25),main="Time vs tempratu
lines(dataSampled$time, pred, lwd = 3, type = "l", col="green")
lines(dataSampled$time, postMean2, lwd = 3, type = "l", col="red")
lines(dataSampled$time, preodicMeanPred, lwd = 3, type = "l", col="blue")
legend(x=1550,y=-5, legend=c("Time Model", "Periodic Kernel Model","Day Model"),col=c("green","blue","re
```



The day model has the highest error and is not smooth. Time model fits the data wl and has the minimum error of all three models.

2.3. GP Classification with kernlab. Download the banknote fraud data

a)You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e., use the vector SelectTraining to subset the training observations):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.

```
library(AtmRay)
```

```
## Warning: package 'AtmRay' was built under R version 3.5.3
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
#for meshgrid
# meshgrid <- function(x, y = x) {
#     if (!is.numeric(x) || !is.numeric(y))
#         stop("Arguments 'x' and 'y' must be numeric vectors.")
#
#     x <- c(x); y <- c(y)
#     n <- length(x)
#     m <- length(y)
#
#     X <- matrix(rep(x, each = m),  nrow = m, ncol = n)
#     Y <- matrix(rep(y, times = n), nrow = m, ncol = n)
#
#     return(list(X = X, Y = Y))
# }
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[-SelectTraining,]
colnames(data)
```

```
## [1] "varWave"     "skewWave"    "kurtWave"    "entropyWave" "fraud"
```

```r
#Using automatic sigma estimation or sigest for RBF or laplace kernel
GPfraud_model <- gausspr(fraud ~ varWave + skewWave, data = train)
```

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

```r
GPfraud_model
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.2043047635594
##
## Number of training instances learned : 1000
## Train error : 0.068
```

```r
summary(GPfraud_model)
```

```
##  Length   Class    Mode
##       1 gausspr      S4
```

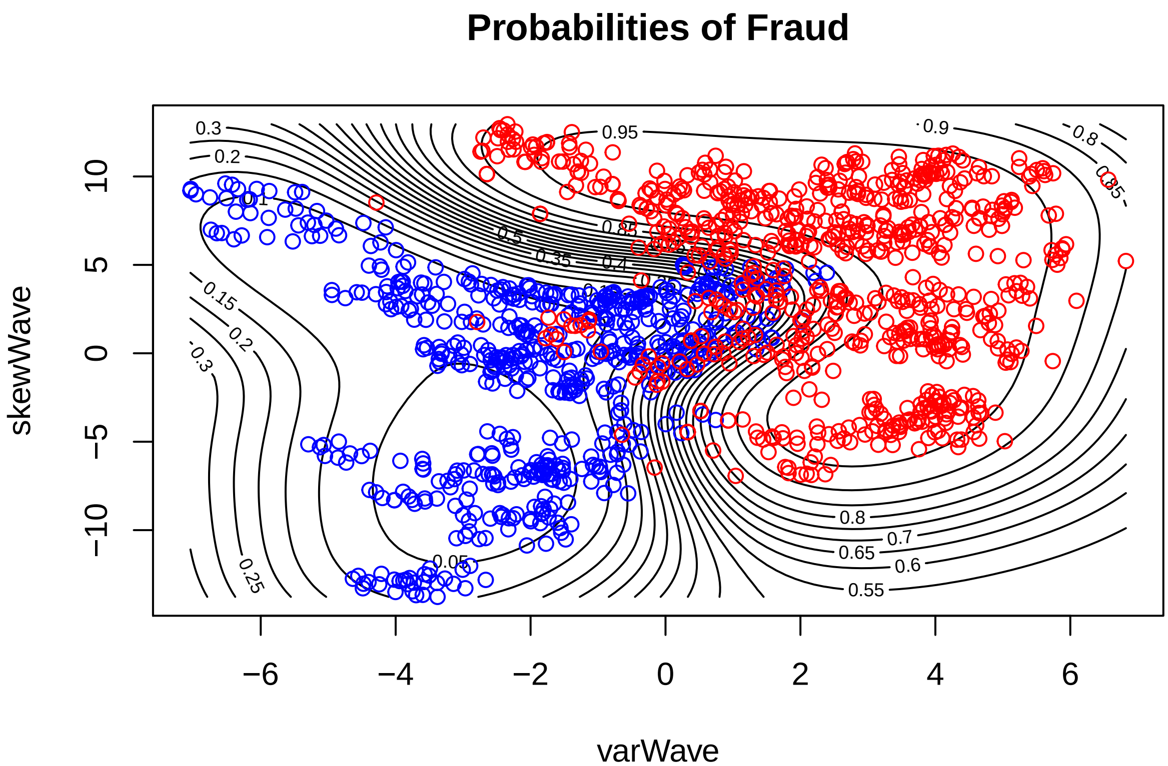
```r
#predict using the test data
GPfraud_pred_train <- predict(GPfraud_model, train[,c("varWave","skewWave")])
#confusion matrix
table(GPfraud_pred_train, train$fraud)
```

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

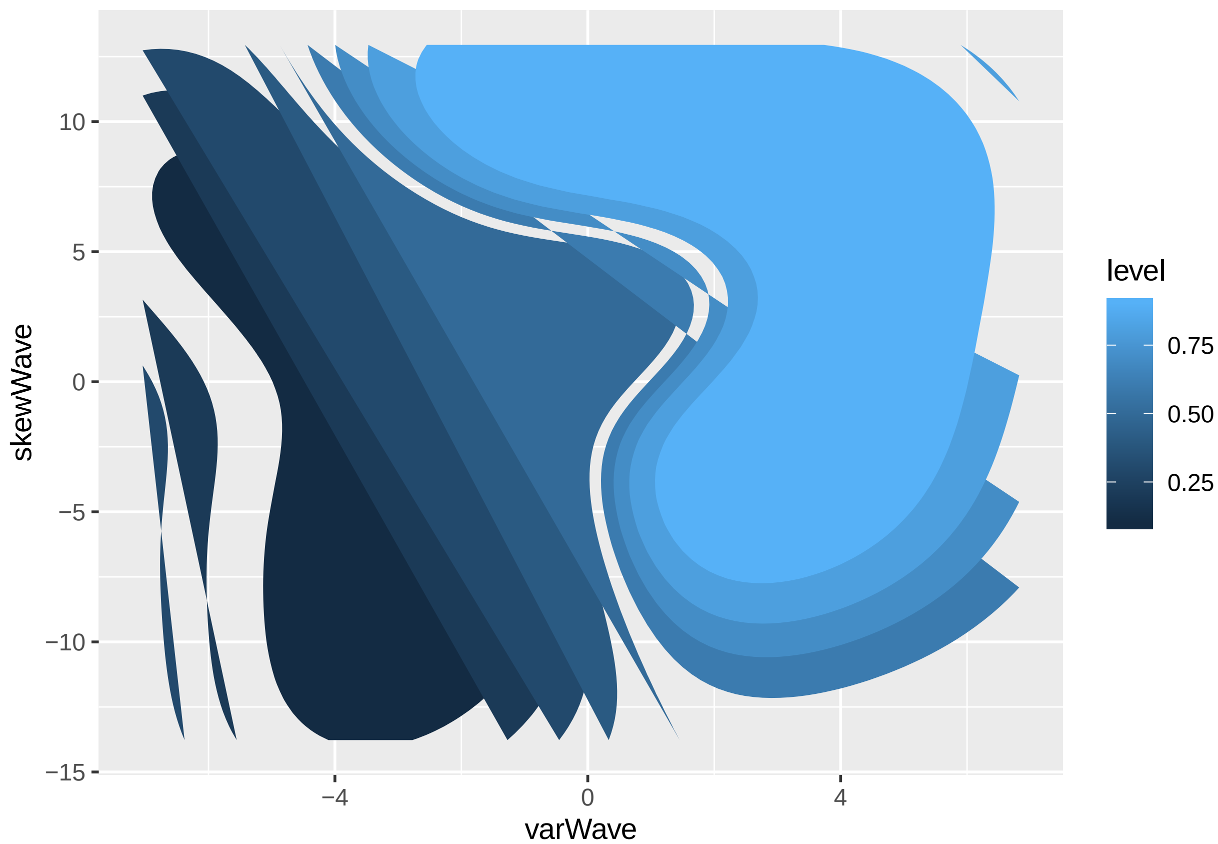
```r
#accuracy
mean(GPfraud_pred_train == train$fraud)
```

```
## [1] 0.932
```

```r
#creating grid points for contour plot
contour_x <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length = 100)
contour_y <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length = 100)
grid <- AtmRay::meshgrid(contour_x,contour_y)
grid <- cbind(c(grid$x),c(grid$y))
grid <- data.frame(grid)
names(grid) <- c("varWave","skewWave")
prediction_probability <- predict(GPfraud_model, grid, type = "probabilities")
z <- t(matrix(prediction_probability[,1],100))
contour(contour_x,contour_y, z ,nlevels = 20,
        xlab = "varWave",ylab = "skewWave",
        main = "Probabilities of Fraud")
points(data[data[,5]==1,"varWave"],data[data[,5]==1,"skewWave"],col = "blue")
points(data[data[,5]==0,"varWave"],data[data[,5]==0,"skewWave"],col = "red")
```




```r
ggplot(grid, aes(x = varWave,y = skewWave, z = prediction_probability[,1]))+
stat_contour(geom="polygon",aes(fill=..level..))
```



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

```r
# predict on the test set
GPfraud_pred_test <- predict(GPfraud_model, test[,c("varWave","skewWave")])
#confusion matrix and accuracy
table(GPfraud_pred_test, test$fraud)
```

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

```r
mean(GPfraud_pred_test == test$fraud)
```

```
## [1] 0.9354839
```

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

```r
GPfraud_model_allvars <- gausspr(fraud ~ ., data = train)
```

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

```r
GPfraud_model_allvars
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.399933221120042
##
## Number of training instances learned : 1000
## Train error : 0.004
```

```r
# prediction using test data
GPfraud_pred_test_allvars<- predict(GPfraud_model_allvars,test[,-ncol(test)])
# confusion matrix and accuracy
table(GPfraud_pred_test_allvars, test$fraud)
```

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

```r
mean(GPfraud_pred_test_allvars == test$fraud)
```

```
## [1] 0.9973118
```

Question: Explain the difference between the results from ii) and iii). Answer: ii) is about the uncertainty of the function f, which is the MEAN of y iii) is about the uncertainty of individual y values. They are uncertain for two reasons: you don't know f at the test point, and you don't know the error (epsilon) that will hit this individual observation

Question: Discuss the differences in results from using the two length scales. Answer: shorter length scale gives less smooth f. We are overfitting the data. Answer: longer length scale gives more smoothness.

Question: Do you think a GP with a squared exponential kernel is a good model for this data? If not, why? Answer: One would have to experiment with other length scales, or estimate the length scales (see question 3c), but this is not likely to help here. The issue is that the data seems to be have different smoothness for small x than it has for large x (where the function seems much more flat) The solution is probably to have different length scales for different x

Question 3(c)

mention EITHER of the following two approaches: 1. The marginal likelihood can be used to select optimal hyperparameters, and also the noise variance. We can optimize the log marginal likelihood with respect to the hyperparameters. In Gaussian Process Regression the marginal likelihood is availble in closed form (a formula). 2. We can use sampling methods (e.g. MCMC) to sample from the marginal posterior of the hyperparameters We need a prior p(theta) for the hyperparameter and then Bayes rule gives the marginal posterior p(theta | data) propto p(data | theta)*p(theta) where p(data | theta) is the marginal likelihood (f has been integrated out).

If the noise variance is unknown, we can treat like any of the kernel hyperparameters and infer the noise variance jointly with the length scale and the prior variance sigma_f

```r
knitr::opts_chunk$set(echo = TRUE)
library("mvtnorm")
library(kernlab)

mySqKernel <- function(x1,x2,sigf=1,l=3){
  K <- matrix(0,nrow = length(x1),ncol = length(x2))
  for (i in 1:ncol(K)){
    K[,i] <- sigf^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

posteriorGP <- function(X,y,Xstar,hyperParam,sigmaNoise){
  # K(x,x)
  k <- mySqKernel(X,X,hyperParam[1],hyperParam[2])
  L <- t(chol(k + diag(sigmaNoise^2,length(X))))
  alpha_ <- solve(L,y)
  alpha <- solve(t(L),alpha_)

  # K(x,xstar)
  kstar <- mySqKernel(X,Xstar,hyperParam[1],hyperParam[2])
  meanVector<- t(kstar)%*%alpha
  v <- solve(L,kstar)
  kstarstar <- mySqKernel(Xstar,Xstar,hyperParam[1],hyperParam[2])
  var <- kstarstar - t(v)%*%v
  return(list("mean" = meanVector, "var" = var))
}

x <- c(0.4)
y <- c(0.719)
Xstar <- seq(-1,1,length=20)
hyperParam <- c(sigf=1,l=0.3)
noise <- 0.1

getBounds <- function(mean,var){
  lower <- mean - 1.96 * sqrt(diag(var))
  upper <-  mean + 1.96 * sqrt(diag(var))
  return(list(lower=lower,upper=upper))
}
plotGP <- function(X,y,Xstar,hyperParam,sigNoise){
  posteriror <- posteriorGP(X,y,Xstar,hyperParam,sigNoise)
  meanVal <- posteriror$mean
  varVal <- posteriror$var
  bounds <- getBounds(meanVal,varVal)

  plot(X,y, type='p' ,ylim=c(min(bounds$lower), max(bounds$upper)))
  lines(Xstar,meanVal,col='green')
  lines(Xstar,bounds$lower,col='blue')
  lines(Xstar,bounds$upper,col='blue')
}
plotGP(x,y,Xstar,hyperParam ,noise)
# Dont know how to update, following hint
x <- c(0.4,-0.6)
```

```r
y <- c(0.719,-0.044)
plotGP(x,y,Xstar,hyperParam,noise)
x <- c(-1,-0.6,-0.2,0.4,0.8)
y <- c(0.768,-0.044,-0.940,0.719,-0.664)
plotGP(x,y,Xstar,hyperParam,noise)
hyperParam <- c(sigf = 1, l = 1)
plotGP(x,y,Xstar,hyperParam,noise)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
data$time <- c(1:nrow(data))
data$day <- c(1:365)

dataSampled = data[seq(1, nrow(data), 5), ]

mykern <- function(sigf = 1, l = 1)
{
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
  return(sigf^2 * exp(-0.5 * ( r / l )^2 ))
  }
  class(rval) <- "kernel"
  return(rval)
}
kernFunc <- mykern(sigf = 1, l = 1)
kernFunc(1,2) # Evaluate Kernel at (1,2)

x <- matrix(c(1,3,4), 3, 1)
xtran <- matrix(c(2,3,4), 3, 1)
K <- kernelMatrix(kernel = kernFunc, x = x, y = xtran)
K

fit <- lm(temp ~ I(time)+I(time^2),data = dataSampled) # According to equation given
sigmaNoise <- sd(resid(fit))
sigmaNoise

# gp with square exp kernel
sigf <- 30
l <- 0.2
hyperParam <- c(sigf,l)
kern = mykern(sigf,l)
gpFit <- gausspr(dataSampled$time,dataSampled$temp,kernel=kern,var = sigmaNoise^2)
pred <- predict(gpFit,dataSampled$time)
{plot(dataSampled$time, dataSampled$temp, ylim=c(-25,30), main="Time Model")
lines(dataSampled$time,pred, col="blue", lwd = 2)}
tempS <- scale(dataSampled$temp)
timeS <- scale(dataSampled$time)

x <- seq(1,2190,length=1000)
xS <- scale(x)

post <- posteriorGP(timeS,tempS,xS,hyperParam,sigmaNoise)
meanPost <- post$mean
varPost <- post$var
```

```r
meanPost <- meanPost *attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')

upper <- matrix(NA,length(meanPost),1)
lower <- matrix(nrow = length(meanPost), ncol = 1 ,0)
for (i in 1:length(post$mean)){
  var <- post$var[i,i]
  var <- var * attr(tempS, 'scaled:scale') + attr(tempS, 'scaled:center')
  upper[i] <- meanPost[i] - 1.96*sqrt(var)
  lower[i] <- meanPost[i] + 1.96*sqrt(var)
}
{plot(dataSampled$time, dataSampled$temp, type="p", ylim=c(min(upper), max(lower)), main="Probabilty ba
lines(x, upper, col="blue")
lines(x, lower, col="blue")}
fit2 <- lm(dataSampled$temp ~ I(dataSampled$day) + I(dataSampled$day^2) )
sigmaNoise = sd(fit2$residuals)

gpfit2 <- gausspr(dataSampled$day, dataSampled$temp, kernel = kernFunc, var = sigmaNoise^2)
postMean2 <- predict(gpfit2,dataSampled$day)
plot(dataSampled$time, dataSampled$temp, main="Day Model")
lines(dataSampled$time,postMean2, col="blue")
sigf <- 20;l1<-1;l2<-10;

fit <- lm(dataSampled$temp ~ dataSampled$time + I(dataSampled$time^2))

timeSd <- sqrt(var(dataSampled$time)); d <- 365/timeSd;
sigNoise = sd(resid(fit))

preodicKern <- function(sigf,l1,l2){
  rval <- function(x, y = NULL) {
  r = sqrt(crossprod(x-y));
    return( sigf^2 * exp(-0.5*( (r)/l2)^2 ) * exp(-2*( (sin(pi*r/d))/l1)^2) )
  }
  class(rval) <- "kernel"
  return(rval)
}

preodKern <- preodicKern(sigf,l1,l2)
gpPrediodic <- gausspr(dataSampled$time, dataSampled$temp, kernel = preodKern, var = sigmaNoise^2)
preodicMeanPred <- predict(gpPrediodic, dataSampled$time)
cat('Time Model Error')
(timeSE = gpFit@error) #time fit
cat('Day Model error')
(datSE = gpfit2@error) # day fit

cat('Periodic Model error')
(errorPrTime = gpPrediodic@error) # preodic time fit

{plot(dataSampled$time, dataSampled$temp, pch=20, col="lightblue", ylim=c(-25,25),main="Time vs temprat
lines(dataSampled$time, pred, lwd = 3, type = "l", col="green")
lines(dataSampled$time, postMean2, lwd = 3, type = "l", col="red")
lines(dataSampled$time, preodicMeanPred, lwd = 3, type = "l", col="blue")
legend(x=1550,y=-5, legend=c("Time Model", "Periodic Kernel Model","Day Model"),col=c("green","blue","r
library(AtmRay)
```

```r
library(ggplot2)
#for meshgrid
# meshgrid <- function(x, y = x) {
#     if (!is.numeric(x) || !is.numeric(y))
#         stop("Arguments 'x' and 'y' must be numeric vectors.")
#
#     x <- c(x); y <- c(y)
#     n <- length(x)
#     m <- length(y)
#
#     X <- matrix(rep(x, each = m),  nrow = m, ncol = n)
#     Y <- matrix(rep(y, times = n), nrow = m, ncol = n)
#
#     return(list(X = X, Y = Y))
# }
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[-SelectTraining,]
colnames(data)
#Using automatic sigma estimation or sigest for RBF or laplace kernel
GPfraud_model <- gausspr(fraud ~ varWave + skewWave, data = train)
GPfraud_model
summary(GPfraud_model)
#predict using the test data
GPfraud_pred_train <- predict(GPfraud_model, train[,c("varWave","skewWave")])
#confusion matrix
table(GPfraud_pred_train, train$fraud)
#accuracy
mean(GPfraud_pred_train == train$fraud)
#creating grid points for contour plot
contour_x <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length = 100)
contour_y <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length = 100)
grid <- AtmRay::meshgrid(contour_x,contour_y)
grid <- cbind(c(grid$x),c(grid$y))
grid <- data.frame(grid)
names(grid) <- c("varWave","skewWave")
prediction_probability <- predict(GPfraud_model, grid, type = "probabilities")
z <- t(matrix(prediction_probability[,1],100))
contour(contour_x,contour_y, z ,nlevels = 20,
        xlab = "varWave",ylab = "skewWave",
        main = "Probabilities of Fraud")
points(data[data[,5]==1,"varWave"],data[data[,5]==1,"skewWave"],col = "blue")
points(data[data[,5]==0,"varWave"],data[data[,5]==0,"skewWave"],col = "red")


ggplot(grid, aes(x = varWave,y = skewWave, z = prediction_probability[,1]))+
stat_contour(geom="polygon",aes(fill=..level..))
# predict on the test set
```

```r
GPfraud_pred_test <- predict(GPfraud_model, test[,c("varWave","skewWave")])
#confusion matrix and accuracy
table(GPfraud_pred_test, test$fraud)
mean(GPfraud_pred_test == test$fraud)
GPfraud_model_allvars <- gausspr(fraud ~ ., data = train)
GPfraud_model_allvars
# prediction using test data
GPfraud_pred_test_allvars<- predict(GPfraud_model_allvars,test[,-ncol(test)])
# confusion matrix and accuracy
table(GPfraud_pred_test_allvars, test$fraud)
mean(GPfraud_pred_test_allvars == test$fraud)
### Question 3(a)
# Change to your path
library(kernlab)
library(mvtnorm)

# Simulating from the prior for l = 0.2
kernel02 <- mykern(sigf = 1, l = 0.2) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel02, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}

# Simulating from the prior for l = 1
kernel1 <- mykern(sigf = 1, l = 1) # This constructs the covariance function
xGrid = seq(-1,1,by=0.1)
K = kernelMatrix(kernel = kernel1, xGrid, xGrid)

colors = list("black","red","blue","green","purple")
f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
plot(xGrid,f, type = "l", ylim = c(-3,3), col = colors[[1]])
for (i in 1:4){
  f = rmvnorm(n = 1, mean = rep(0,length(xGrid)), sigma = K)
  lines(xGrid,f, col = colors[[i+1]])
}


# Computing the correlation functions

# l = 0.2
kernel02(0,0.1) # Note: here correlation=covariance since sigf = 1
kernel02(0,0.5)

# l = 1
kernel1(0,0.1) # Note: here correlation=covariance since sigf = 1
kernel1(0,0.5)
```

```r
# The correlation between the function at x = 0 and x=0.1 is much higher than
# between x = 0 and x=0.5, since the latter points are more distant.
# Thus, the correlation decays with distance in x-space. This decay is much more
# rapid when l = 0.2 than when l = 1. This is also visible in the simulations
# where realized functions are much more smooth when l = 1.


### Question 3(b)

load("GPdata.RData")

### l = 0.2

sigmaNoise = 0.2

# Set up the kernel function
kernelFunc <- mykern(sigf = 1, l = 0.2)

# Plot the data and the true
plot(x, y, main = "", cex = 0.5)

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
# Alternative: GPfit <- gausspr(y ~ x, kernel = mykern, kpar = list(sigf = 1, l = 0.2), var = sigmaNois
xs = seq(min(x),max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)



### l = 1

sigmaNoise = 0.2
```

```r
# Set up the kernel function
kernelFunc <- mykern(sigf = 1, l = 1)

# Plot the data and the true
plot(x,y, main = "", cex = 0.5)
#lines(xGrid,fVals, type = "l", col = "black", lwd = 3) # true mean

GPfit <- gausspr(x, y, kernel = kernelFunc, var = sigmaNoise^2)
xs = seq(min(x), max(x), length.out = 100)
meanPred <- predict(GPfit, data.frame(x = xs)) # Predicting the training data. To plot the fit.
lines(xs, meanPred, col="blue", lwd = 2)

# Compute the covariance matrix Cov(f)
n <- length(x)
Kss <- kernelMatrix(kernel = kernelFunc, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = kernelFunc, x = x, y = x)
Kxs <- kernelMatrix(kernel = kernelFunc, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)

# Probability intervals for f
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "red")
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "red")

# Prediction intervals for y
lines(xs, meanPred - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(xs, meanPred + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")


legend("topright", inset = 0.02, legend = c("data","post mean","95% intervals for f", "95% predictive i
       col = c("black", "blue", "red", "purple"),
       pch = c('o',NA,NA,NA), lty = c(NA,1,1,1), lwd = 2, cex = 0.55)
```