GP_lab_Milda

Milda Poceviciute

16 October 2018

Question 1

Part 1

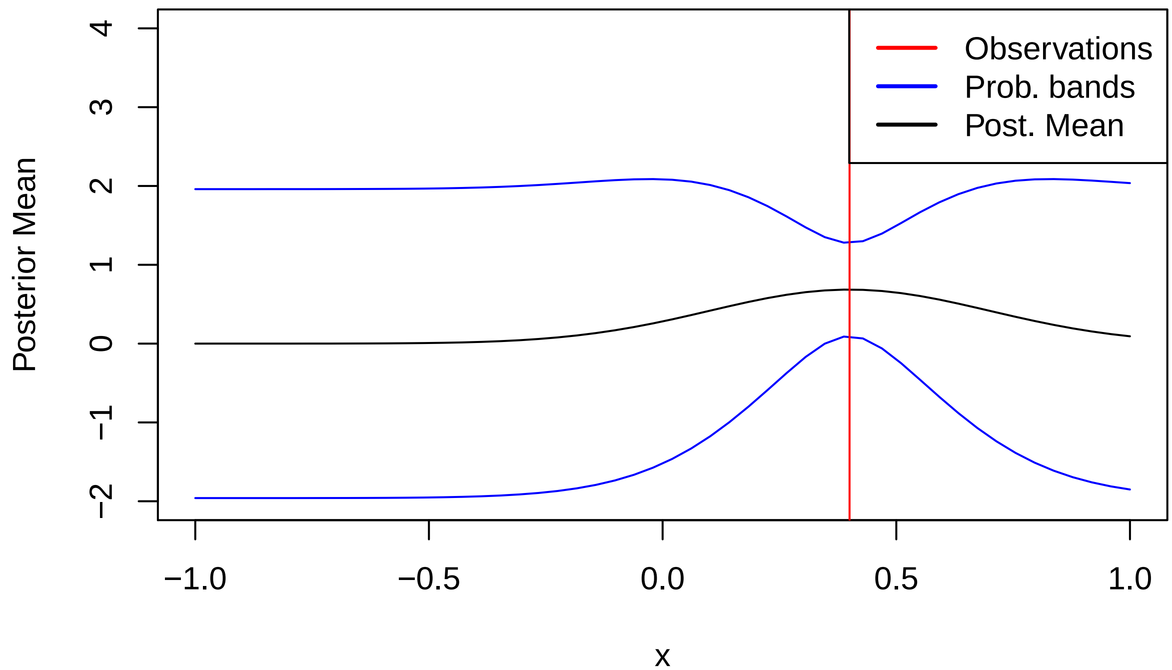
The code below is the implementation of the Gaussian process regression model:

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

```
# Setting up the kernel
SquaredExpKernel <- 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)
}


posteriorGP <- function(X, y, XStar, hyperParam, sigmaNoise) {
    k_star <- SquaredExpKernel(X, XStar, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    K <- SquaredExpKernel(X, X, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    noise <- diag(nrow(K)) * sigmaNoise
    L <- t(chol(K + noise))

    # Predictive mean
    alpha <- solve(L) %*% y
    posterior_mean <- t(k_star) %*% alpha
    # Predictive variance
    v <- solve(L) %*% k_star
    kk <- SquaredExpKernel(XStar, XStar, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    posterior_var <- kk - t(v) %*% v

    return(list(posterior_mean = posterior_mean, posterior_var = posterior_var))

}
```

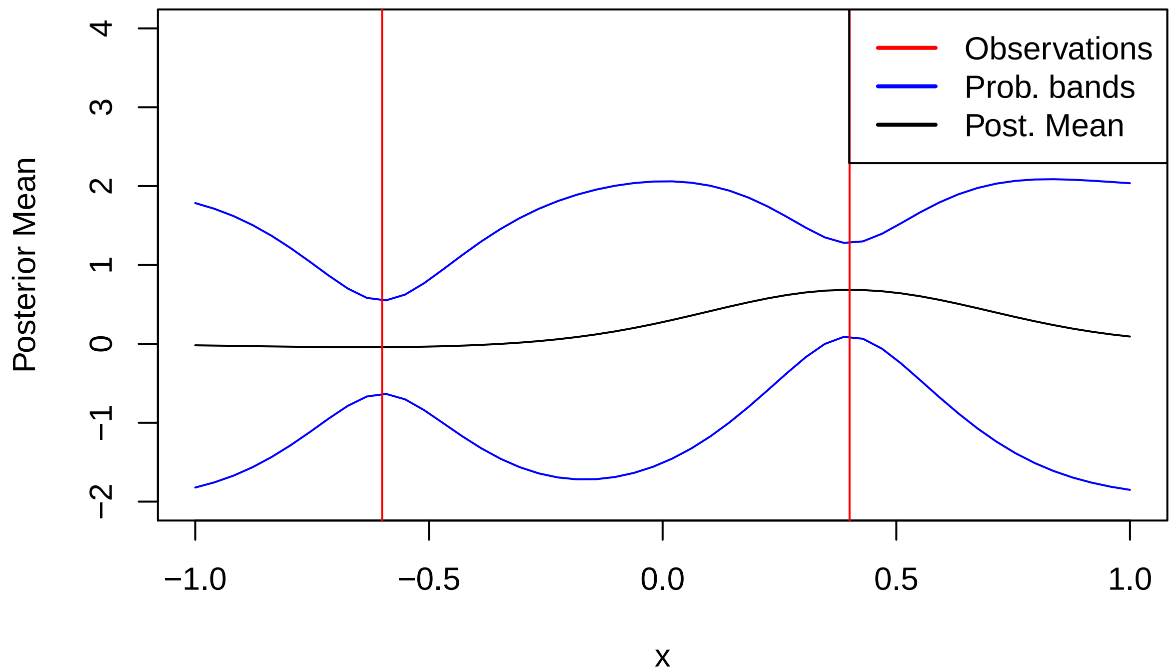
Part 2

In this part we assume that prior hyper-parameters are $\sigma_f = 1$, $\ell = 0.3$, and we update the posterior with observation $(x, y) = (0.4, 0.719)$. Here σ_n is assumed to be 0.1. The plot below shows the observation, the posterior mean (the black line), and the 95% probability bands (point-wise) plotted over a grid $[-1, 1]$. The probability bands are quite wide for most of the x grid points, except from the location of the observed point (x, y). This observation decreases the uncertainty and bands become narrower.



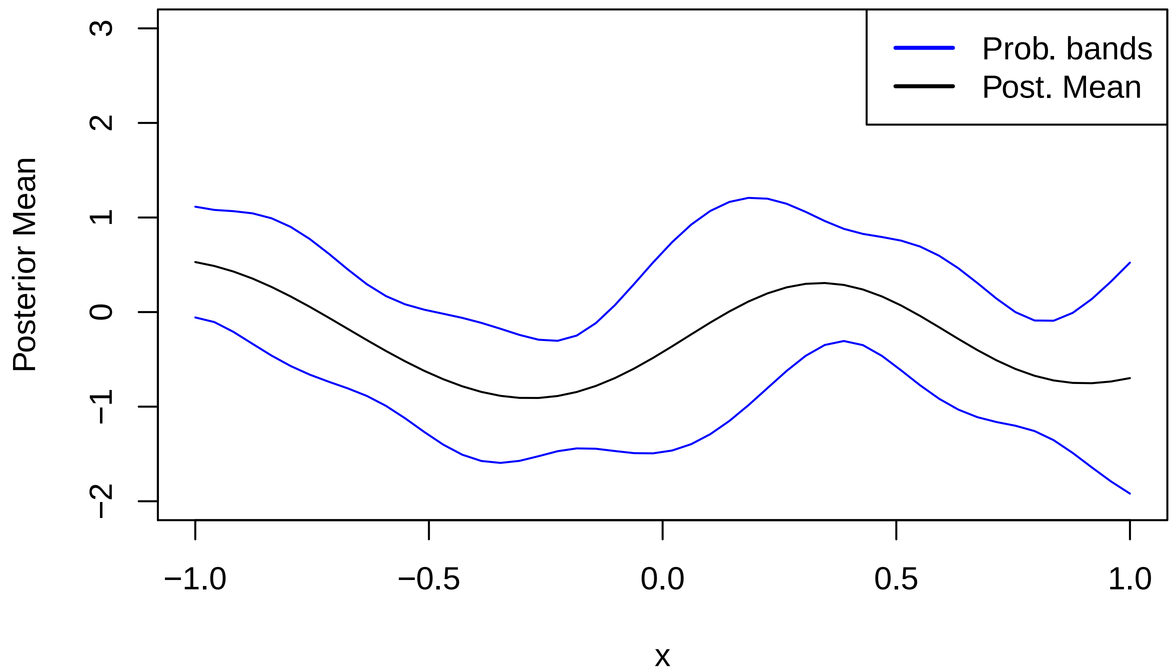
Part 3

The posterior is updated with one more observation $(x, y) = (-0.6, -0.044)$. The plot below shows again the observations, the posterior mean(the black line), and the 95% probability bands (point-wise) over the grid $[-1, 1]$. This time the uncertainty is smaller for the two observed x values: the probability bands are narrower there.



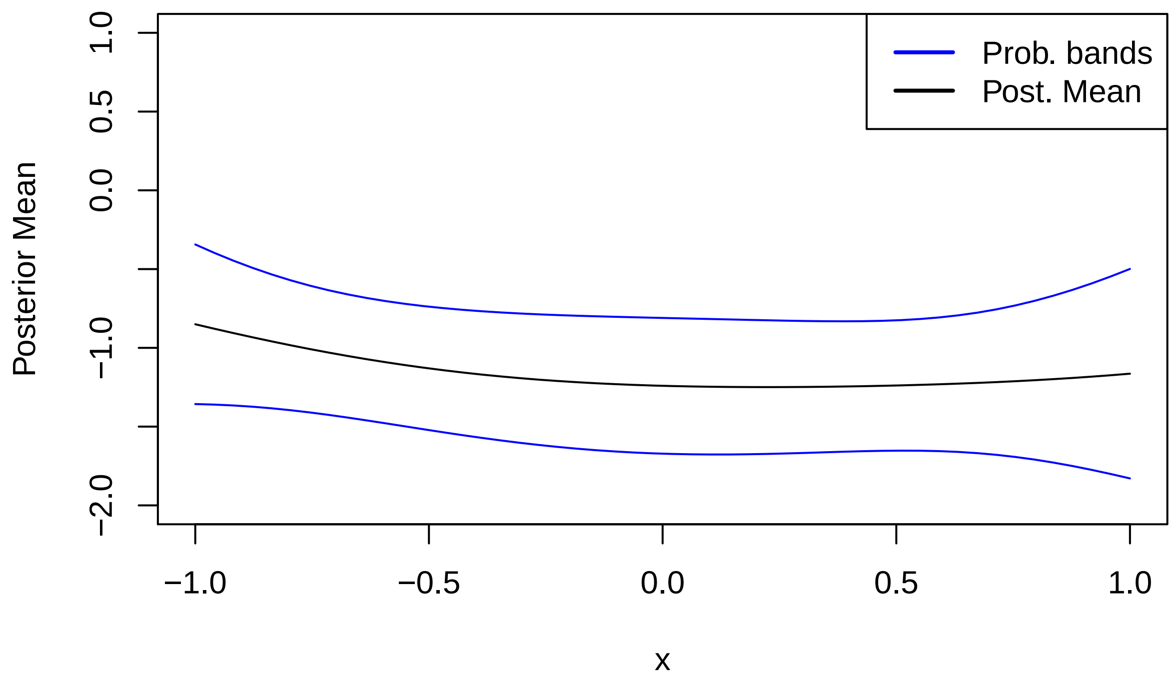
Part 4

The plot shows the posterior mean and the 95% probability bands after having 5 observations. The uncertainty has decreased a lot in comparison to parts 2 and 3.



Part 5

The plot below shows the posterior mean and the 95% probability bands after having the 5 observations as in part 4, but changing the hyper-parameters to $\sigma_f = 1$, $\ell = 1$. The higher the ℓ value, the smoother the posterior mean becomes. In the kernel the distance between the points is divided by ℓ. This means that the higher the value of this hyper-parameter is, the kernel values converge to the σ_f: the distance between the observations becomes less and less important. Hence, the posterior mean becomes more smooth.



Question 2

Part 1

In this question, the exponential kernel function is defined:

```
# Function that returns kernel class object
Matern32 <- function(sigmaf = 1, ell = 1)
{
  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)
}

MaternFunc = Matern32(sigmaf = 1, ell = 2)
```

It is used to compute the covariance, firstly between (1,2), and afterwards between points $x = (1, 3, 4)$ and $x^* = (2, 3, 4)$. The results are printed:

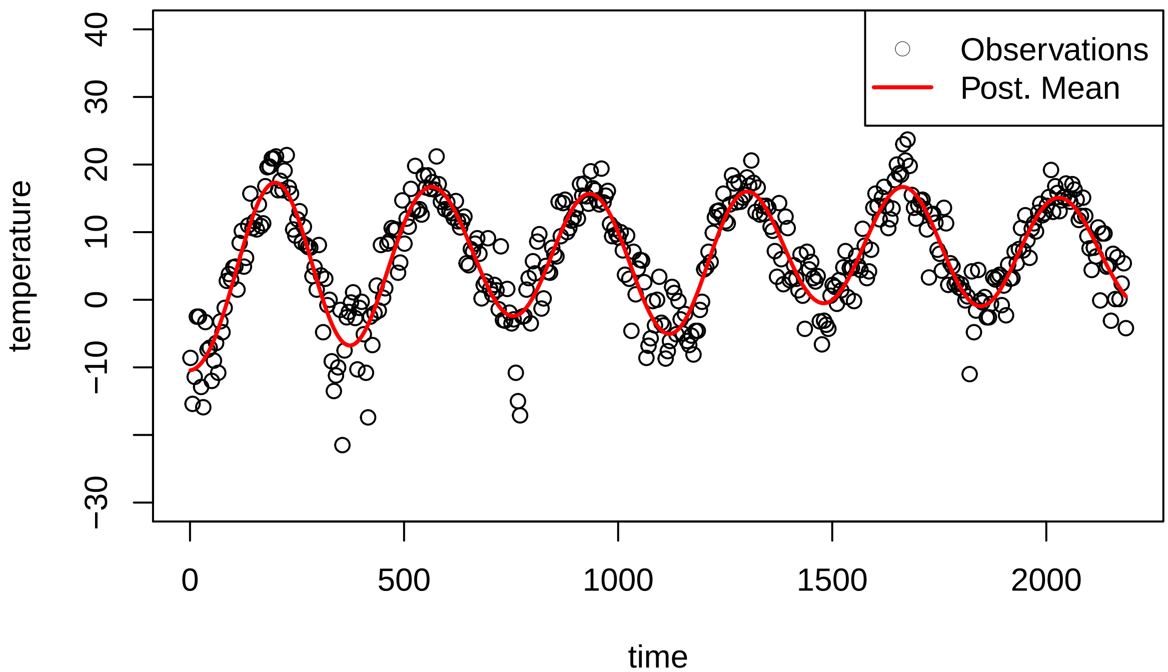
```
## The kernel between points (1,2) is : 0.8824969

## The kernel between points (1,3,4) and (2,3,4) is:

## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.8824969 0.6065307 0.3246525
## [2,] 0.8824969 1.0000000 0.8824969
## [3,] 0.6065307 0.8824969 1.0000000
```

Part 2

The following model is used for obtaining the posterior mean:

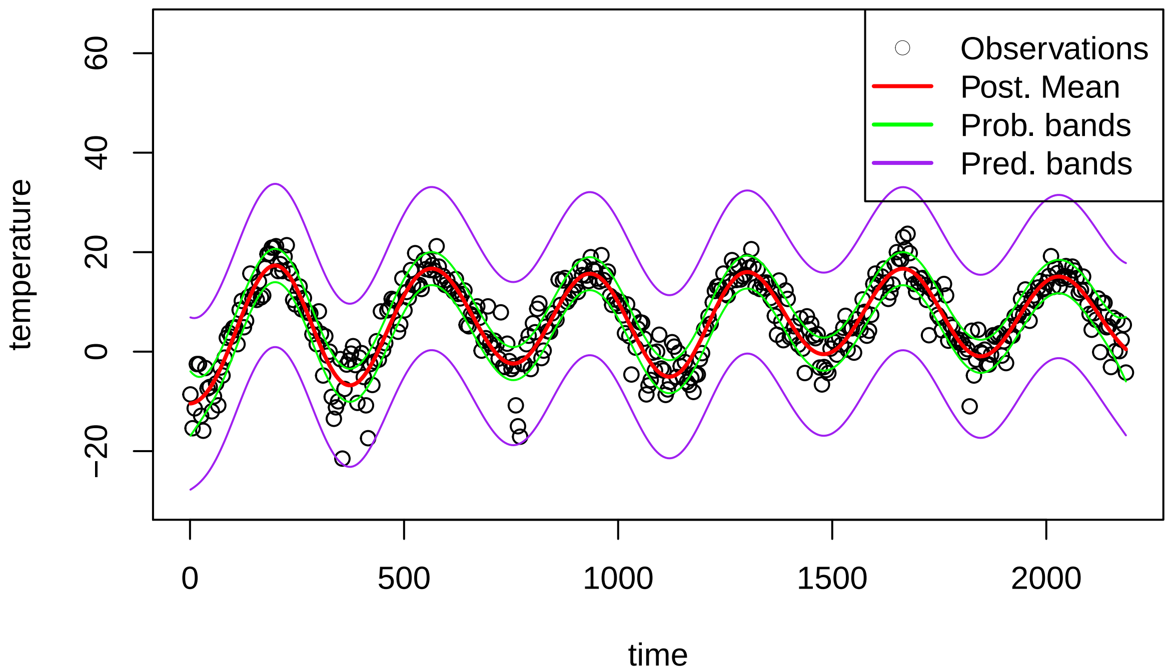
$temp = f(time) + \epsilon$ with $\epsilon \sim N(0, \sigma_n^2)$, $f \sim GP(0, k(time, time^{'}))$

The plot below shows the observations plotted with a fitted posterior mean at each data point. The model fits the data quite well as it predicts the underlying trends (periodicity).



Part 3

From the plot below, we see that the probability and prediction bands fit the data quite well. Though, the prediction bands are quite wide.

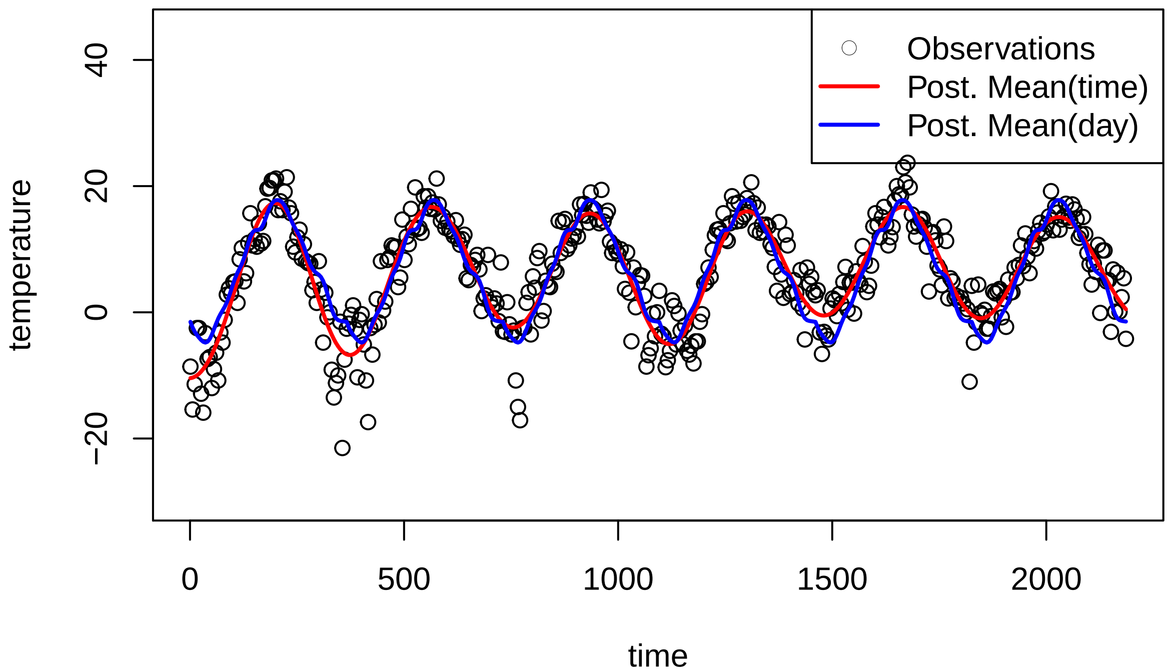


Part 4

The following model is used for obtaining the posterior mean:

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

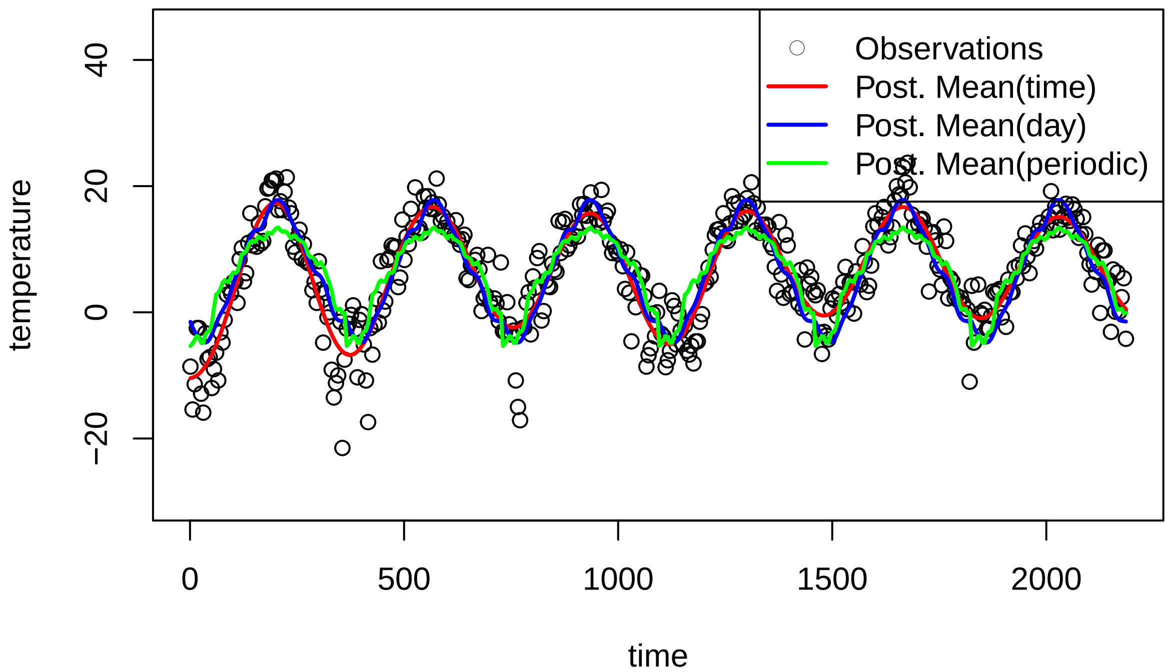
From the plot, we see that both models performed quite well and the posterior means follow closely the seasonality of the temperatures. The *Time* model was fitting to the all observed information, hence it shows the overall trend of the temperature fluctuations over the whole time period. The *Day* model captures the seasonality during the year well, as the variable $1, 2, ..., 365$ was repeated 6 times and fitted with the temperatures, it fits exactly the same curve 6 times.



Part 5

The GP with a generalization of the periodic kernel is used in this part to model the temperature against the time.

From the plot, we see that the posterior mean from *Time* model predicts the underlying trends over all time period, the *Day* model is good at capturing the yearly seasonality. The *Periodic* model of this part seems to be performing quite well too. This is reasonable as the temperature data is a periodic data.



Question 3

```
##   varWave skewWave kurtWave entropyWave fraud
## 1 3.62160   8.6661  -2.8073    -0.44699     0
## 2 4.54590   8.1674  -2.4586    -1.46210     0
## 3 3.86600  -2.6383   1.9242     0.10645     0
## 4 3.45660   9.5228  -4.0112    -3.59440     0
## 5 0.32924  -4.4552   4.5718    -0.98880     0
## 6 4.36840   9.6718  -3.9606    -3.16250     0
```

Part 1

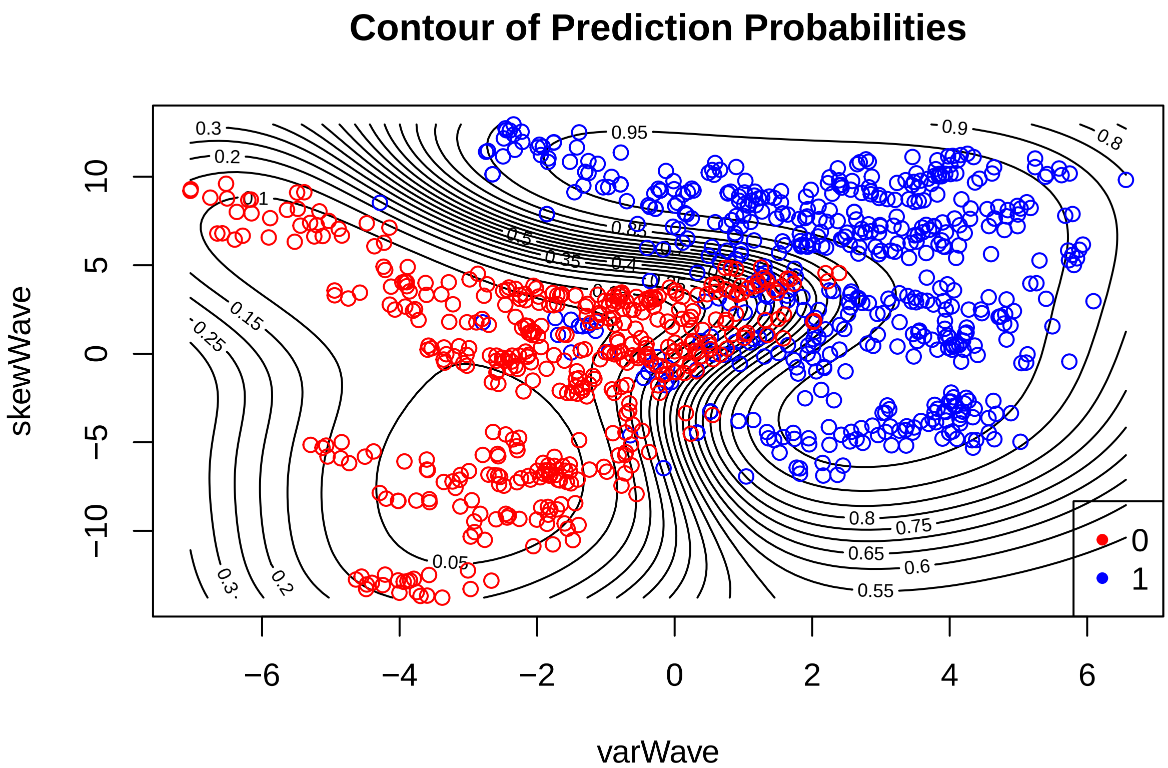
The package *kenrlab* is used to fit the Gaussian Process Classification on Fraud data-set.

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

The model gives the corresponding confusion matrix on the training data:

```
##
## pred1   0   1
##     0 512  24
##     1  44 420
```

Hence, the accuracy is 0.932. From the plot below, we see that classifier works quite well and there are two contour peaks that concentrate the most of the points from the different class. However, the overlap region is quite big, and that explains the lower accuracy rate of the classifier.



Part 2

The confusion matrix on the testing data is:

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

The accuracy is 0.9354839, which is pretty close to the accuracy rate obtained on the training data. This is a good indication that the classification works well, and it is not overfitted.

Part 3

In this part all available information in the data-set is used to classify the fraud. The resulting confusion matrix on the test data is:

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

```
##
## pred3   0   1
##     0 205   0
##     1   1 166
```

The accuracy is 0.9973118, which is higher than in the previous parts. It is reasonable that the classification model improved once we included all the available information.

Appendix

```
knitr::opts_chunk$set(echo = FALSE)
library(kernlab)
library(AtmRay)
set.seed(12345)
# Setting up the kernel
SquaredExpKernel <- 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)
}


posteriorGP <- function(X, y, XStar, hyperParam, sigmaNoise) {
    k_star <- SquaredExpKernel(X, XStar, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    K <- SquaredExpKernel(X, X, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    noise <- diag(nrow(K)) * sigmaNoise
    L <- t(chol(K + noise))

    # Predictive mean
    alpha <- solve(L) %*% y
    posterior_mean <- t(k_star) %*% alpha
    # Predictive variance
    v <- solve(L) %*% k_star
    kk <- SquaredExpKernel(XStar, XStar, sigmaF = hyperParam$sigmaF, l = hyperParam$l)
    posterior_var <- kk - t(v) %*% v

    return(list(posterior_mean = posterior_mean, posterior_var = posterior_var))

}

# Information for updating the prior

X_observations <- c(0.4)
y_observations <- c(0.719)
hyperParam2 <- list(sigmaF = 1, l = 0.3)

# Use the data set and new given point to compute the posterior over the grid [-1,1]
xGrid <- seq(-1,1,length=50)
posteriorP2 <- posteriorGP(X = X_observations, y = y_observations, XStar = xGrid, hyperParam = hyperPara

# find probability bounds
error <- qnorm(0.975)*sqrt(diag(posteriorP2$posterior_var))
upp_band <- as.vector(posteriorP2$posterior_mean) + as.vector(error)
low_band <- as.vector(posteriorP2$posterior_mean) - as.vector(error)

# Plot the posterior means over a grid
plot(x=xGrid,y=posteriorP2$posterior_mean, type="l",ylim=c(-2,4),xlab="x", ylab="Posterior Mean")
```

```r
# Add 95% probability bounds (check bayesian course, we must have doen it there)
lines(x=xGrid,y=upp_band, type="l",col="blue")
lines(x=xGrid,y=low_band, type="l",col="blue")
abline(v=0.4,col="red")
legend("topright",c("Observations","Prob. bands","Post. Mean"),
       col = c("red","blue","black"),lty = c(1, 1, 1), lwd=c(2,2,2))


# Update the information

X_observations[2] <- -0.6
y_observations[2] <- -0.044

posteriorP3 <- posteriorGP(X = X_observations, y = y_observations, XStar = xGrid, hyperParam = hyperPara
# find probability bounds
error3 <- qnorm(0.975)*sqrt(diag(posteriorP3$posterior_var))
upp_band3 <- as.vector(posteriorP3$posterior_mean) + as.vector(error3)
low_band3 <- as.vector(posteriorP3$posterior_mean) - as.vector(error3)

# Plot the posterior means over a grid
plot(x=xGrid,y=posteriorP3$posterior_mean, type="l",ylim=c(-2,4),xlab="x", ylab="Posterior Mean")
# Add 95% probability bounds (check bayesian course, we must have doen it there)
lines(x=xGrid,y=upp_band3, type="l",col="blue")
lines(x=xGrid,y=low_band3, type="l",col="blue")
abline(v=0.4,col="red")
abline(v=-0.6,col="red")
legend("topright",c("Observations","Prob. bands","Post. Mean"),
       col = c("red","blue","black"),lty = c(1, 1, 1), lwd=c(2,2,2))

X_observations[1] <- -1
X_observations[3:5] <- c(-0.2,0.4,0.8)
y_observations[3:5] <- c(-0.940, 0.719, -0.664)

posteriorP4 <- posteriorGP(X = X_observations, y = y_observations, XStar = xGrid, hyperParam = hyperPara
# find probability bounds
error4 <- qnorm(0.975)*sqrt(diag(posteriorP4$posterior_var))
upp_band4 <- as.vector(posteriorP4$posterior_mean) + as.vector(error4)
low_band4 <- as.vector(posteriorP4$posterior_mean) - as.vector(error4)

# Plot the posterior means over a grid
plot(x=xGrid,y=posteriorP4$posterior_mean, type="l",ylim=c(-2,3),xlab="x", ylab="Posterior Mean")
# Add 95% probability bounds (check bayesian course, we must have doen it there)
lines(x=xGrid,y=upp_band4, type="l",col="blue")
lines(x=xGrid,y=low_band4, type="l",col="blue")
legend("topright",c("Prob. bands","Post. Mean"),
       col = c("blue","black"),lty = c(1, 1), lwd=c(2,2))

hyperParam5 <- list(sigmaF = 1, l = 1)

posteriorP5 <- posteriorGP(X = X_observations, y = y_observations, XStar = xGrid, hyperParam = hyperPara
# find probability bounds
error5 <- qnorm(0.975)*sqrt(diag(posteriorP5$posterior_var))
upp_band5 <- as.vector(posteriorP5$posterior_mean) + as.vector(error5)
```

```r
low_band5 <- as.vector(posteriorP5$posterior_mean) - as.vector(error5)

# Plot the posterior means over a grid
plot(x=xGrid,y=posteriorP5$posterior_mean, type="l", ylim = c(-2,1),xlab="x", ylab="Posterior Mean")
# Add 95% probability bounds (check bayesian course, we must have doen it there)
lines(x=xGrid,y=upp_band5, type="l",col="blue")
lines(x=xGrid,y=low_band5, type="l",col="blue")
legend("topright",c("Prob. bands","Post. Mean"),
        col = c("blue","black"),lty = c(1, 1), lwd=c(2,2))

dataQ2 <- read.csv("TempTullinge.csv", header=TRUE, sep=";")
time <- 1:2190
day <- 1:365
Stime <- seq(1,2190,by=5)
Sday <- seq(1,365,by=5)
x1 <- c(1,3,4)
x2 <- c(2,3,4)


# Function that returns kernel class object
Matern32 <- function(sigmaf = 1, ell = 1)
{
  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)
}

MaternFunc = Matern32(sigmaf = 1, ell = 2)

#evaluate it in the point x = 1; x' = 2
cat("The kernel between points (1,2) is :", MaternFunc(1,2))
cat("\n")
cat("\n")
# use the kernelMatrix function to compute the covariance matrix K(X;X*)
covM <- kernelMatrix(MaternFunc, x1,x2)
cat("The kernel between points (1,3,4) and (2,3,4) is:")
cat("\n")
covM

# Subset data
temp <- dataQ2$temp[Stime]
# Find the noise value
polyFit <- lm(temp ~  Stime + Stime^2)
sigmaNoise = sd(polyFit$residuals)
MaternFunc2 = Matern32(sigmaf = 20, ell = 0.2)
```

```r
# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
GPfit <- gausspr(Stime, temp, kernel = MaternFunc2, var = sigmaNoise^2)
postMean2 <- predict(GPfit,Stime)
plot(Stime, temp, ylim=c(-30,40),xlab="time",ylab="temperature")
lines(Stime,postMean2, col="red", lwd = 2)
legend("topright",c("Observations","Post. Mean"),
        col = c("black","red"), pch = c(1,-1), lty = c(0, 1), lwd=c(0,2))

x<-scale(Stime)
xs<-scale(Stime) # XStar
n <- length(x)
SEkernel <- MaternFunc2
Kss <- kernelMatrix(kernel = SEkernel, x = xs, y = xs)
Kxx <- kernelMatrix(kernel = SEkernel, x = x, y = x)
Kxs <- kernelMatrix(kernel = SEkernel, x = x, y = xs)
Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs) # Covariance matrix of fStar

plot(Stime, temp, ylim=c(-30,65),xlab="time",ylab="temperature")
lines(Stime,postMean2, col="red", lwd = 2)
# Probability intervals for fStar
lines(Stime, postMean2 - 1.96*sqrt(diag(Covf)), col = "green")
lines(Stime, postMean2 + 1.96*sqrt(diag(Covf)), col = "green")
# Prediction intervals for yStar
lines(Stime, postMean2 - 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
lines(Stime, postMean2 + 1.96*sqrt((diag(Covf) + sigmaNoise^2)), col = "purple")
legend("topright",c("Observations","Post. Mean","Prob. bands","Pred. bands"),
        col = c("black","red","green","purple"), pch = c(1,-1,-1,-1), lty = c(0, 1,1,1), lwd=c(0,2,2,2))


# Find the noise value
polyFit4 <- lm(temp ~  rep.int(Sday,6) )
sigmaNoise4 = sd(polyFit4$residuals)

# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
GPfit4 <- gausspr(rep.int(Sday,6), temp, kernel = MaternFunc2, var = sigmaNoise4^2)
postMean4 <- predict(GPfit4,rep.int(Sday,6))

plot(Stime, temp,xlab="time",ylab="temperature",ylim=c(-30,45))
lines(Stime,postMean2, col="red", lwd = 2)
lines(Stime,postMean4, col="blue", lwd = 2)
legend("topright",c("Observations","Post. Mean(time)","Post. Mean(day)"),
        col = c("black","red", "blue"), pch = c(1,-1,-1), lty = c(0, 1,1), lwd=c(0,2,2))

sd <- sqrt(var(time))
d <- 365/sd
# Function that returns kernel class object
Matern5 <- function(sigmaf = 1, ell1 = 1, ell2 = 1)
{
  rval <- function(x, y = NULL) {
    r = sqrt(crossprod(x-y));
    return(sigmaf^2*exp( -2*(sin(pi*abs(x-y)/d))^2/ell1)*exp(-0.5*(x-y)^2/ell2))
  }
  class(rval) <- "kernel"
```

```r
  return(rval)
}

MaternFunc5 = Matern5(sigmaf = 20, ell1 = 1, ell2=10)
GPfit5 <- gausspr(rep.int(Sday,6), temp, kernel = MaternFunc5, var = sigmaNoise4^2)
postMean5 <- predict(GPfit5,rep.int(Sday,6))
plot(Stime, temp,ylim=c(-30,45),xlab="time",ylab="temperature")
lines(Stime,postMean2, col="red", lwd = 2)
lines(Stime,postMean4, col="blue", lwd = 2)
lines(Stime,postMean5, col="green", lwd = 2)
legend("topright",c("Observations","Post. Mean(time)","Post. Mean(day)","Post. Mean(periodic)"),
        col = c("black","red","blue","green"), pch = c(1,-1,-1,-1), lty = c(0, 1,1,1), lwd=c(0,2,2,2))
# read data from github
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])
head(data)
set.seed(111);
SelectTraining <- sample(1:dim(data)[1], size = 1000,
                                        replace = FALSE)
training <- data[SelectTraining,]
testing <- data[-SelectTraining,]

accuracy <- function (conf_mat){(conf_mat[1,1] + conf_mat[2,2]) / sum(conf_mat)}
GPfitFraud <- gausspr(fraud ~ varWave + skewWave, data=training)

# class probabilities
probPreds <- predict(GPfitFraud, training, type="probabilities")
x1 <- seq(min(training[,1]),max(training[,1]),length=100)
x2 <- seq(min(training[,2]),max(training[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

# predict on the testing set
pred1 <- predict(GPfitFraud,training)
conM1 <- table(pred1, training$fraud) # confusion matrix
conM1

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Contour of Prediction Probabilities')
points(training[training[,5]==0,1],training[training[,5]==0,2],col="blue")
points(training[training[,5]==1,1],training[training[,5]==1,2],col="red")
legend("bottomright", legend=c("0","1"), pch=c(20,20), col=c("red","blue"))
# predict on the testing set
predT <- predict(GPfitFraud,testing)
conM2 <- table(predT, testing$fraud) # confusion matrix
conM2
```

```r
GPfitFraud3 <- gausspr(fraud ~ ., data=training)

# predict on the testing set
pred3 <- predict(GPfitFraud3,testing)
conM3 <- table(pred3, testing$fraud) # confusion matrix
conM3
```