Aswath_MandakathGopinathLab4

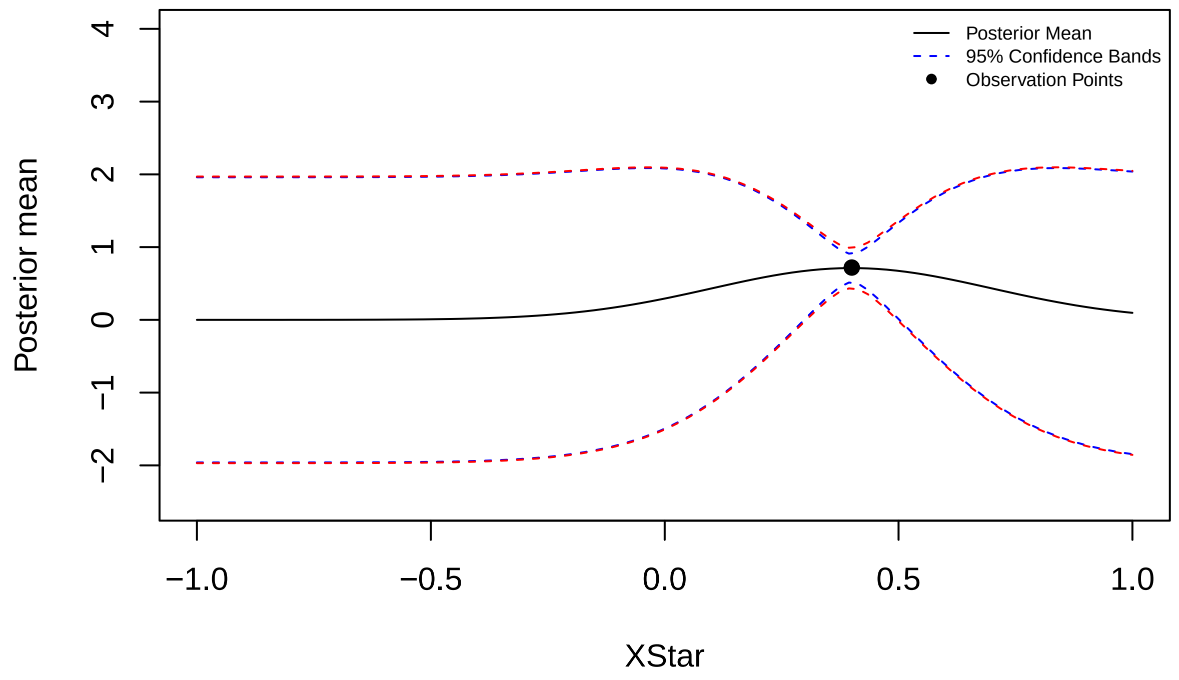
aswma317

2023-10-06

2.1: Implementing GP Regression

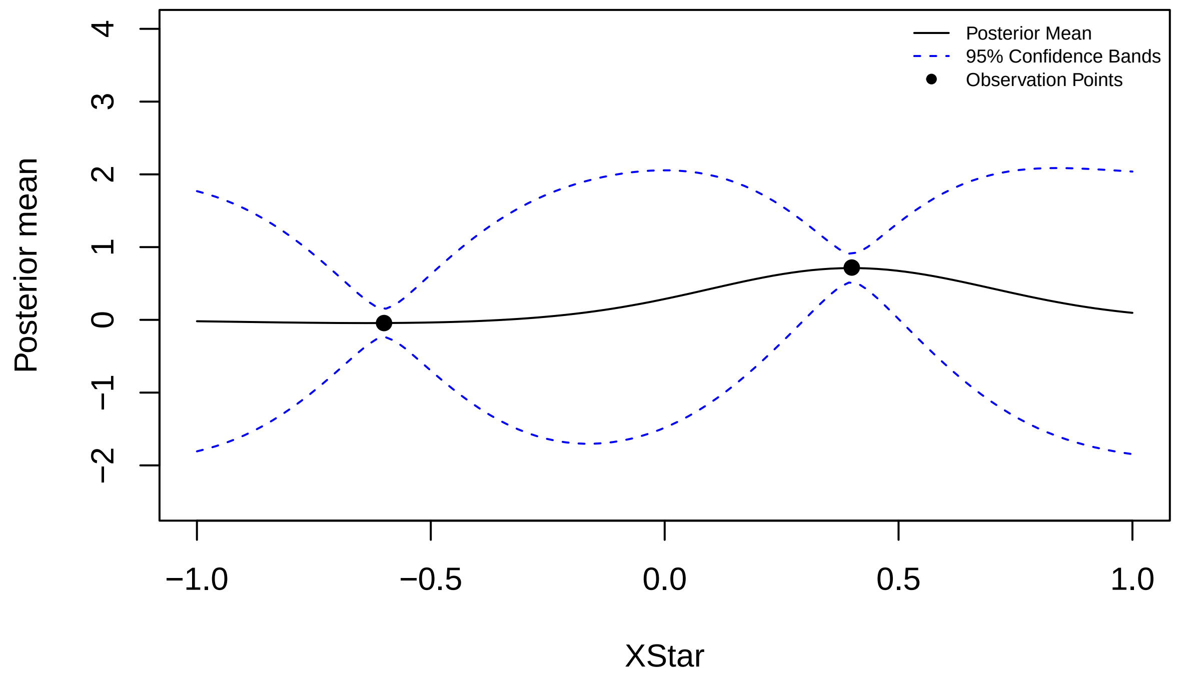
```r
#library(kernlab)
#Subq1: Write your own code for simulating from the posterior distribution of f
#Gets the covariance function or kernel K
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=0.3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){

    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l )^2 )#Hyperparameter, S4
  }
  return(K)
}

#Gets the posterior mean and variance
posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...){
  #sigmaNoise is the sd
  K <- k(X, X, ...)
  n <- length(X)
  #Hyperparameter, S11
  L <- t(chol(K+((sigmaNoise**2) * diag(n))))#Upper to Lower Triangular matrix

  #Predicting mean
  alpha <- solve(t(L) ,solve(L,y))
  KStar <- k(X, XStar, ...)
  FStar <- t(KStar) %*% alpha

  #Predicting variance
  v <- solve(L, KStar)
  V_FStart <- k(XStar, XStar, ...) - (t(v) %*% v)

  logmar <- -0.5*(t(y)%*%alpha)-sum(log(diag(L)))-(n/2)*log(2*pi)

  return(list(pos_mean = FStar,
              pos_var = V_FStart))
}
```

```r
#Subq2: Plot the posterior mean of f over the interval x  [-1, 1]
xTrain <- c(0.4)
y <- c(0.719)
sigmaF <- 1#Prior hyperparameters
l <- 0.3#Prior hyperparameters
sigmaNoise <- 0.1#Assumed
XStar <- seq(-1, 1, length = 100)#New points for prediction
post_GP <- posteriorGP(xTrain, y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)
plot(XStar,post_GP$pos_mean, type = 'l', ylab = 'Posterior mean',ylim=c(-2.5,4))

# Probability intervals for fStar
lines(XStar, post_GP$pos_mean - 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
lines(XStar, post_GP$pos_mean + 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)

# Prediction intervals for yStar
lines(XStar, post_GP$pos_mean - 1.96*sqrt(diag(post_GP$pos_var)+sigmaNoise^2),
      col = "red", lty=2)
lines(XStar, post_GP$pos_mean + 1.96*sqrt(diag(post_GP$pos_var)+sigmaNoise^2),
      col = "red", lty=2)
points(xTrain,y,col='black', pch = 19)
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
       col = c("black", "blue", "black"), lty = c(1, 2, NA), pch = c(NA, NA, 19),
       cex = 0.6,
       bty = 'n')
```

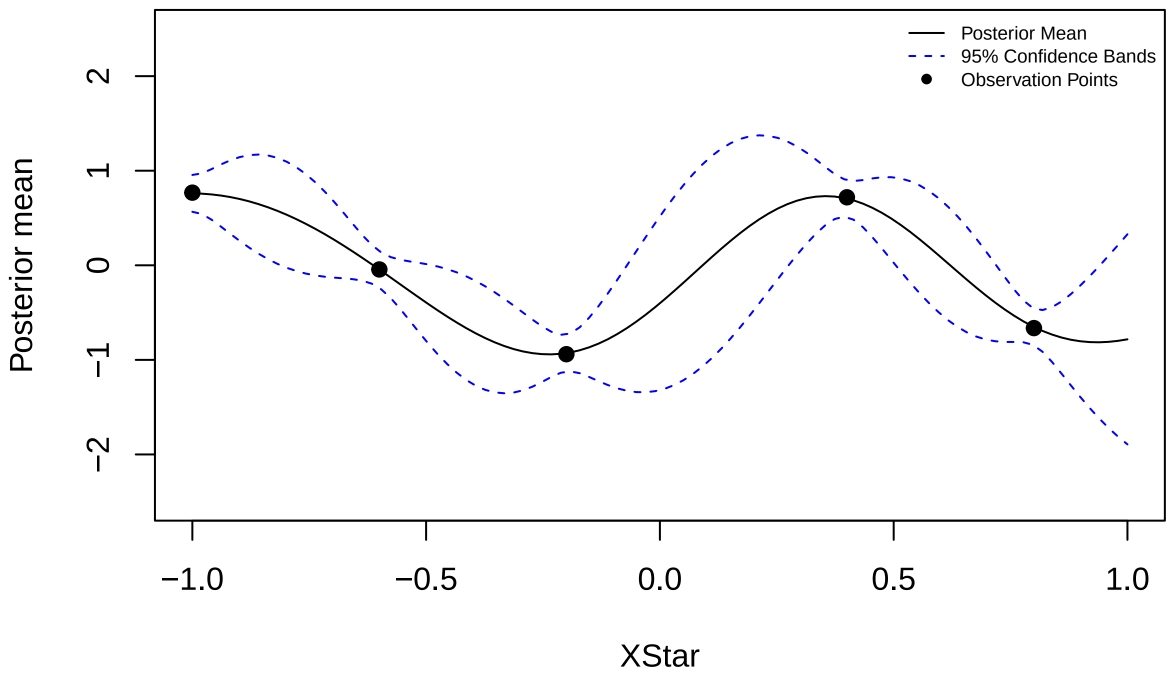


```r
# Covariance matrix of fStar.
#Covf = Kss-t(Kxs)%*%solve(Kxx + sigmaNoise^2*diag(n), Kxs)
```

```r
#Subq3: Update your posterior and Plot the posterior mean of f
xTrain <- c(0.4, -0.6)
y <- c(0.719, -0.044)
post_GP <- posteriorGP(xTrain, y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)
plot(XStar,post_GP$pos_mean, type = 'l', ylab = 'Posterior mean',ylim=c(-2.5,4))
lines(XStar, post_GP$pos_mean - 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
lines(XStar, post_GP$pos_mean + 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
points(xTrain,y,col='black', pch = 19)
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
       col = c("black", "blue", "black"), lty = c(1, 2, NA), pch = c(NA, NA, 19),
       cex = 0.6,
       bty = 'n')
```

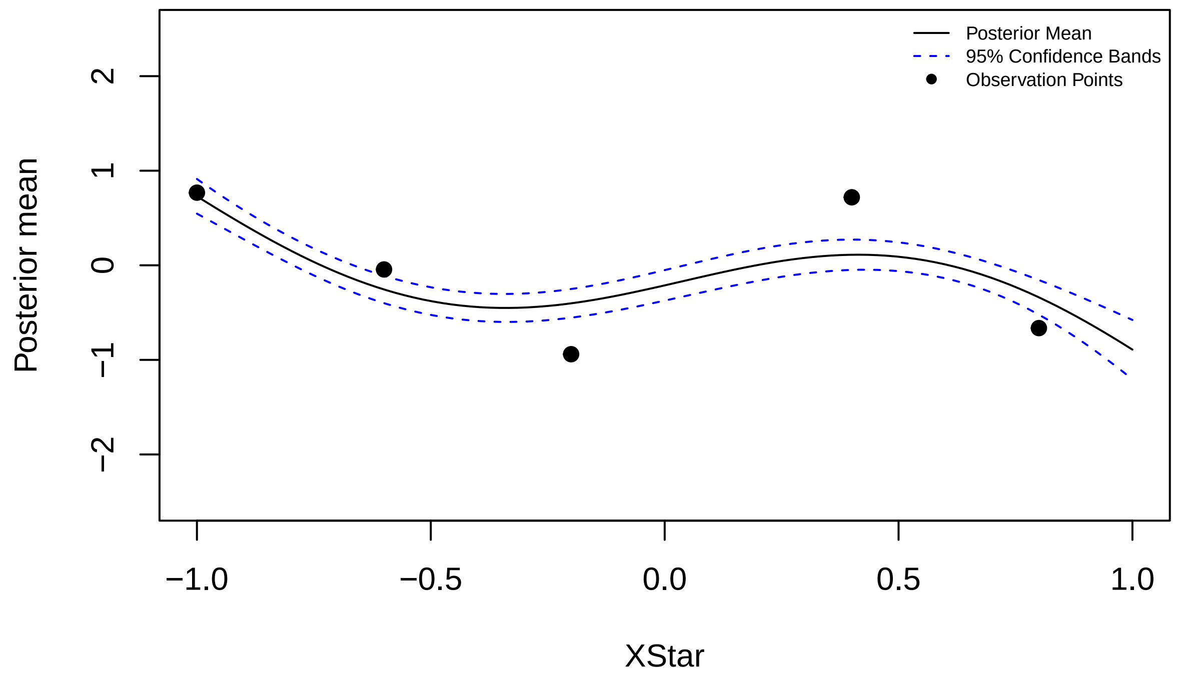


```
#Subq4: Update your posterior and Plot the posterior mean of f
xTrain <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
post_GP <- posteriorGP(xTrain, y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)
plot(XStar,post_GP$pos_mean, type = 'l', ylab = 'Posterior mean',ylim=c(-2.5,2.5))
lines(XStar, post_GP$pos_mean - 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
lines(XStar, post_GP$pos_mean + 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
points(xTrain,y,col='black', pch = 19)
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
       col = c("black", "blue", "black"), lty = c(1, 2, NA), pch = c(NA, NA, 19),
       cex = 0.6,
       bty = 'n')
```



From the CIs in the above plot, we can observe that the model is more certain at points where we have observed the prior, i.e. (-1.0, -0.6, -0.2, 0.4, 0.8). Hence the variance reduces at these points.

```r
#Subq5: Repeat (4), this time with hyperparameters sigmaF = 1 and l = 1
sigmaF <- 1#Prior hyperparameters
l <- 1#Prior hyperparameters
post_GP <- posteriorGP(xTrain, y, XStar, sigmaNoise, SquaredExpKernel, sigmaF, l)
plot(XStar,post_GP$pos_mean, type = 'l', ylab = 'Posterior mean',ylim=c(-2.5,2.5))
lines(XStar, post_GP$pos_mean - 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
lines(XStar, post_GP$pos_mean + 1.96*sqrt(diag(post_GP$pos_var)),
      col = "blue", lty = 2)
points(xTrain,y,col='black', pch = 19)
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands", "Observation Points"),
       col = c("black", "blue", "black"), lty = c(1, 2, NA), pch = c(NA, NA, 19),
       cex = 0.6,
       bty = 'n')
```



The ‘l’ hyperparameter controls the complexity of the model. A smaller l value indicates more complicated model wherein the posterior mean function is more sensitive to small scale variations. That’s why in the previous subquestion, we observed that the posterior mean function is oscillatory and more complex. This may lead to overfitting.

Whereas when we increase l, the posterior mean function is no longer sensitive to small scale variations, rather correlation between nearby points decreases more slowly with distance, emphasizing longer-range dependencies, resulting in a smoother function.

2.2: GP Regression with kernlab

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.
         header=TRUE, sep=";")
time <- 1:nrow(data) #1... 2190
temp <- data[,2]
day <- (1:nrow(data))%%365 #1...365 1...365
day[day == 0] <- 365

#Consider only time = 1, 6, 11, . . ., 2186
indx <-  seq(1, length(time), 5)
time_sub <- time[indx]
day_sub <- day[indx]
temp_sub <- data[indx,2]
```

```r
#Subq1: Define SE Kernel Function and compute covariance matrix K(X, Xstar)
SquaredExpKer <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    #Question: How is this same as ||x-x_prime|| in slide 4
    #r = sqrt(crossprod(x-y));
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaf^2*exp(-0.5*( (x-y[i])/ell )^2 )
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

Matern32 <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf^2*(1+sqrt(3)*r/ell)*exp(-sqrt(3)*r/ell))
    }
  class(rval) <- "kernel"
  return(rval)
}

# Testing our own defined kernel function.
X <- matrix(data = c(1,3,4), ncol = 1) # Given
Xstar <- matrix(data = c(2,3,4), ncol = 1) # Given
SquaredExpFunc = SquaredExpKer(sigmaf = 1, ell = 1) # SquaredExpFunc is a kernel FUNCTION.
SquaredExpFunc(c(1),c(2)) # Evaluating the kernel in x=c(1), x'=c(2).
```

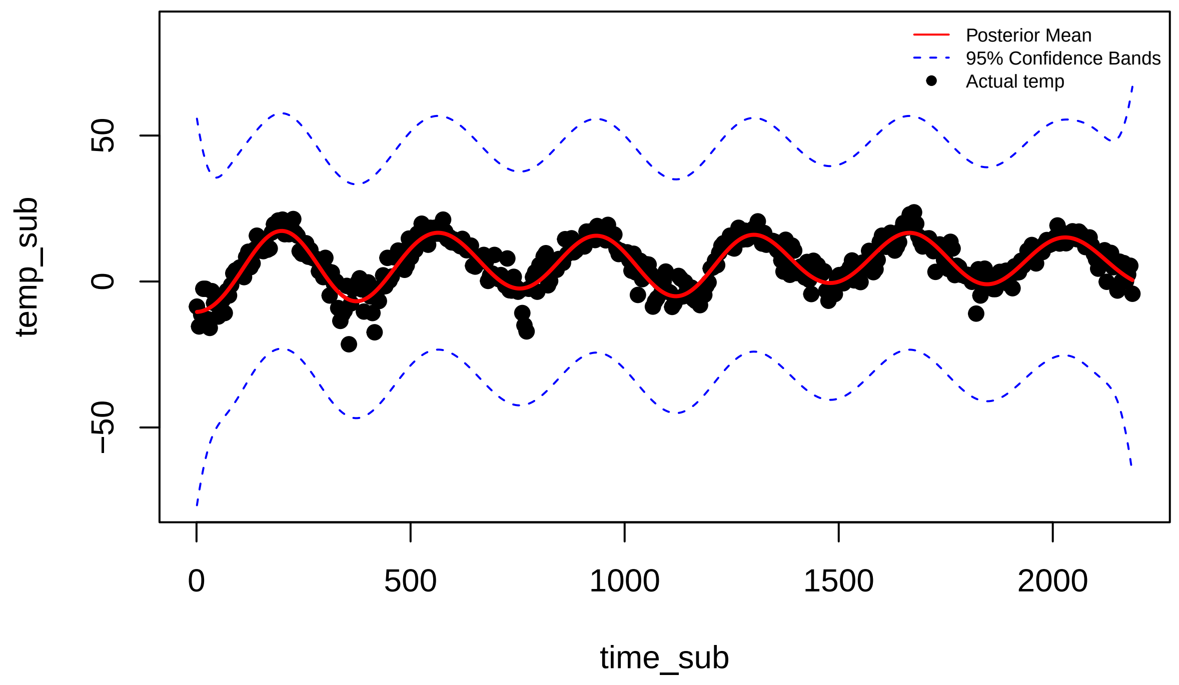
```
##           [,1]
## [1,] 0.6065307
```

```r
# Computing the whole covariance matrix K from the kernel.
kernelMatrix(kernel = SquaredExpFunc, x = X, y = Xstar) # So this is K(X,Xstar).
```

```
## 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
```

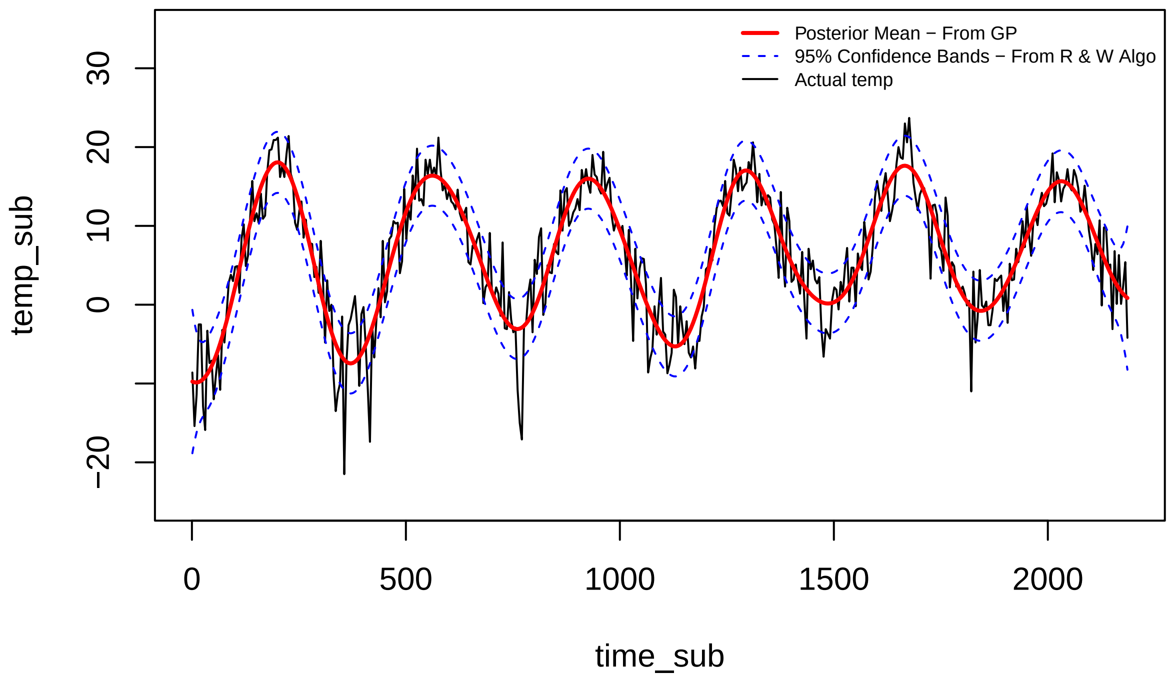
```r
#Subq2: For the given model, get the posterior mean at every data point using kernlab
#Get the sigma2NoiseTime(residual var) from a simple quadratic regression fit
quadFit <- lm(temp_sub ~  time_sub + I(time_sub^2))
sigma2NoiseTime = var(quadFit$residuals)

# Fit the GP with custom square exponential kernel
```

```r
sigmaf <-  20;ell <-  0.2
SquaredExpFunc = SquaredExpKer(sigmaf, ell) # SquaredExpFunc is a kernel FUNCTION.
GPfit <- gausspr(x = time_sub,
                 y = temp_sub,
                 scaled = TRUE,
                 type = 'regression',
                 kernel = SquaredExpFunc,
                 var = sigma2NoiseTime,
                 variance.model = TRUE)
meanPredTime <- predict(GPfit, time_sub) # Predicting the training data.
plot(time_sub, temp_sub, ylim = c(-76,86), pch = 19)
lines(time_sub, meanPredTime, col="red", lwd = 2)
#Question: The below gives an error - use variance.model = TRUE
lines(time_sub, meanPredTime+1.96*predict(GPfit,time_sub, type="sdeviation"),
      col="blue", lty = 2)
lines(time_sub, meanPredTime-1.96*predict(GPfit,time_sub, type="sdeviation"),
      col="blue", lty = 2)
legend("topright", legend = c("Posterior Mean", "95% Confidence Bands",
                              "Actual temp"),
       col = c("red", "blue", "black"),
       lty = c(1, 2, NA), pch = c(NA, NA, 19),
       cex = 0.6,
       bty = 'n')
```



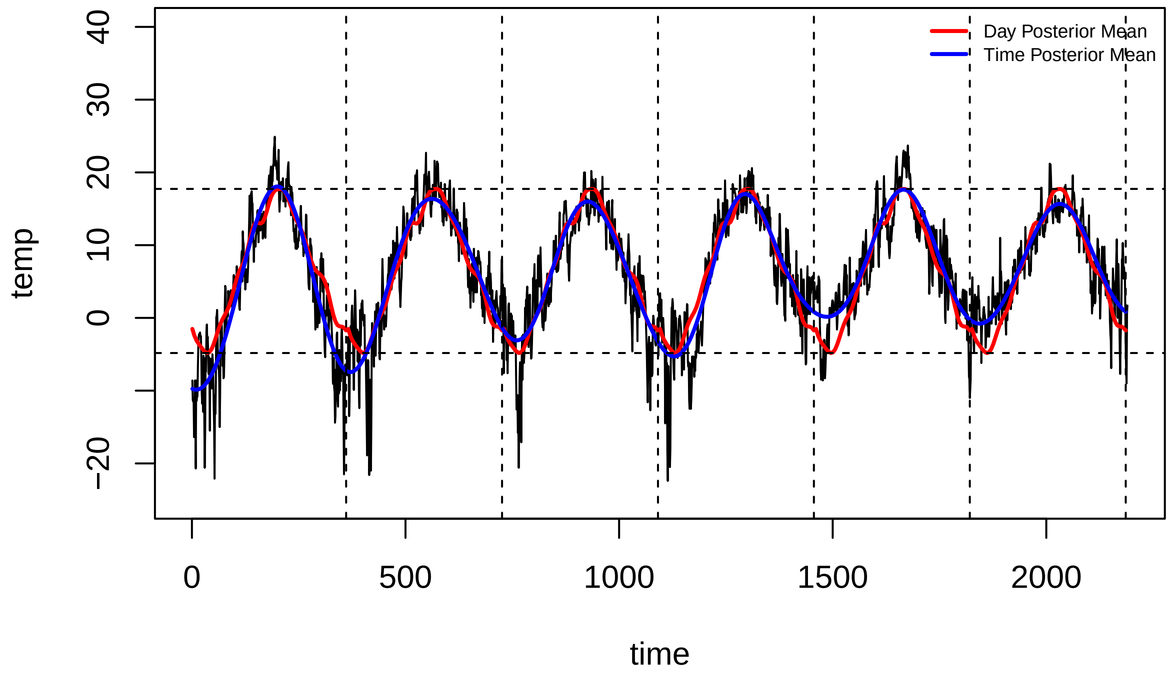
As seen above, the CI's are wrong.

```r
#Subq3: Get the posterior mean and variance using Rasmussen algorithm
sigmaf <-  20;ell <-  0.2
SquaredExpFunc = SquaredExpKer(sigmaf, ell) # SquaredExpFunc is a kernel FUNCTION.

#Scale the x and y -> (val-mean)/sd
scaled_time_sub <- scale(time_sub)
#scaled_time <- scale(time)
scaled_temp_sub <- scale(temp_sub)

#Get the new noise for scaled data
quadFit <- lm(scaled_temp_sub ~  scaled_time_sub + I(scaled_time_sub^2))
sigma2NoiseScaled = var(quadFit$residuals)

post_GP <- posteriorGP(X = scaled_time_sub,
                      y = scaled_temp_sub,
                      XStar = scaled_time_sub, #Pass the entire time for prediction
                      sigmaNoise = sqrt(sigma2NoiseScaled), #Variance
                      k = SquaredExpFunc)
#Unscale the posterior mean - this will be same as the gausspr posterior mean
unscaled_post_mean_time <- (post_GP$pos_mean * sd(temp_sub)) + mean(temp_sub)

#Unscale the posterior sd - this is the correction from gausspr
unscaled_post_sd <- sqrt(diag(post_GP$pos_var)) * sd(temp_sub)

plot(time_sub, temp_sub,  ylim = c(-25,35), type = 'l')
lines(time_sub, unscaled_post_mean_time, type = 'l', col = 'red', lwd = 2)
#lines(time_sub, unscaled_post_mean_time, type = 'l', col = 'yellow')
lines(time_sub, unscaled_post_mean_time - 1.96*unscaled_post_sd,
      col="blue", lty = 2)
lines(time_sub, unscaled_post_mean_time + 1.96*unscaled_post_sd,
      col="blue", lty = 2)
legend("topright", legend = c("Posterior Mean - From GP",
                              "95% Confidence Bands - From R & W Algo",
                              "Actual temp"),
       col = c("red", "blue", "black"),
       lty = c(1, 2, 1),
       lwd = c(2, 1, 1),
       cex = 0.6,
       bty = 'n')
```



As seen above, the CIs are better than the GP CIs,

```
#Subq4: Get the posterior mean of the day
#Question: Should sigma2NoiseDay be recalculated? - sigma2NoiseTime can be used
#But better to restimate. sigma2Noise is just a rough estimate.
#Get the sigma2NoiseDay(residual var) from a simple quadratic regression fit
quadFit <- lm(temp_sub ~  day_sub + I(day_sub^2))
sigma2NoiseDay = var(quadFit$residuals)

sigmaf <-  20;ell <-  0.2
SquaredExpFunc = SquaredExpKer(sigmaf, ell) # SquaredExpFunc is a kernel FUNCTION.
GPfit <- gausspr(x = day_sub,
                 y = temp_sub,
                 scaled = TRUE,
                 type = 'regression',
                 kernel = SquaredExpFunc,
                 var = sigma2NoiseDay)
post_mean_day <- predict(GPfit, day_sub) # Predicting the training data.
plot(time, temp, type = 'l', ylim = c(-25, 40))
lines(time_sub, post_mean_day, col="red", lwd = 2)
lines(time_sub, unscaled_post_mean_time, col="blue", lwd = 2)
legend('topright',
       legend = c('Day Posterior Mean','Time Posterior Mean'),
       col = c('red','blue'),
       lty = c(1, 1),
       lwd = c(2, 2),
       cex = 0.6,
       bty = 'n')
#time_sub[which(day_sub == 361)]
abline(v = c(time_sub[which(day_sub == 361)]), lty = 2)
abline(h = c(max(post_mean_day), min(post_mean_day)), lty = 2)
```



Compare the results of both models. What are the pros and cons of each model?

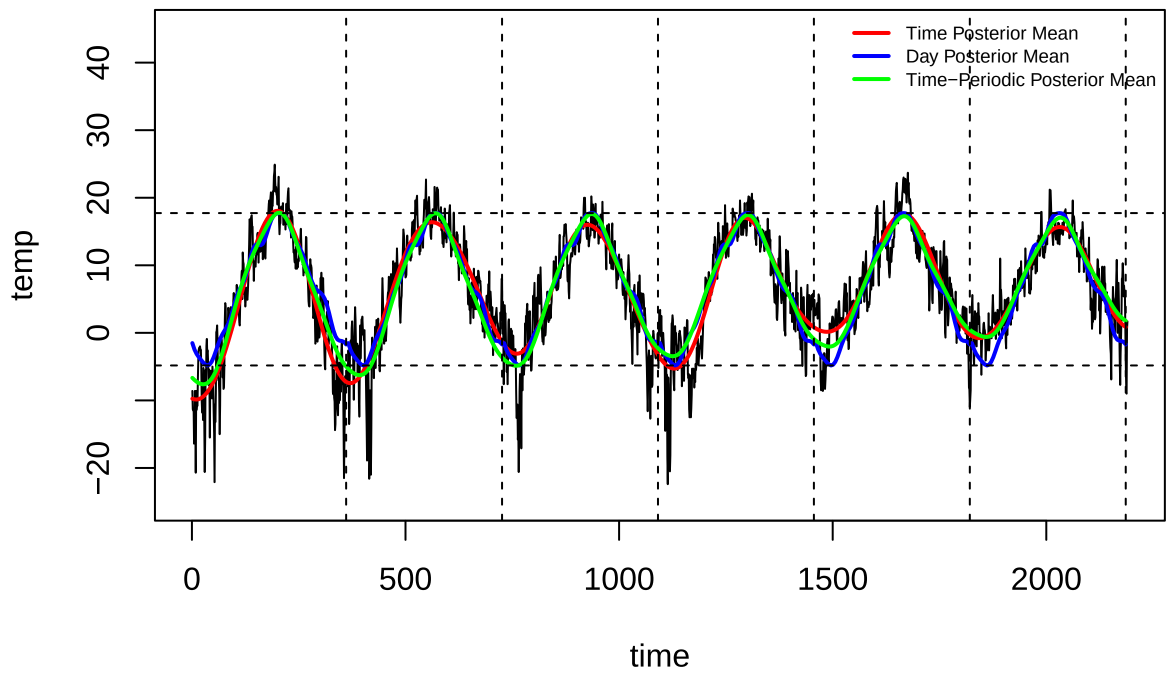
In time model, we have modelled time as sequence of values starting from 1 to 2190. Because of this, the model is able to capture the trend of the data.

Whereas in the day model, we have modelled the day as a sequence of values from 1 to 365 for each year. So, although day 1 of 2010 is far off from day 1 of 2011-15, they are highly correlated due to our data setup. Because of this correlation, the day model is able to capture the periodicity.

Pros and cons: As mentioned above, the time model is able to capture the short-term trend in the data, but is not able to capture the periodicity/seasonality. But in the day model is able to capture the periodicity well, but comparatively it does not capture the short-term trend.

```
#Subq5: Extension of the squared exponential kernel
SquaredExpPeriodicKer <- function(sigmaf = 1, ell1 = 1, ell2 = 10, d = 365/2)
{
  rval <- function(x, y = NULL) {
    #Question: How is this same as ||x-x_prime|| in slide 4
    #r = sqrt(crossprod(x-y));
    n1 <- length(x)
    n2 <- length(y)
```

```r
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaf^2*exp(-0.5*( (x-y[i])/ell2 )^2 )*
        exp(-2 * (sin(( pi*(x-y[i]) )/d )/ell1)^2)
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

sigmaf <- 20;ell1 <- 1;ell2 <- 10; d = 365/sd(time)
SquaredExpPeriodicFunc = SquaredExpPeriodicKer(sigmaf, ell1, ell2, d) # SquaredExpPeriodicFunc is a ker
GPfit <- gausspr(x = time_sub,
                 y = temp_sub,
                 scaled = TRUE,
                 type = 'regression',
                 kernel = SquaredExpPeriodicFunc,
                 var = sigma2NoiseTime)
post_mean_time_periodic <- predict(GPfit, time_sub) # Predicting the training data.
plot(time, temp, type = 'l', ylim = c(-25, 45))
lines(time_sub, unscaled_post_mean_time, col="red", lwd = 2)
lines(time_sub, post_mean_day, col="blue", lwd = 2)
lines(time_sub, post_mean_time_periodic, col="green", lwd = 2)
legend('topright',
       legend = c('Time Posterior Mean','Day Posterior Mean',
                  'Time-Periodic Posterior Mean'),
       col = c('red','blue', 'green'),
       lty = c(1, 1, 1),
       lwd = c(2, 2, 2),
       cex = 0.6,
       bty = 'n')
abline(v = c(time_sub[which(day_sub == 361)]), lty = 2)
abline(h = c(max(post_mean_day), min(post_mean_day)), lty = 2)
```



Compare and discuss the results:

As shown above, the Time-Periodic Posterior mean is able to capture both the trend and the seasonality in the data, because the local periodic kernel is a combination of the squared exponential kernel(which capture the trend), and the periodic kernel(which capture the seasonality.)

2.3: GP Classification with kernlab

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])#For classification

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

```
#Subq1:
#Use the R package kernlab to fit a Gaussian process classification model
#library(kernlab)
GPfit = gausspr(fraud~varWave + skewWave, data_train)
```

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

```r
pred_fraud <- predict(GPfit, data_train[, c('varWave', 'skewWave')])

tab1 <- table('prediction' = pred_fraud, 'true'=data_train$fraud)
cat('The confusion matrix:', '\n')
```

```
## The confusion matrix:
```

```r
tab1
```

```
##           true
## prediction   0   1
##          0 503  18
##          1  41 438
```

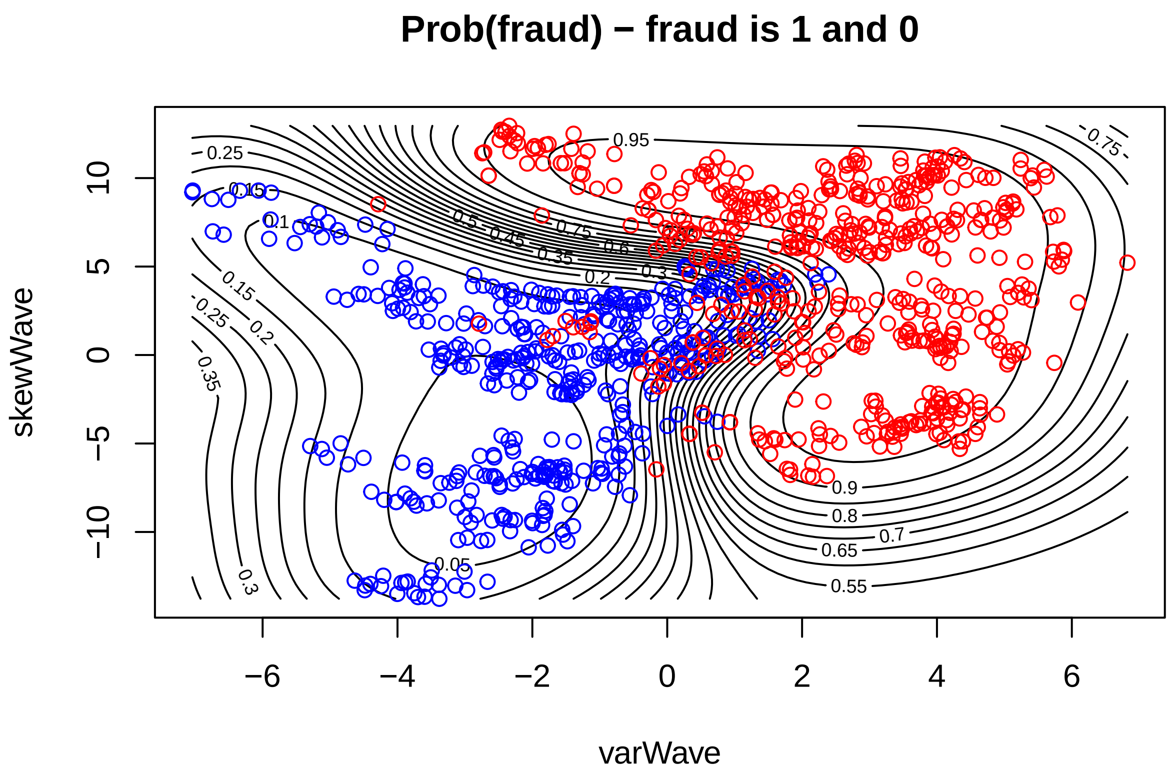
```r
acc1 <- sum(diag(tab1))/sum(tab1)
cat('The accuracy rate:','\n')
```

```
## The accuracy rate:
```

```r
acc1
```

```
## [1] 0.941
```

```r
#Plot contours of the prediction probabilities over a suitable grid of values
#for varWave and skewWave
library(AtmRay)
x1 <- seq(min(data_train$varWave),max(data_train$varWave),length=100)
x2 <- seq(min(data_train$skewWave),max(data_train$skewWave),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- c('varWave', 'skewWave')
probPreds <- predict(GPfit, gridPoints, type="probabilities")

# Plotting for Prob(fraud)
#Question: fraud == 0 or 1?
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'Prob(fraud) - fraud is 1 and 0')
points(data_train[data_train$fraud ==1, 1],data_train[data_train$fraud ==1, 2],col="blue")
points(data_train[data_train$fraud ==0, 1],data_train[data_train$fraud ==0, 2],col="red")
```



```r
#Subq2: Using the estimated model from (1), make predictions for the test set.
SelectTest = setdiff(1:dim(data)[1], SelectTraining)
data_test <- data[SelectTest,]

pred_fraud_test <- predict(GPfit, data_test[, c('varWave', 'skewWave')])

tab2 <- table('prediction' = pred_fraud_test, 'true'=data_test$fraud)
cat('The confusion matrix:', '\n')
```

```
## The confusion matrix:
```

```r
tab2
```

```
##             true
## prediction    0   1
##           0 199   9
##           1  19 145
```

```r
cat('The accuracy rate:','\n')
```

```
## The accuracy rate:
```

```r
mean(pred_fraud_test == data_test$fraud)
```

```
## [1] 0.9247312
```

```r
#Subq3:  Train a model using all four covariates
GPfit2 = gausspr(fraud~., data_test)
```

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

```r
pred_fraud_test2 <- predict(GPfit2, data_test[,])

tab2 <- table('prediction' = pred_fraud_test2, 'true'=data_test$fraud)
cat('The confusion matrix:', '\n')
```

```
## The confusion matrix:
```

```r
tab2
```

```
##           true
## prediction   0   1
##          0 218   0
##          1   0 154
```

```r
cat('The accuracy rate:','\n')
```

```
## The accuracy rate:
```

```r
mean(pred_fraud_test2 == data_test$fraud)
```

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

Compare the accuracy to the model with only two covariates:

As seen in the contour plot, there are quite a lot data points which are near the decision boundary when only 2 covariates are considered for the classification. Some of the these points in the decision boundary were getting missclassified.

Whereas, when the covariates are increased to 4, these data points at the decision boundaries are correctly classified, as the 4 covarites are needed to capture the true decision boundary between the classes.