Gaussian Processes - Individual report

Nicolas Taba (nicta839)

12/10/2021

Implementing GP regression

Question 1

We are asked to implement code simulating from the posterior distribution using the squared exponential kernel. We use the kernel function from the course page in this implementation.

```r
# (from course page)
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 )
  }
  return(K)
}
```

```r
posteriorGP <- function(X, y, XStar, hyperParam, sigmaNoise, k){
  # Calculate K
  K <- k(x1 = X, x2 = X, sigmaF = hyperParam[1], l = hyperParam[2]) + sigmaNoise^2*diag(length(X))
  # Cholesky decomposition for numerical stability
  L <- t(chol(K))
  alpha <- solve(t(L), solve(L, y))

  # posterior mean
  kstar <- k(x1 = X, x2 = XStar, sigmaF = hyperParam[1], l = hyperParam[2])
  #variance
  v <- solve(L, kstar)
  Vfstar <- k(x1 = XStar, x2 = XStar, sigmaF = hyperParam[1], l = hyperParam[2]) - t(v)%*%v

  fstar <- t(kstar)%*%alpha

  result <- list("mean" = fstar, "variance" = Vfstar)

  return(result)
}
```

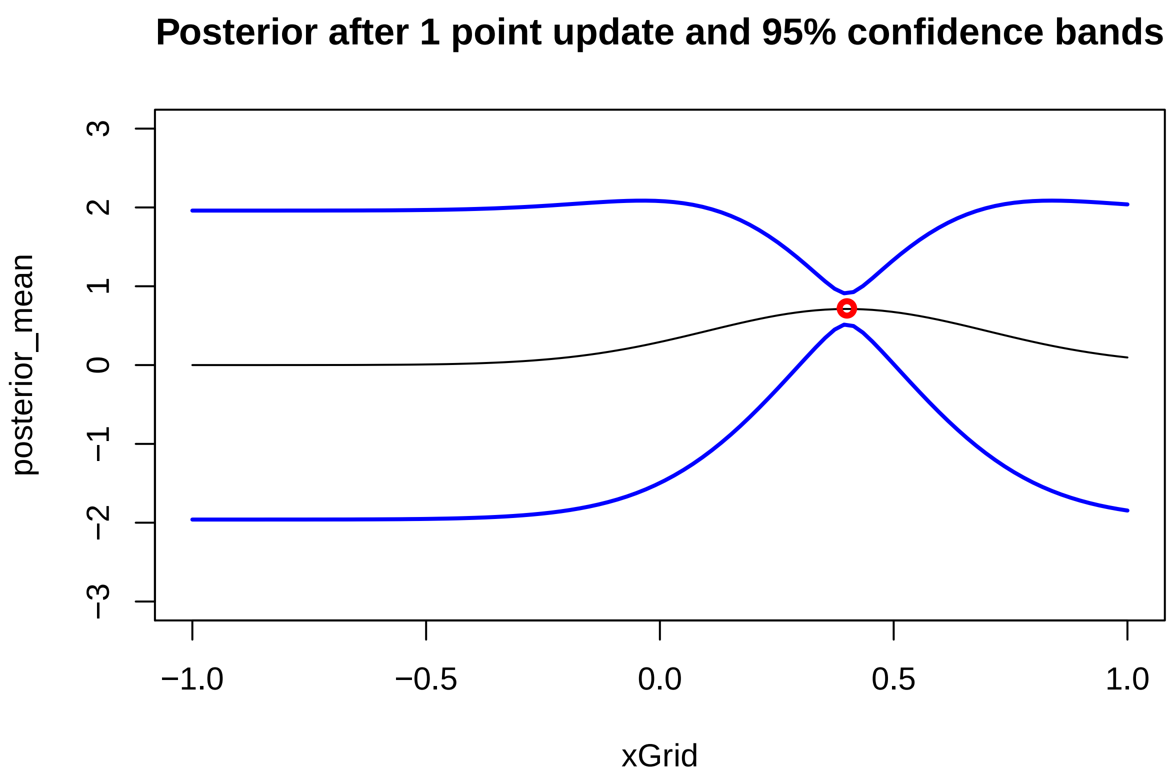
Question 2

We are asked to compute the GP regression after updating with one data point. We reuse the code from the course page to plot.

```r
# Setting up parameters
hyperparameters <- c(1, 0.3)
x <- 0.4
y <- 0.719
sigma_n <- 0.1
xGrid <- seq(-1,1,length=100)
```

```r
# Calculate the posterior mean/variance for 1 point update
update <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigma_n, 
```

```r
posterior_mean <- update$mean
posterior_variance <- diag(update$variance)

# Plot posterior mean
plot(xGrid, posterior_mean, type="l", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
points(x=0.4, y=0.719, col="red", lwd=3)
title("Posterior after 1 point update and 95% confidence bands")
```



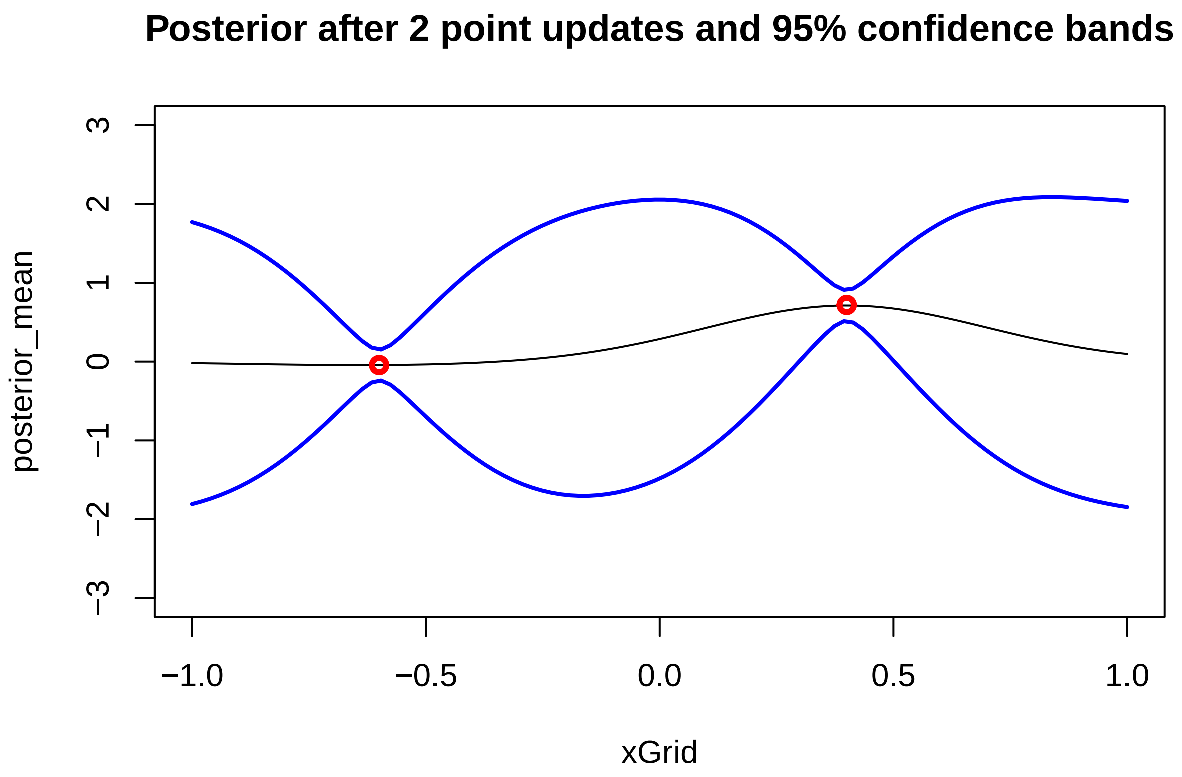
Question 3

We repeat the exercise with 2 points.

```
hyperparameters <- c(1, 0.3)
x <- c(0.4, -0.6)
y <- c(0.719, -0.044)
sigma_n <- 0.1
xGrid <- seq(-1,1,length=100)
```

```
update2 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigma_n,
```

```
posterior_mean <- update2$mean
posterior_variance <- diag(update2$variance)

# Plot posterior mean
plot(xGrid, posterior_mean, type="l", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
points(x=x, y=y, col="red", lwd=3)
title("Posterior after 2 point updates and 95% confidence bands")
```



Question 4

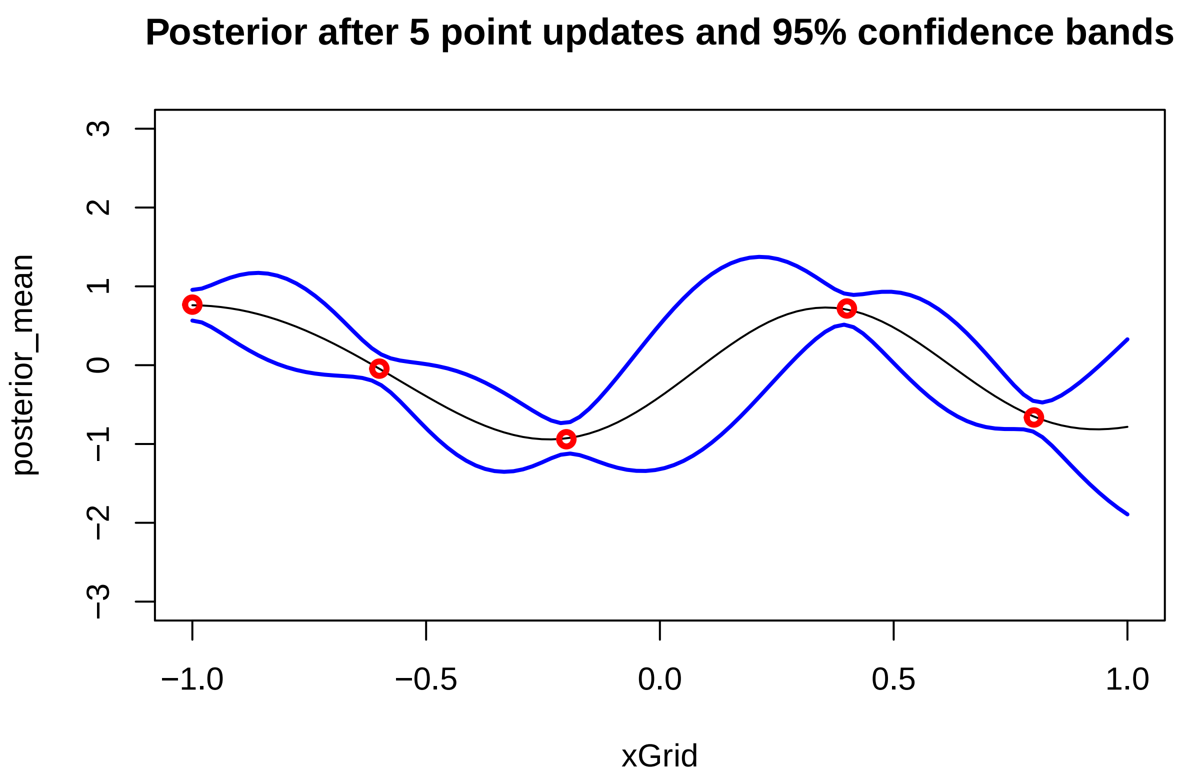
We repeat the exercise with 5 points.

```
hyperparameters <- c(1, 0.3)
x <- c(-1.0 ,-0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
```

```
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```
update3 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigma_n,
```

```
posterior_mean <- update3$mean
posterior_variance <- diag(update3$variance)

# Plot posterior mean
plot(x = xGrid,y = posterior_mean, type="l", ylim = c(-3,3))

# Plot posterior 95% confidence intervals
lines(x = xGrid, y = posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(x = xGrid, y = posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
points(x=x, y=y, col="red", lwd= 3)
title("Posterior after 5 point updates and 95% confidence bands")
```



The more points we add, the "tighter" the confidence bands.

Question 5

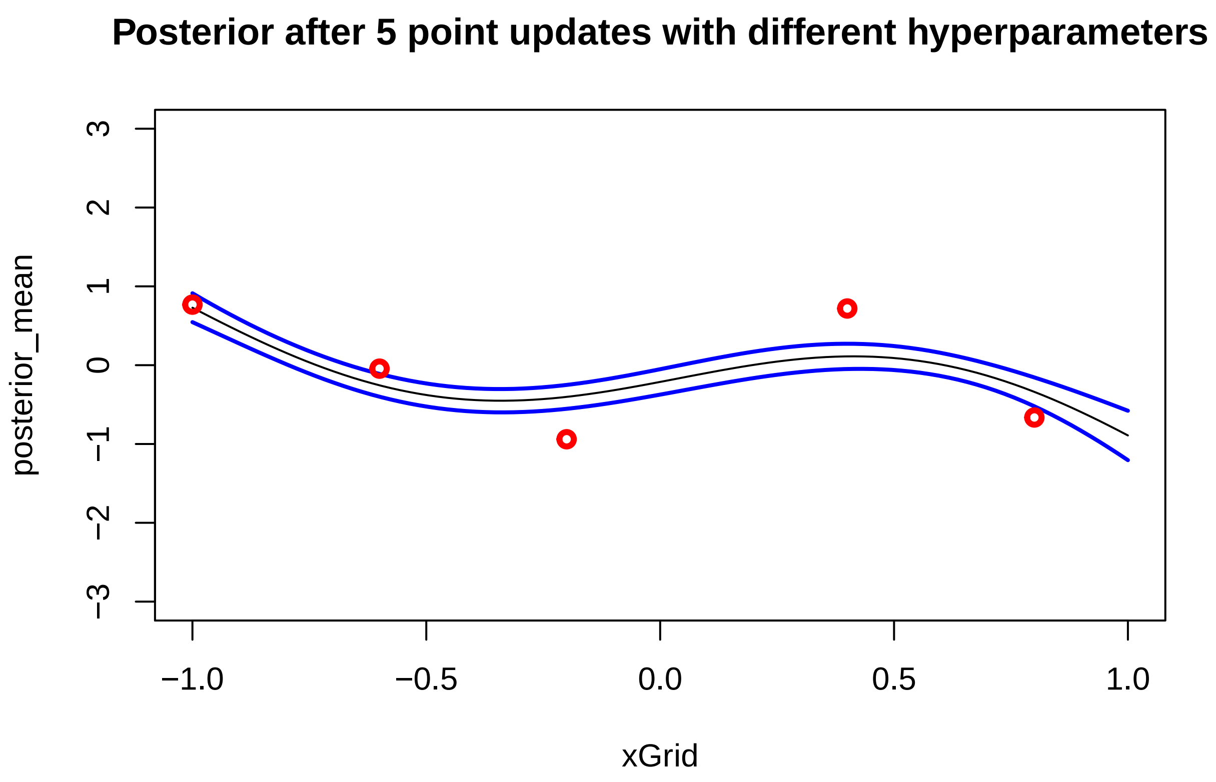
We repeat the exercise with different parameters.

```
hyperparameters <- c(1, 1)
x <- c(-1.0 ,-0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719, -0.664)
```

```
sigmaN <- 0.1
xGrid <- seq(-1,1,length=100)
```

```
update5 <- posteriorGP(X = x, y = y, XStar = xGrid, hyperParam = hyperparameters, sigmaNoise = sigma_n,
```

```
posterior_mean <- update5$mean
posterior_variance <- diag(update5$variance)

# Plot posterior mean
plot(xGrid, posterior_mean, type="l", ylim = c(-3, 3)) #set y axis to be the same

# Plot posterior 95% confidence intervals
lines(xGrid, posterior_mean - 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
lines(xGrid, posterior_mean + 1.96*sqrt(posterior_variance), col = "blue", lwd = 2)
points(x=x, y=y, col="red", lwd= 3)
title("Posterior after 5 point updates with different hyperparameters")
```



From this graph, we can see that the general trend of the curve is not followed and the confidence bands are quite narrow. The parameter l controls the smoothness of the function. With a high value of l, the smoother the fit, but it does not follow well the behavior of the data points.

GP regression with kernlab

```
temperature <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTu
temperature$time <- as.numeric(rownames(temperature))
temperature$day <- rep(c(1:365))
index <- seq(from=1, to=nrow(temperature), by = 5)
temperature <- temperature[index,]
```

Question 1

We are asked to implement our own square exponential kernel.

```
library(kernlab)
```

```
## Warning: package 'kernlab' was built under R version 4.0.3
```

```
# Gaussian kernel
Gkernel <- function(sigmaf=1,ell=1){
    rval <- function(x1, x2){
      K <- sigmaf^2*exp(-crossprod(x1-x2)/(2*ell^2) )
      return(K)
      }
    class(rval) <- "kernel" #kernelMatrix() needs this to be of the kernel class
    return(rval)
}
```

```
kernel_function <- Gkernel(sigmaf = 1, ell = 1)
kernel_function(x1 = 1, x2 = 2)
```

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

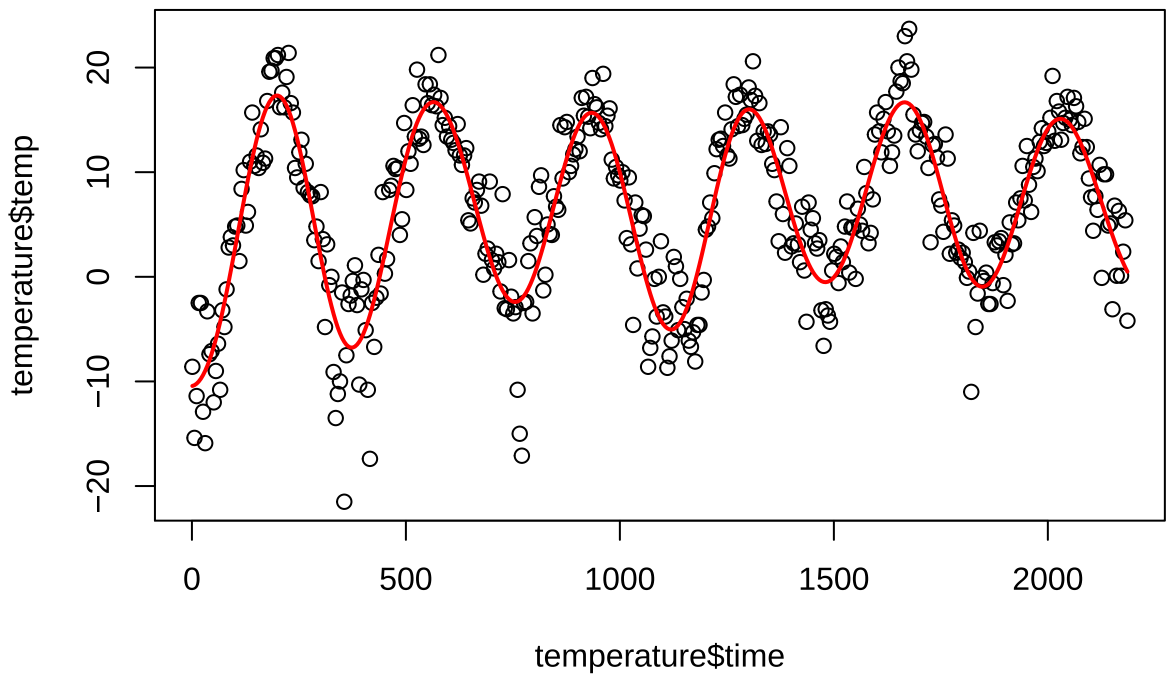
Now we compute the covariance matrix.

```
X <- c(1,3,4)
Xstar <- c(2,3,4)

K <- kernelMatrix(kernel = kernel_function, x = X, y = Xstar)
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
```

Question 2

```
# Fit the GP with home made SE Kernel
sigmaf <- 20
ell <- 0.2
Fit <- lm(temperature$temp ~ temperature$time + I(temperature$time^2))
sigmaNoise = sd(Fit$residuals)

GPfit <- gausspr(temperature$time, temperature$temp, kernel = Gkernel, kpar = list(sigmaf = sigmaf, ell=
meanPred <- predict(GPfit, temperature$time)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="red", lwd = 2)
```

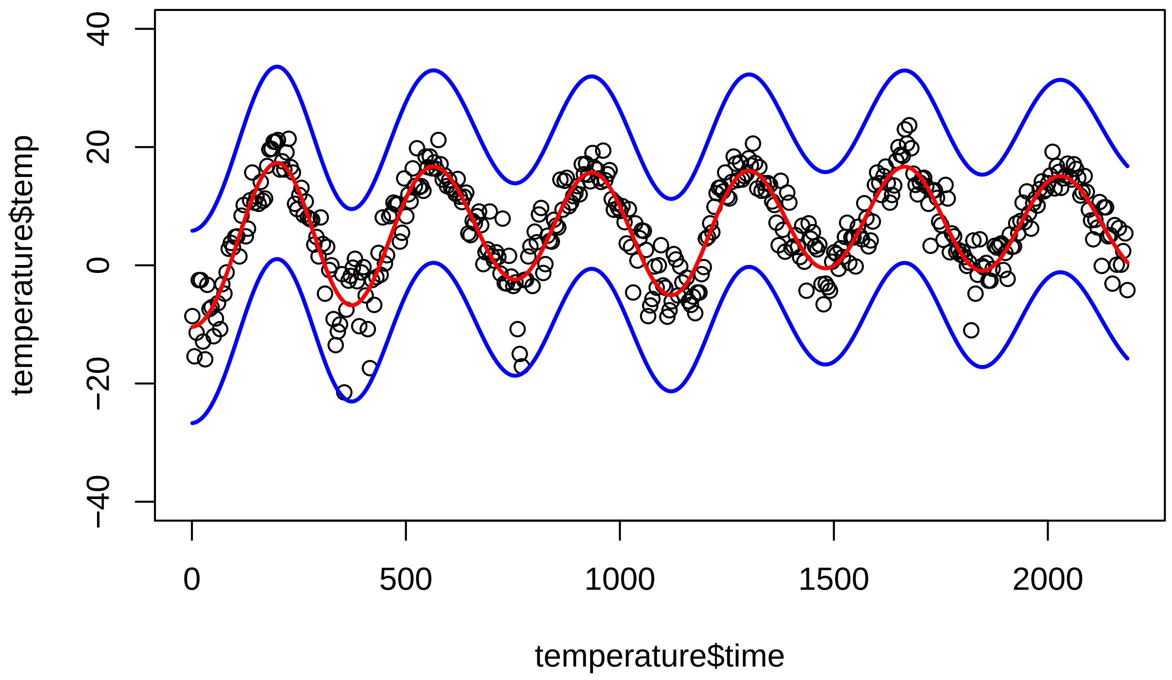


Question 3

we now have to compute the posterior variance

```r
XStar <- seq(1, length(temperature$time), by = 1)
hyperparameters <- c(20, 0.2)
# We use the scale() method as explained in the exercise sheet
# reload data if not positive semi-definite
posterior <- posteriorGP(X = scale(temperature$time), y = temperature$temp, XStar = scale(XStar), hyperP


posterior_variance <- diag(posterior$variance)
variance_temp <- var(temperature$temp)
```

```r
# plot everything
plot(x = temperature$time, y = temperature$temp, ylim = c(-40, 40))
lines(temperature$time, meanPred, col="red", lwd = 2)
lines(temperature$time, meanPred + 1.96*sqrt(variance_temp), col="blue", lwd=2)
lines(temperature$time, meanPred - 1.96*sqrt(variance_temp), col="blue", lwd=2)
```

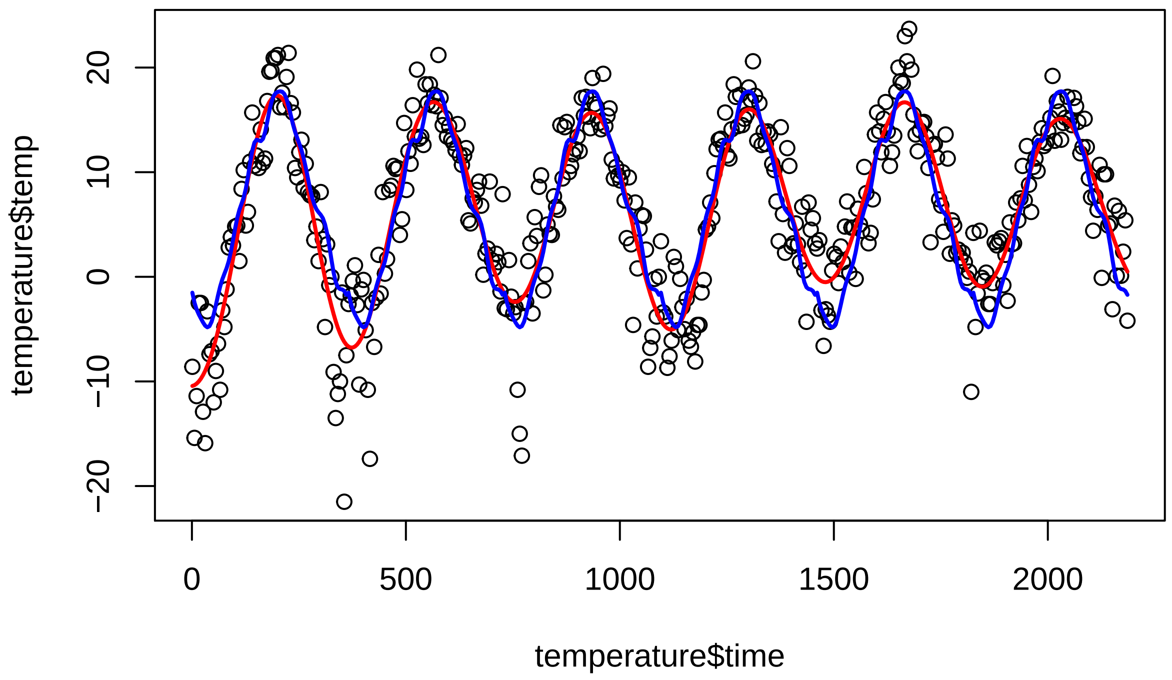


There is a lot of uncertainty about the model so it is difficult to say anything about predictions given this model since the function can take values that have a large range.

Question 4

We are asked to compare the previous model with one that uses day as a variable instead in the squared exponential model.

```
sigmaf <- 20
ell <- 0.2

Fit <- lm(temperature$temp ~ temperature$day + I(temperature$day^2))
sigmaNoise = sd(Fit$residuals)
GPfit <- gausspr(temperature$day, temperature$temp, kernel = Gkernel, kpar = list(sigmaf = sigmaf, ell=
meanPred1 <- predict(GPfit, temperature$day)


plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred, col="red", lwd = 2)
lines(temperature$time, meanPred1, col="blue", lwd = 2)
```



The first model captures the general behavior of the data, but fails to capture deviations from year to year. The confidence bands are also quite large so there is a lot of uncertainty on the actual value of the temperature. The second model seems to capture better variations between the years. (Need to do the same work with the confidence bands)

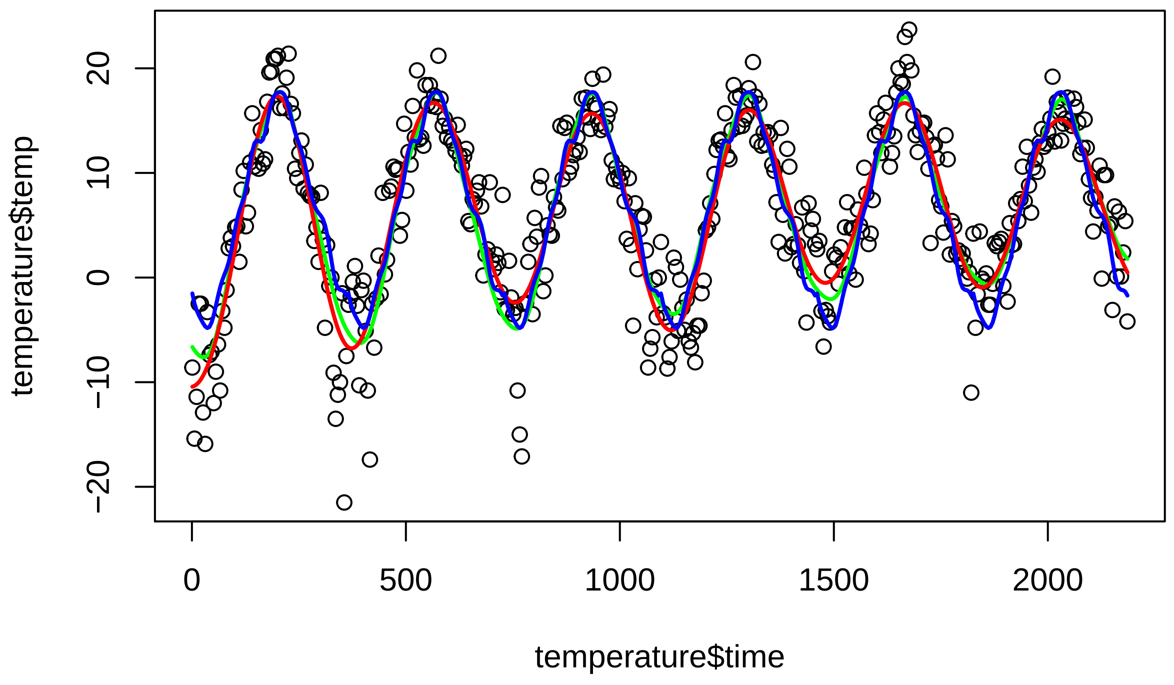
Question 5

We are asked to implement a periodic kernel.

```r
# SE kernel, so that it can go into kernelmatrix function
Periodkernel <- function(sigmaf,ell1, ell2, d)
  {
    kernel <- function(x1, x2){
      num1 <- 2*(sin(pi*abs(x1-x2)/d)^2)
      denom1 <- ell1^2
      num2 <- abs(x1-x2)^2
      denom2 <- ell2^2
      K <- sigmaf^2*exp(-num1 / denom1)*exp(-0.5*(num2)/denom2)
      return(K)
      }
    class(kernel) <- "kernel"
    return(kernel)
}
```

```r
sigmaf <- 20
ell1 <- 1
ell2 <- 10
d <- 365/sd(temperature$time)
```

```r
Fit <- lm(temperature$temp ~ temperature$time + I(temperature$time^2))
sigmaNoise = sd(Fit$residuals)
GPfit <- gausspr(temperature$time, temperature$temp, kernel = Periodkernel, kpar = list(sigmaf = sigmaf

meanPred2 <- predict(GPfit, temperature$time)
plot(x = temperature$time, y = temperature$temp)
lines(temperature$time, meanPred2, col="green", lwd = 2)
lines(temperature$time, meanPred, col="red", lwd = 2)
lines(temperature$time, meanPred1, col="blue", lwd = 2)
```



The periodic kernel seems to balance the good points of both previous kernel. It varies from year to year capturing the trend of the data but is also not too much influenced by the local density of points.

GP classification with kernlab

Question 1

We are asked to use kernlab to perform a GP classification for fraud data.

```r
library(pracma)
```

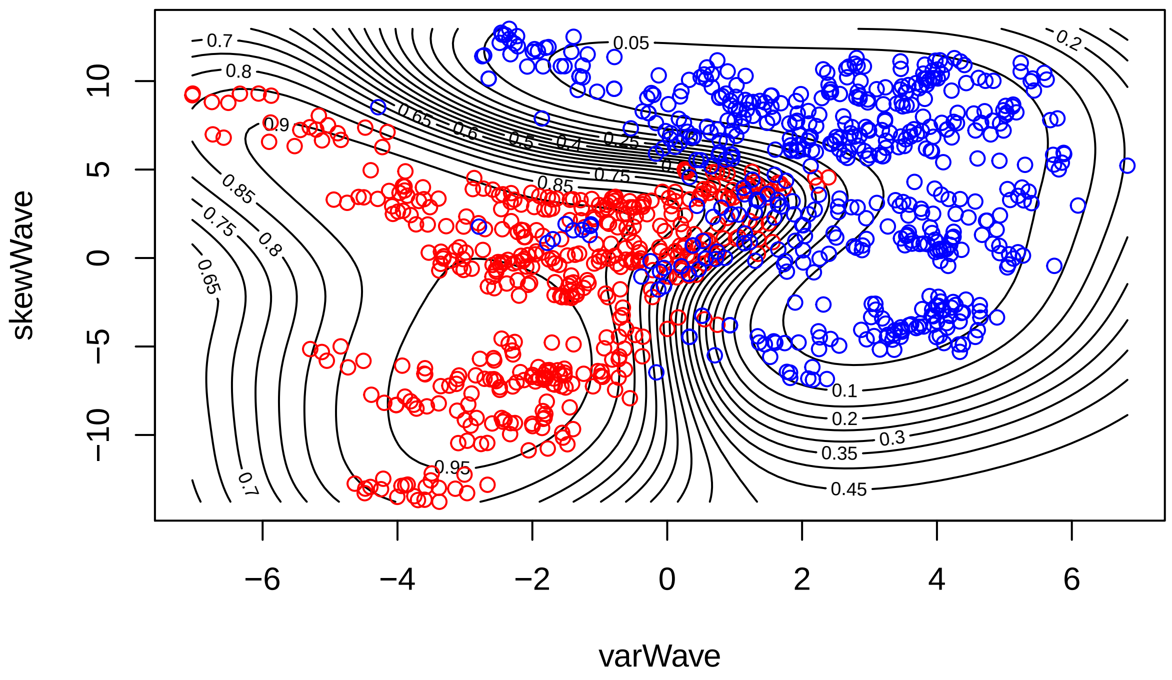
```
## Warning: package 'pracma' was built under R version 4.0.5

##
## Attaching package: 'pracma'
## The following objects are masked from 'package:kernlab':
##
##     cross, eig, size
```

```
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])

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

```
grid_x <-range(Train[, 1])
grid_x <-seq(grid_x[1], grid_x[2], length = 100)
grid_y <-range(Train[, 2])
grid_y <-seq(grid_y[1], grid_y[2], length = 100)
gridPoints <-meshgrid(grid_x, grid_y)
gridPoints <-cbind(c(gridPoints$X),c(gridPoints$Y))
gridPoints <-data.frame(gridPoints)
names(gridPoints) <-c("varWave","skewWave")

GPfit <- gausspr(fraud ~ varWave + skewWave, data=Train)
```

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

```
probabilities <- predict(GPfit, gridPoints, type="probabilities")

contour(grid_x, grid_y, matrix(probabilities[, 2], 100, byrow = TRUE),
        20, xlab = "varWave", ylab = "skewWave")
points(Train[Train[, 5] == 1, "varWave"], Train[Train[, 5] == 1, "skewWave"],
       col = "red")
points(Train[Train[, 5] == 0, "varWave"], Train[Train[, 5] == 0, "skewWave"],
       col = "blue")
```



```r
# Fit using only varWave and skewWave
GPfit <- gausspr(fraud ~ varWave + skewWave, data=Train)
```

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

```r
classification <- predict(GPfit, Train[,1:2])

CM1 <- table(classification, Train[, 5])
acc1 <- (CM1[1,1]+CM1[2,2])/sum(CM1)

CM1
```

```
##
## classification   0   1
##              0 503  18
##              1  41 438
```

```r
acc1
```

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

Question 2

```r
classification2 <- predict(GPfit, Test[,1:2])



CM2 <- table(classification2, Test[, 5])
acc2 <- (CM2[1,1]+CM2[2,2])/sum(CM2)
```

```r
CM2
```

```
##
## classification2   0   1
##               0 199   9
##               1  19 145
```

```r
acc2
```

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

Question 3

```r
GPfit_all <- gausspr(fraud ~ ., data=Train)
```

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

```r
classification_all <- predict(GPfit_all, Test[,1:4])

CM3 <- table(classification_all, Test[, 5])
acc3 <- (CM3[1,1]+CM3[2,2])/sum(CM3)

CM3
```

```
##
## classification_all   0   1
##                  0 216   0
##                  1   2 154
```

```r
acc3
```

```
## [1] 0.9946237
```

Accuracy increases by allowing more covariates to be taken into account.