# Advanced Machine Learning

Hugo Morvan (hugmo418)

2024-10-14

## Lab 4: Gaussian Processes

### 1. Implementing GP Regression.

This first exercise will have you writing your own code for the Gaussian process regression model: $y = f(x) + \epsilon$ with $\epsilon \sim N(0, \sigma_n^2)$ and $f \sim GP(0, k(x, x'))$

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 qmatrix. 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:

#### 1.1)

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:

- `X`: Vector of training inputs.
- `y`: Vector of training targets/outputs.
- `XStar`: Vector of inputs where the posterior distribution is evaluated, i.e. X* .
- `sigmaNoise`: Noise standard deviation $\sigma_n$ .
- `k`: Covariance function or kernel. That is, the kernel should be a separate function (see the file `GaussianProcesses.R` on the course web page).

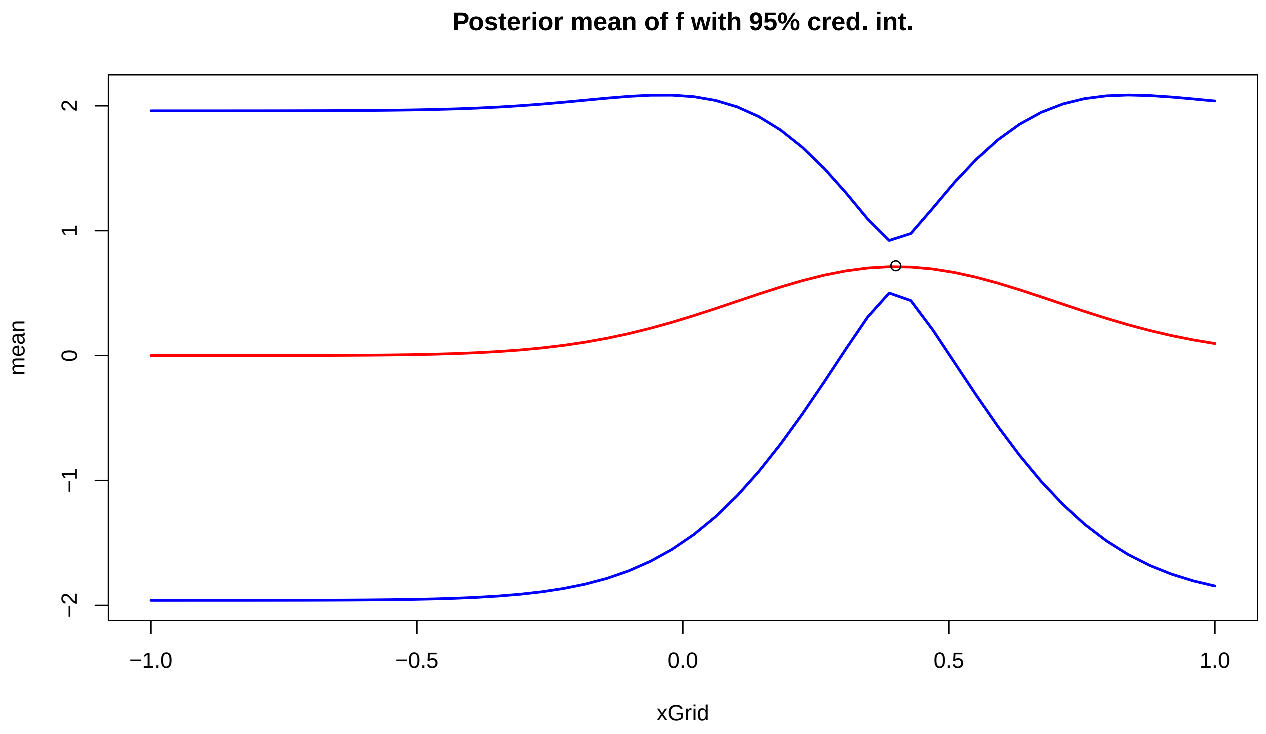
```r
library(kernlab)
library(AtmRay)
```

```r
posteriorGP <- function(X, y, Xstar, sigmaNoise, sigmaF, l){
  #Algorithm 2.1 on page 19 of Rasmussen and Willams' book.

  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)
  }

  K = SquaredExpKernel(x1 = X, x2 = X, sigmaF = sigmaF, l=l) #Covariance K(X,X)
  K_star = SquaredExpKernel(x1 = X, x2 = Xstar, sigmaF = sigmaF, l=l) #Covariance K(X,X*)
  K_star_star = SquaredExpKernel(x1 = Xstar, x2 = Xstar, sigmaF = sigmaF, l=l) #Covariance K(X*,X*)

  L = chol(K + diag(sigmaNoise^2, nrow(K))) #Cholesky

  #Note that L in the algorithm is a lower triangular matrix,
  #whereas the R function returns an upper triangular qmatrix
  L = t(L)

  #Predictive mean
  alpha = solve(t(L),solve(L,y))
  pred_mean = t(K_star) %*% alpha

  #Predictive variance
  v = solve(L, K_star)
  pred_var = diag(K_star_star - t(v)%*%v)

  return(list("mean" = pred_mean, "var" = pred_var))
}
```

**1.2)**

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

```
sigmaf = 1
sigma_n = 0.1
ell = 0.3
xGrid <- matrix(seq(-1,1,length=50))

obs = data.frame(X=c(0.4), y=c(0.719))
#Updating prior with a single observation
updated_prior = posteriorGP(obs$X, obs$y, Xstar = xGrid, sigmaNoise = sigma_n,
                            sigmaF = sigmaf, l=ell )

plot_res <- function(res, obs){
  mean = res$mean
  std = sqrt(res$var)
  plot(xGrid, mean, col="red", type='l', lwd = 2,
       ylim = range(mean, mean+1.96*std, mean-1.96*std),
       main = "Posterior mean of f with 95% cred. int.")
  lines(xGrid, mean+1.96*std,col="blue", lw=2)
  lines(xGrid, mean-1.96*std,col="blue", lw=2)
  points(obs)
}

plot_res(updated_prior, obs)
```

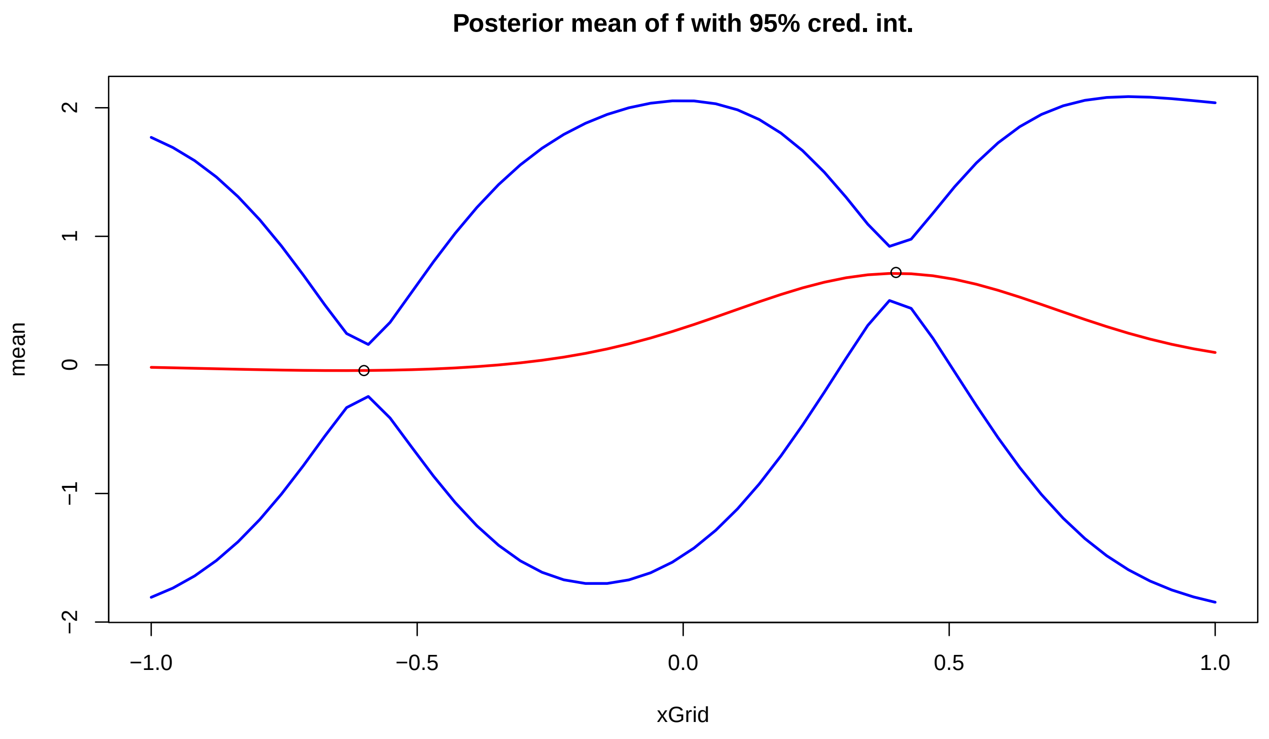


**1.3)**

Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of $f$ over the interval $x \in [-1, 1]$. Plot also 95 % probability (point-wise) 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.

```
obs = data.frame(X=c(0.4, -0.6), y=c(0.719, -0.044))
updated_prior = posteriorGP(obs$X, obs$y, Xstar=xGrid, sigmaNoise=sigma_n, sigmaF=sigmaf, l=ell )
plot_res(updated_prior, obs)
```

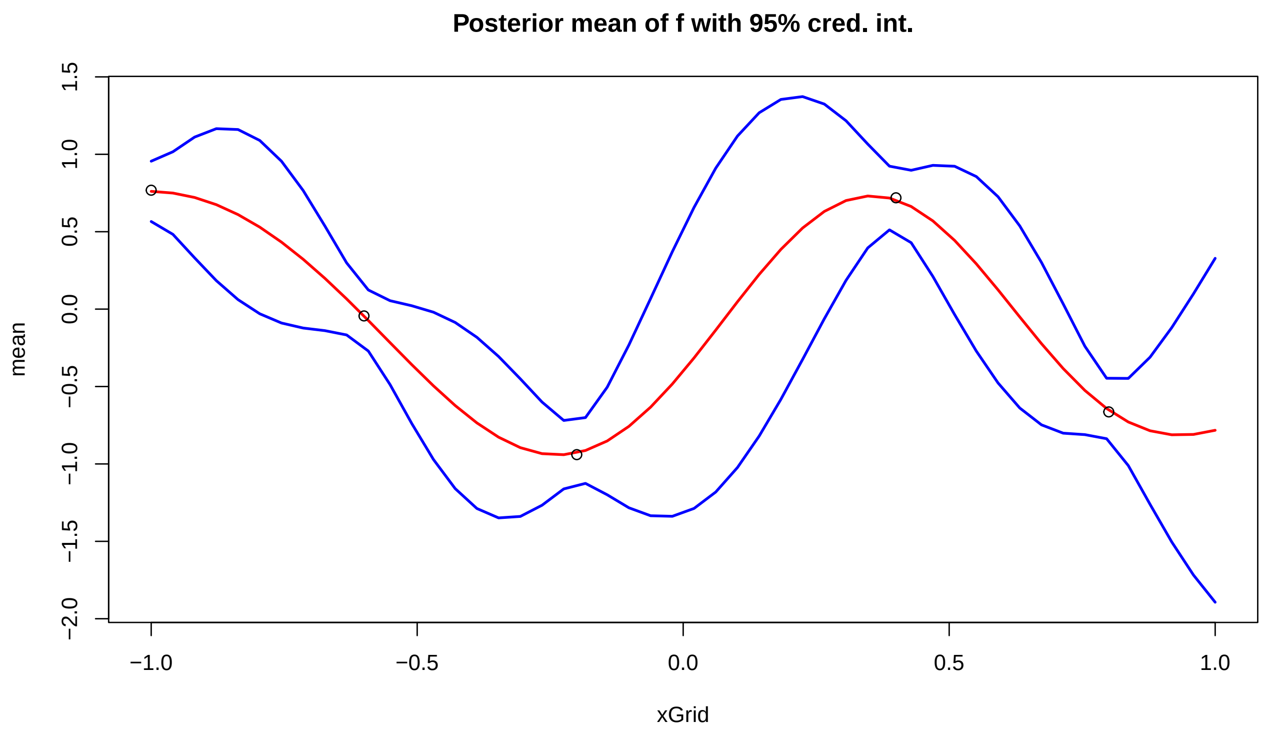


### 1.4)

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). Plot the posterior mean of $f$ over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for $f$.

| x | -1.0 | -0.6 | -0.2 | 0.4 | 0.8 |
|---|---|---|---|---|---|
| y | 0.768 | -0.044 | -0.940 | 0.719 | -0.664 |

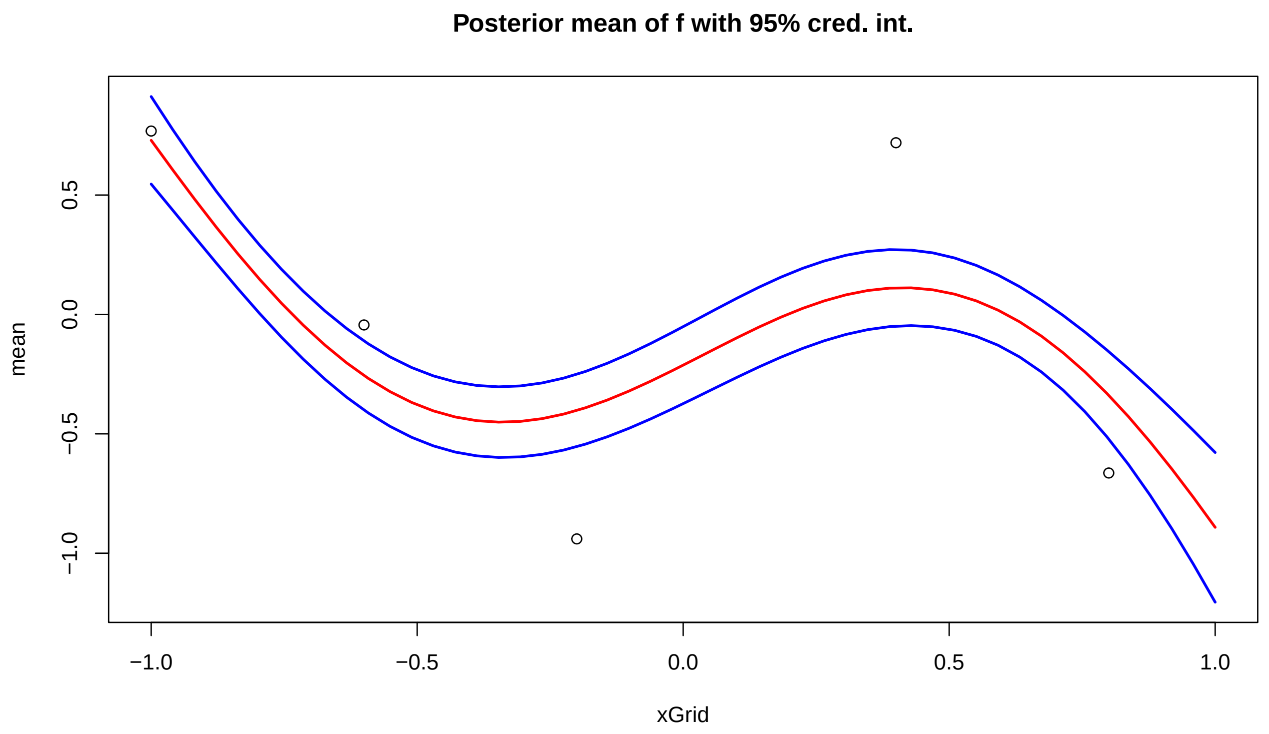
```
obs = data.frame(X=c(-1.0 , -0.6 , -0.2 , 0.4 , 0.8), y=c(0.768 , -0.044 , -0.940 , 0.719 , -0.664))
#Updating prior with a single observation
updated_prior = posteriorGP(obs$X, obs$y, Xstar=xGrid, sigmaNoise=sigma_n, sigmaF=sigmaf, l=ell )
plot_res(updated_prior, obs)
```



**1.5)**

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

```
obs = data.frame(X=c(-1.0 , -0.6 , -0.2 , 0.4 , 0.8), y=c(0.768 , -0.044 , -0.940 , 0.719 , -0.664))
updated_prior = posteriorGP(obs$X, obs$y, Xstar=xGrid, sigmaNoise=sigma_n, sigmaF=1, l=1 )
plot_res(updated_prior, obs)
```



Comparaison of the results:

Increasing $l$ in the later plot gives us a smoother function, which is to be expected given that $l$ is responsible for the smoothness of the posterior distribution.

## 2. 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. You can read the dataset with the command:

`read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv", header=TRUE, sep=";")`

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.

```r
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
dates = data$date
dates = as.Date(dates, "%d/%m/%y")
temps = as.numeric(data$temp)

t0 = dates[1]

days_to_t0 = function(day, t0){
  return(as.numeric(difftime(day, t0, units = "days"))+1)
}
time = sapply(dates, days_to_t0, t0 = t0)

days_to_newyear = function(day, t0){
  return(as.numeric(difftime(day, t0, units = "days"))%%365+1)
}
day = sapply(dates, days_to_newyear, t0 = t0)

#Subsampling every 5 datapoints
subsample <- function(x,by){
  end = length(x)
  idx = seq.int(1, end, by = by)
  return(x[idx])
}

time = subsample(time, 5)
day = subsample(day, 5)
temp = subsample(temps, 5)
```

### 2.1)

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 $X* = (2, 3, 4)^T$.

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

k = SEkernel()
print(k(1,2))
```

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

```r
X = matrix(t(c(1,3,4)))
Xstar = matrix(t(c(2,3,4)))
print(kernelMatrix(kernel = k, x = X, y = 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
```

### 2.2)

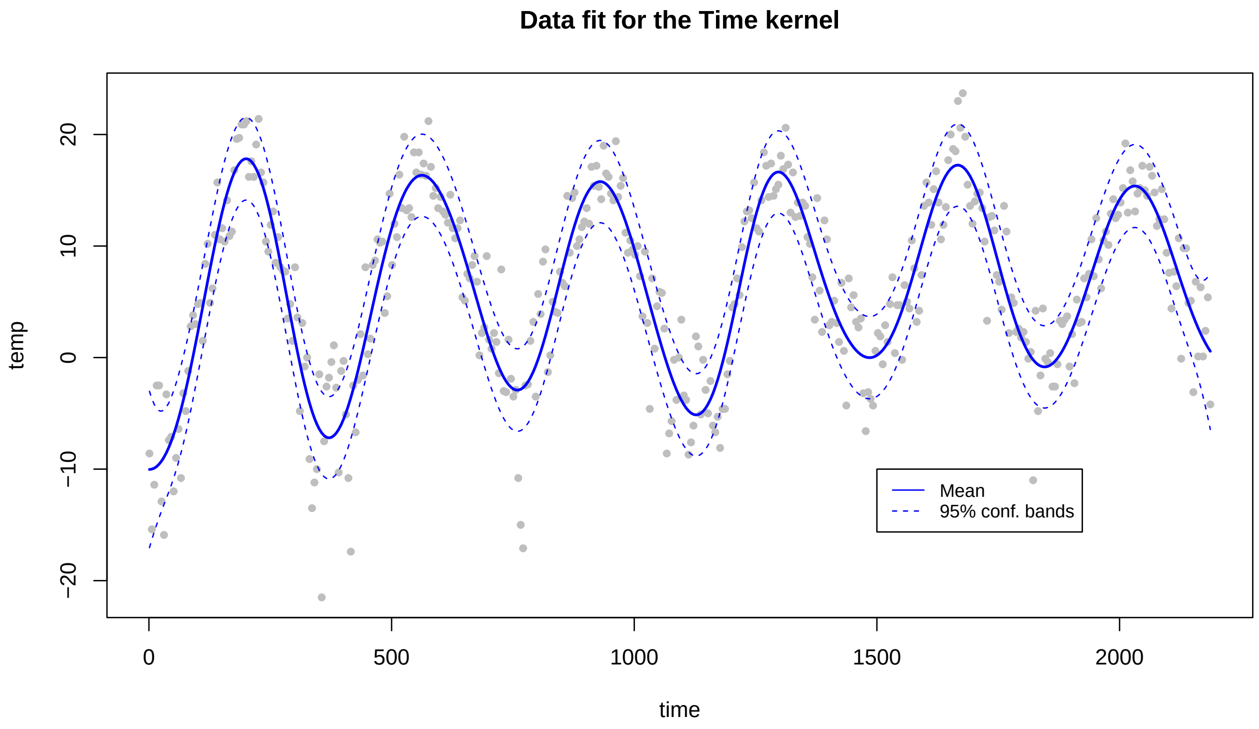
Consider first the following model:

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

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 `gausspr` function with the squared exponential function from (1) with $\sigma_f = 20$ and $l = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these $\sigma_f$ and $l$ values are not suitable). 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). Plot also the 95 % probability (pointwise) bands for $f$. Play around with different values on $\sigma_f$ and $l$ (no need to write this in the report though).

```r
quadFit <- lm(temp ~  time + I(time^2))
sigmaNoise = sd(quadFit$residuals)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
GPfit <- gausspr(time, temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE
meanPred1 <- predict(GPfit, time)
stdPred1 <- predict(GPfit,time, type="sdeviation")
```

```r
plot(time,temp, pch=20, col="grey",main = "Data fit for the Time kernel")
lines(time, meanPred1, col="blue", lwd = 2)
lines(time, meanPred1+1.96*stdPred1,col="blue", lty=2)
lines(time, meanPred1-1.96*stdPred1,col="blue", lty=2)
legend(1500, -10, legend=c("Mean", "95% conf. bands"),
       col=c("blue", "blue"), lty=c(1,2), cex=0.8)
```

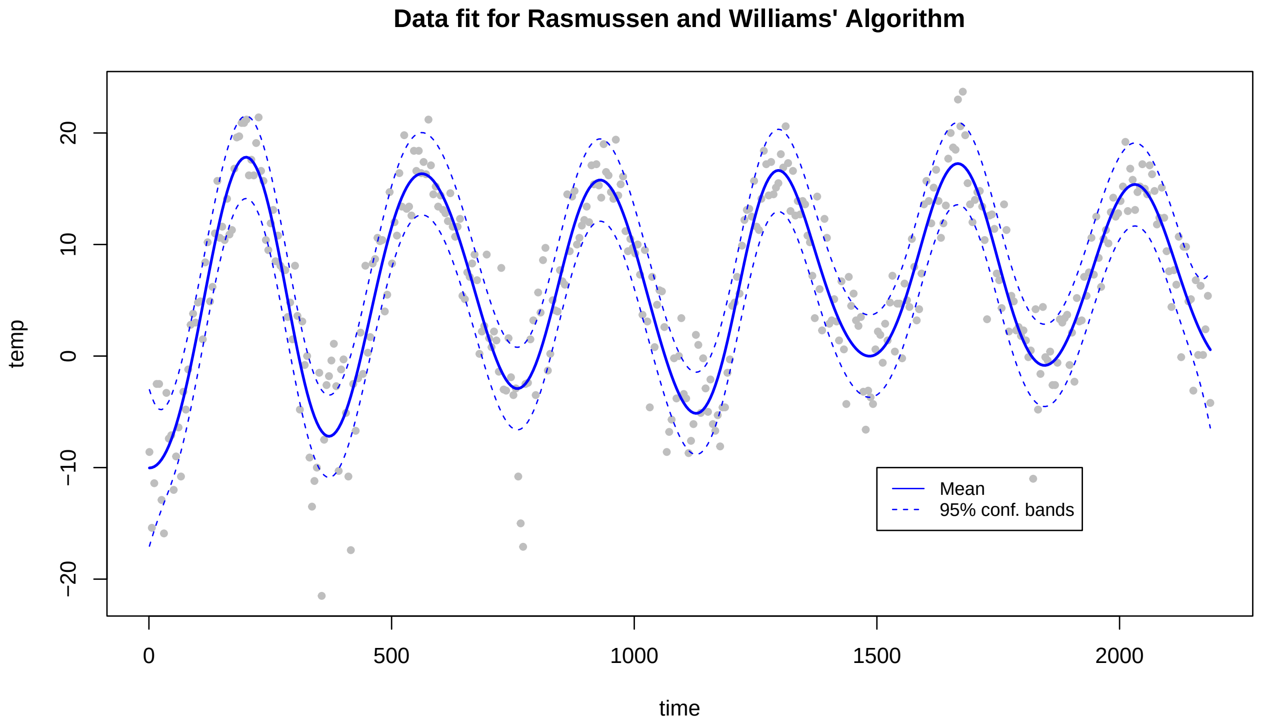


**2.3)**

Repeat the previous exercise, but now use Algorithm 2.1 on page 19 of Rasmussen and Willams' book to compute the posterior mean and variance of $f$ .

```r
quadFit <- lm(temp ~  time + I(time^2))
sigmaNoise = sd(quadFit$residuals)

# Fit the GP with built-in square exponential kernel (called rbfdot in kernlab).
#k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
#GPfit <- gausspr(time,temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE
Xstar = time
resZ = posteriorGP(X = time, y = temp, Xstar, sigmaNoise = sigmaNoise, sigmaF = 20, l=100)
meanPredZ <- resZ$mean
varPredZ <- resZ$var

plot(time, temp, pch=20, col ="grey",main = "Data fit for Rasmussen and Williams' Algorithm")
lines(time, meanPredZ, col="blue", lwd = 2)
lines(time, meanPredZ+1.96*sqrt(varPredZ),col="blue", lty = 2)
lines(time, meanPredZ-1.96*sqrt(varPredZ),col="blue", lty = 2)
```

```r
legend(1500, -10, legend=c("Mean", "95% conf. bands"),
       col=c("blue", "blue"), lty=c(1,2), cex=0.8)
```



**2.4)**

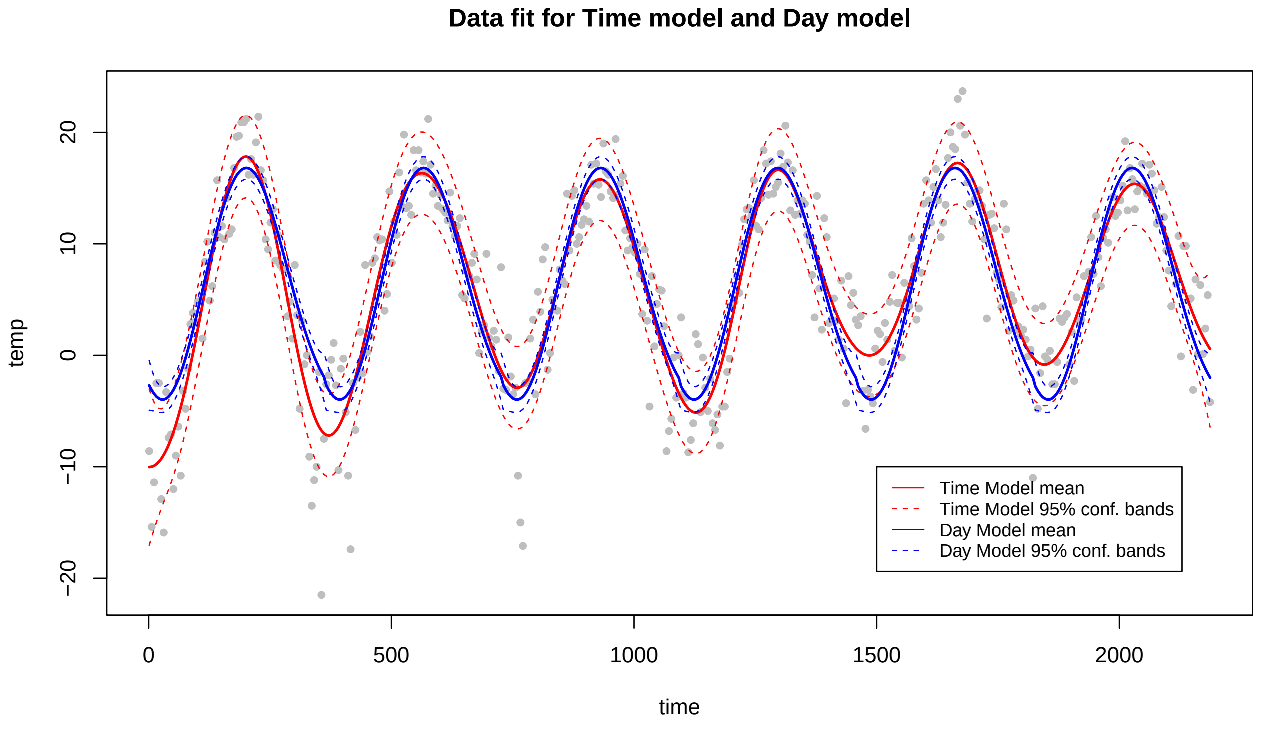
Consider now the following model:

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

Estimate the model using the `gausspr` function with the squared exponential function from (1) with $\sigma_f = 20$ and $l = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these $\sigma_f$ and $l$ values are not suitable). 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.

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


# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
k_20_100 <- SEkernel(sigmaf = 20, ell = 100)
GPfit2 <- gausspr(day, temp, kernel = k_20_100 , var = sigmaNoise^2, variance.model = TRUE,scaled=FALSE
meanPred2 <- predict(GPfit2, day)
varPred2 <- predict(GPfit2, day, type="sdeviation")

plot(time,temp, pch=20, col="grey", main="Data fit for Time model and Day model")
lines(time, meanPred1, col="red", lwd = 2)
lines(time, meanPred1+1.96*stdPred1,col="red", lwd = 1, lty = 2)
lines(time, meanPred1-1.96*stdPred1,col="red", lwd = 1, lty = 2)

lines(time, meanPred2, col="blue", lwd = 2)
lines(time, meanPred2+1.96*varPred2,col="blue", lwd = 1, lty = 2)
lines(time, meanPred2-1.96*varPred2,col="blue", lwd = 1, lty = 2)
legend(1500, -10, legend=c("Time Model mean", "Time Model 95% conf. bands",
                           "Day Model mean", "Day Model 95% conf. bands"),
       col=c("red", "red", "blue", "blue"), lty=c(1,2,1,2), cex=0.8)
```



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

For model based on day of the year, the pro is that it is good at ignoring outliers from one year to another since in learns a pattern that is the same from year to year. But that could also be a con since that pattern HAS to be the same between different years. For the model based on time since start, the pro is that it is able to change pattern from one year to another, but a con it that it is more susceptible to outliers.

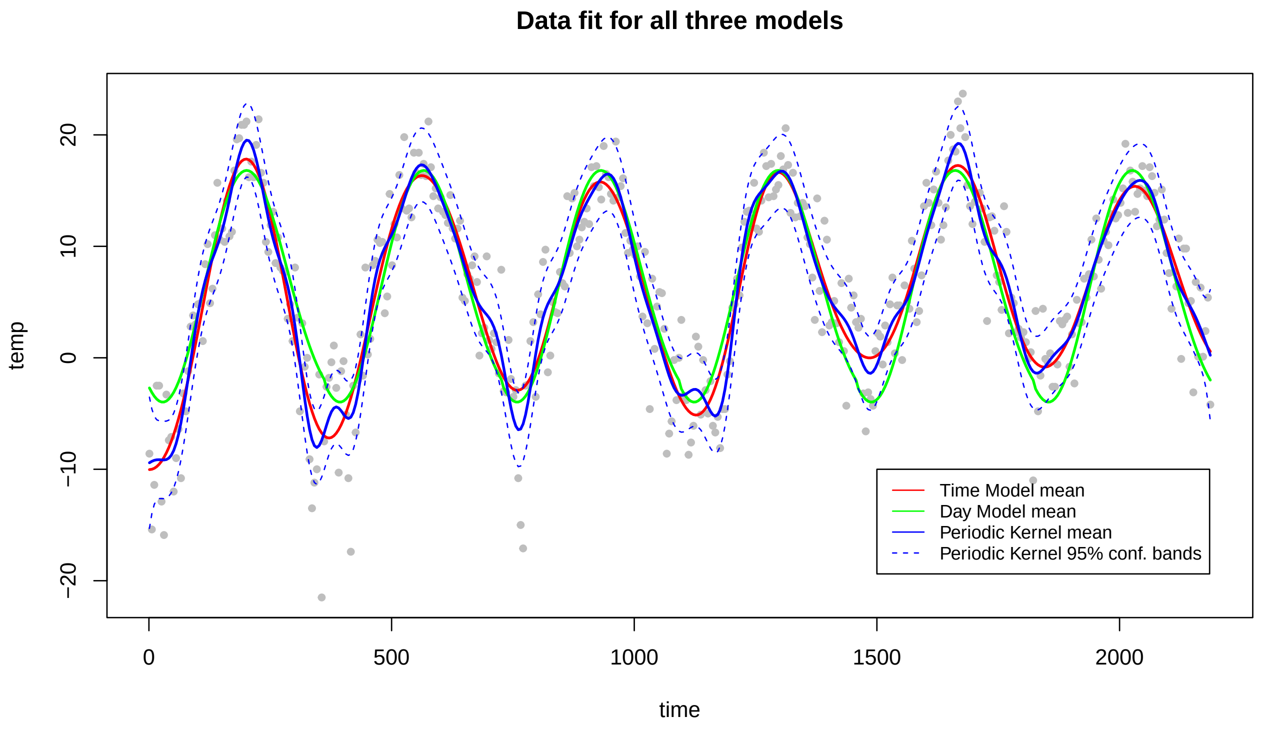
**2.5)**

Finally, implement the following extension of the squared exponential kernel with a periodic kernel (a.k.a. locally periodic kernel):

$$k(x,x') = \sigma_f^2 \exp\{-\frac{2\sin^2(\pi|x-x'|/d)}{l_1^2}\} \exp\{-\frac{1}{2}\frac{|x-x'|^2}{l_2^2}\}$$

Note that we have two different length scales in the kernel. Intuitively, $l_1$ controls the correlation between two days in the same year, 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 $\sigma_f = 20$, $l_1 = 1$, $l_2 = 100$ and $d = 365$. Use the `gausspr` function with the option `scaled=FALSE`, otherwise these $\sigma_f$ , $l_1$ and $l_2$ values are not suitable. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 100$). Discuss the results.

```
SEkernelPeriodic <- function(sigmaf = 20, ell1 = 1, ell2 = 100, d = 365){
  rval <- function(x, y = NULL) {
      r = sqrt(crossprod(x-y));
      return(sigmaf**2 * exp(-(2*sin(pi*abs(x-y)/d)**2)/ell1**2) * exp(-0.5*(abs(x-y)**2)/ell2**2))
    }
  class(rval) <- "kernel"
  return(rval)
}

k_periodic <- SEkernelPeriodic()
GPfit3 <- gausspr(time, temp, kernel = k_periodic , var = sigmaNoise^2, variance.model = TRUE,scaled=FA
meanPred3 <- predict(GPfit3, time)
varPred3 <- predict(GPfit3, time, type="sdeviation")

plot(time,temp, pch=20, col="grey", main = "Data fit for all three models")
lines(time, meanPred1, col="red", lwd = 2)
lines(time, meanPred2, col="green", lwd = 2)
lines(time, meanPred3, col="blue", lwd = 2)
lines(time, meanPred3+1.96*varPred3,col="blue", lwd = 1, lty = 2)
lines(time, meanPred3-1.96*varPred3,col="blue", lwd = 1, lty = 2)
legend(1500, -10, legend=c("Time Model mean", "Day Model mean",
                           "Periodic Kernel mean", "Periodic Kernel 95% conf. bands"),
       col=c("red", "green", "blue", "blue"), lty=c(1,1,1,2), cex=0.8)
```



Discussion of the results:

The locally periodic kernel seems to have the "best of both worlds" compared to the two previous kernels as it takes into account both the time since start and day of the year.

## 3. GP Classification with kernlab.

Download the banknote fraud data:

```r
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])
```

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):

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

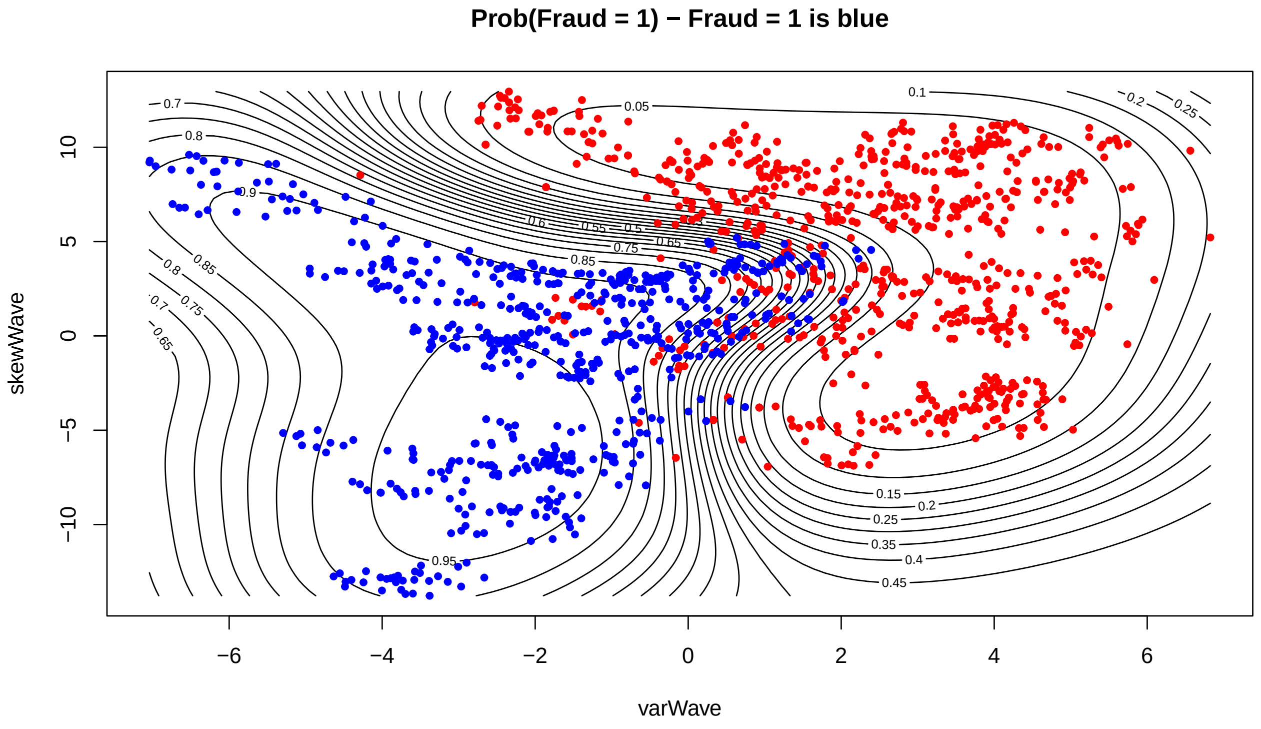
### 3.1)

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.

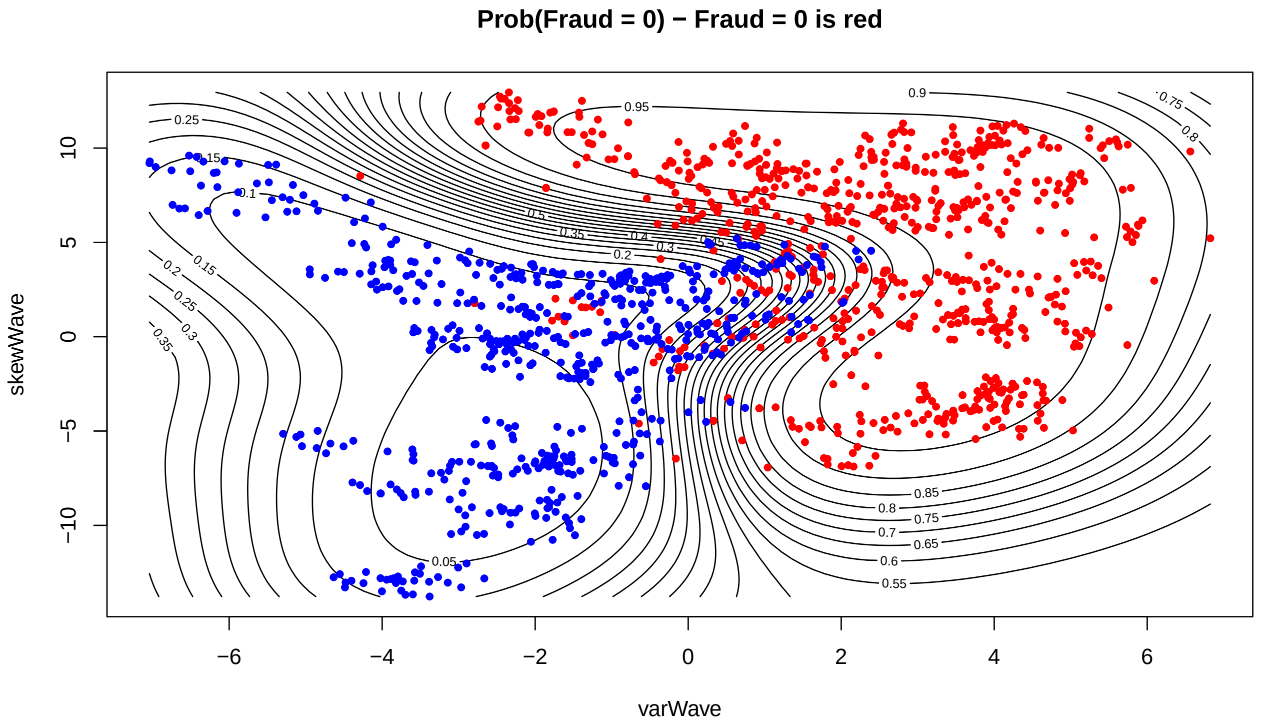
```r
#Fitting on train dataset
GPfitfraud <- gausspr(fraud ~  varWave + skewWave , data[SelectTraining,])
```

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

```r
# class probabilities
probPreds <- predict(GPfitfraud, data[SelectTraining,1:2], type="probabilities")

# Obtaining predictions by getting the factor (class) with the highest probability
predsTrain = as.factor(colnames(probPreds)[apply(probPreds,1,which.max)])

#Contour plot stuff
x1 <- seq(min(data[,1]),max(data[,1]),length=100)
x2 <- seq(min(data[,2]),max(data[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

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

# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 1) - Fraud = 1 is blue')
points(data[data[,5]==0,1],data[data[,5]=='0',2],col="red", pch=20) # Not fraud
points(data[data[,5]==1,1],data[data[,5]=='1',2],col="blue", pch=20) #Fraud
```



```
# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud = 0) - Fraud = 0 is red')
points(data[data[,5]==0,1],data[data[,5]=='0',2],col="red", pch=20) # Not fraud
points(data[data[,5]==1,1],data[data[,5]=='1',2],col="blue", pch=20) #Fraud
```



```r
# Accuracy and Confusion matrix:
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```r
confMat = table(True = data[SelectTraining,5], Predicted = predsTrain)
print(confMat)
```

```
##     Predicted
## True   0   1
##    0 503  41
##    1  18 438
```

```r
print("Accuracy:")
```

```
## [1] "Accuracy:"
```

```r
print(sum(diag(confMat))/sum(confMat))
```

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

**3.2)**

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

```r
#Getting the test dataset
idx = 1:dim(data)[1]
test_idx = idx[which(!idx %in% SelectTraining)]

#Predicting on the test dataset using learned weights from train dataset
probPredstest <- predict(GPfitfraud, data[test_idx,1:2], type="probabilities")

# Obtaining predictions by getting the factor (class) with the highest probability
predsTest = as.factor(colnames(probPredstest)[apply(probPredstest,1,which.max)])

#Confusion Matrix and Accuracy
cm = table(True = data[test_idx,5], Prediction = predsTest)
print("Confusion Matrix")
```

```
## [1] "Confusion Matrix"
```

```r
print(cm)
```

```
##     Prediction
## True   0   1
##    0 199  19
##    1   9 145
```

```
print("Accuracy")
```

```
## [1] "Accuracy"
```

```
print(sum(diag(cm))/sum(cm))
```

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

**3.3)**

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

```
#Fitting model using all covariates on training data
GPfitfraud <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data=data[SelectTraining,])
```

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

```
#Predicting on the test dataset using learned weights from train dataset
probPredstest <- predict(GPfitfraud, data[test_idx,1:4], type="probabilities")

# Obtaining predictions by getting the factor (class) with the highest probability
predsTest = as.factor(colnames(probPredstest)[apply(probPredstest,1,which.max)])

#Confusion Matrix and Accuracy
cm = table(True = data[test_idx,5], Prediction = predsTest)
print("Confusion Matrix")
```

```
## [1] "Confusion Matrix"
```

```
print(cm)
```

```
##     Prediction
## True   0   1
##    0 216   2
##    1   0 154
```

```
print("Accuracy")
```

```
## [1] "Accuracy"
```

```
print(sum(diag(cm))/sum(cm))
```

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

Comparaison of the two models: While the model with only two covariates was already really good (92.5% accuracy on test data), the model with four covariates is even better with 99.5% accuracy. Only 2 test points were misclassified. One could argue that the accuracy gained from adding to two covariates may not be worth the additional computational cost of fitting the model with the two extra covariates, since it was already really good without. (Had to run `ulimit -s hard` in terminal to compute the last question)