LAB 4: GAUSSIAN PROCESSES

Phillip Hölscher

2019-10-16

Contents

```r
# libraries
library(kernlab)
library(AtmRay)
library(ggplot2)
```

Assaginment 1 - Implementing GP Regression

Exercise 1.1

Write your own code for the Gaussian process regression model:

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

The implementation of the Gaussian process, use following algorithm:

2.3 Varying the Hyperparameters

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

2: $L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$ } predictive mean eq. (2.25)

4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$

$\mathbf{v} := L \backslash \mathbf{k}_*$ } predictive variance eq. (2.26)

6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$ eq. (2.30)

8: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

Algorithm 2.1: Predictions and log marginal likelihood for Gaussian process regression. The implementation addresses the matrix inversion required by eq. (2.25) and (2.26) using Cholesky factorization, see section A.4. For multiple test cases lines 4-6 are repeated. The log determinant required in eq. (2.30) is computed from the Cholesky factor (for large n it may not be possible to represent the determinant itself). The computational complexity is $n^3/6$ for the Cholesky decomposition in line 2, and $n^2/2$ for solving triangular systems in line 3 and (for each test case) in line 5.

```r
# seperate kernel function
####### use the SquaredExpKernel form the lecture
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF,l){
  # input:
  # sigmaF  - influences the variance of the function (how variate is the function)
  # it influences also the 95% CI
  # l - influences the smoothnes
  # both have influance to the smoothnes


  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){
  # Function input
  # X: Vector of training inputs.
  # y: Vector of training targets/outputs
  # XStar: Vector of inputs where the posterior distribution is evaluated ie X_*
  # hyperparm: vector with two elements, with sigma_f (sigmaF) & l (ell)
  # sigmaNois: Noise standard deviation sigma_f (sigmaF)

  # before we can start with the step 2, where we compute L,
  # we need to calculate K, the kernel
  # we are suppost to use the squared exponentional kernel


  # compute the covariance matrix K or K(X,X)
  # late on we also need the covariance for k(x,x_*) & K(x_*,x_*)
  # K = the covariance which represents the distances between
  # all combination of points in the hyperspace
  K = SquaredExpKernel(x1 = X, x2 = X, sigmaF = hyperParam[1],
                       l = hyperParam[2])


  # next step compute L
  # we compute Cholesky decomposition + K
  n = length(X)

  ## 2: L
  L_trans = chol(K + (sigmaNoise^2 * diag(n)))
  #L_trans = chol(K + diag(sigmaNoise^2, nrow = length(X), ncol = length(X)))
  #L_trans = chol(K + (sigmaNoise^2)*diag(n))
  # L in the algorithm is a lower triangular matrix,
  # whereas the R function returns an upper triangular matrix.
  L = t(L_trans)

  ###### predictive mean
  # A\b means the vector x that solves the equation Ax = b - use function solve
  # alpha = L^T \ (L\y)
  alpha = solve(L_trans, solve(L,y))

  ## 4: f_bar_star
  # compute k^T_* to compute f_bar_star
  k_X_Xstart = SquaredExpKernel(x1 = X, x2 = XStar,
                                sigmaF = hyperParam[1],
                                l = hyperParam[2])
  f_bar_star = t(k_X_Xstart) %*% alpha


  ###### predictive variance
```

```
  # compute v
  v = solve(L, k_X_Xstart)

  ## 6: V[f_*]

  # compute k(x_*,x_*) to compute V
  k_Xstar_Xstar = SquaredExpKernel(x1 = XStar, x2 = XStar,
                                   sigmaF = hyperParam[1],
                                   l = hyperParam[2])

  # compute V[f_*]
  V_f_star = k_Xstar_Xstar - t(v) %*% v

  # we are only interested in variance of f,
  # therefore extract, take diagonal of covariance matrix of f
  V_f_star = diag(V_f_star)

  # note:
  # we do not need to compute the log marginal likelihood

  #save the pred mean and pred varaince
  result = list("predictive_mean" = f_bar_star,
                "predictive_var" = V_f_star)

}
```

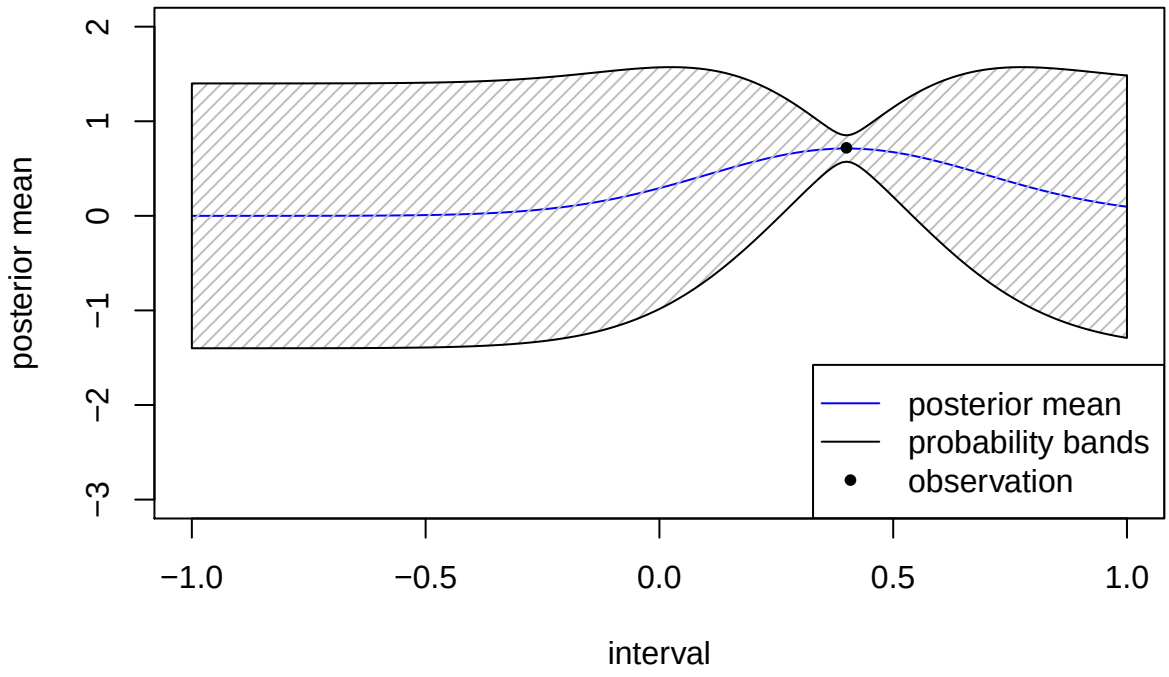
Exercise 1.2

After the function has been implemented, it should now be used. The hyperparameters should be set to $\sigma_f = 1$ and $l = 0.3$. The observations are given with $x = 0.4$ and $y = 0.719$. Furthermore it is assumed that nois with $\sigma_n = 0.1$. A visualization of the results follows.

The code to compute the posterior:

```
# init given values
sigmaF = 1
ell = 0.3
#obs_x_y = c(0.4,0.719)
obs_x_y = data.frame(x = 0.4, y = 0.719)
sigmanois = 0.1
xgrid = seq(from = -1, to = 1 ,by = 0.01)

# compute the posterior
posterior_E1.2 = posteriorGP(X = obs_x_y[1,1], y = obs_x_y[1,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)
```

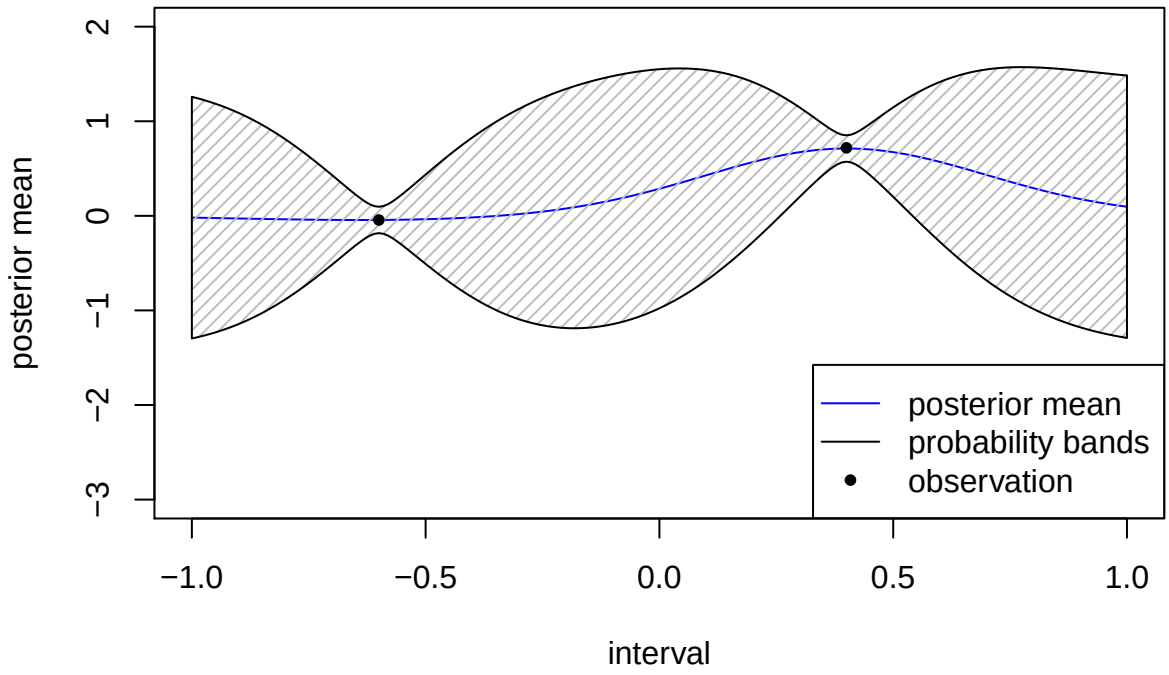
The plot of the posterior mean and 95% probability bands



Exercise 1.3

Update the prior information with the observation (x,y) = (-0.6,-0.044). For this the given observations in the data frame are added and the prior recalculated. The new plot can be viewed under this text.

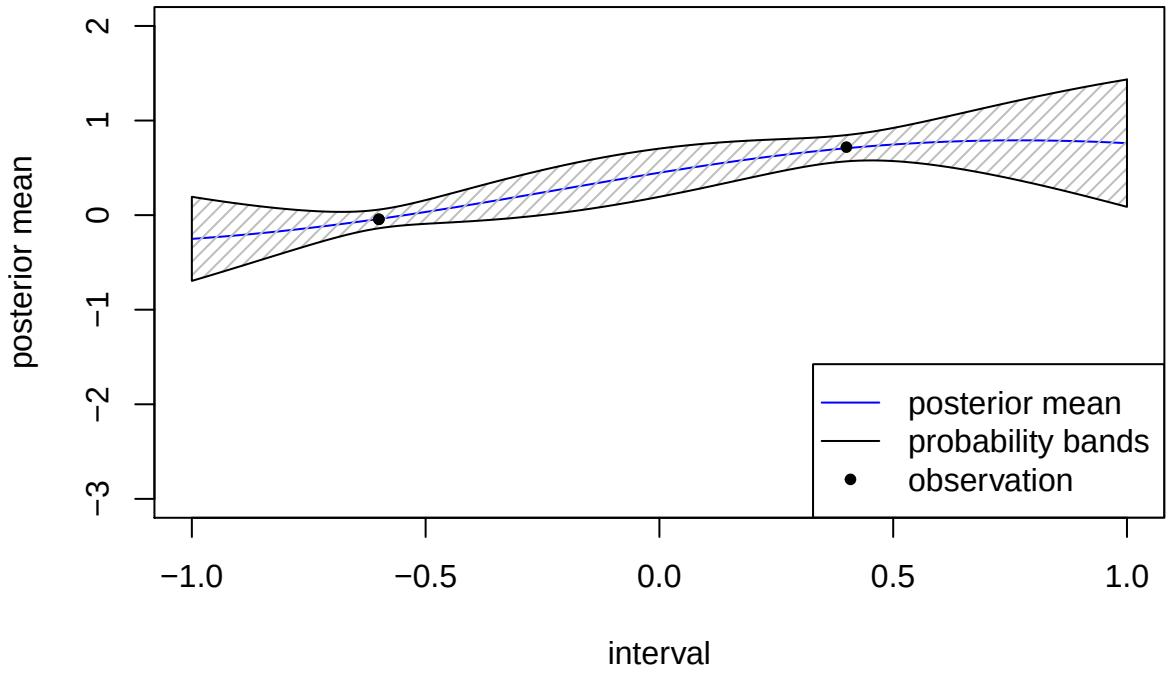
```
# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)

# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)
```



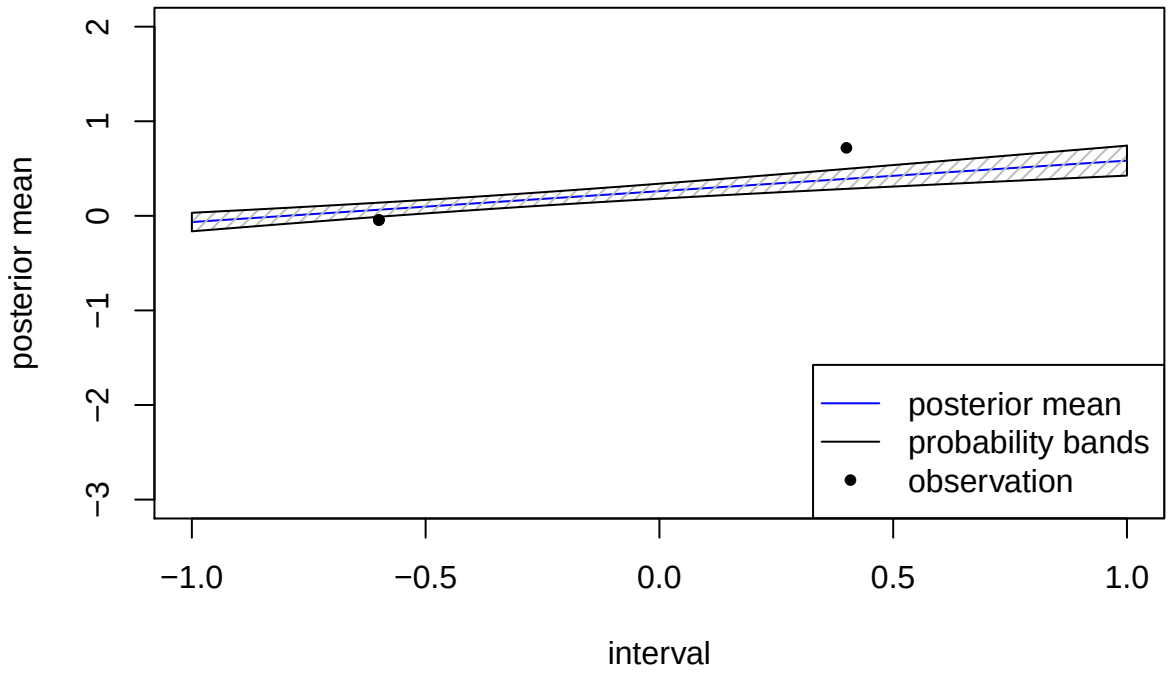
In the lab, we weren't told we had to create the following plot. But out of interest I changed the parameter l to see what the plot looks like.

In this part I change the hyperparameter l and let sigma equal to 1.

- l = 1

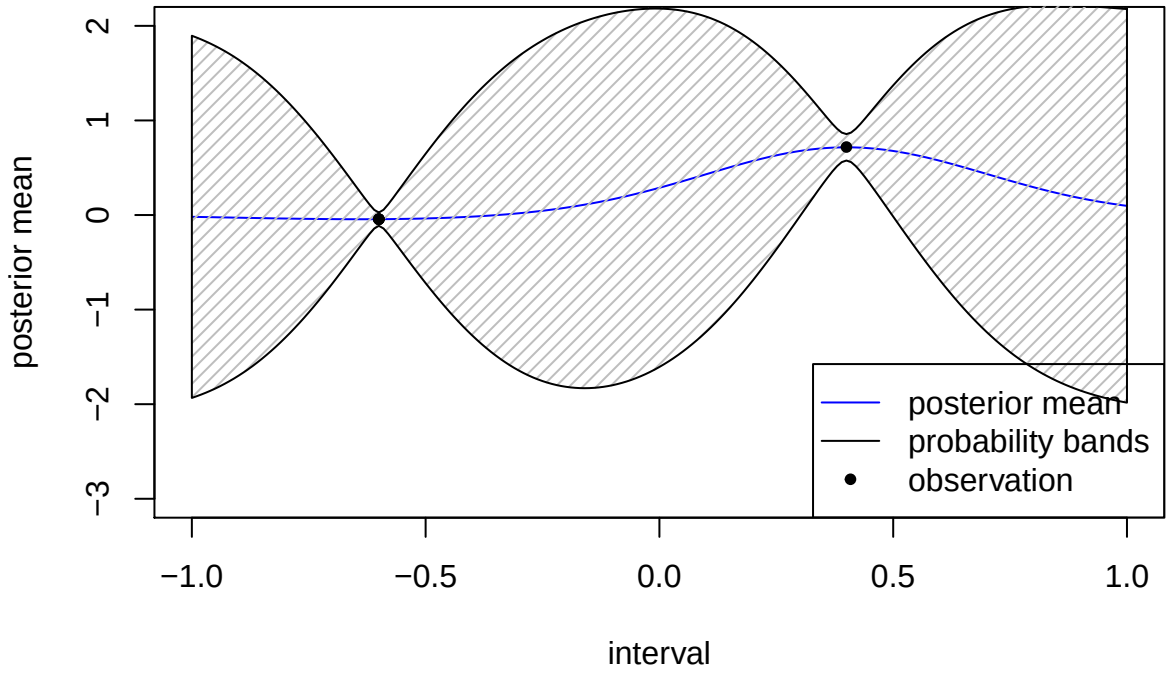


- l = 10

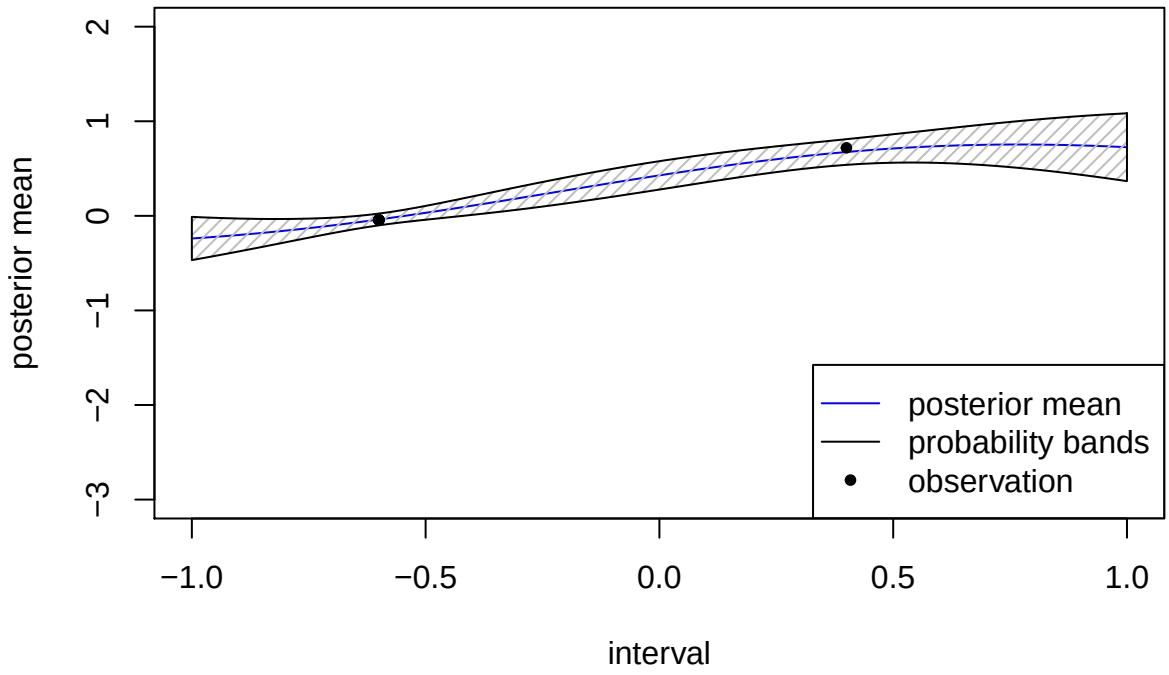


In this part I change the hyperparameter simga and let l equal to 0.3.

- $\sigma_f = 1.5$



- $\sigma_f = 0.5$



- Interpretation of the plots:

In the previous part I changed the parameters to see what effects this has on the plot. Remember, we are using a squared exponential kernel from the lecture.

$k(x, x') = cov(f(x), f(x')) = \sigma_f^2 exp(-\frac{||x-x'||^2}{2l^2})$

Where the l parameter affects the smoothnes of the function and σ_f^2 does change the variance of the function.

At the plot with $\sigma_f^2 = 1$ and $l = 1$ we can see that the function runs smoother. And because l is now larger, the term of σ_f^2 has more weight.
The following plot $\sigma_f^2 = 1$ and $l = 10$ shows that the function was *overfitted* and therefore no longer goes through the given observations. However, if we change the parameter values of σ_f^2, we can see that this has a direct effect on the accuracy of the prediction, as I mentioned before.

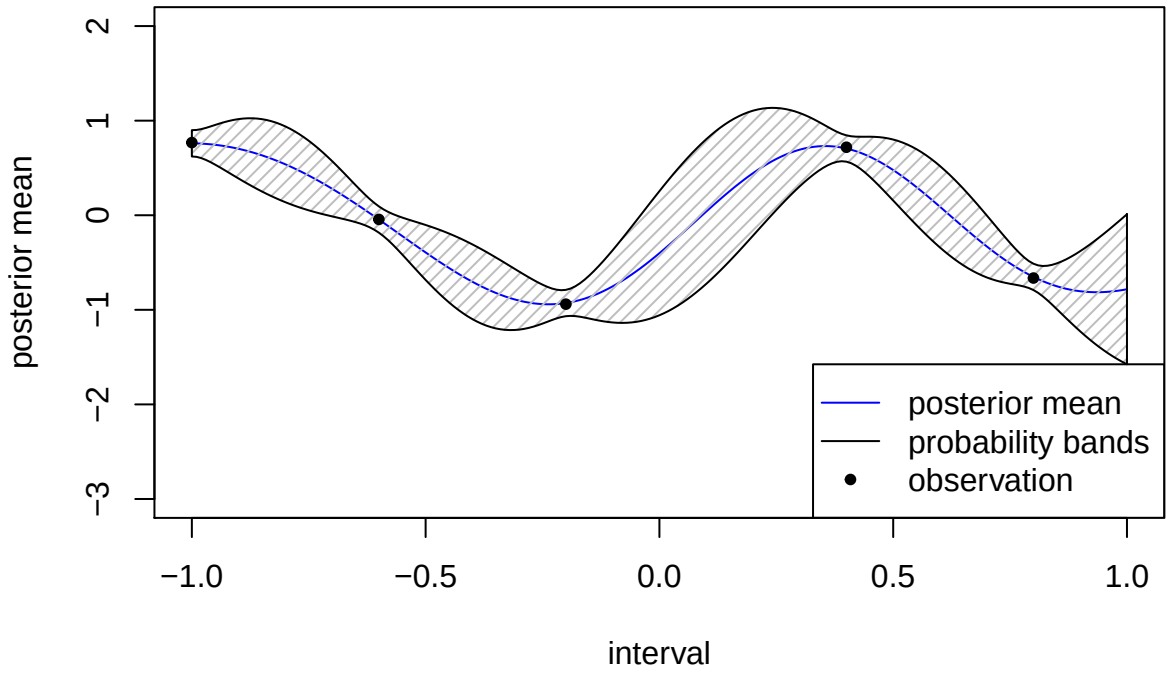
Exercise 1.4

In this task part several observations are given. After these were created in the form of a data frame, the posterior was recalculated. The plot can be viewed under this text.

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

```
# init given data frame
obs_x_y_E1.4 = data.frame(x = c(-1, -0.6, -0.2, 0.4, 0.8),
                          y = c(0.768, -0.044, -0.940, 0.719, -0.664))


# compute the posterior
posterior_E1.4 = posteriorGP(X = obs_x_y_E1.4[,1], y = obs_x_y_E1.4[,2],
                             XStar = xgrid,
```

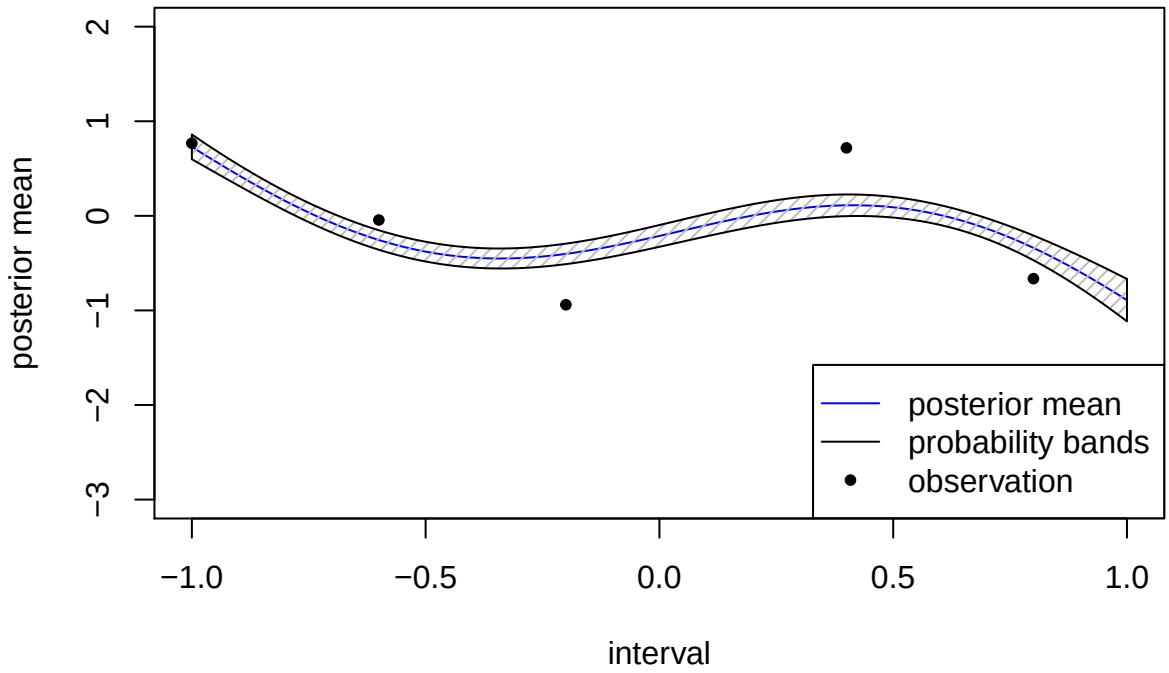
```
                                hyperParam = c(sigmaF, ell),
                                sigmaNoise = sigmanois)
```



Exercise 1.5

In the last task we shall repeat the task part 1.4, but with different hyperparameters values. σ_f^2 and l should both be set to 1.

```
# init new hyperparameter
sigmaF = ell = 1

# compute the posterior
posterior_E1.5 = posteriorGP(X = obs_x_y_E1.4[,1], y = obs_x_y_E1.4[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)
```



In this part of the task similar changes as I did in 1.3 have been made. The parameter l was raised from 0.3 to 1. As mentioned before, the parameter l influences the smoothnes of the function, as the correlation increases due to the change in relation between covariances and variances, becoming more equal. Thus the function becomes smoother and no longer runs through the observations. The confidence interval also runs smoother and no longer catches the 95% of all observations.

Assaginment 2 - GP Regression with kernlab

Data preparation

Before we start work on this part of the lab, do we have to get the data, for this is a GitHub link provided. Furthermore, do we need some data preparation, otherways the gaussian process demands a lot of time.

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

# create vector time (nr of days) & day (of the year 1:365)
time = 1:nrow(data)
day = rep(1:365, 6)

# GP could take lot of computation time
# subsample the data, take every 5s element
time_5s = seq(from = time[1], to = time[length(time)], by = 5)
day_5s = rep(seq(from = 1, to = 365, by = 5), times = 6)
# we need to extracted temp data for the linear regression
temp = data$temp
# case time - Exercise 2.2
temp_5s = temp[time_5s]
# case time - Exercise 2.4
temp_5s_day = temp[day_5s]
```

Exercise 2.1

Now we have to implement a square exponential kernel function, with the parameter l and σ_f. This function should be evaluated with $x = 1, x' = 2$. With a function *kernelMatrix* are suppost to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

```
# init given values
x = 1
x_prime = 2
X = c(1,3,4)
X_prime = c(2,3,4)

# use lecture function
# Squared Exponential Kernel function
SEkernel = function(sigmaF = 1, ell = 1)
{
  SquaredExpKernel <- function(x, y = NULL) {
    n1 = length(x)
    n2 = length(y)
    K = matrix(NA, n1, n2)
    for(i in 1:n2) {
      for(j in 1:n1) {
        K[j,i] = sigmaF^2 * exp(-0.5 * ( (x[i] - y[j]) / ell)^2 )
      }
    }
    return(K)
  }
```

```
  class(SquaredExpKernel) <- "kernel"
  return(SquaredExpKernel)
}
```

Here you can see the Covariance matrix $K(X, X_*)$

```
# sigmaF = ell = 1
k = SEkernel()

## Evaluate kernel
#k(x,x_prime)

# Computing the whole covariance matrix K from the kernel
K = kernelMatrix(kernel = k, x = X, y = X_prime)
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
```

Exercise 2.2

- Fit the quadratic regression

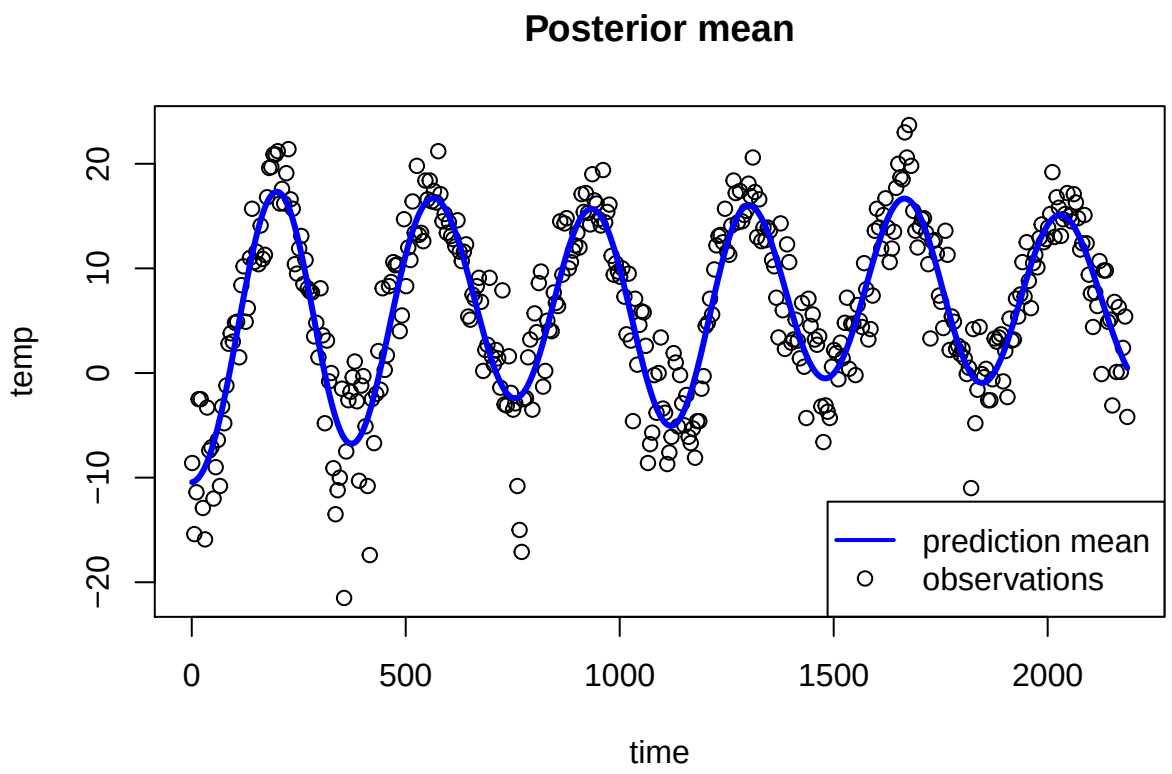
Given model $temp = f(time) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ and $f \sim \mathcal{GP}(0, k(time, time'))$.

```
# fit quadratic regression
qr_fit = lm(temp_5s ~ time_5s + I(time_5s)^2)

# compute the residual vairance
sigmaNoise = sd(qr_fit$residuals)
```

Esimate Gaussian proces regression model & compute the postetior mean. The results will then be visualized in the form of the posterior means. The hyperparameters are given with $\sigma_f = 20$ and $l = 0.2$.

```
#### Estimate Gaussian process regression model using the squared exponential function

# we need to extracted temp data for the linear regression
temp = data$temp
temp_5s = temp[time_5s]

## given values
ell = 0.2
var_f = 20

# fit GP
GPfit = gausspr(x = time_5s,
                y = temp_5s,
                kernel = SEkernel(sigmaF = var_f, ell = ell),
                #kpar = list(sigma = 1/(2*ell^2)),
                var = sigmaNoise^2)
```

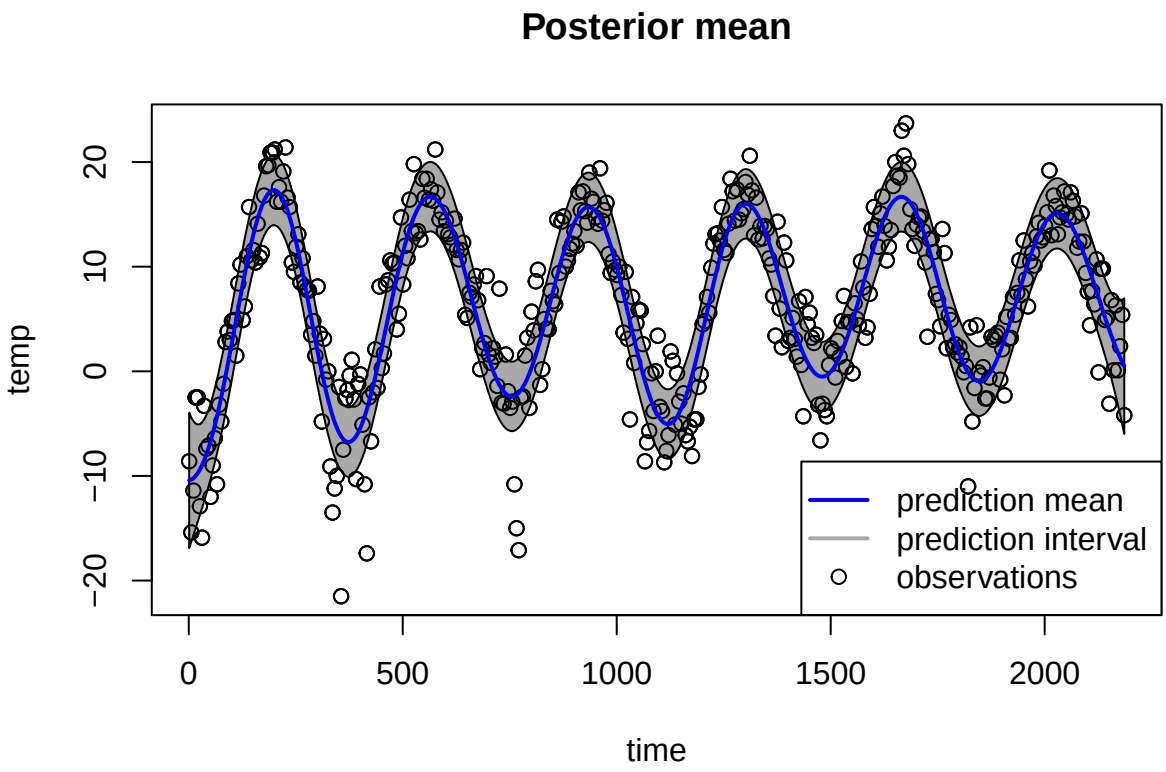
```r
# compute the posterior mean at every data point in the training dataset
meanPred = predict(GPfit, time_5s)
```



Exercise 2.3

Compute the posterior variance of f & the 95% confidence interval and visualize afterwards.

```r
# init given values
sigmaF = 20
ell = 0.2
obs_x_y = data.frame(x = scale(time_5s), y = scale(temp_5s))

# compute the posterior
posterior_E2.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = scale(time_5s),
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmaNoise)
CI_lower = meanPred - 1.96 * sqrt(posterior_E2.3$predictive_var)
CI_upper = meanPred + 1.96 * sqrt(posterior_E2.3$predictive_var)
```

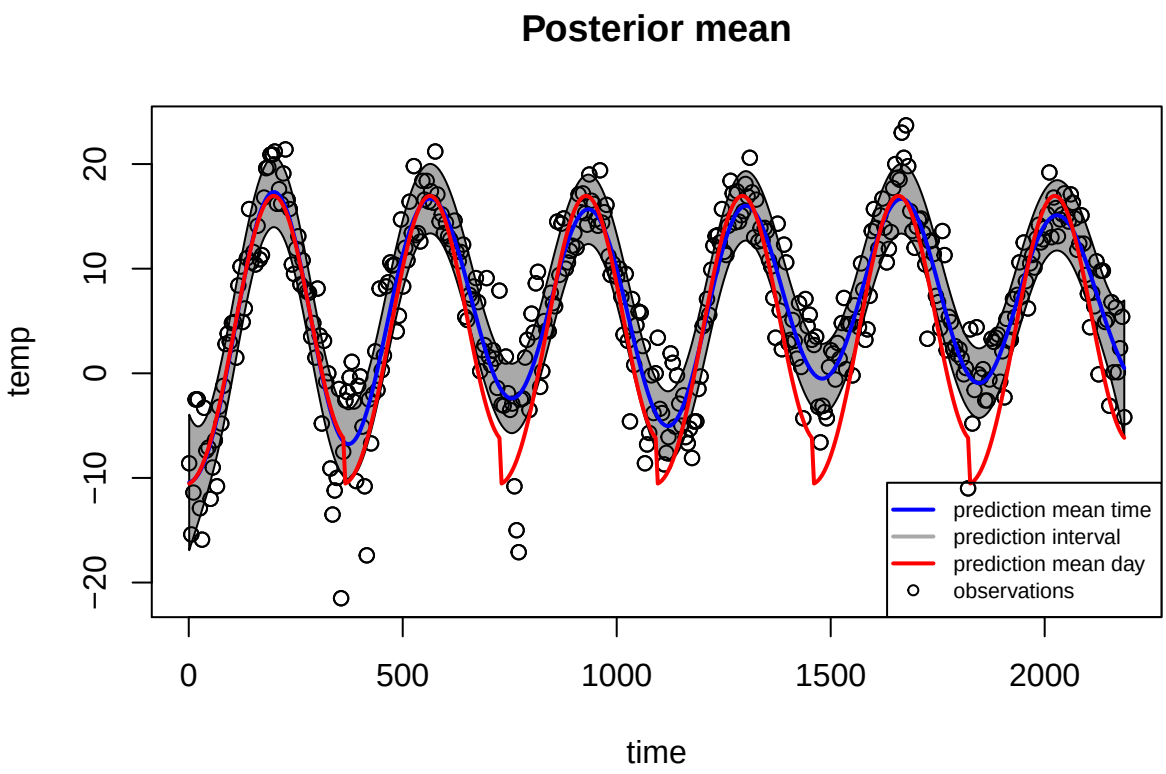


Exercise 2.4

For the following task we shall implement a new model. This is almost the same as the previous model, but in this case the variable *time* was replaced with the variable *day*.

Given model $temp = f(day) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_f^2)$ and $f \sim \mathcal{GP}(0, k(day, day'))$.

```
# fit quadratic regression
qr_fit_day = lm(temp_5s_day ~ day_5s + I(day_5s)^2)

# given values
ell = 0.2
var_f = 20
```



Exercise 2.5

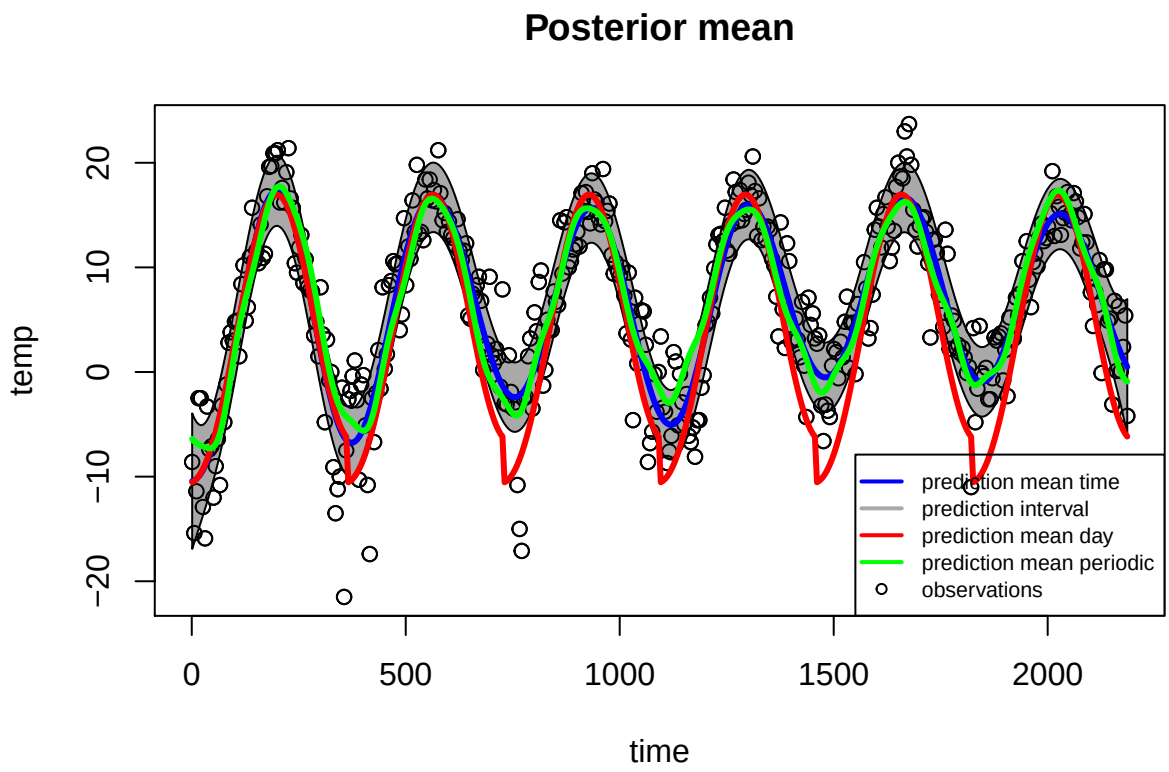
In the final task, we're supposed to to implement a generalization of the periodic kernel:

$$k(x, x') = \sigma_f^2 exp\left(- \frac{2sin^2(\pi|x - x'|/d)}{\ell_1^2} \right) exp\left(- \frac{1}{2} \frac{|x - x'|^2}{\ell_2^2} \right)$$

The hyperparameters should be initialized with the values $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = \frac{350}{sd(time)}$.

```
# init given values
sigmaF = 20
ell1 = 1
ell2 = 10
d = 350/sd(time)

# implement generalization of periodic kernel

# periodic kernel

periodic_kernel = function(sigmaF, l1, l2, d) {
  P_Kernel = function(x1,x2) {
    result = (sigmaF^2)*
      exp(-2*((sin(pi*abs(x1-x2)/d)^2)/(l1^2)))*
      exp(-0.5*((x1 -x2)^2)/(l2^2))
    return(result)
  }
  class(P_Kernel) = "kernel"
```

```
  return(P_Kernel)
}
```



The course of all functions is more or less similar. With *prediction mean day* it can be seen that the course repeats itself again and again. This is because we follow a time interval from 1 to 365. The course *prediction mean periodic kernel* represents the observations best but is slightly less smoother than *prediction mean time*.

Assaginment 3 - GP Classification with kernlab

Data preparation

Before we can begin with the task, we must also prepare data in these new tasks.

```
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])

# create training sample
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
# create training und test data
train = data[SelectTraining,]
test = data[-SelectTraining,]
```

Exercise 3.1

- Fit Gaussian process classification model

```
######## fit gaussian process
# fraud on training
# kernel & hyperparameter = default
GPfit = gausspr(fraud ~ .,
                data = train)


# only use covariates varWave & sekwWave
GPfit_var_sekw = gausspr(fraud ~ varWave + skewWave,
                         data = train)
```

- Compute predictions & accuracy

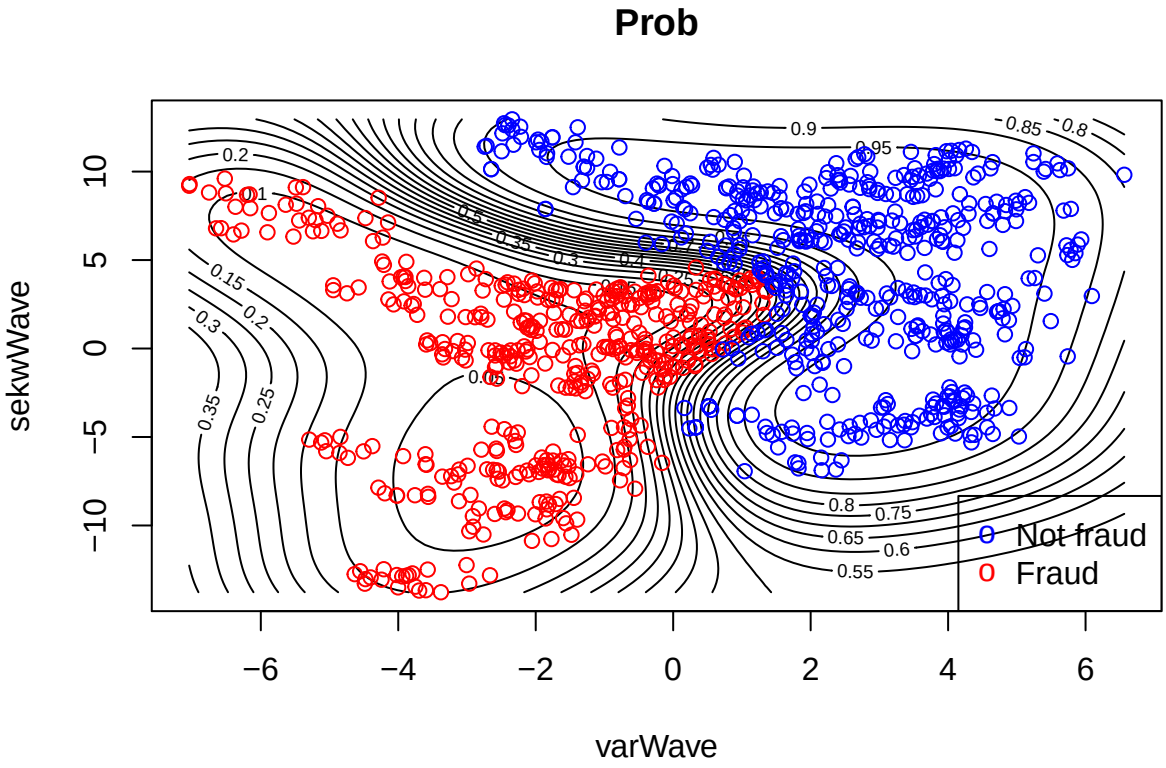
```
###### make predcition on the training & check accuracy
########## GPfit
pred_GPfit = predict(GPfit,
                     train)
cm_PGfit = table(pred_GPfit, train$fraud)
accuracy_GPfit = sum(diag(cm_PGfit))/sum(cm_PGfit)

########## GPfit_var_sekw
pred_GPfit_var_sekw = predict(GPfit_var_sekw,
                              train)
cm_PGfit_var_sekw = table(pred_GPfit_var_sekw, train$fraud)
accuracy_GPfit_var_sekw = sum(diag(cm_PGfit_var_sekw))/sum(cm_PGfit_var_sekw)
```

- Plot contours of the prediction probabilities

** remove this code after the question is solved**

```
# make prediction of class probabilities
probPreds <- predict(GPfit_var_sekw, train, type="probabilities")
```

```r
# init suitable grid of values for varWave and skewWave
# therefore, check min and max values of these two variables
x1 <- seq(min(train$varWave),max(train$varWave),length=100)
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100)

# Creates 2D matrices for accessing images and 2D matrices
gridPoints <- meshgrid(x1, x2)# output of mashgrid:
# list of every of the 100 points repeated 10 times
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y)) #output
# x1 and x2 in just 2 columns with length 10000

# transform to make new predictions
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]

probPreds <- predict(GPfit_var_sekw, gridPoints, type="probabilities")

# Plotting for Prob
contour(x = x1, y = x2,
        z = matrix(probPreds[,1],100,byrow = TRUE),
        nlevel = 20,
        xlab = "varWave", ylab = "sekwWave",
        main = 'Prob')
points(x = train$varWave[pred_GPfit_var_sekw == 1],
       y = train$skewWave[pred_GPfit_var_sekw == 1], col = "red")
points(x = train$varWave[pred_GPfit_var_sekw == 0],
       y = train$skewWave[pred_GPfit_var_sekw == 0], col = "blue")
legend("bottomright",
       pch = c("o","o"),
       legend = c("Not fraud", "Fraud"),
       col = c("blue", "red"))
```



- Confusion Matrix of the classifier

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

- Accuracy of the classifier

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

Exercise 3.2 & 3.3

- Accuracy of the classifier for the test set

```
##                               Train  Test
## accuracy.all.covariates       0.996 0.997
## accuracy.selected.covariates 0.932 0.938
```

From the results we can see that in both cases the accuracy is very high. In the case where all covariates are used, even an accuracy of 99% is achieved. If we use only the two covariates varWave and skewWave, only a little worse accuracy has been achieved, 93%. So it can be said that with all covariates a better accuracy is achieved, but it seems that with varWave and skewWave the most important information has been retained.

Lecture code

Gaussian Process Regression -

```
#install.packages("mvtnorm")
library("mvtnorm")


######## Gaussian Process Regression
### A function f(x) consist of
# gaussian process G(m(x), k(x,x´))
# m(x) is the mean function
# k(x,x´) is the covariance function


# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  # input:
  # sigmaF  - influences the variance of the function (how variate is the function)
  # it influences also the 95% CI
  # l - influences the smoothnes
  # both have influance to the smoothnes


  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)
}

# Mean function
MeanFunc <- function(x){
  m <- sin(x)
  return(m)
}

# Simulates nSim realizations (function) from a GP with mean m(x) and covariance K(x,x')
# over a grid of inputs (x)
SimGP <- function(m = 0,K,x,nSim,...){
  n <- length(x)
  if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x)
  covMat <- K(x,x,...)
  f <- rmvnorm(nSim, mean = meanVector, sigma = covMat)
  return(f)
}

xGrid <- seq(-5,5,length=20) # change the nr of points smapled here

# Plotting one draw
sigmaF <- 1 #### affect of the variance of the function & 95% CI
```

```
l <- 1 ##### affect to the smoothnes
nSim <- 1
fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)
plot(xGrid, fSim[1,], type="p", ylim = c(-3,3))
if(nSim>1){
  for (i in 2:nSim) {
    lines(xGrid, fSim[i,], type="p")
  }
}
lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3)
lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
      col = "blue", lwd = 2)
lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
      col = "blue", lwd = 2)

# Plotting using manipulate package
library(manipulate)

plotGPPrior <- function(sigmaF, l, nSim){
  fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)
  plot(xGrid, fSim[1,], type="l", ylim = c(-3,3), ylab="f(x)", xlab="x")
  if(nSim>1){
    for (i in 2:nSim) {
      lines(xGrid, fSim[i,], type="l")
    }
  }
  lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3)
  lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
        col = "blue", lwd = 2)
  lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
        col = "blue", lwd = 2)
  title(paste('length scale =',l,', sigmaf =',sigmaF))
}

manipulate(
  plotGPPrior(sigmaF, l, nSim = 10),
  sigmaF = slider(0, 2, step=0.1, initial = 1, label = "SigmaF"),
  l = slider(0, 2, step=0.1, initial = 1, label = "Length scale, l")
)
```

Kernlab Demo

```
#########################################################################################################
## Kernlab demo for Gaussian processes regression and classfication
## Author: Mattias Villani, Linkoping University. http://mattiasvillani.com
#########################################################################################################


#############################################################################
###    Prelims: Setting path and installing/loading packages            #####
#############################################################################
setwd('/Users/mv/Dropbox/Teaching/AdvML/GaussianProcess/Code')
```

```r
#install.packages('kernlab')
#install.packages("AtmRay") # To make 2D grid like in Matlab's meshgrid
library(kernlab)
library(AtmRay)


################################################
###    Messin' around with kernels        #####
################################################
# This is just to test how one evaluates a kernel function
# and how one computes the covariance matrix from a kernel function
X <- matrix(rnorm(12), 4, 3)
Xstar <- matrix(rnorm(15), 5, 3)
ell <- 1
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdot
#(which is the SE kernel) in kernlab
SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2
# Computing the whole covariance matrix K from the kernel. Just a test.
K <- kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar)

# Own implementation of Matern with nu = 3/2 (See RW book equation 4.17)
# Note that a call of the form kernelFunc <- Matern32(sigmaf = 1, ell = 0.1) r
#eturns a kernel FUNCTION.
# You can now evaluate the kernel at inputs: kernelFunc(x = 3, y = 4).
# Note also that class(kernelFunc) is of class "kernel", which is a class defined
#by kernlab.
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(rnorm(12), 4, 3) # Simulating some data
Xstar <- matrix(rnorm(15), 5, 3)
MaternFunc = Matern32(sigmaf = 1, ell = 2) # MaternFunc is a kernel FUNCTION
MaternFunc(c(1,1),c(2,2)) # Evaluating the kernel in x=c(1,1), x'=c(2,2)
# Computing the whole covariance matrix K from the kernel.
K <- kernelMatrix(kernel = MaternFunc, x = X, y = Xstar) # So this is K(X,Xstar)




################################################
###      Regression on the LIDAR data       ###
################################################
#lidarData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess
```

```
lidarData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/
                        master/GaussianProcess/Code/LidarData',
                        header = T)
LogRatio <- lidarData$LogRatio
Distance <- lidarData$Distance

# Estimating the noise variance from a third degree polynomial fit
polyFit <- lm(LogRatio ~  Distance + I(Distance^2) + I(Distance^3) )
sigmaNoise = sd(polyFit$residuals)

plot(Distance,LogRatio)

# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = rbfdot, kpar =
                   list(sigma = 1/(2*ell^2)), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance) # Predicting the training data. To plot the fit.
lines(Distance, meanPred, col="blue", lwd = 2)

# Fit the GP with home made Matern
sigmaf <- 1
ell <- 2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1))
# NOTE: this also works and is the same as the next line.
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="purple", lwd = 2)

# Trying another length scale
sigmaf <- 1
ell <- 1
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="green", lwd = 2)

# And now with a different sigmaf
sigmaf <- 0.1
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="black", lwd = 2)









################################################
####       Classification on Iris data      ###
```

```
################################################
data(iris)
GPfitIris <- gausspr(Species ~  Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                     data=iris)
GPfitIris

# predict on the training set
predict(GPfitIris,iris[,1:4])
table(predict(GPfitIris,iris[,1:4]), iris[,5]) # confusion matrix

# Now using only Sepal.Length and Sepal.Width to classify
GPfitIris <- gausspr(Species ~  Sepal.Length + Sepal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris,iris[,1:2])
table(predict(GPfitIris,iris[,1:2]), iris[,5]) # confusion matrix

# Now using only  Petal.Length + Petal.Width to classify
GPfitIris <- gausspr(Species ~ Petal.Length + Petal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris,iris[,3:4])
table(predict(GPfitIris,iris[,3:4]), iris[,5]) # confusion matrix

# class probabilities
probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(iris[,3]),max(iris[,3]),length=100)
x2 <- seq(min(iris[,4]),max(iris[,4]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(iris)[3:4]
probPreds <- predict(GPfitIris, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Setosa) - Setosa is red')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

# Plotting for Prob(Versicolor)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Versicolor) - Versicolor is green')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")


# Plotting for Prob(virginica)
```

```r
contour(x1,x2,matrix(probPreds[,3],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Virginica) - Virginica is blue')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

# Plotting the decision boundaries
meanPred <- matrix(max.col(probPreds),100,byrow = TRUE)
plot(gridPoints,  pch=".", cex=3, col=ifelse(meanPred==1, "red",
                                             ifelse(meanPred==2, "green", "blue")))
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],
       col="red", cex=10, pch=".")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],
       col="blue",  cex=10, pch=".")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],
       col="green",  cex=10, pch=".")

########################################
# Author: Jose M. Pena, jose.m.pena@liu.se
# GP regression on the canadian wages data
########################################

library(kernlab)
#CWData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess/Co
CWData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/
                     master/GaussianProcess/Code/CanadianWages.dat',
                     header = T)
logWage<-CWData$logWage
age<-CWData$age
age<-(age-mean(age))/sd(age) # Standarize the age

# Estimating the noise variance from a third degree polynomial fit.
#I() is needed because, otherwise
# age^2 reduces to age in the formula, i.e. age^2 means adding the main
#effect and the second order
# interaction, which in this case do not exist. See ?I.
polyFit <- lm(logWage ~  age + I(age^2) + I(age^3))
sigmaNoise = sd(polyFit$residuals)
plot(age,logWage)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab)
ell <- 0.5
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note the reparametrization
GPfit <- gausspr(age,logWage, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, age) # Predicting the training data
lines(age, meanPred, col="red", lwd = 2)

# The implementation of kernlab for the probability and prediction intervals seem
#to have a bug: The intervals
# seem to be too wide, e.g. replace 1.96 with 0.1 to see something.
# GPfit <- gausspr(age,logWage, kernel = SEkernel, var = sigmaNoise^2, variance.model = TRUE)
# meanPred <- predict(GPfit, age)
```

```r
# lines(age, meanPred, col="red", lwd = 2)
# lines(age, meanPred+1.96*predict(GPfit,age, type="sdeviation"),col="blue")
# lines(age, meanPred-1.96*predict(GPfit,age, type="sdeviation"),col="blue")

# Probability and prediction interval implementation.
x<-age
xs<-age # XStar
n <- length(x)
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

# Probability intervals for fStar
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)

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

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
set.seed(12345)
rm(list = ls())
# install.packages
#install.packages('kernlab')
#install.packages("AtmRay") # To make 2D grid like in Matlab's meshgrid
#install.packages("ggplot2")

# libraries
library(kernlab)
library(AtmRay)
library(ggplot2)
set.seed(12345)
rm(list = ls())
knitr::include_graphics("Algo2.3.png")

# seperate kernel function
####### use the SquaredExpKernel form the lecture
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF,l){
  # input:
  # sigmaF  - influences the variance of the function (how variate is the function)
  # it influences also the 95% CI
  # l - influences the smoothnes
  # both have influance to the smoothnes


  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){
  # Function input
  # X: Vector of training inputs.
  # y: Vector of training targets/outputs
  # XStar: Vector of inputs where the posterior distribution is evaluated ie X_*
  # hyperparm: vector with two elements, with sigma_f (sigmaF) & l (ell)
  # sigmaNois: Noise standard deviation sigma_f (sigmaF)

  # before we can start with the step 2, where we compute L,
  # we need to calculate K, the kernel
  # we are suppost to use the squared exponentional kernel


  # compute the covariance matrix K or K(X,X)
```

```
# late on we also need the covariance for k(x,x_*) & K(x_*,x_*)
# K = the covariance which represents the distances between
# all combination of points in the hyperspace
K = SquaredExpKernel(x1 = X, x2 = X, sigmaF = hyperParam[1],
                     l = hyperParam[2])


# next step compute L
# we compute Cholesky decomposition + K
n = length(X)

## 2: L
L_trans = chol(K + (sigmaNoise^2 * diag(n)))
#L_trans = chol(K + diag(sigmaNoise^2, nrow = length(X), ncol = length(X)))
#L_trans = chol(K + (sigmaNoise^2)*diag(n))
# L in the algorithm is a lower triangular matrix,
# whereas the R function returns an upper triangular matrix.
L = t(L_trans)

###### predictive mean
# A\b means the vector x that solves the equation Ax = b - use function solve
# alpha = L^T \ (L\y)
alpha = solve(L_trans, solve(L,y))

## 4: f_bar_star
# compute k^T_* to compute f_bar_star
k_X_Xstart = SquaredExpKernel(x1 = X, x2 = XStar,
                              sigmaF = hyperParam[1],
                              l = hyperParam[2])
f_bar_star = t(k_X_Xstart) %*% alpha


###### predictive variance

# compute v
v = solve(L, k_X_Xstart)

## 6: V[f_*]

# compute k(x_*,x_*) to compute V
k_Xstar_Xstar = SquaredExpKernel(x1 = XStar, x2 = XStar,
                                 sigmaF = hyperParam[1],
                                 l = hyperParam[2])

# compute V[f_*]
V_f_star = k_Xstar_Xstar - t(v) %*% v

# we are only interested in variance of f,
# therefore extract, take diagonal of covariance matrix of f
V_f_star = diag(V_f_star)

# note:
# we do not need to compute the log marginal likelihood
```

```
  #save the pred mean and pred varaince
  result = list("predictive_mean" = f_bar_star,
                "predictive_var" = V_f_star)

}

# init given values
sigmaF = 1
ell = 0.3
#obs_x_y = c(0.4,0.719)
obs_x_y = data.frame(x = 0.4, y = 0.719)
sigmanois = 0.1
xgrid = seq(from = -1, to = 1 ,by = 0.01)

# compute the posterior
posterior_E1.2 = posteriorGP(X = obs_x_y[1,1], y = obs_x_y[1,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)


# create a plot function

plot_function = function(mean,var, xgrid, obs_x_y){

  # compute the confidence bands
  CI_lower = mean - sqrt(1.96 * var)
  CI_upper = mean + sqrt(1.96 * var)


  plot(x = xgrid, y = mean, ylim = c(-3,2),
       xlab = "interval", ylab = "posterior mean",
       col = "blue", type = "l")
  polygon(x = c(rev(xgrid), xgrid),
          y = c(rev(CI_upper), CI_lower),
          col = "grey",
          border = "black",
          density = c(20))
  points(x = obs_x_y[,1], y = obs_x_y[,2], pch = 20)
  legend("bottomright",
         legend = c("posterior mean", "probability bands", "observation"),
         col = c("blue", "black", "black"),
         lty = c(1,1,NA),
         pch = c(NA,NA,20))

}

plot_function(posterior_E1.2$predictive_mean, posterior_E1.2$predictive_var,
              xgrid, obs_x_y)

# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)
```

```
# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)

plot_function(posterior_E1.3$predictive_mean, posterior_E1.3$predictive_var,
              xgrid, obs_x_y)
# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)

# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, 1),
                             sigmaNoise = sigmanois)
plot_function(posterior_E1.3$predictive_mean, posterior_E1.3$predictive_var,
              xgrid, obs_x_y)

# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)

# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, 10),
                             sigmaNoise = sigmanois)
plot_function(posterior_E1.3$predictive_mean, posterior_E1.3$predictive_var,
              xgrid, obs_x_y)
# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)

# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(1.5, ell),
                             sigmaNoise = sigmanois)
plot_function(posterior_E1.3$predictive_mean, posterior_E1.3$predictive_var,
              xgrid, obs_x_y)

# update observation
obs_E1.3 = c(-0.6,-0.044)
obs_x_y = rbind(obs_x_y, obs_E1.3)

# compute the posterior
posterior_E1.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = xgrid,
                             hyperParam = c(0.5, 1),
                             sigmaNoise = sigmanois)
plot_function(posterior_E1.3$predictive_mean, posterior_E1.3$predictive_var,
```

```
              xgrid, obs_x_y)
#### just some test
x = 1
x_star = 3

l_high = 1
l_low = 0.3

(x - x_star)^2

(x - x_star)^2/l_high
(x - x_star)^2/l_low

exp((x - x_star)^2/l_high)
exp((x - x_star)^2/l_low)

exp(-100)

knitr::include_graphics("dataframe_E14.png")
# init given data frame
obs_x_y_E1.4 = data.frame(x = c(-1, -0.6, -0.2, 0.4, 0.8),
                          y = c(0.768, -0.044, -0.940, 0.719, -0.664))


# compute the posterior
posterior_E1.4 = posteriorGP(X = obs_x_y_E1.4[,1], y = obs_x_y_E1.4[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)

plot_function(posterior_E1.4$predictive_mean, posterior_E1.4$predictive_var,
              xgrid, obs_x_y_E1.4)
# init new hyperparameter
sigmaF = ell = 1

# compute the posterior
posterior_E1.5 = posteriorGP(X = obs_x_y_E1.4[,1], y = obs_x_y_E1.4[,2],
                             XStar = xgrid,
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmanois)

plot_function(posterior_E1.5$predictive_mean, posterior_E1.5$predictive_var,
              xgrid, obs_x_y_E1.4)
set.seed(12345)
rm(list = ls())
# get the data
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/
                GaussianProcess/Code/TempTullinge.csv", header=TRUE, sep=";")

# create vector time (nr of days) & day (of the year 1:365)
time = 1:nrow(data)
day = rep(1:365, 6)
```

```r
# GP could take lot of computation time
# subsample the data, take every 5s element
time_5s = seq(from = time[1], to = time[length(time)], by = 5)
day_5s = rep(seq(from = 1, to = 365, by = 5), times = 6)
# we need to extracted temp data for the linear regression
temp = data$temp
# case time - Exercise 2.2
temp_5s = temp[time_5s]
# case time - Exercise 2.4
temp_5s_day = temp[day_5s]


# get the data
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.cs

# create vector time (nr of days) & day (of the year 1:365)
time = 1:nrow(data)
day = rep(1:365, 6)

# GP could take lot of computation time
# subsample the data, take every 5s element
time_5s = seq(from = time[1], to = time[length(time)], by = 5)
day_5s = rep(seq(from = 1, to = 365, by = 5), times = 6)
# we need to extracted temp data for the linear regression
temp = data$temp
# case time - Exercise 2.2
temp_5s = temp[time_5s]
# case time - Exercise 2.4
temp_5s_day = temp[day_5s]


# check needed functions
?gausspr
?kernelMatrix
# init given values
x = 1
x_prime = 2
X = c(1,3,4)
X_prime = c(2,3,4)

# use lecture function
# Squared Exponential Kernel function
SEkernel = function(sigmaF = 1, ell = 1)
{
  SquaredExpKernel <- function(x, y = NULL) {
    n1 = length(x)
    n2 = length(y)
    K = matrix(NA, n1, n2)
    for(i in 1:n2) {
      for(j in 1:n1) {
        K[j,i] = sigmaF^2 * exp(-0.5 * ( (x[i] - y[j]) / ell)^2 )
      }
    }
```

```
    return(K)
  }
  class(SquaredExpKernel) <- "kernel"
  return(SquaredExpKernel)
}

# sigmaF = ell = 1
k = SEkernel()

## Evaluate kernel
#k(x,x_prime)

# Computing the whole covariance matrix K from the kernel
K = kernelMatrix(kernel = k, x = X, y = X_prime)
K
# fit quadratic regression
qr_fit = lm(temp_5s ~ time_5s + I(time_5s)^2)

# compute the residual vairance
sigmaNoise = sd(qr_fit$residuals)
#### Estimate Gaussian process regression model using the squared exponential function

# we need to extracted temp data for the linear regression
temp = data$temp
temp_5s = temp[time_5s]

## given values
ell = 0.2
var_f = 20

# fit GP
GPfit = gausspr(x = time_5s,
                y = temp_5s,
                kernel = SEkernel(sigmaF = var_f, ell = ell),
                #kpar = list(sigma = 1/(2*ell^2)),
                var = sigmaNoise^2)


# compute the posterior mean at every data point in the training dataset
meanPred = predict(GPfit, time_5s)


# scatterplot of the data and superimpose the posterior mean
plot(time_5s, temp_5s, main = "Posterior mean", ylab = "temp", xlab = "time")
lines(time_5s, meanPred, col="blue", lwd = 3)
legend("bottomright",
       legend=c("prediction mean", "observations"),
       col=c("blue", "black"), lty=c(1,NA), pch=c(NA,1), lwd=c(2,1))


####### same function as in E1.1, but I clean the envorionemnt and therefore create the function again
```

```r
# seperate kernel function
####### use the SquaredExpKernel form the lecture
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF,l){
  # input:
  # sigmaF  - influences the variance of the function (how variate is the function)
  # it influences also the 95% CI
  # l - influences the smoothnes
  # both have influance to the smoothnes


  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){
  # Function input
  # X: Vector of training inputs.
  # y: Vector of training targets/outputs
  # XStar: Vector of inputs where the posterior distribution is evaluated ie X_*
  # hyperparm: vector with two elements, with sigma_f (sigmaF) & l (ell)
  # sigmaNois: Noise standard deviation sigma_f (sigmaF)

  # before we can start with the step 2, where we compute L,
  # we need to calculate K, the kernel
  # we are suppost to use the squared exponentional kernel


  # compute the covariance matrix K or K(X,X)
  # late on we also need the covariance for k(x,x_*) & K(x_*,x_*)
  # K = the covariance which represents the distances between
  # all combination of points in the hyperspace
  K = SquaredExpKernel(x1 = X, x2 = X, sigmaF = hyperParam[1],
                       l = hyperParam[2])


  # next step compute L
  # we compute Cholesky decomposition + K
  n = length(X)

  ## 2: L
  L_trans = chol(K + (sigmaNoise^2 * diag(n)))
  #L_trans = chol(K + diag(sigmaNoise^2, nrow = length(X), ncol = length(X)))
  #L_trans = chol(K + (sigmaNoise^2)*diag(n))
  # L in the algorithm is a lower triangular matrix,
  # whereas the R function returns an upper triangular matrix.
  L = t(L_trans)
```

```
  ###### predictive mean
  # A\b means the vector x that solves the equation Ax = b - use function solve
  # alpha = L^T \ (L\y)
  alpha = solve(L_trans, solve(L,y))

  ## 4: f_bar_star
  # compute k^T_* to compute f_bar_star
  k_X_Xstart = SquaredExpKernel(x1 = X, x2 = XStar,
                                sigmaF = hyperParam[1],
                                l = hyperParam[2])
  f_bar_star = t(k_X_Xstart) %*% alpha


  ###### predictive variance

  # compute v
  v = solve(L, k_X_Xstart)

  ## 6: V[f_*]

  # compute k(x_*,x_*) to compute V
  k_Xstar_Xstar = SquaredExpKernel(x1 = XStar, x2 = XStar,
                                   sigmaF = hyperParam[1],
                                   l = hyperParam[2])

  # compute V[f_*]
  V_f_star = k_Xstar_Xstar - t(v) %*% v

  # we are only interested in variance of f,
  # therefore extract, take diagonal of covariance matrix of f
  V_f_star = diag(V_f_star)

  # note:
  # we do not need to compute the log marginal likelihood

  #save the pred mean and pred varaince
  result = list("predictive_mean" = f_bar_star,
                "predictive_var" = V_f_star)

}
# init given values
sigmaF = 20
ell = 0.2
obs_x_y = data.frame(x = scale(time_5s), y = scale(temp_5s))

# compute the posterior
posterior_E2.3 = posteriorGP(X = obs_x_y[,1], y = obs_x_y[,2],
                             XStar = scale(time_5s),
                             hyperParam = c(sigmaF, ell),
                             sigmaNoise = sigmaNoise)
CI_lower = meanPred - 1.96 * sqrt(posterior_E2.3$predictive_var)
CI_upper = meanPred + 1.96 * sqrt(posterior_E2.3$predictive_var)
```

```r
# bill verion
# we have to compute the variance
# therefore we fit a linear regression with the scaled data

# scale the data
temp_5s_scale = scale(temp_5s)
time_5s_scale = scale(time_5s)

# fit quadratic regression with the scaled data
qr_fit_scaled = lm(temp_5s_scale ~ time_5s_scale + I(time_5s_scale)^2)

# compute the residual vairance
sigmaNoise_scaled = sd(qr_fit_scaled$residuals)

# fit GP
hyperParam <- c(20, 0.3)

GPfit_sacled = gausspr(time_5s_scale,
                       temp_5s_scale,
                       kernel=SEkernel,
                       kpar=list(sigmaF=hyperParam[1], ell=hyperParam[2]),
                       var=sigmaNoise_scaled,
                       variance.model=TRUE,
                        scaled=F)
meanPred_scaled <- predict(GPfit_sacled, time_5s_scale)

N <- length(temp_5s_scale)
sek <- SEkernel(hyperParam[1], hyperParam[2])
K <- kernelMatrix(sek, time_5s_scale)
L <- t( chol( K+sigmaNoise_scaled*diag(N) ) )

# X is a scaled arithmetic sequence from 1 to 2186
# we can just use X as XStar

XStar <- time_5s_scale
kStar <- kernelMatrix(sek, time_5s_scale, XStar)
v <- solve(L) %*% kStar
vf <- kernelMatrix(sek, XStar, XStar) - t(v)%*%v
vf = diag(vf)



# Compute Upper & Lower Confidence Interval
CI_lower = meanPred - 1.96 * sqrt(vf)
CI_upper = meanPred + 1.96 * sqrt(vf)

# Superimpose the predictive interval
plot(time_5s, temp_5s, main = "Posterior mean", ylab = "temp", xlab = "time")
polygon(c(time_5s, rev(time_5s)),
        c(CI_lower, rev(CI_upper)),
        col = "darkgrey")
points(time_5s, temp_5s)
lines(time_5s, meanPred, col="blue", lwd = 2)
```

```
legend("bottomright",
       legend=c("prediction mean", "prediction interval", "observations"),
       col=c("blue", "darkgrey", "black"), lty=c(1,1,NA), pch=c(NA,NA,1), lwd=c(2,2,1))

# fit quadratic regression
qr_fit_day = lm(temp_5s_day ~ day_5s + I(day_5s)^2)

# given values
ell = 0.2
var_f = 20
######## do the same as in exercise 2.2


# compute the residual vairance
sigmaNoise_day = sd(qr_fit_day$residuals)

#### Estimate Gaussian process regression model using the squared exponential function

# fit GP
GPfit_day = gausspr(x = time_5s,
                    y = temp_5s,
                    kernel = SEkernel(sigmaF = var_f, ell = ell),
                    #kpar = list(sigma = 1/(2*ell^2)),
                    var = sigmaNoise_day^2)


# compute the posterior mean at every data point in the training dataset
meanPred_day = predict(GPfit_day, day_5s)
# scatterplot of the data and superimpose the posterior mean
plot(time_5s, temp_5s, main = "Posterior mean", ylab = "temp", xlab = "time")
polygon(c(time_5s, rev(time_5s)),
        c(CI_lower, rev(CI_upper)),
        col = "darkgrey")
points(time_5s, temp_5s)
lines(time_5s, meanPred, col="blue", lwd = 2)
lines(time_5s, meanPred_day, col="red", lwd = 2)
legend("bottomright",
       legend=c("prediction mean time", "prediction interval",
                "prediction mean day" , "observations"),
       col=c("blue", "darkgrey", "red" , "black"),
       lty=c(1,1,1,NA), pch=c(NA,NA, NA,1), lwd=c(2,2,2,1), cex=0.7)
####### for the implementation:
# the periodic kernel: is like a outer function to convert the result as an kernel object
# inside follows the normal function implementation
# every x1 and x2 is going to be assignt via gausspr function
# init given values
sigmaF = 20
ell1 = 1
ell2 = 10
d = 350/sd(time)

# implement generalization of periodic kernel
```

```
# periodic kernel

periodic_kernel = function(sigmaF, l1, l2, d) {
  P_Kernel = function(x1,x2) {
    result = (sigmaF^2)*
      exp(-2*((sin(pi*abs(x1-x2)/d)^2)/(l1^2)))*
      exp(-0.5*((x1 -x2)^2)/(l2^2))
    return(result)
  }
  class(P_Kernel) = "kernel"
  return(P_Kernel)
}
# fit GP
GP_periodic = gausspr(x = time_5s,
                      y = temp_5s,
                      kernel = periodic_kernel(sigmaF = sigmaF,
                                               l1 = ell1,
                                               l2 = ell2,
                                               d = d),
                      var = sigmaNoise^2)


# compute the posterior mean at every data point in the training dataset
meanPred_periodic = predict(GP_periodic, time_5s)


# scatterplot of the data and superimpose the posterior mean
plot(time_5s, temp_5s, main = "Posterior mean", ylab = "temp", xlab = "time")
polygon(c(time_5s, rev(time_5s)),
        c(CI_lower, rev(CI_upper)),
        col = "darkgrey")
points(time_5s, temp_5s)
lines(time_5s, meanPred, col="blue", lwd = 3)
lines(time_5s, meanPred_day, col="red", lwd = 3)
lines(x = time_5s, meanPred_periodic, col = "green", lwd = 3)
legend("bottomright",
       legend=c("prediction mean time", "prediction interval",
                "prediction mean day" , "prediction mean periodic", "observations"),
       col=c("blue", "darkgrey", "red" , "green", "black"),
       lty=c(1,1,1,1,NA), pch=c(NA,NA, NA,NA,1), lwd=c(2,2,2,2,1), cex=0.7)
set.seed(12345)
rm(list = ls())
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])

# create training sample
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
# create training und test data
train = data[SelectTraining,]
test = data[-SelectTraining,]
```

```
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])

# create training sample
set.seed(111)
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
# create training und test data
train = data[SelectTraining,]
test = data[-SelectTraining,]

######## fit gaussian process
# fraud on training
# kernel & hyperparameter = default
GPfit = gausspr(fraud ~ .,
                data = train)


# only use covariates varWave & sekwWave
GPfit_var_sekw = gausspr(fraud ~ varWave + skewWave,
                         data = train)

###### make predcition on the training & check accuracy
########## GPfit
pred_GPfit = predict(GPfit,
                     train)
cm_PGfit = table(pred_GPfit, train$fraud)
accuracy_GPfit = sum(diag(cm_PGfit))/sum(cm_PGfit)

########## GPfit_var_sekw
pred_GPfit_var_sekw = predict(GPfit_var_sekw,
                              train)
cm_PGfit_var_sekw = table(pred_GPfit_var_sekw, train$fraud)
accuracy_GPfit_var_sekw = sum(diag(cm_PGfit_var_sekw))/sum(cm_PGfit_var_sekw)

# make prediction of class probabilities
probPreds <- predict(GPfit_var_sekw, train, type="probabilities")

# init suitable grid of values for varWave and skewWave
# therefore, check min and max values of these two variables
x1 <- seq(min(train$varWave),max(train$varWave),length=100)
x2 <- seq(min(train$skewWave),max(train$skewWave),length=100)

# Creates 2D matrices for accessing images and 2D matrices
gridPoints <- meshgrid(x1, x2)# output of mashgrid:
# list of every of the 100 points repeated 10 times
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y)) #output
# x1 and x2 in just 2 columns with length 10000

# transform to make new predictions
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train)[1:2]
```

```r
probPreds <- predict(GPfit_var_sekw, gridPoints, type="probabilities")

# Plotting for Prob
contour(x = x1, y = x2,
        z = matrix(probPreds[,1],100,byrow = TRUE),
        nlevel = 20,
        xlab = "varWave", ylab = "sekwWave",
        main = 'Prob')
points(x = train$varWave[pred_GPfit_var_sekw == 1],
       y = train$skewWave[pred_GPfit_var_sekw == 1], col = "red")
points(x = train$varWave[pred_GPfit_var_sekw == 0],
       y = train$skewWave[pred_GPfit_var_sekw == 0], col = "blue")
legend("bottomright",
       pch = c("o","o"),
       legend = c("Not fraud", "Fraud"),
       col = c("blue", "red"))


################### 3D
#install.packages("plot3D")
x_3d = gridPoints[,1]
y_3d = gridPoints[,2]
z_3d = matrix(probPreds[,1],100,byrow = TRUE)

library("plot3D")
scatter3D(x = x1,
          y = x2,
          z = z_3d)
cm_PGfit_var_sekw
accuracy_GPfit_var_sekw

########## GPfit_var_sekw
pred_GPfit_var_sekw_test = predict(GPfit_var_sekw,
                                   test)
cm_PGfit_var_sekw_test = table(pred_GPfit_var_sekw_test, test$fraud)
accuracy_GPfit_var_sekw_test = round(sum(diag(cm_PGfit_var_sekw_test))/sum(cm_PGfit_var_sekw_test), 3)


###### make predcition on the training & check accuracy
########## GPfit
pred_GPfit_test = predict(GPfit,
                          test)
cm_PGfit_test = table(pred_GPfit_test, test$fraud)
accuracy_GPfit_test = round(sum(diag(cm_PGfit_test))/sum(cm_PGfit_test), 3)

# create a tables with all results
result_df = data.frame("accuracy all covariates" = c(accuracy_GPfit,
                                                     accuracy_GPfit_test),
                       "accuracy selected covariates" = c(accuracy_GPfit_var_sekw,
                                                          accuracy_GPfit_var_sekw_test))

result_df = t(result_df)
colnames(result_df) = c("Train", "Test")
```

```r
result_df
#knitr::kable(result_df)
#install.packages("mvtnorm")
library("mvtnorm")


######## Gaussian Process Regression
### A function f(x) consist of
# gaussian process G(m(x), k(x,x´))
# m(x) is the mean function
# k(x,x´) is the covariance function


# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  # input:
  # sigmaF  - influences the variance of the function (how variate is the function)
  # it influences also the 95% CI
  # l - influences the smoothnes
  # both have influance to the smoothnes


  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)
}

# Mean function
MeanFunc <- function(x){
  m <- sin(x)
  return(m)
}

# Simulates nSim realizations (function) from a GP with mean m(x) and covariance K(x,x')
# over a grid of inputs (x)
SimGP <- function(m = 0,K,x,nSim,...){
  n <- length(x)
  if (is.numeric(m)) meanVector <- rep(0,n) else meanVector <- m(x)
  covMat <- K(x,x,...)
  f <- rmvnorm(nSim, mean = meanVector, sigma = covMat)
  return(f)
}

xGrid <- seq(-5,5,length=20) # change the nr of points smapled here

# Plotting one draw
sigmaF <- 1 #### affect of the variance of the function & 95% CI
l <- 1 ##### affect to the smoothnes
nSim <- 1
```

```r
fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)
plot(xGrid, fSim[1,], type="p", ylim = c(-3,3))
if(nSim>1){
  for (i in 2:nSim) {
    lines(xGrid, fSim[i,], type="p")
  }
}
lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3)
lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
      col = "blue", lwd = 2)
lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
      col = "blue", lwd = 2)

# Plotting using manipulate package
library(manipulate)

plotGPPrior <- function(sigmaF, l, nSim){
  fSim <- SimGP(m=MeanFunc, K=SquaredExpKernel, x=xGrid, nSim, sigmaF, l)
  plot(xGrid, fSim[1,], type="l", ylim = c(-3,3), ylab="f(x)", xlab="x")
  if(nSim>1){
    for (i in 2:nSim) {
      lines(xGrid, fSim[i,], type="l")
    }
  }
  lines(xGrid,MeanFunc(xGrid), col = "red", lwd = 3)
  lines(xGrid, MeanFunc(xGrid) - 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
        col = "blue", lwd = 2)
  lines(xGrid, MeanFunc(xGrid) + 1.96*sqrt(diag(SquaredExpKernel(xGrid,xGrid,sigmaF,l))),
        col = "blue", lwd = 2)
  title(paste('length scale =',l,', sigmaf =',sigmaF))
}

manipulate(
  plotGPPrior(sigmaF, l, nSim = 10),
  sigmaF = slider(0, 2, step=0.1, initial = 1, label = "SigmaF"),
  l = slider(0, 2, step=0.1, initial = 1, label = "Length scale, l")
)
#############################################################################################
## Kernlab demo for Gaussian processes regression and classfication
## Author: Mattias Villani, Linkoping University. http://mattiasvillani.com
#############################################################################################


#############################################################################
###    Prelims: Setting path and installing/loading packages            #####
#############################################################################
setwd('/Users/mv/Dropbox/Teaching/AdvML/GaussianProcess/Code')
#install.packages('kernlab')
#install.packages("AtmRay") # To make 2D grid like in Matlab's meshgrid
library(kernlab)
library(AtmRay)
```

```
################################################
###    Messin' around with kernels        #####
################################################
# This is just to test how one evaluates a kernel function
# and how one computes the covariance matrix from a kernel function
X <- matrix(rnorm(12), 4, 3)
Xstar <- matrix(rnorm(15), 5, 3)
ell <- 1
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdot
#(which is the SE kernel) in kernlab
SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2
# Computing the whole covariance matrix K from the kernel. Just a test.
K <- kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar)

# Own implementation of Matern with nu = 3/2 (See RW book equation 4.17)
# Note that a call of the form kernelFunc <- Matern32(sigmaf = 1, ell = 0.1) r
#eturns a kernel FUNCTION.
# You can now evaluate the kernel at inputs: kernelFunc(x = 3, y = 4).
# Note also that class(kernelFunc) is of class "kernel", which is a class defined
#by kernlab.
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(rnorm(12), 4, 3) # Simulating some data
Xstar <- matrix(rnorm(15), 5, 3)
MaternFunc = Matern32(sigmaf = 1, ell = 2) # MaternFunc is a kernel FUNCTION
MaternFunc(c(1,1),c(2,2)) # Evaluating the kernel in x=c(1,1), x'=c(2,2)
# Computing the whole covariance matrix K from the kernel.
K <- kernelMatrix(kernel = MaternFunc, x = X, y = Xstar) # So this is K(X,Xstar)




################################################
###      Regression on the LIDAR data       ###
################################################
#lidarData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess
lidarData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/
                        master/GaussianProcess/Code/LidarData',
                        header = T)
LogRatio <- lidarData$LogRatio
Distance <- lidarData$Distance
```

```
# Estimating the noise variance from a third degree polynomial fit
polyFit <- lm(LogRatio ~  Distance + I(Distance^2) + I(Distance^3) )
sigmaNoise = sd(polyFit$residuals)

plot(Distance,LogRatio)

# Fit the GP with built in Square expontial kernel (called rbfdot in kernlab)
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = rbfdot, kpar =
                   list(sigma = 1/(2*ell^2)), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance) # Predicting the training data. To plot the fit.
lines(Distance, meanPred, col="blue", lwd = 2)

# Fit the GP with home made Matern
sigmaf <- 1
ell <- 2
# GPfit <- gausspr(Distance, LogRatio, kernel = Matern32(ell=1))
# NOTE: this also works and is the same as the next line.
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="purple", lwd = 2)

# Trying another length scale
sigmaf <- 1
ell <- 1
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="green", lwd = 2)

# And now with a different sigmaf
sigmaf <- 0.1
ell <- 2
GPfit <- gausspr(Distance, LogRatio, kernel = Matern32, kpar =
                   list(sigmaf = sigmaf, ell=ell), var = sigmaNoise^2)
meanPred <- predict(GPfit, Distance)
lines(Distance, meanPred, col="black", lwd = 2)









################################################
####       Classification on Iris data      ###
################################################
data(iris)
GPfitIris <- gausspr(Species ~  Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
                     data=iris)
GPfitIris
```

```
# predict on the training set
predict(GPfitIris,iris[,1:4])
table(predict(GPfitIris,iris[,1:4]), iris[,5]) # confusion matrix

# Now using only Sepal.Length and Sepal.Width to classify
GPfitIris <- gausspr(Species ~  Sepal.Length + Sepal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris,iris[,1:2])
table(predict(GPfitIris,iris[,1:2]), iris[,5]) # confusion matrix

# Now using only  Petal.Length + Petal.Width to classify
GPfitIris <- gausspr(Species ~ Petal.Length + Petal.Width, data=iris)
GPfitIris
# predict on the training set
predict(GPfitIris,iris[,3:4])
table(predict(GPfitIris,iris[,3:4]), iris[,5]) # confusion matrix

# class probabilities
probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(iris[,3]),max(iris[,3]),length=100)
x2 <- seq(min(iris[,4]),max(iris[,4]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(iris)[3:4]
probPreds <- predict(GPfitIris, gridPoints, type="probabilities")

# Plotting for Prob(setosa)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Setosa) - Setosa is red')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")

# Plotting for Prob(Versicolor)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Versicolor) - Versicolor is green')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")


# Plotting for Prob(virginica)
contour(x1,x2,matrix(probPreds[,3],100,byrow = TRUE), 20,
        xlab = "Petal.Length", ylab = "Petal.Width",
        main = 'Prob(Virginica) - Virginica is blue')
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],col="red")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],col="blue")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],col="green")
```

```
# Plotting the decision boundaries
meanPred <- matrix(max.col(probPreds),100,byrow = TRUE)
plot(gridPoints,  pch=".", cex=3, col=ifelse(meanPred==1, "red",
                                             ifelse(meanPred==2, "green", "blue")))
points(iris[iris[,5]=='setosa',3],iris[iris[,5]=='setosa',4],
       col="red", cex=10, pch=".")
points(iris[iris[,5]=='virginica',3],iris[iris[,5]=='virginica',4],
       col="blue",  cex=10, pch=".")
points(iris[iris[,5]=='versicolor',3],iris[iris[,5]=='versicolor',4],
       col="green",  cex=10, pch=".")

########################################
# Author: Jose M. Pena, jose.m.pena@liu.se
# GP regression on the canadian wages data
########################################

library(kernlab)
#CWData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/master/GaussianProcess/Co
CWData <- read.table('https://raw.githubusercontent.com/STIMALiU/AdvMLCourse/
                     master/GaussianProcess/Code/CanadianWages.dat',
                     header = T)
logWage<-CWData$logWage
age<-CWData$age
age<-(age-mean(age))/sd(age) # Standarize the age

# Estimating the noise variance from a third degree polynomial fit.
#I() is needed because, otherwise
# age^2 reduces to age in the formula, i.e. age^2 means adding the main
#effect and the second order
# interaction, which in this case do not exist. See ?I.
polyFit <- lm(logWage ~  age + I(age^2) + I(age^3))
sigmaNoise = sd(polyFit$residuals)
plot(age,logWage)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab)
ell <- 0.5
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note the reparametrization
GPfit <- gausspr(age,logWage, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, age) # Predicting the training data
lines(age, meanPred, col="red", lwd = 2)

# The implementation of kernlab for the probability and prediction intervals seem
#to have a bug: The intervals
# seem to be too wide, e.g. replace 1.96 with 0.1 to see something.
# GPfit <- gausspr(age,logWage, kernel = SEkernel, var = sigmaNoise^2, variance.model = TRUE)
# meanPred <- predict(GPfit, age)
# lines(age, meanPred, col="red", lwd = 2)
# lines(age, meanPred+1.96*predict(GPfit,age, type="sdeviation"),col="blue")
# lines(age, meanPred-1.96*predict(GPfit,age, type="sdeviation"),col="blue")

# Probability and prediction interval implementation.
x<-age
xs<-age # XStar
```

```
n <- length(x)
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

# Probability intervals for fStar
lines(xs, meanPred - 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)
lines(xs, meanPred + 1.96*sqrt(diag(Covf)), col = "blue", lwd = 2)

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