Aruup817@student.liu.se

Lab 4 Gaussian Processes

ArunUppugunduri

2020-10-16

Assignment 1

Write your own code for the gaussian process regression model:

y = f(x) + epsilon with epsilon ∼ N (0, σ 2 n) and f ∼ GP(0, k(x, x′))

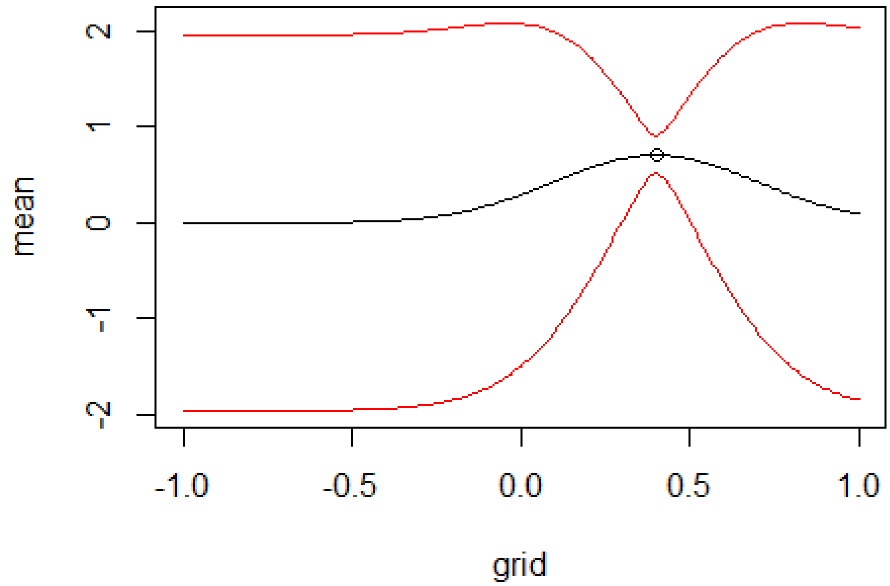
```
library(kernlab)


### Assignment 1###


# Simulate nSim realizations (functions) from a GP wiht mean 0 and covariance
K(x,x')


# Covariance function
# The function takes in input values x1, x2 and computes the exponential
kernel which gives
# the resulting covariance between the two input values
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}


PosteriorGP = function(X, y, XStar, sigmaNoise, hyperParameter){
  #n is how many functions we will need, as many as the nr of inputs
  n = length(X)

  #Compute the covarance matrix [K = K(X,X)]
  K = SquaredExpKernel(X,X, hyperParameter[1], hyperParameter[2])
  sigmaNoise = sigmaNoise^2

  #Compute L by using the cholesky decomposition
```

```
  L_trans = chol(K + sigmaNoise*diag(n))
  #Need to take the transpose since L in the algorithm is a lower triangular 
matrix where as
  # the R function returns an uper triangular matrix
  L = t(L_trans)

  ### Predictive mean f_bar*
  ##Comute the predictive mean by solving the equations
  # L\y means the vector x that solves the equation Lx = y. On paper it can 
be solved by
  # multiplying by the inverse but is better solved by using the function 
solve
  # [alpha = t(L)\(L\y)]
  alpha = solve(L_trans, solve(L,y))

  ##Compute f_bar*
  #f_bar* = alpha * t(K*)
  # [K* = K(X, X*) => t(K*) = K(X*, X)]
  K_X_Xstar = SquaredExpKernel(X, XStar, hyperParameter[1], 
hyperParameter[2])
  f_bar_star = t(K_X_Xstar) %*% alpha

  ### Predictive variance f_star
  #Compute v, [v = L\K*]
  v = solve(L, K_X_Xstar)

  ## Compute the variance of f*
  #V[f_star] = K(X*, X*) - t(v)*v
  # First need to compute K[X*, X*]
  K_XStar_XStar = SquaredExpKernel(XStar, XStar, hyperParameter[1], 
hyperParameter[2])
  #Compute V_f*
  v_f_star = K_XStar_XStar - t(v) %*% v
  #To draw from the posterior we only need the variance of f
  v_f_star = diag(v_f_star)

  result = list("Predictive mean" = f_bar_star,
                "Predicitive variance" = v_f_star)
}
```

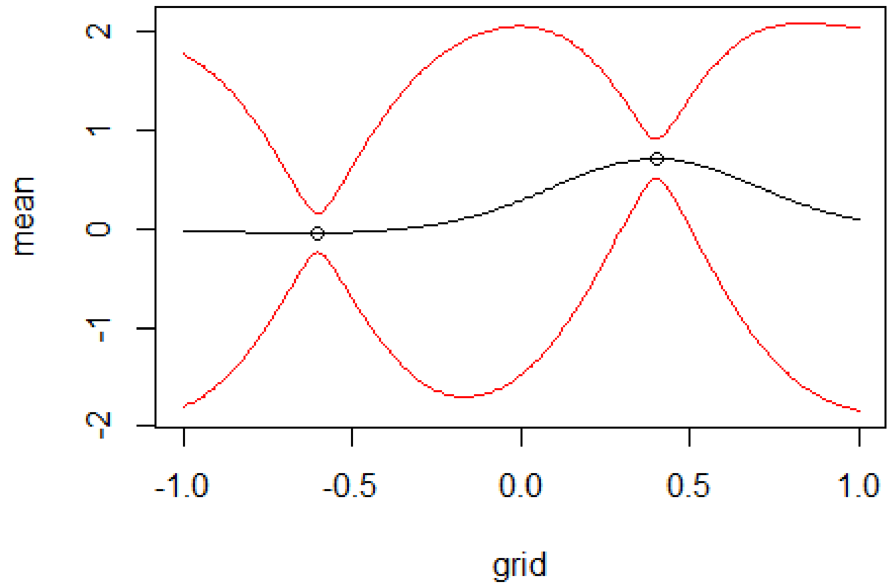
Aruup817@student.liu.se

Part 1.2

```r
### Part 1.2
# Let the hyperparameters be the following: sigmaf = 1, el = 0,3, using a singel observation (x,y) = (0.4, 0.719), sigmanoise = 0.1
# Plot the posterior

#Function for plotting the result
plotGP = function(mean,variance,grid,x,y){
  plot(grid,mean,ylim = c(min(mean-1.96*sqrt(variance))
                          ,max(mean+1.96*sqrt(variance))),
       type = "l")
  lines(grid,
        mean+1.96*sqrt(variance),
        col = "red")
  lines(grid,
        mean-1.96*sqrt(variance),
        col = "red")
  points(x,y)
}

sigmaF = 1
ell = 0.3
obs = data.frame(0.4, 0.719)
sigmanoise = 0.1
xGrid = seq(-1,1,0.01)

GP = PosteriorGP(obs[,1], obs[,2], xGrid, sigmanoise,c(sigmaF, ell))
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs[1,1], obs[1,2])
```



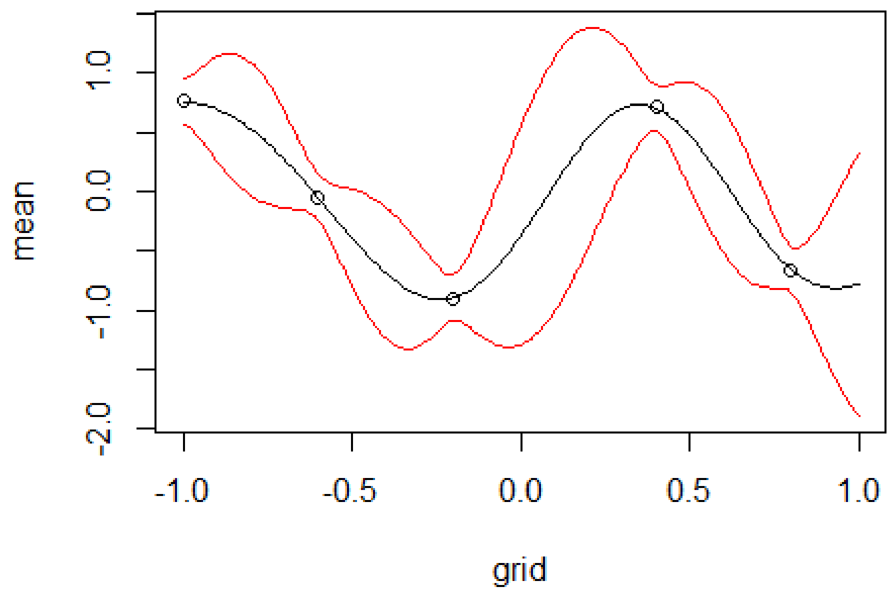

```
### Part 1.3
#Update the posterior with another observation (x,y) = (-0.6, -0.044) and
plot the posterior mean of f and the probability bands

newobs = c(-0.6, -0.044)
obs_1.3 = rbind(obs, newobs)
GP = PosteriorGP(obs_1.3[,1], obs_1.3[,2], xGrid, sigmanoise,c(sigmaF, ell))
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs_1.3[,1],
obs_1.3[,2])
```




```
### Part 1.4
#Compute now the posterior distirbution of f using all available data points

obs_1.4 = data.frame(x=c(-1,-0.6, -0.2, 0.4, 0.8), y=c(0.768, -0.044, -0.904,
0.719,-0.664))
GP = PosteriorGP(obs_1.4[,1], obs_1.4[,2], xGrid, sigmanoise,c(sigmaF, ell))
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs_1.4[,1],
obs_1.4[,2])
```

As more data points is are fed the resulting variance is changes since the covariance is now computed for more values giving us more information. Consequently we can se how the resulting bands become more narrow where the model is now better at making interpolarization.

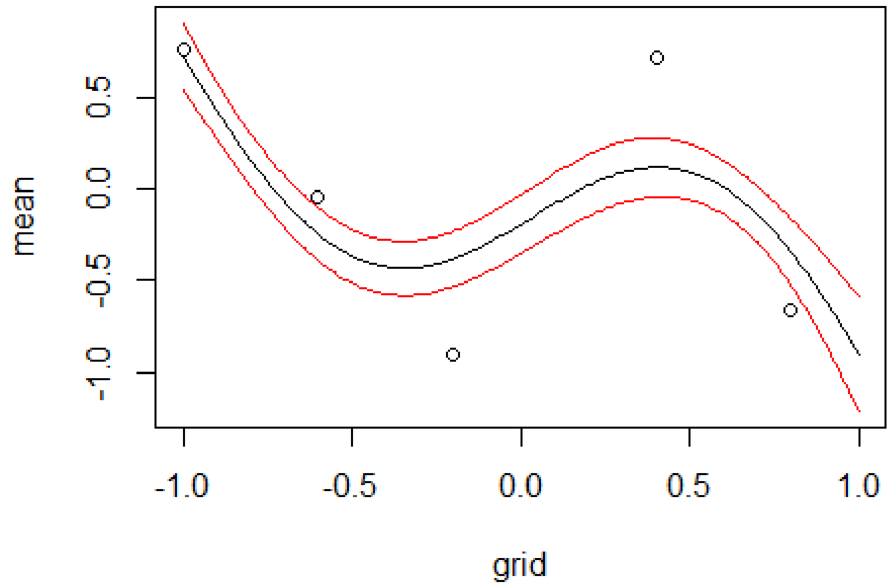
Aruup817@student.liu.se




```
#Part 1.5
#Repeat the exercise, this time wiht hyperparameters sigmaf = 1 and ell = 1
# Compare the results
sigmaF = 1
ell = 1
GP = PosteriorGP(obs_1.4[,1], obs_1.4[,2], xGrid, sigmanoise,c(sigmaF, ell))
plotGP(GP$`Predictive mean`, GP$`Predicitive variance`, xGrid, obs_1.4[,1],
obs_1.4[,2])
```

As the l value is increased from 0.3 to 1 the plot becomes a lot more smooth which is due to that the correlation between points becomes higher and resultingly the function values will have less variance. This high value then results in an underfit since it now fails to capture the true data points.

Aruup817@student.liu.se



Assignment 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.

Part 2.1

Define your own square exponential kernel function (with parameters l` (ell) and σf (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

Aruup817@student.liu.se

```
##################### Assignment 2##################################
#################### GP Regresssion withKernlab#########################

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

library(kernlab)

time = seq(1, nrow(tempData))

day = c()
counter = 1
for (i in 1:nrow(tempData)){

  if(counter > 365){
    counter = 1
  }

  day = append(day, counter)
  counter = counter +1

}


five_sequence = seq(1, nrow(tempData), 5)
time_selection = time[five_sequence]
day_selection = day[five_sequence]
temperature = tempData$temp
temperature_selection = temperature[five_sequence]
```

Part 2.1

```
### 2.1
# Define your own square exponential kernel function (with parameters ` (ell)
and
# σf (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

x = 1
x_prime = 2
X = c(1,3,4)
X_prime = c(2,3,4)
```

Aruup817@student.liu.se

```r
SE_Kernel <- function(sigmaF=1,l=1){
  rval = function(x, y = NULL){
    n1 <- length(x)
    n2 <- length(y)
    res = sigmaF^2*exp(-0.5*( (x-y)/l)^2 )
    return(res)
  }
  class(rval) = "kernel"
  return(rval)

}
```

```r
## Compute the covariance for (x, x')
# Initialize the kernel, values ell and sigmaF = 1
kernel = SE_Kernel()

# Evalute the kernel in x = 1 and x' = 2
covariance = kernel(x = 1, y = 2)
covariance
```

Kernel evaluated in (1,2)

```r
## [1] 0.6065307

# Compute the covariance matrix for the input vectors X, X_prime K[X,
X_prime]
# X = c(1,3,4)
# X_prime = c(2,3,4)

cov_matrix = kernelMatrix(kernel = SE_Kernel(), x = X, y = X_prime)
cov_matrix

## 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
```

Above can be seen the result from evaluating the input vector X, Xstar.

Aruup817@student.liu.se

Part 2.2

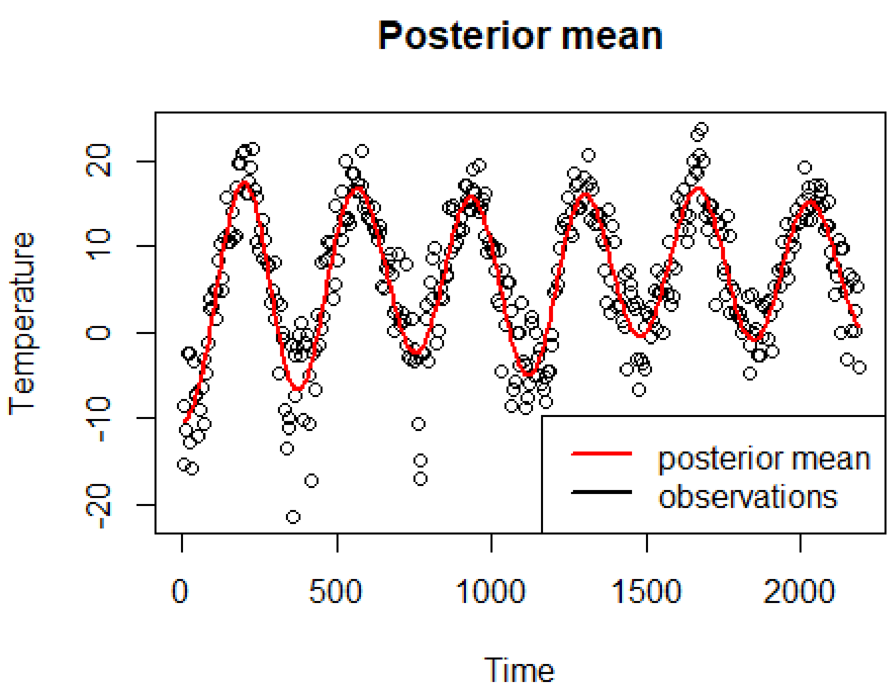
Consider the following model

temp = f(time) + with ∼ N (0, σ 2 n) and f ∼ GP(0, k(time, time′))

Let σ 2 n 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 squared exponential function from (1) with σf = 20 and ` = 0.2. 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). Play around with different values on σf and ` (no need to write this in the report though).

```
### 2.2
#Consider the following model:
#temp = f(time) + epsilon with epsilon ∼ N (0, σ2n) and f ∼ GP(0, k(time,
time’))

#Let σ2n 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
squaredexponential function from (1) with σf = 20 and ` = 0.2.
#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). Play around with different values on σf and
` (no needto write this in the report though).

sigmaf = 20
ell = 0.2

# Fit quadratic regression with the scaled data
regression_fit = lm(temperature_selection ~ time_selection +
I(time_selection)^2)

#Compute the residual variance
sigmaNoise = sd(regression_fit$residuals)

hyperparam = c(sigmaf, ell)

#Compute GP regression

GP_fit = gausspr(x = time_selection,
                 y = temperature_selection,
```

```
                 kernel = SE_Kernel(sigmaF = sigmaf, l = ell),
                 var = sigmaNoise^2)

#Compute the posterior mean at every data point in the training
# dataset
meanPred = predict(GP_fit, time_selection)

plot(time_selection, temperature_selection, main = "Posterior mean",
     ylab = "Temperature", xlab = "Time")
lines(time_selection, meanPred, col = "red", lwd = 2)
legend("bottomright", legend = c("posterior mean",
"observations"),col=c("red", "black"),
       lty = c(1, 1), lwd = c(2,2))
```



Part 2.3

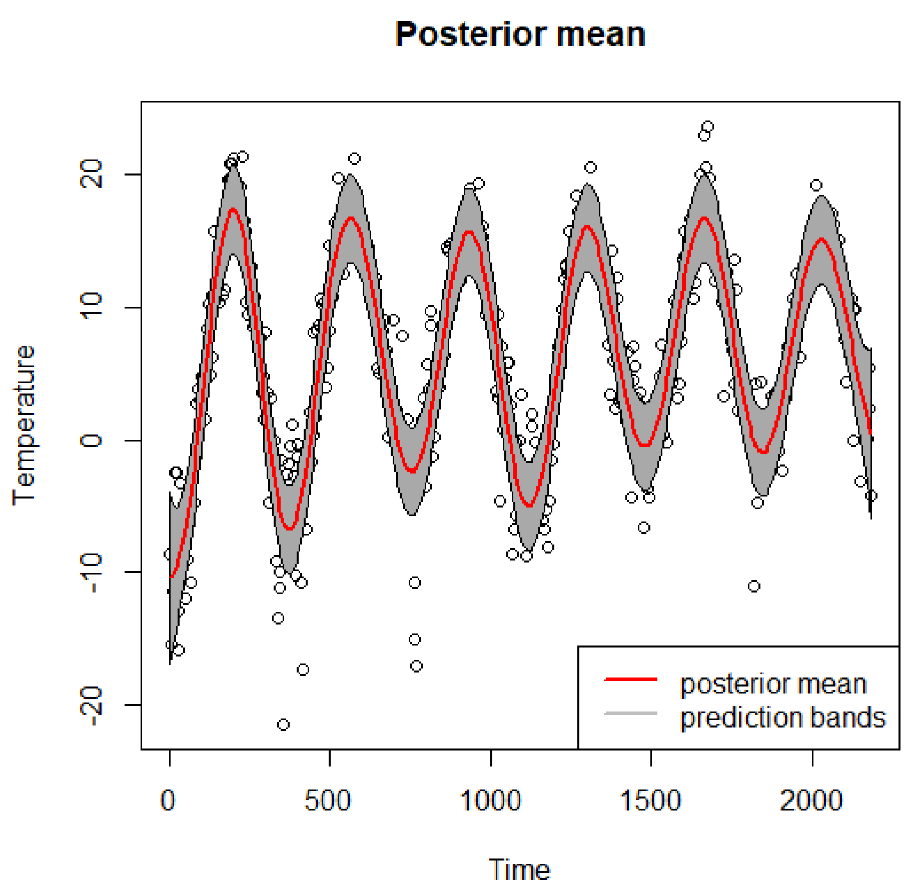
Compute the posterior variance of f & and the 95% confidence interval.

```
### 2.3 ###
# Make own computations to obtain the posterior variance of f and plot the
# 95 % probability bands for f. To do this we can use the prviously computed
# function PosteriorGP

sigmaf = 20
ell = 0.2
```

Aruup817@student.liu.se

```
hyperparam = c(sigmaf, ell)

SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}

PosteriorGP = function(X, y, XStar, sigmaNoise, hyperParameter){

  n = length(X)
  #Compute the covarance matrix [K = K(X,X)]
  K = SquaredExpKernel(X,X, hyperParameter[1], hyperParameter[2])

  L_trans = chol(K + sigmaNoise^2*diag(n))
  #Need to take the transpose since L in the algorithm is a lower triangular
matrix where as
  # the R function returns an uper triangular matrix
  L = t(L_trans)
  alpha = solve(L_trans, solve(L,y))

  ##Compute f_bar*
  K_X_Xstar = SquaredExpKernel(X, XStar, hyperParameter[1],
hyperParameter[2])

  f_bar_star = t(K_X_Xstar) %*% alpha

  v = solve(L, K_X_Xstar)

  K_XStar_XStar = SquaredExpKernel(XStar, XStar, hyperParameter[1],
hyperParameter[2])
  #Compute V_f*
  v_f_star = K_XStar_XStar - t(v) %*% v
  #To draw from the posterior we only need the variance of f
  v_f_star = diag(v_f_star)

  result = list("Predictive mean" = f_bar_star,
                "Predicitive variance" = v_f_star)
}


posterior  = PosteriorGP(X = scale(time_selection),
                         y = scale(temperature_selection),
                         XStar = scale(time_selection),
                         sigmaNoise = sigmaNoise,
```

```
                             hyperParameter = hyperparam)


# Compute the variance for f
posterior_variance = posterior$`Predicitive variance`
posterior_mean = posterior$`Predictive mean`
posterior_mean = posterior_mean*sd(temperature_selection) +
mean(temperature_selection)

L = posterior_mean - 1.96*sqrt(posterior_variance)
U = posterior_mean + 1.96*sqrt(posterior_variance)

# Plot the meanPred, and the prediction bands for the posterior variance
plot(time_selection, temperature_selection, main = "Posterior mean",
     ylab = "Temperature", xlab = "Time")
lines(time_selection, posterior_mean, col = "red", lwd = c(2,2))
legend("bottomright", legend = c("posterior mean", "prediction
bands"),col=c("red", "blue"),
       lty = c(1, 1), lwd = c(2,2))
lines(time_selection, meanPred + 1.96*sqrt(posterior_variance), col = "blue")
lines(time_selection, meanPred - 1.96*sqrt(posterior_variance), col = "blue")
```



Aruup817@student.liu.se

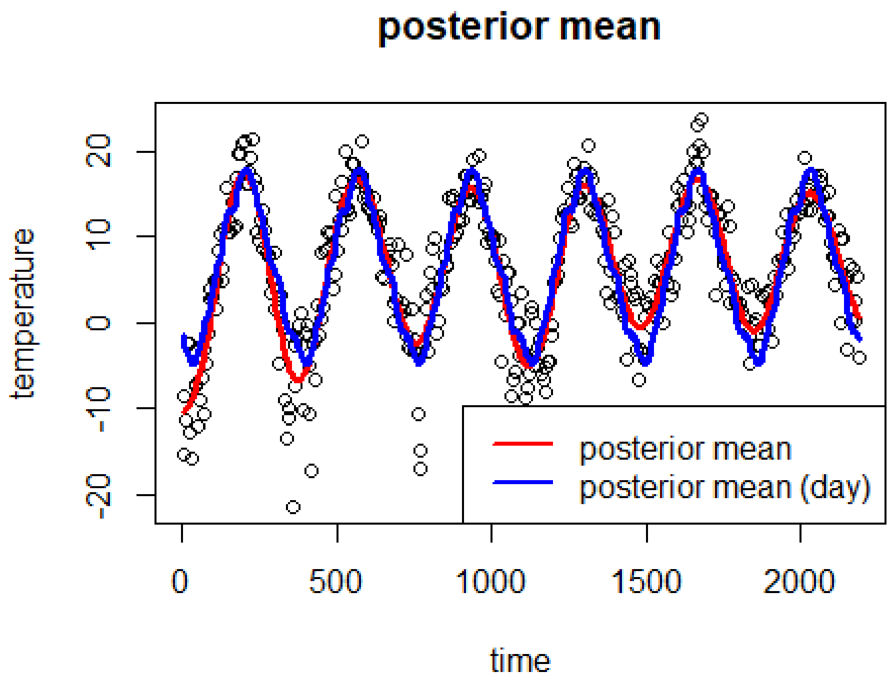
Part 2.4

Consider now the following model:

temp = f(day) + epsilon with ∼ N (0, σ 2 n) and f ∼ GP(0, k(day, day′))

Estimate the model using the squared exponential function with σf = 20 and ` = 0.2. 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. Compare the results of both models. What are the pros and cons of each model?

```
### Part 2.4 ###
regression_fit_4 = lm(temperature_selection ~ day_selection +
I(day_selection)^2)
sigmaNoise_day = sd(regression_fit_4$residuals)

ell = 0.2
sigmaf = 20

GP_fit_day = gausspr(x = day_selection,
                     y = temperature_selection,
                     kernel = SE_Kernel(sigmaF = sigmaf, l = ell),
                     var = sigmaNoise_day)

meanPred_day = predict(GP_fit_day, day_selection)

plot(time_selection, temperature_selection, main = "posterior mean",
     ylab = "temperature", xlab = "time")
points(time_selection, temperature_selection)
lines(time_selection, meanPred, col = "red", lwd = 3)
lines(time_selection, meanPred_day, col = "blue", lwd = 3)
legend("bottomright", legend = c("posterior mean", "posterior mean
(day)"),col=c("red", "blue"),
       lty = c(1, 1), lwd = c(2,2))
```



The model using the days we can see that the prediction repeats itself every year since it is following an interval from 1 to 365. This is good in the sense that it captures a correlation between the same day but different years but it fails to capture that the mean of the data appears to be increasing every year.

Part 2.5

Finally, implement a generalization of the periodic kernel given in the lectures:

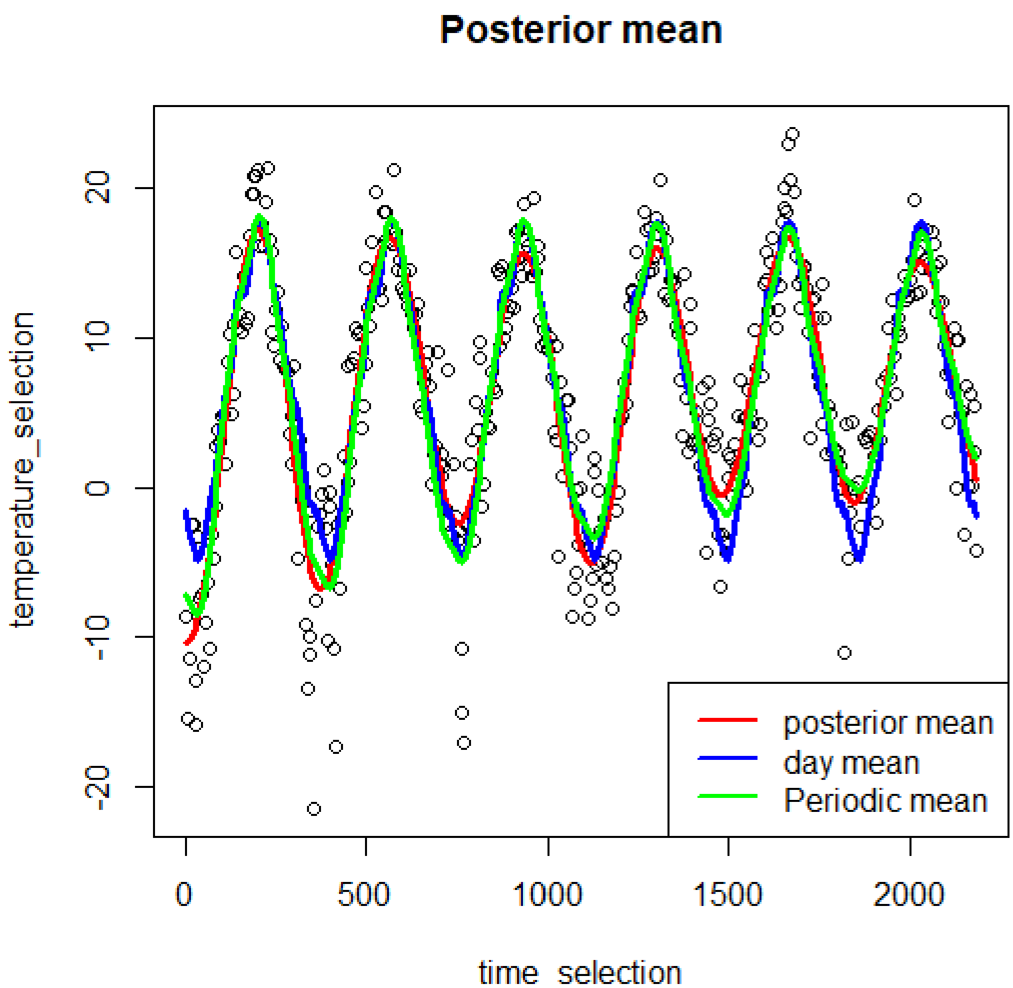
$$k(x, x') = \sigma_f^2 \exp\left\{-\frac{2\sin^2(\pi|x - x'|/d)}{\ell_1^2}\right\} \exp\left\{-\frac{1}{2}\frac{|x - x'|^2}{\ell_2^2}\right\}$$

Note that we have two different length scales here, and `2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters σf = 20, `1 = 1, `2 = 10 and d = 365/sd(time). The reason for the rather strange period here is that kernlab standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with σf = 20 and ` = 0.2). Discuss the results.

Aruup817@student.liu.se

```
### Part 2.5 ###
#Implement a generalization of the periodic kernel given in the lectures
# Note that we have two different l which controls the correlation between
# the same day in different years. Estimate the GP mmodel using the time
variable
# with this kernel and the hyperparamerers:

sigmaf = 20
ell1 = 1
ell2 = 10
d = 365/sd(time_selection)

#Create the periodic kernel
periodic_kernel = function(sigmaf, l1, l2, d){
  Periodic_K = function(x,y){
    result = (sigmaf^2)*exp(-2*((sin(pi*abs(x-y)/d)^2)/(l1^2)))*
      exp(-0.5*((x-y)^2)/(l2^2))
    return(result)
  }
  class(Periodic_K) = "kernel"
  return(Periodic_K)
}

GP_periodic = gausspr(x = time_selection,
                      y = temperature_selection,
                      kernel = periodic_kernel(sigmaf, ell1, ell2, d),
                      var = sigmaNoise)

meanPred_periodic = predict(GP_periodic, time_selection)

plot(time_selection, temperature_selection, main = "Posterior mean")
lines(time_selection, meanPred, col = "red", lwd = 2)
lines(time_selection, meanPred_day, col = "blue", lwd = 2)
lines(time_selection, meanPred_periodic, col = "green", lwd = 2)
legend("bottomright", legend = c("posterior mean", "day mean", "Periodic
mean"),col=c("red", "blue", "green"), lwd = c(2,2))
```

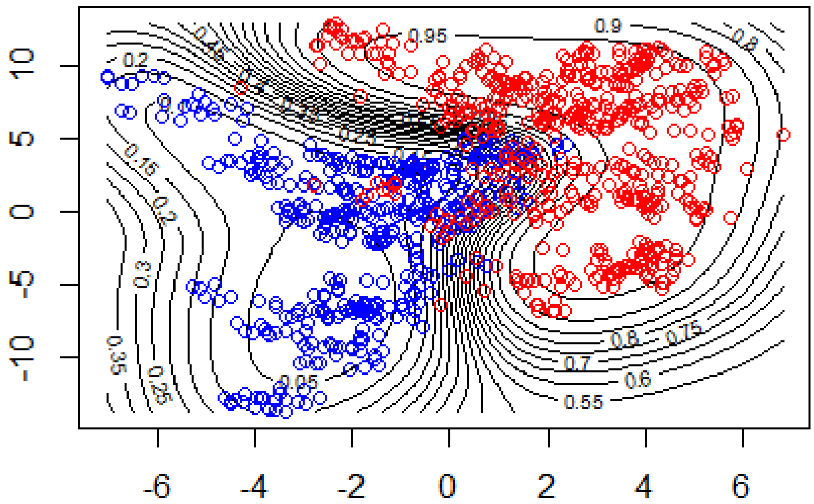
Aruup817@student.liu.se



Comparing all of the results from using the different kernels we see that they are fairly similar with the difference that the day kernel is periodically repeating itself each year while the other two kernles gradually get an increased lowest point.

```
####################### Assignment 3#####################
################## GP Classicatio with kernlab##############################

library(kernlab)
library(AtmRay)

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])

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

# Part 3.1
GP.fit = gausspr(fraud ~ varWave + skewWave, data = train)

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

x1 <- seq(min(train$varWave), max(train$varWave), length = 100)
x2 <- seq(min(train$skewWave), max(train$skewWave), 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(GP.fit, gridPoints, type = "probabilities")

contour(x1, x2, matrix(probPreds[, 1], 100, byrow = TRUE), 20)
points(train[train$fraud == 1, "varWave"], train[train$fraud == 1,
"skewWave"], col = "blue")
points(train[train$fraud == 0, "varWave"], train[train$fraud == 0,
"skewWave"], col = "red")
```



```r
### Part 3.2
# Compute the accuracy on predicting the test data
test_prediction = predict(GP.fit, test, type = "response")
accuracy = mean(test_prediction == test$fraud)
accuracy

## [1] 0.9247312

### Part 3.3
GP.fit_all = gausspr(fraud ~ ., data = train)

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

test_prediction_all = predict(GP.fit_all, test, type = "response")
accuracy_all = mean(test_prediction_all == test$fraud)
accuracy_all

## [1] 0.9946237
```