Advanced Machine Learning Lab 4

Mikael Montén

2024-10-23

Assignments

Assignment 1 - Implementing GP Regression

Gaussian process regression model: $y = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right)$ and $f \sim \mathcal{GP}\left(0, k\left(x, x'\right)\right)$

The task is to implement the following algorithm with your own code. It uses Cholesky chol() decomposition to attain numerical stability. Note that L in the algorithm is a lower triangular matrix, whereas the R function returns an upper triangular matrix. So, you need to transpose the output of the R function. In the algorithm, the notation $A \backslash b$ means the vector x that solves the equation Ax = b. This is implemented in R with solve().

```
input: $X$ (inputs), $\mathbf{y}$ (targets), $k$ (covariance function), $\sigma_n^2$ (noise level), $\mathbf{x}_*$ (test input)
$L := \text{cholesky}(K + \sigma_n^2 I)$
$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$                         } predictive mean eq. (2.25)
$\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$
$\mathbf{v} := L \backslash \mathbf{k}_*$                                    } predictive variance eq. (2.26)
$\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$                 eq. (2.30)
return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)
```

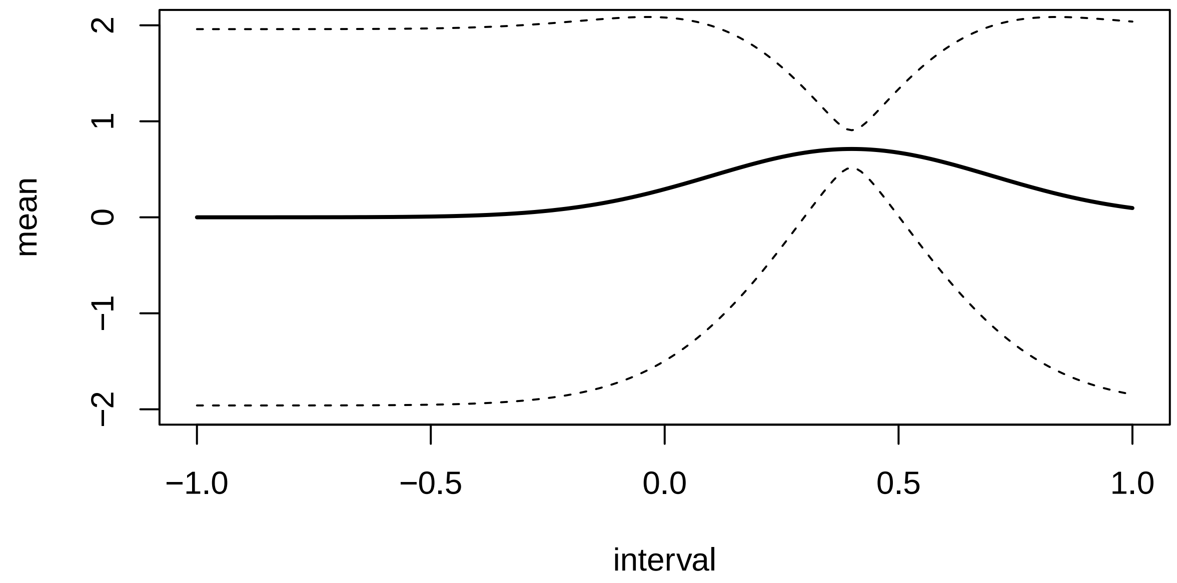
1. Simulate from the posterior distribution of f using the squared exponential kernel. The function should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values (X_*). You can assume that the prior mean of f is zero for all x. The function should have inputs $x, y, XStar, sigmaNoise, k$.

```r
library(mvtnorm)
library(kernlab)
library(scales)
library(AtmRay)

posteriorGP <- function(X,y,XStar,sigmaNoise,k,sigmaF,l){
  # X: vector of training inputs
  # y: vector of training targets/outputs
  # XStar: vector of inputs where the posterior distribution is evaluated
  # sigmaNoise: noise standard deviation
  # k: covariance function or kernel, separate function

  # this part is added to handle question 2.3 with my own kernel function
  # all the steps are the same except the kernelMatrix function to create the matrices
  if(class(k) == "kernel"){
    K <- kernelMatrix(k,X,X)
    L <- t(chol(K+sigmaNoise^2*diag(nrow(K))))

    # pred mean
    a <- solve(t(L), solve(L,y))
    pred_mean <- t(kernelMatrix(k,X,XStar)) %*% a

    # pred variance
    v <- solve(L,kernelMatrix(k,X,XStar))
    pred_var <- kernelMatrix(k,XStar,XStar) - t(v) %*% v
  }

  else{
    K <- k(X,X,sigmaF,l) # create K(X,X) of only training inputs
    L <- t(chol(K+sigmaNoise^2*diag(nrow(K))))

    # pred mean
    a <- solve(t(L), solve(L,y))
    pred_mean <- t(k(X,XStar,sigmaF,l)) %*% a

    # pred variance
    v <- solve(L,k(X,XStar,sigmaF,l))
    pred_var <- k(XStar,XStar,sigmaF,l) - t(v) %*% v
  }

  # log marginal likelihood
  n <- length(XStar)
  llik <- -(1/2)*t(y) %*% a - sum(diag(L)) - (n/2)*log(2*pi)

  return(list(pred_mean, pred_var, llik))
}
```

2. Now, let the prior hyperparameters be $\sigma_f = 1$ and $l = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that $\sigma_n = 0.1$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot the 95% probability (pointwise) bands for f.

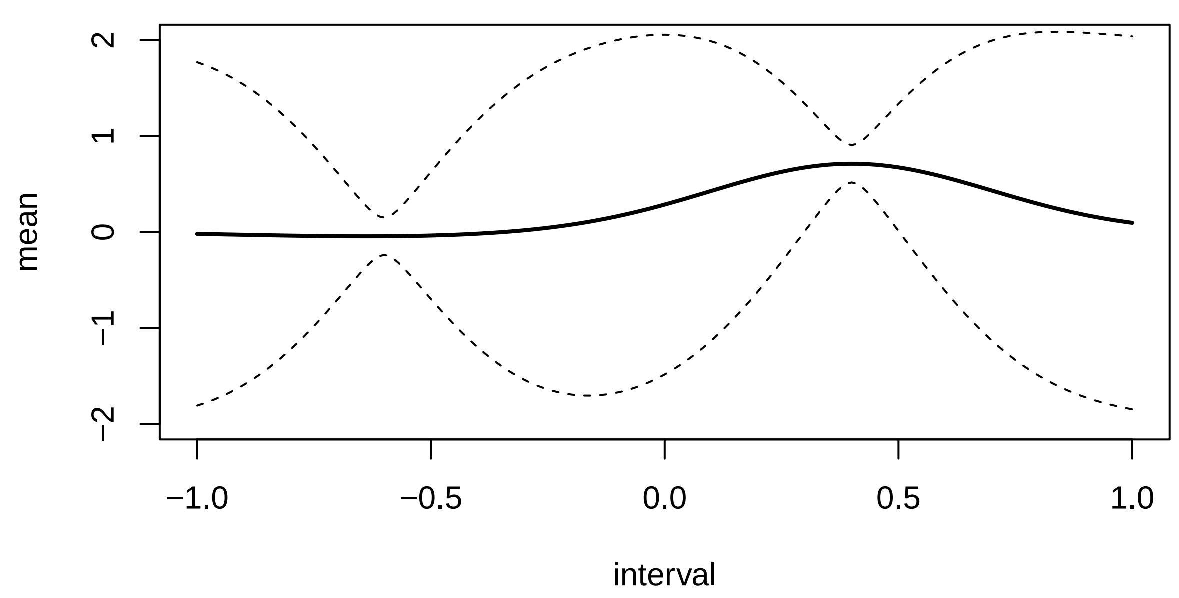
```r
sigma_f <- 1
l <- 0.3
x <- 0.4
y <- 0.719
sigma_n <- 0.1
x_seq <- seq(-1,1,0.01)

gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f
```

```r
plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type = "l", lwd = 2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed") # 95% probability bands
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
```



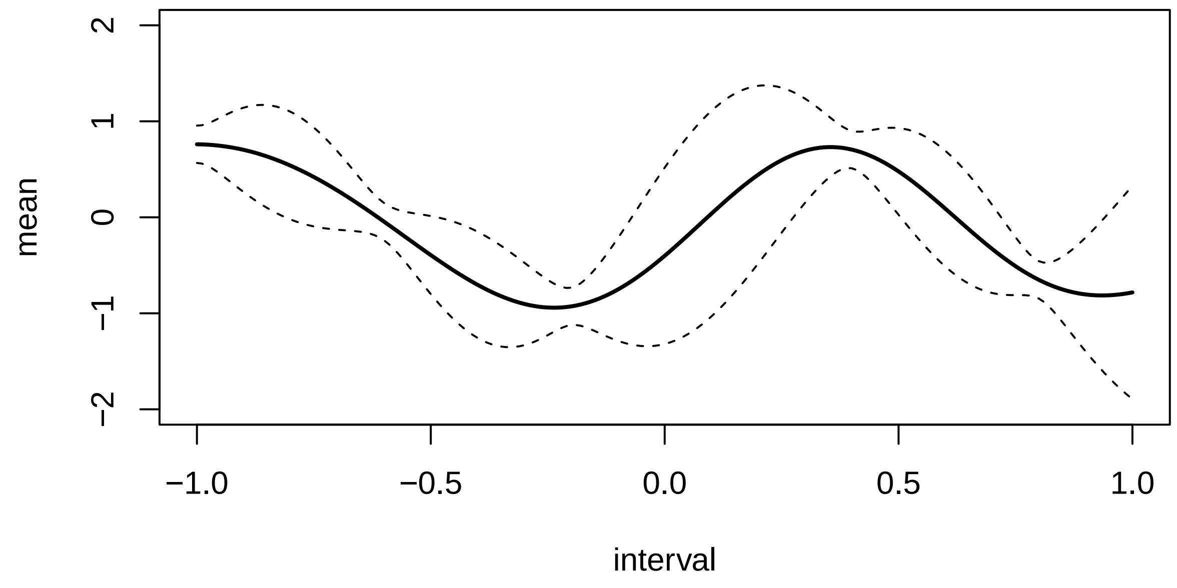
3. Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$. Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot the 95% probability (pointwise) bands for f.

```
x <- c(0.4,-0.6)
y <- c(0.719,-0.044)
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2),type="l", lwd=2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
```



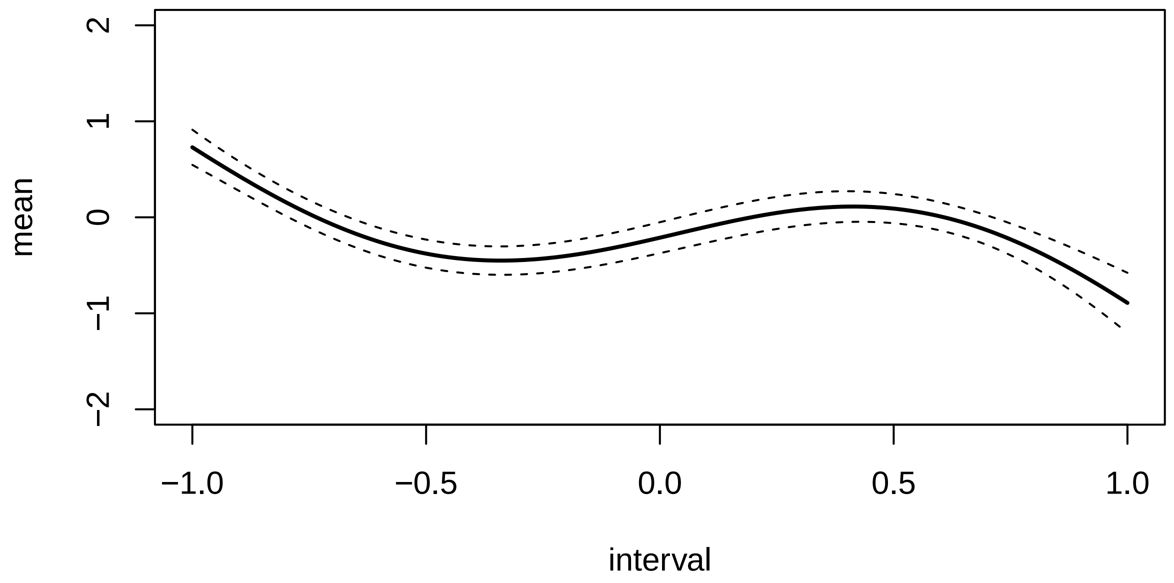
4. Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observatijons are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot the 95% probability (pointwise) bands for f.

```
x <- c(-1.0,-0.6,-0.2,0.4,0.8)
y <- c(0.768, -0.044, -0.940, 0.719,-0.664)
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type = "l", lwd = 2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
```



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

```
sigma_f <- 1
l <- 1
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type ="l", lwd =2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
```



* Compare the results.

Comparing the latest plots, the increase in l results in a smoother mean prediction that is more centered towards 0. The confidence interval is also more smooth as a result of the covariance function considering more future values for each current value, reducing the individual weight point.

The length scale l controls how quickly the correlation between function values decays with distance in the input space. Specifically, when l is larger at 1, the function values at different input points remain correlated over longer distances. This results in the seen smoother and more gradually varying functions with slower variations over the interval. When its lower, at 0.3, the correlation decays more rapidly with distance and capture more local variations in data.

Assignment 2 - GP Regression with kernlab

In this exercise you will work with daily mean temperatures in Stockholm during the period January 1 2010 - December 31 2015. We have removed the leap year day February 29, 2012 to make things simpler.

Create the variable time which records the day number since the start of the dataset (i.e., $time = 1, 2, ..., 365 \frown 6 = 2190$). 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 $day = 1, 6, 11, ..., 361, 1, 6, 11, ..., 361$.

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

# create time and day variables and sequence data to every 5th day
temp_data <- temp_data[seq.int(1,2190,5),]
temp_data$time <- seq.int(1,2190,5)
temp_data$day <- rep(seq.int(1,365,5),6)
```

1. Familiarize yourself with the functions gausspr and kernelMatrix in kernlab. Go through the file KernLabDemo.R. Now, define your own square exponential kernel function with parameters ℓ (ell) and σ_f $(sigmaf)$, evaluate it in the point $x = 1, x' = 2$ and use the kernelMatrix function to compute the covariance matrix $K(X, X_*)$ for the input vectors $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

```
# own square exponential kernel function
# created to create a function which to be called with input
sq_exp_kern <- function(ell=1,sigmaf=1){
  kern <- function(x,y=NULL){
    return(sigmaf^2*exp(-sum((x-y)^2)/(2*ell^2)))
  }
  class(kern) <- "kernel"
  return(kern)
}

# define x and x'
x <- 1
xp <- 2

# create kernel and evaluate at points
kernelFunc <- sq_exp_kern()
kernelFunc(x,xp)
```

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

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

# compute covariance matrix
kernelMatrix(kernelFunc, X, Xstar)
```

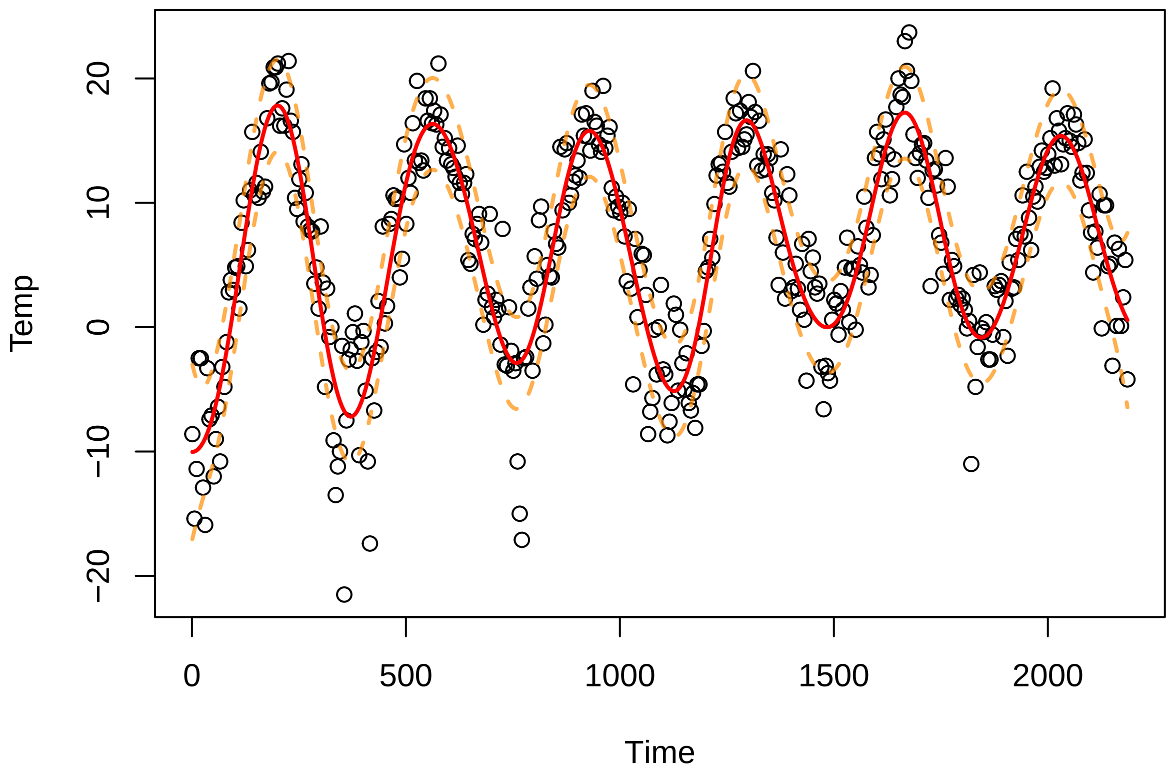
```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.6065307 0.1353353 0.0111090
## [2,] 0.6065307 1.0000000 0.6065307
## [3,] 0.1353353 0.6065307 1.0000000
```

2. Consider first the following Gaussian process regression model: $y = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right)$ and $f \sim$

$\mathcal{GP}\left(0, k\left(x, x'\right)\right)$

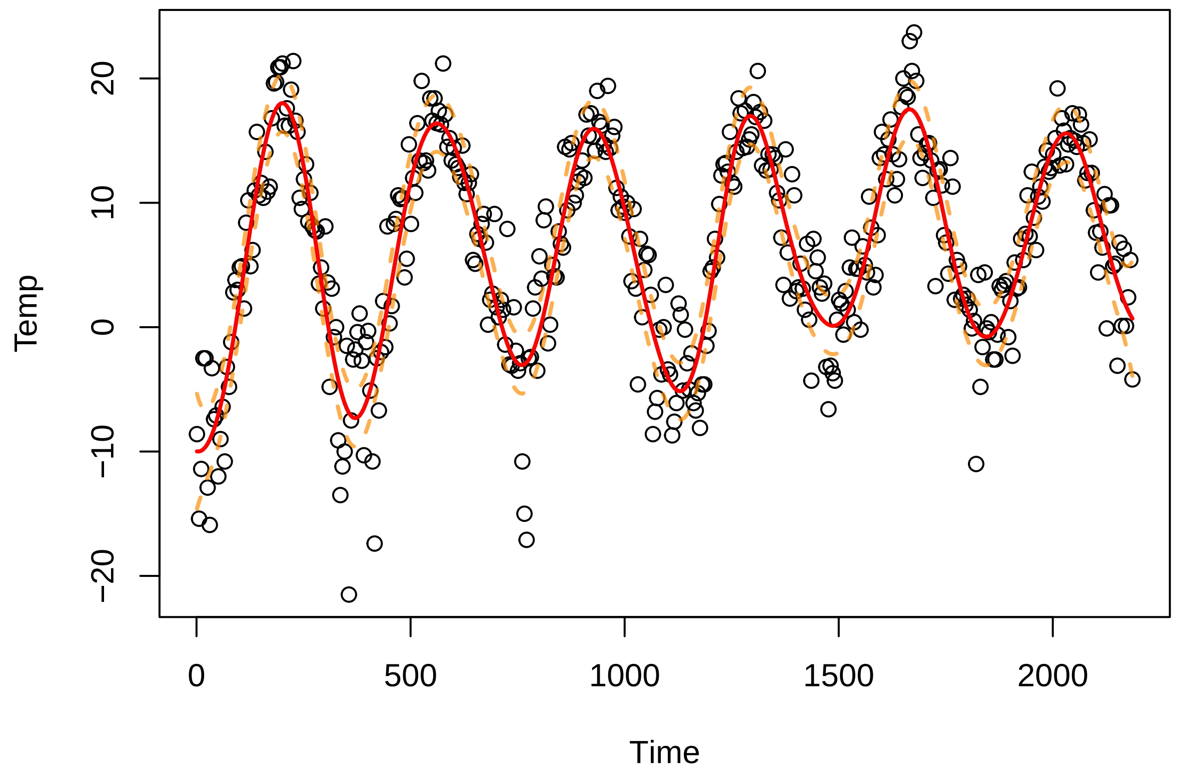
Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the lm function in R). Estimate the above Gaussian process regression model using the gausspr function with the squared exponential function from (1) with $\sigma_f = 20$ and $\ell = 20$ (use option scaled=FALSE in the gausspr function). Use the predict function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use type="l" in the plot function). Plot also the 95 % probability (pointwise) bands for f. Play around with different values on σ_f and ℓ.

```r
# quadratic regression fit
quad_reg <- lm(temp ~ time + I(time^2), data = temp_data)
sigma2n <- var(quad_reg$residuals) # residual variance
kernelFunc <- sq_exp_kern(ell = 100, sigmaf = 20) # kernelFunc with given ell and sigma

# gausspr didnt work properly with vectors inside data
Time <- temp_data$time
Temp <- temp_data$temp

# gp regression
pr_reg <- gausspr(x = Time, y = Temp, kernel = kernelFunc, var = sigma2n,
                  type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean <- predict(pr_reg, Time) # posterior means
post_sd <- predict(pr_reg, Time, type = "sdeviation") # posterior std devs

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,post_mean, type ="l", col = "red", lwd = 2)
lines(Time,post_mean+1.96*post_sd,col=alpha("darkorange",0.7),lwd=2,lty="dashed")
lines(Time,post_mean-1.96*post_sd,col=alpha("darkorange",0.7),lwd=2,lty="dashed")
```



3. Repeat the previous exercise, but now use Algorithm 2.1 again to compute the posterior mean and variance of f.

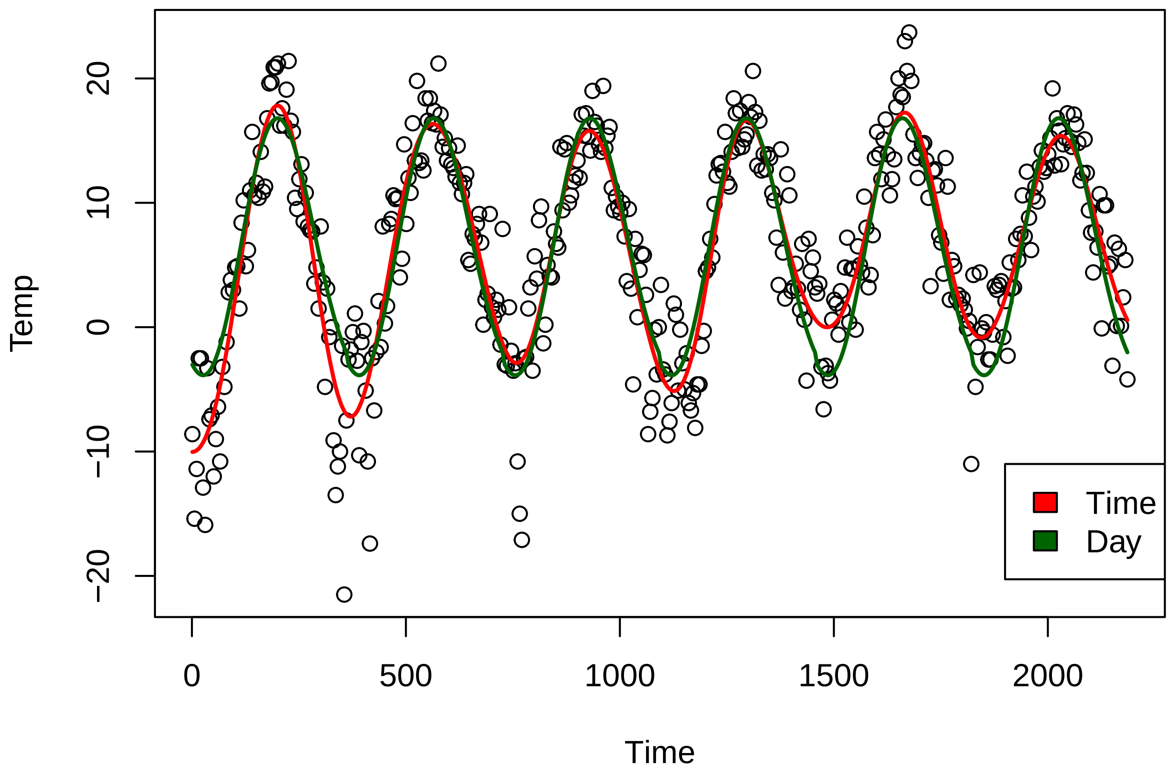
```
quad_reg_day <- lm(temp ~ day + I(day^2), data = temp_data)
sigma2n_day <- var(quad_reg_day$residuals) # residual variance
kernelFunc <- sq_exp_kern(ell = 100, sigmaf = 20) # kernelFunc with given ell and sigma

gp_func <- posteriorGP(Time, Temp, Time, sqrt(sigma2n_day), kernelFunc)
func_mean <- unlist(gp_func[1])
func_var <- diag(gp_func[[2]])


plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,func_mean, type ="l", col = "red", lwd = 2)
lines(Time,func_mean+1.96*sqrt(func_var),col=alpha("darkorange",0.7),lwd=2,lty="dashed")
lines(Time,func_mean-1.96*sqrt(func_var),col=alpha("darkorange",0.7),lwd=2,lty="dashed")
```



4. Consider now the following model, temp $= f($ day $)+\epsilon$ with $\epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right)$ and $f \sim \mathcal{GP}(0, k($ day, day' $))$

Estimate the model using the gausspr function with the squared exponential function from (1) with $\sigma_f = 20$ and $l = 100$ (use the option scaled=FALSE in the gausspr function). 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?

```r
Day <- temp_data$day
# gp regression
pr_reg_day <- gausspr(x = Day, y = Temp, kernel = kernelFunc, var = sigma2n,
                  type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean_day <- predict(pr_reg_day, Day) # posterior means

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,post_mean, type ="l", col = "red", lwd = 2)
lines(Time,post_mean_day, type ="l", col = "darkgreen", lwd = 2)
legend(1900,-11,legend=c("Time","Day"),fill=c("red","darkgreen"))
```



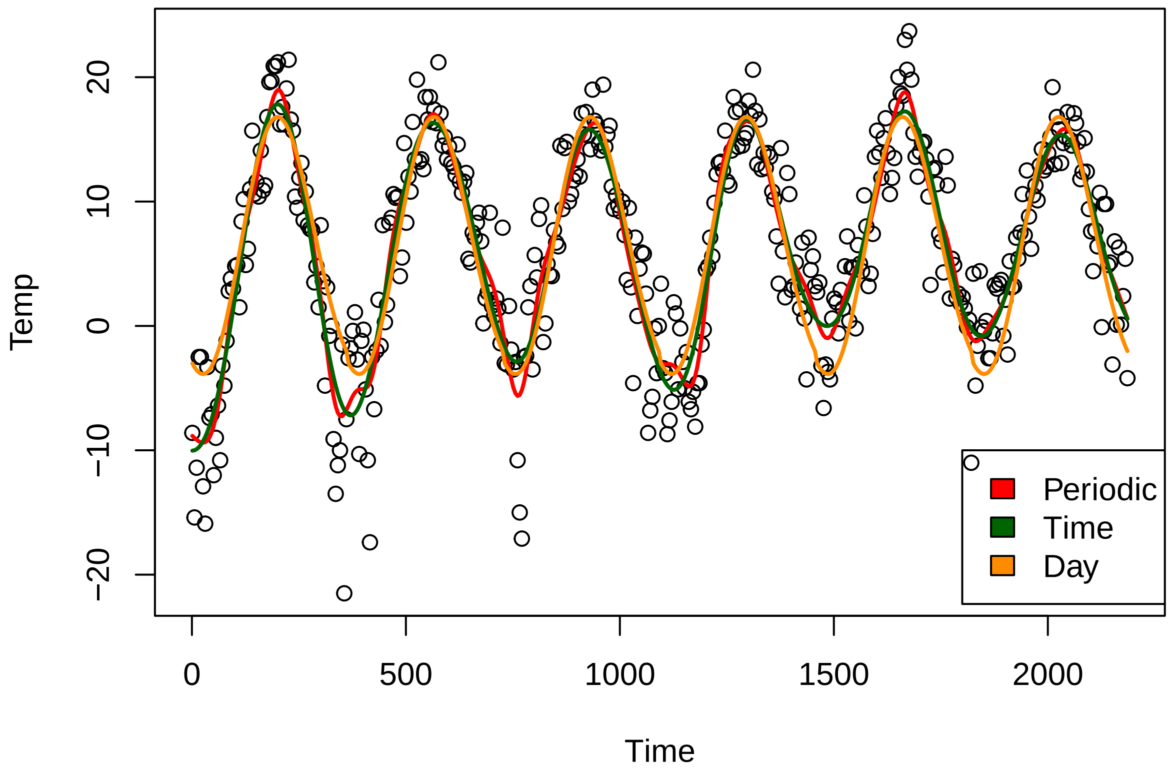
The estimates for both models are very similar and follow each other closely for most parts. The difference is in how the models handle repeating data; the *Time* model uses a continuous iterating value thus only see each value once and the *Day* model see each value 5 times. The day model automatically averages information across years for each calendar day as all dates across the years are treated as the same x-value. This is more "stable" since each prediction is based on multiple observations from the same day. Instead, for the time model, each observations is treated as occurring at a unique time point which means it has to learn the seasonal pattern purely from the correlations (and by extensions seasonal patterns through the covariance function) in data rather than explicit seasonal dependency. This results in more variability in the predictions due to temporally local observations influencing the results more.

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

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

Note that we have two different length scales in the kernel. Intuitively, l_1 controls the correlation between two days in the same year, and l_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20, l_1 = 1, l_2 = 100, d = 365$. Use the gausspr function with the option scaled=FALSE. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 100$). Discuss the results.

```r
# function for local periodic kernel
loc_per_kern <- function(sigmaf, ell1, ell2, d){
  kern <- function(x,y=NULL){
    return(sigmaf^2*exp(-(2*sin(pi*abs(x-y)/d)^2)/ell1^2)*exp(-(1/2)*abs(x-y)^2/ell2^2))
  }
  class(kern) <- "kernel"
  return(kern)
}

# kernel evaluated at the values given
lockernelFunc <- loc_per_kern(20,1,100,365)

loc_reg <- gausspr(x = Time, y = Temp, kernel = lockernelFunc, var = sigma2n,
                   type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean_loc <- predict(loc_reg, Time) # posterior means

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p",)
lines(Time,post_mean_loc, type ="l", col = "red", lwd = 2)
lines(Time,post_mean, type ="l", col = "darkgreen", lwd = 2)
lines(Time,post_mean_day, type ="l", col = "darkorange", lwd = 2)
legend(1800,-10,legend=c("Periodic","Time","Day"),fill=c("red","darkgreen","darkorange"))
```



Comparing the different fits shows that the periodic model is less smooth, i.e. giving more weight to observations deviating from the mean prediction. This can be seen at the changes of directions, especially those with high density of points outside, where the periodic table has the highest maximum and lowest minimum. This is due to the periodicity explicitly encoded in the kernel, where it adapts its prediction more to how the period looks. This makes it particularly well-suited for data with known periodic components like for this data.

The combination of the squared exponential kernel and the local periodic kernel enables the modeling of both periodic patterns and overall trends in the data where the kernels excel at different aspects of modeling. The use of two different length scales in the periodic kernel, one to control correlation between days within year and one to control correlation between days between years provides good flexibility for the model to both capture short-term and long-term patterns in data.

Assignment 3 - GP Classification with kernlab

Download the banknote fraud data and subset 1000 observations as training data using SelectTraining.

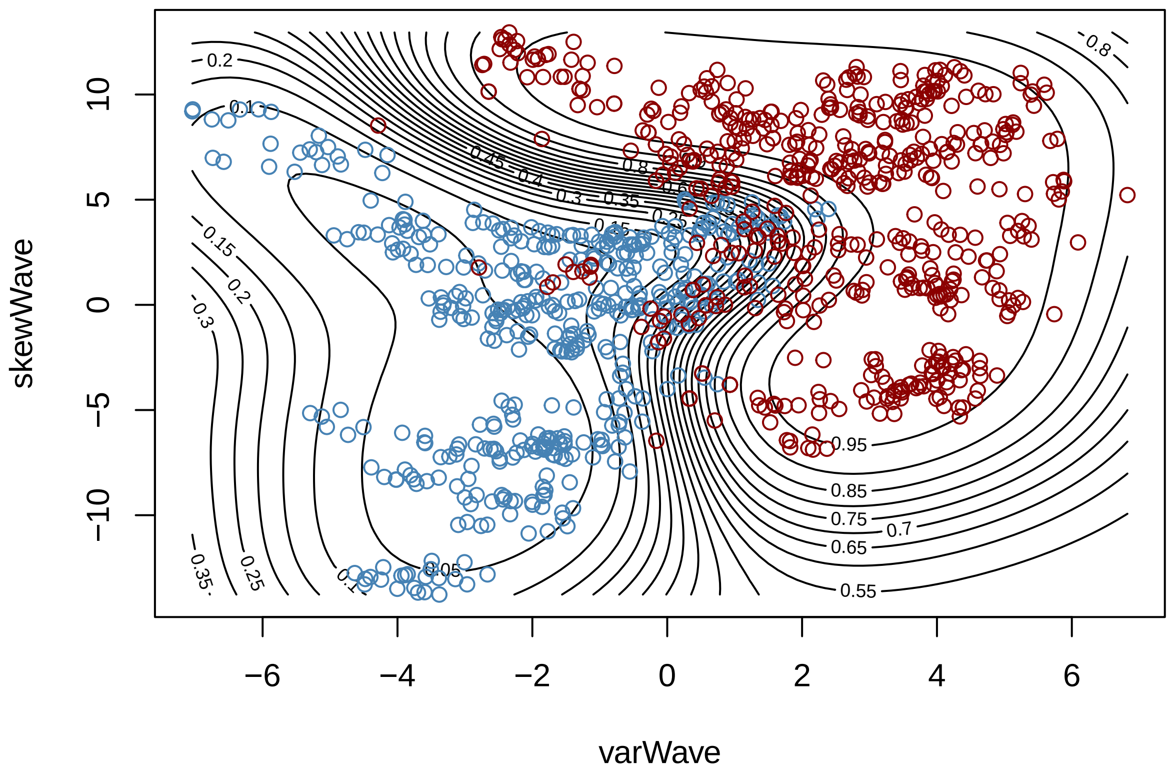
```r
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])

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

train <- data[SelectTraining,]
test <- data[-SelectTraining,]
```

1. Use the R package kernlab to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file KernLabDemo.R available on the course website. Compute the confusion matrix for the classifier and its accuracy.

```r
GPfitFraud <- gausspr(fraud ~ varWave + skewWave, data = data) # model
```

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

```r
class_pred <- predict(GPfitFraud,train)

# suitable grids for x vars
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) <- c("varWave","skewWave")

# predict class probabilities on the grid training points
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")

# contour plot of points and decision boundaries
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave")
points(train[train$fraud==1,1], train[train$fraud==1,2],col="steelblue")
points(train[train$fraud==0,1], train[train$fraud==0,2],col="darkred")
```



```r
conf <- table(class_pred,train$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
```

```
##
## class_pred   0   1
##          0 504  22
##          1  40 434
```

```r
acc
```

```
## [1] 0.938
```

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

```r
test_pred <- predict(GPfitFraud,test)
conf <- table(test_pred,test$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
```

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

```
acc
```

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

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

```
GPfitFraud <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data = data) # model
```

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

```
test_pred <- predict(GPfitFraud,test)
conf <- table(test_pred,test$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
```

```
##
## test_pred   0   1
##         0 216   0
##         1   2 154
```

```
acc
```

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

The test accuracy improves from 92.4% to 99.5%. The addition of two covariates enables the classifier to better classify data, and probably in particular the data points that are close to the class boundaries in the grid plot.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE,fig.height = 4,set.seed(1234567890))
library(mvtnorm)
library(kernlab)
library(scales)
library(AtmRay)
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)
}
library(mvtnorm)
library(kernlab)
library(scales)
library(AtmRay)

posteriorGP <- function(X,y,XStar,sigmaNoise,k,sigmaF,l){
  # X: vector of training inputs
  # y: vector of training targets/outputs
  # XStar: vector of inputs where the posterior distribution is evaluated
  # sigmaNoise: noise standard deviation
  # k: covariance function or kernel, separate function

  # this part is added to handle question 2.3 with my own kernel function
  # all the steps are the same except the kernelMatrix function to create the matrices
  if(class(k) == "kernel"){
    K <- kernelMatrix(k,X,X)
    L <- t(chol(K+sigmaNoise^2*diag(nrow(K))))

    # pred mean
    a <- solve(t(L), solve(L,y))
    pred_mean <- t(kernelMatrix(k,X,XStar)) %*% a

    # pred variance
    v <- solve(L,kernelMatrix(k,X,XStar))
    pred_var <- kernelMatrix(k,XStar,XStar) - t(v) %*% v
  }

  else{
    K <- k(X,X,sigmaF,l) # create K(X,X) of only training inputs
    L <- t(chol(K+sigmaNoise^2*diag(nrow(K))))

    # pred mean
    a <- solve(t(L), solve(L,y))
    pred_mean <- t(k(X,XStar,sigmaF,l)) %*% a

    # pred variance
    v <- solve(L,k(X,XStar,sigmaF,l))
    pred_var <- k(XStar,XStar,sigmaF,l) - t(v) %*% v
  }

  # log marginal likelihood
  n <- length(XStar)
  llik <- -(1/2)*t(y) %*% a - sum(diag(L)) - (n/2)*log(2*pi)

  return(list(pred_mean, pred_var, llik))
}
sigma_f <- 1
l <- 0.3
x <- 0.4
y <- 0.719
sigma_n <- 0.1
x_seq <- seq(-1,1,0.01)

gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f
plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type = "l", lwd = 2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed") # 95% probability bands
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
x <- c(0.4,-0.6)
y <- c(0.719,-0.044)
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2),type="l", lwd=2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
x <- c(-1.0,-0.6,-0.2,0.4,0.8)
y <- c(0.768, -0.044, -0.940, 0.719,-0.664)
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type = "l", lwd = 2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
sigma_f <- 1
l <- 1
gp <- posteriorGP(x,y,x_seq,sigma_n,SquaredExpKernel,sigma_f,l)
meanx <- unlist(gp[1]) # mean of f
varx <- diag(gp[[2]]) # var of f

plot(x_seq,meanx,ylab="mean",xlab="interval",ylim=c(-2,2), type ="l", lwd =2)
lines(x_seq,meanx+1.96*sqrt(varx), lty = "dashed")
lines(x_seq,meanx-1.96*sqrt(varx), lty = "dashed")
link <- "https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.csv"
temp_data <- read.csv(link, header=TRUE, sep=";")

# create time and day variables and sequence data to every 5th day
```

```r
temp_data <- temp_data[seq.int(1,2190,5),]
temp_data$time <- seq.int(1,2190,5)
temp_data$day <- rep(seq.int(1,365,5),6)
# own square exponential kernel function
# created to create a function which to be called with input
sq_exp_kern <- function(ell=1,sigmaf=1){
  kern <- function(x,y=NULL){
    return(sigmaf^2*exp(-sum((x-y)^2)/(2*ell^2)))
  }
  class(kern) <- "kernel"
  return(kern)
}

# define x and x'
x <- 1
xp <- 2

# create kernel and evaluate at points
kernelFunc <- sq_exp_kern()
kernelFunc(x,xp)

# input vectors
X <- c(1,3,4)
Xstar <- c(2,3,4)

# compute covariance matrix
kernelMatrix(kernelFunc, X, Xstar)
# quadratic regression fit
quad_reg <- lm(temp ~ time + I(time^2), data = temp_data)
sigma2n <- var(quad_reg$residuals) # residual variance
kernelFunc <- sq_exp_kern(ell = 100, sigmaf = 20) # kernelFunc with given ell and sigma

# gausspr didnt work properly with vectors inside data
Time <- temp_data$time
Temp <- temp_data$temp

# gp regression
pr_reg <- gausspr(x = Time, y = Temp, kernel = kernelFunc, var = sigma2n,
                  type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean <- predict(pr_reg, Time) # posterior means
post_sd <- predict(pr_reg, Time, type = "sdeviation") # posterior std devs

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,post_mean, type ="l", col = "red", lwd = 2)
lines(Time,post_mean+1.96*post_sd,col=alpha("darkorange",0.7),lwd=2,lty="dashed")
lines(Time,post_mean-1.96*post_sd,col=alpha("darkorange",0.7),lwd=2,lty="dashed")
quad_reg_day <- lm(temp ~ day + I(day^2), data = temp_data)
sigma2n_day <- var(quad_reg_day$residuals) # residual variance
kernelFunc <- sq_exp_kern(ell = 100, sigmaf = 20) # kernelFunc with given ell and sigma

gp_func <- posteriorGP(Time, Temp, Time, sqrt(sigma2n_day), kernelFunc)
```

```r
func_mean <- unlist(gp_func[1])
func_var <- diag(gp_func[[2]])


plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,func_mean, type ="l", col = "red", lwd = 2)
lines(Time,func_mean+1.96*sqrt(func_var),col=alpha("darkorange",0.7),lwd=2,lty="dashed")
lines(Time,func_mean-1.96*sqrt(func_var),col=alpha("darkorange",0.7),lwd=2,lty="dashed")
Day <- temp_data$day
# gp regression
pr_reg_day <- gausspr(x = Day, y = Temp, kernel = kernelFunc, var = sigma2n,
                  type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean_day <- predict(pr_reg_day, Day) # posterior means

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p", ylim = c(-21.5,23.7))
lines(Time,post_mean, type ="l", col = "red", lwd = 2)
lines(Time,post_mean_day, type ="l", col = "darkgreen", lwd = 2)
legend(1900,-11,legend=c("Time","Day"),fill=c("red","darkgreen"))
# function for local periodic kernel
loc_per_kern <- function(sigmaf, ell1, ell2, d){
  kern <- function(x,y=NULL){
    return(sigmaf^2*exp(-(2*sin(pi*abs(x-y)/d)^2)/ell1^2)*exp(-(1/2)*abs(x-y)^2/ell2^2))
  }
  class(kern) <- "kernel"
  return(kern)
}

# kernel evaluated at the values given
lockernelFunc <- loc_per_kern(20,1,100,365)

loc_reg <- gausspr(x = Time, y = Temp, kernel = lockernelFunc, var = sigma2n,
                   type = "regression", scaled = FALSE, variance.model = TRUE)

post_mean_loc <- predict(loc_reg, Time) # posterior means

# plot of data, superimposed means and 95% prob bands
plot(Time,Temp, type = "p",)
lines(Time,post_mean_loc, type ="l", col = "red", lwd = 2)
lines(Time,post_mean, type ="l", col = "darkgreen", lwd = 2)
lines(Time,post_mean_day, type ="l", col = "darkorange", lwd = 2)
legend(1800,-10,legend=c("Periodic","Time","Day"),fill=c("red","darkgreen","darkorange"))
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])

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

train <- data[SelectTraining,]
test <- data[-SelectTraining,]
GPfitFraud <- gausspr(fraud ~ varWave + skewWave, data = data) # model
```

```r
class_pred <- predict(GPfitFraud,train)

# suitable grids for x vars
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) <- c("varWave","skewWave")

# predict class probabilities on the grid training points
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")

# contour plot of points and decision boundaries
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave")
points(train[train$fraud==1,1], train[train$fraud==1,2],col="steelblue")
points(train[train$fraud==0,1], train[train$fraud==0,2],col="darkred")

conf <- table(class_pred,train$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
acc

test_pred <- predict(GPfitFraud,test)
conf <- table(test_pred,test$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
acc
GPfitFraud <- gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data = data) # model
test_pred <- predict(GPfitFraud,test)
conf <- table(test_pred,test$fraud) # confusion matrix
acc <- sum(diag(conf))/sum(conf) # accuracy

conf
acc
```