TDDE15-Lab 4

frera064, oscho091, hjaoh082, olosw720

Contribution

All four members solved the problems on their own. We then discussed the answers and compiled the group report by combining our solutions. We used olosw720 for task 1. hjaoh082 for task 2, and oscho091 for task 3. Frera064 contributed by suggesting interesting insights and valuable reflections.

2.1 Implementing GP Regression

This first exercise will have you writing your own code for the Gaussian process regression model:

$$y = f(x) + \epsilon \quad \text{with} \quad \epsilon \sim N(0, \sigma_n^2) \quad \text{and} \quad f \sim GP(0, k(x, x'))$$

You must implement Algorithm 2.1 on page 19 of Rasmussen and Williams' book. The algorithm uses the Cholesky decomposition (`chol` in `R`) to attain numerical stability. Note that $\mathbf{L}$ 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 $\mathbf{A}/\mathbf{b}$ means the vector $\mathbf{x}$ that solves the equation $\mathbf{Ax} = \mathbf{b}$ (see p. xvii in the book). This is implemented in R with the help of the function `solve`.

Here is what you need to do:

Task (1)

Write your own code for simulating from the posterior distribution of $\mathbf{f}$ using the squared exponential kernel. The function (name it `posteriorGP`) should return a vector with the posterior mean and variance of $\mathbf{f}$, both evaluated at a set of x-values ($\mathbf{X}^*$). You can assume that the prior mean of $\mathbf{f}$ is zero for all $\mathbf{x}$. The function should have the following inputs:

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

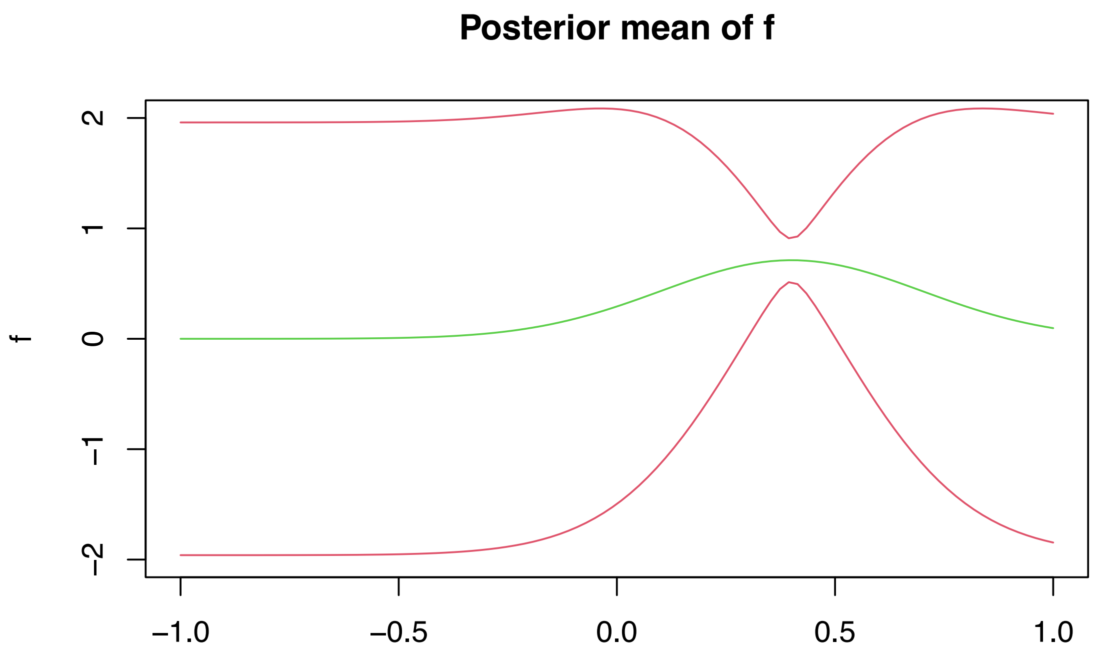
```r
posteriorGP = function(X, y, sigmaNoise, XStar, k, ...) {

  # Line 2
  n = length(X) # No of training points
  K = k(X,X,...)    # Covariance for training points
  kStar = k(X,XStar,...) # Covariance for training and test points
  # Cholesky decomposition, Lower triangular matrix
  L = t(chol(K + sigmaNoise**2 * diag(n)))
  alpha = solve(t(L), solve(L, y))

  # Line 4
  fStar = t(kStar)%*%alpha #posterior mean
  v = solve(L, kStar)

  # Line 6 :  Posterior variance
  V_fStar = k(XStar, XStar,...) - t(v)%*%v
  log_marg_likelihood = -(1/2)*t(y)%*%alpha - sum(log(diag(L))) - (n/2)*log(2*pi)

  return(list(mean = fStar, variance = V_fStar, log_likelihood = log_marg_likelihood))
}
```

Task (2)

Let the prior hyperparameters be $\sigma_f = 1$ and $\ell = 0.3$. 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 $x \in [-1, 1]$ and also plot the 95% probability (pointwise) bands for f.

We will use the squared exponential kernel function provided in the `GaussianProcesses.R` file.

```r
########################################
###### Code from Lab instructions ######
######### GaussianProcesses.R ##########
########################################

library("mvtnorm")

# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,ell=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])/ell)^2 )
  }
  return(K)
}

########################################
########################################

# Initialize paramters
sigmaF = 1
ell = 0.3
x = 0.4
y = 0.719
sigmaN = 0.1

xGrid = seq(-1,1,length = 100)

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN, XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)

plot(x = xGrid, y = posterior$mean, type = "l", col = 3, ylim = c(-2,2), ylab = "f", xlab = "", main =
lines(x = xGrid, y =posterior$mean +1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
lines(x = xGrid, y =posterior$mean -1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
```

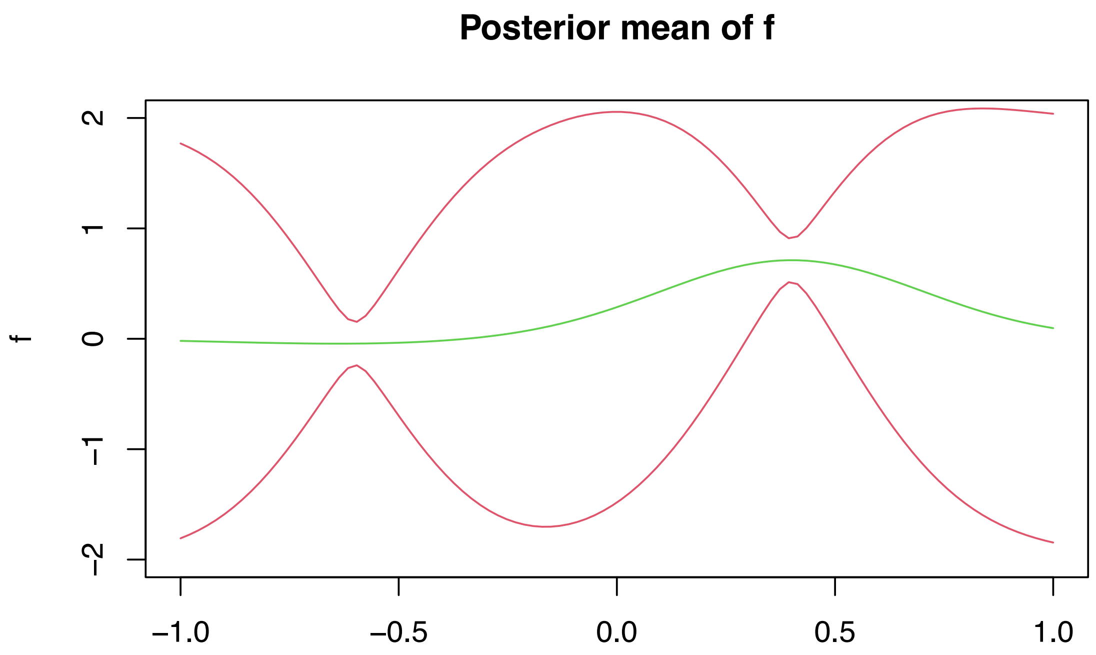


Task (3)

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

Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations.

```r
# Initialize paramters
x = c(0.4, -0.6)
y = c(0.719, -0.044)

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN, XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)

plot(x = xGrid, y = posterior$mean, type = "l", col = 3, ylim = c(-2,2), ylab = "f", xlab = "", main =
lines(x = xGrid, y =posterior$mean +1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
lines(x = xGrid, y =posterior$mean -1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
```

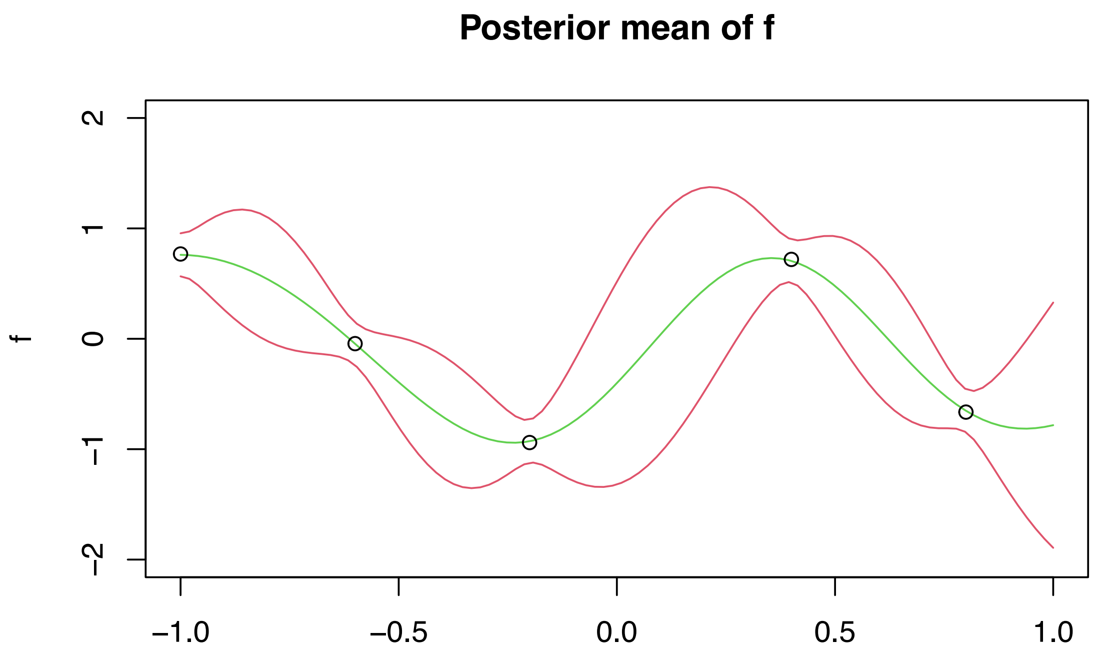


Task (4)

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

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

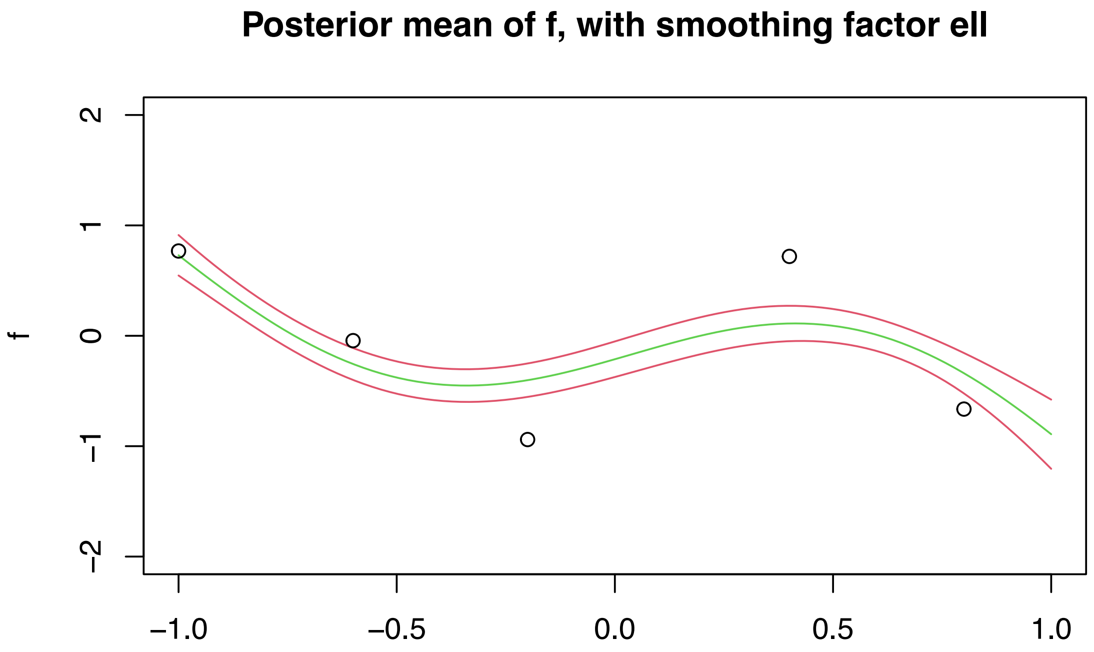
```
# Initialize paramters
x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.94, 0.719, -0.664)

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN, XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)

plot(x = xGrid, y = posterior$mean, type = "l", col = 3, ylim = c(-2,2), ylab = "f", xlab = "", main = 
lines(x = xGrid, y =posterior$mean +1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
lines(x = xGrid, y =posterior$mean -1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
points(x,y)
```



Task (5)

Repeat the previous step, but this time with hyperparameters $\sigma_f = 1$ and $\ell = 1$. Compare the results.

```
# Initialize paramters
sigmaF = 1
ell = 1

posterior = posteriorGP(X=x, y=y, sigmaNoise=sigmaN, XStar=xGrid, k = SquaredExpKernel, sigmaF, ell)

plot(x = xGrid, y = posterior$mean, type = "l", col = 3, ylim = c(-2,2), ylab = "f", xlab = "", main =
lines(x = xGrid, y =posterior$mean +1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
lines(x = xGrid, y =posterior$mean -1.96*sqrt(diag(posterior$variance)), type = "l", col = 2)
points(x,y)
```



As visualized, the smoothing factor ell makes the postieror underfit to the data, because it cant adjust enough after the first datapoint due to the generalizing prior.

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. The leap day (February 29, 2012) has been removed to simplify the dataset.

Create the variable **time** which records the day number since the start of the dataset (i.e., time $= 1, 2, \ldots, 365 \times 6 = 2190$). Also, create the variable **day** that records the day number since the start of each year (i.e., day $= 1, 2, \ldots, 365, 1, 2, \ldots, 365$). 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 $= 1, 6, 11, \ldots, 2186$ and day $= 1, 6, 11, \ldots, 361, 1, 6, 11, \ldots, 361$.

```r
rm(list = ls())
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c

data$time = 1:nrow(data)
data$day = ((data$time - 1) %% 365) + 1
data = data[seq(1, nrow(data), by=5), ]
```

2.1

Go through the file *KernLabDemo.R* available on the course website. You will need to understand it. Now, define your own square exponential kernel function (with parameters ℓ (ell) and 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 $X^* = (2, 3, 4)^T$.

```r
library(kernlab)

SquareExponential <- function(sigmaf, ell)
{
  rval <- function(x, y = NULL) {
    n1 <- length(x)
    n2 <- length(y)
    k <- matrix(NA,n1,n2)
    for (i in 1:n2){
      k[,i] <- sigmaf^2*exp(-0.5*( (x-y[i])/ell)^2 )
    }
    return(k)
  }
  class(rval) <- "kernel"
  return(rval)
}

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

SEkernel = SquareExponential(sigmaf = 1, ell = 1) # SEkernel is a kernel FUNCTION.
SEkernel(1, 2) # Evaluating the kernel in x=1, x'=2.
```

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

```r
# Computing the whole covariance matrix K from the kernel.
kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(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
```

```r
# kernelMatrix(kernel = SquareExponential(1,2), x = X, y = Xstar) # Also works.
```

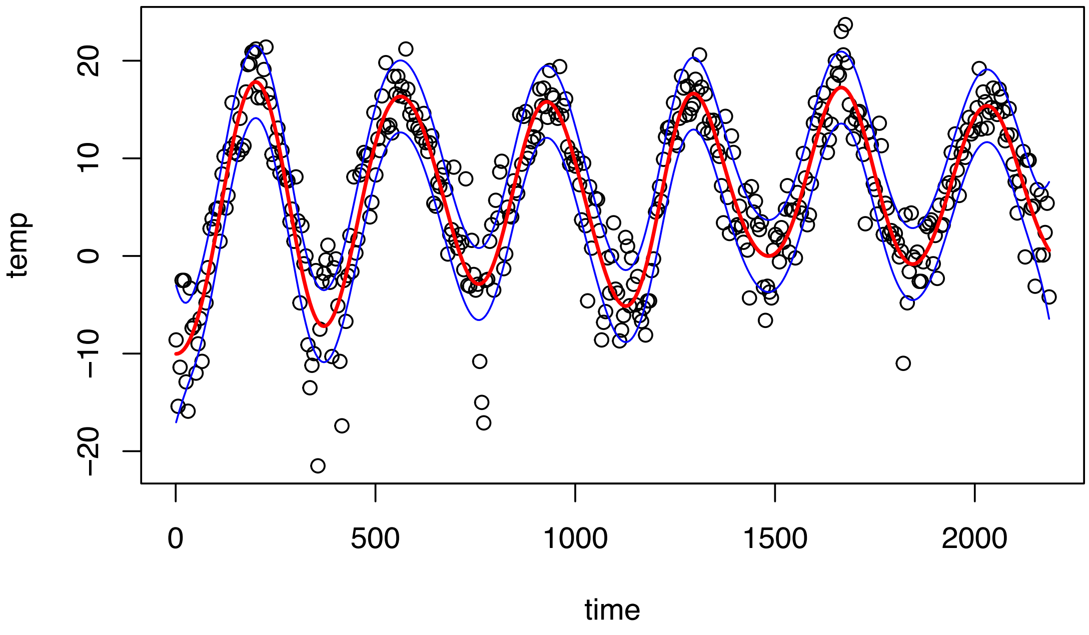
2.2 Gaussian Process Regression Model

Consider first the following model:

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

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 $\ell = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these σ_f and ℓ values are not suitable). Use the `predict` function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use `type="l"` in the `plot` function). Plot also the 95% probability (pointwise) bands for f. Play around with different values on σ_f and ℓ (no need to write this in the report though).

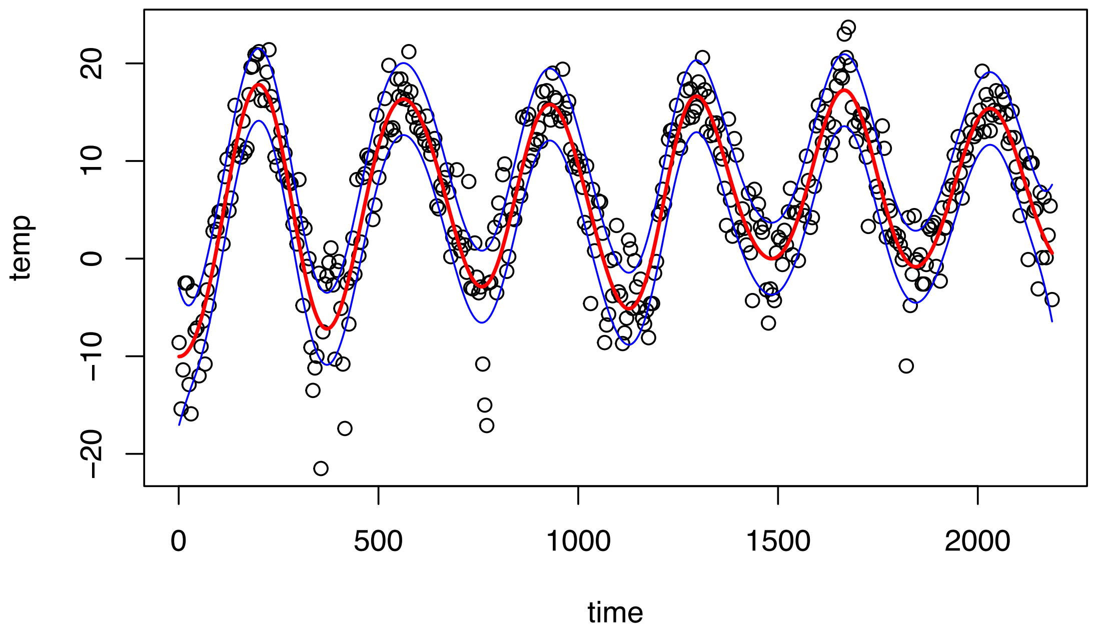
```r
temp = data$temp
time = data$time

polyFit = lm(temp~I(time) + I(time**2))
sigmaNoise = sd(polyFit$residuals)
plot(time, temp)

GPfit = gausspr(time, temp, kernel = SquareExponential(ell = 100, sigmaf = 20), var = sigmaNoise^2, var

time_meanPred <- predict(GPfit, time)
lines(time, time_meanPred, col="red", lwd = 2)
lines(time, time_meanPred+1.96*predict(GPfit,time, type="sdeviation"),col="blue")
lines(time, time_meanPred-1.96*predict(GPfit,time, type="sdeviation"),col="blue")
```



2.3 Algorithm 2.1

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

```
posteriorGP = function(x, y, XStar, sigmaNoise, k, ...){
  K = k(x, x, ...)
  L = t(chol(K + diag(sigmaNoise**2, length(x))))
  alpha = solve(t(L), solve(L,y))

  kStar = k(x, XStar, ...)
  predictive_mean = t(kStar)%*%alpha

  v = solve(L, kStar)
  predictive_variance = k(XStar, XStar, ...) - t(v)%*%v

  return(list(mean = predictive_mean, variance = predictive_variance))
}

predictions = posteriorGP(x = time, y = temp, XStar = time, sigmaNoise = sigmaNoise, k = SquareExponent
plot(time, temp)
lines(time, predictions$mean, col="red", lwd = 2)
lines(time, predictions$mean+1.96*sqrt(diag(predictions$variance)),col="blue")
lines(time, predictions$mean-1.96*sqrt(diag(predictions$variance)),col="blue")
```



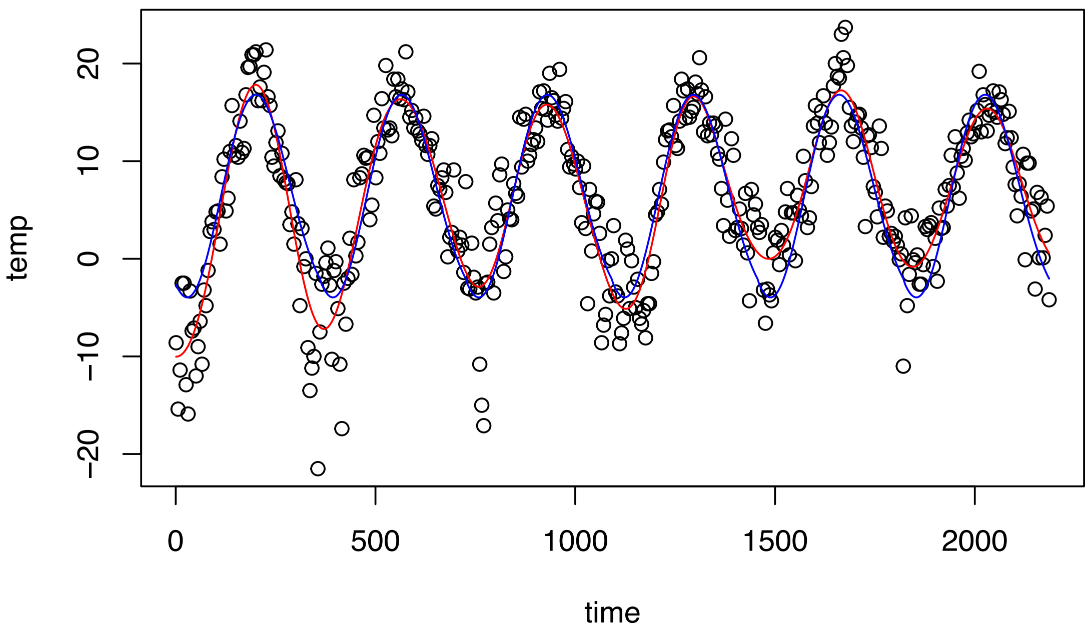
2.4 New Model Based on Day

Consider now the following model:

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

Estimate the model using the `gausspr` function with the squared exponential function from (1) with $\sigma_f = 20$ and $\ell = 100$ (use the option `scaled=FALSE` in the `gausspr` function, otherwise these σ_f and ℓ values are not suitable). Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis.

```
temp = data$temp
day = data$day

polyFit = lm(temp~I(day) + I(day**2))
sigmaNoise = sd(polyFit$residuals)

GPfit = gausspr(day, temp, kernel = SquareExponential(ell = 100, sigmaf = 20), var = sigmaNoise^2, vari

day_meanPred <- predict(GPfit, day)
plot(time, temp)
lines(time, time_meanPred, col = "red")
lines(time, day_meanPred, col = "blue")
```



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

A: The model with time as its input feature seem to have slight more variance than the one with day. Day values repeats itself, meaning that the 1st of January 2010 is the same value as 1st of January 2011. The model therefore gets multiple temperatures to train on for the same day value, while for the model with time, 1st of Jan each year is distinct.

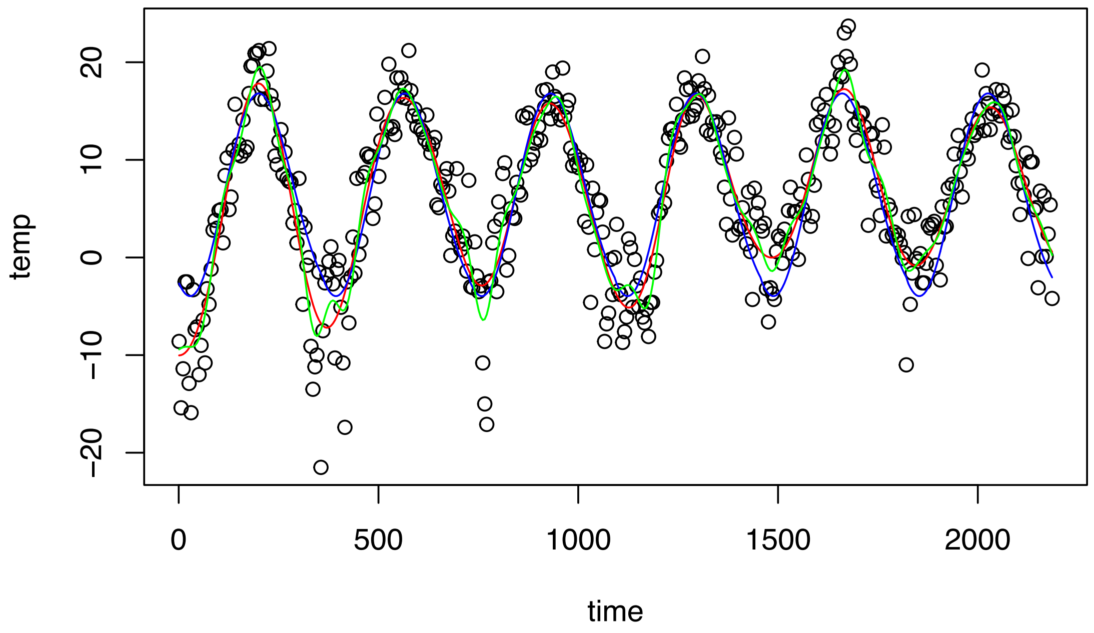
2.5 Extended Squared Exponential Kernel

Implement the following periodic kernel (a.k.a. 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 $\sigma_f = 20$, $\ell_1 = 1$, $\ell_2 = 100$, and $d = 365$. Use the `gausspr` function with the option `scaled=FALSE`, otherwise these σ_f, ℓ_1, and ℓ_2 values are not suitable.

```r
periodic <- function(sigmaf, ell1, ell2, d)
{
  rval <- function(x, y = NULL) {
    factor1 = sigmaf^2*exp(-0.5*( (x-y)/ell2)^2 )
    factor2 = exp((-2*sin(pi*abs(x-y)/d)^2)/(ell1^2))
    return(factor1*factor2)
  }
  class(rval) <- "kernel"
  return(rval)
}

GPfit <- gausspr(time, temp, kernel = periodic(sigmaf=20, ell1=1, ell2=100, d=365), var = sigmaNoise^2,

periodic_meanPred = predict(GPfit, time)

plot(time, temp)
lines(time, time_meanPred, col = "red")
lines(time, day_meanPred, col = "blue")
lines(time, periodic_meanPred, col = "green")
```



Q: Compare the fit to the previous two models (with $\sigma_f = 20$ and $\ell = 100$). Discuss the results. A: The new model with a periodic kernel allows to capture periodic patterns, creating a model with higher variance.

2.3 GP Classification with kernlab

Task 1: Fit GP Classification Model Use the `kernlab` package to fit a Gaussian process classification model for fraud, and plot contours of prediction probabilities.

```
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)
```

```r
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train_data = data[SelectTraining, ]
test_data = data[-SelectTraining, ]

GPfit <- gausspr(fraud ~ varWave + skewWave, kernel = "rbfdot", data=train_data)
```

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

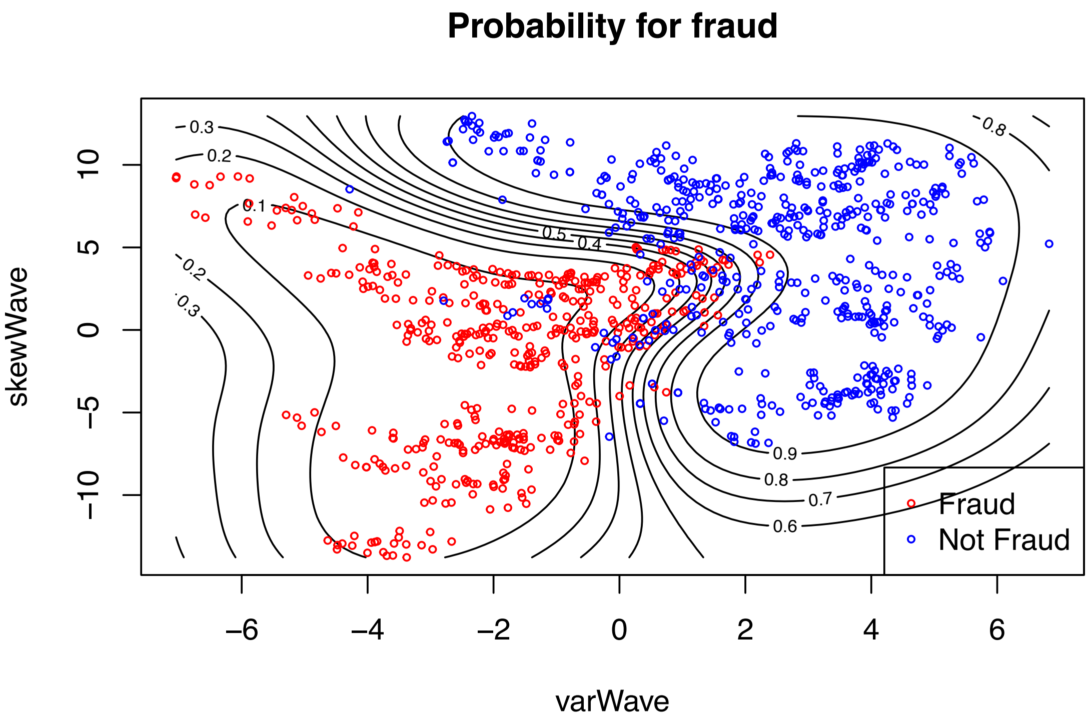
```r
pred = predict(GPfit, train_data)
cm = table(pred, train_data$fraud)
cm
```

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

```r
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set:", accuracy*100, "%\n")
```

```
## Accuracy on the training set: 94.1 %
```

```r
x1 <- seq(min(train_data[,1]),max(train_data[,1]),length=100)
x2 <- seq(min(train_data[,2]),max(train_data[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train_data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")

# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 10, xlab = "varWave", ylab = "skewWave", main = 'l
points(train_data[train_data[,5]==1,1],train_data[train_data[,5]==1,2],col="red", cex = 0.5)
points(train_data[train_data[,5]==0,1],train_data[train_data[,5]==0,2],col="blue", cex = 0.5)
legend("bottomright", legend = c("Fraud", "Not Fraud"), col = c("red", "blue"), pch = 1, pt.cex = 0.5)
```



Task 2: Make Predictions for Test Set Make predictions for the test set and compute the accuracy.

```
pred_test = predict(GPfit, test_data)
cm = table(pred_test, test_data$fraud)
cm
```

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

```
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set using two covariates:", round(accuracy*100, 1), "%\n")
```

```
## Accuracy on the training set using two covariates: 92.5 %
```

Task 3: Train Model with All Covariates Train a model using all four covariates and compare the accuracy to the model with only two covariates.

```
GPfit <- gausspr(fraud ~., kernel = "rbfdot", data=train_data)
```

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

```r
pred_test_all = predict(GPfit, test_data)
cm = table(pred_test_all, test_data$fraud)
cm
```

```
##
## pred_test_all   0   1
##             0 216   0
##             1   2 154
```

```r
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set using all covariates:", round(accuracy*100, 1), "%\n")
```

```
## Accuracy on the training set using all covariates: 99.5 %
```