Lab4

Olof Swedberg

2024-10-08

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 $\mathbf{A/b}$ 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:

- $\mathbf{X}$: Vector of training inputs.
- $\mathbf{y}$: 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.
- $\mathbf{k}$: Covariance function or kernel. That is, the kernel should be a separate function (see the file GaussianProcesses.R on the course web page).

```
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
```

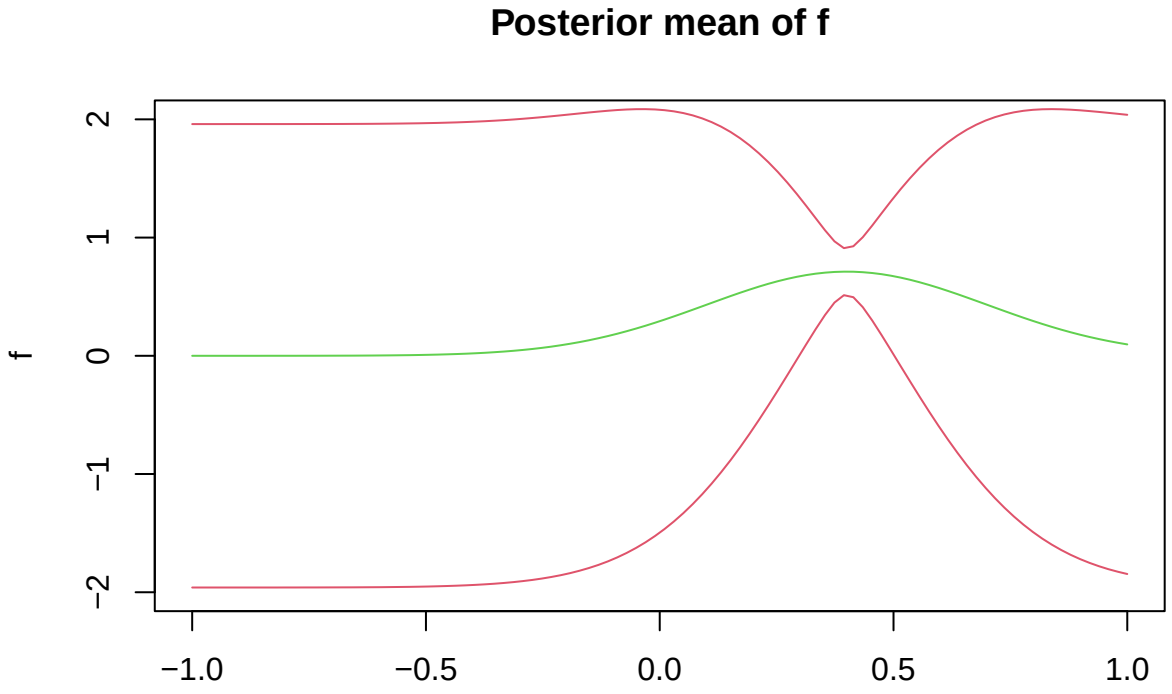
```
  log_marg_likelihood = -(1/2)*t(y)%*%alpha - sum(log(diag(L))) - (n/2)*log(2*pi)

  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.

```
########################################
###### 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)
```

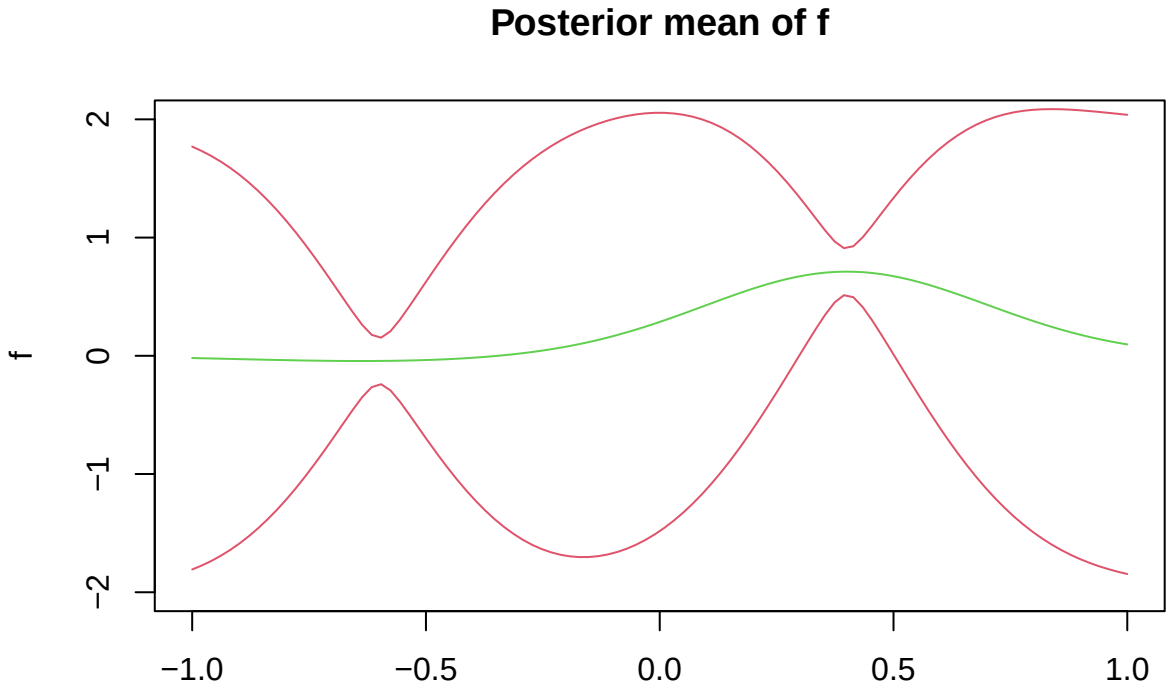


Task (3)

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.

```
# 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)
```

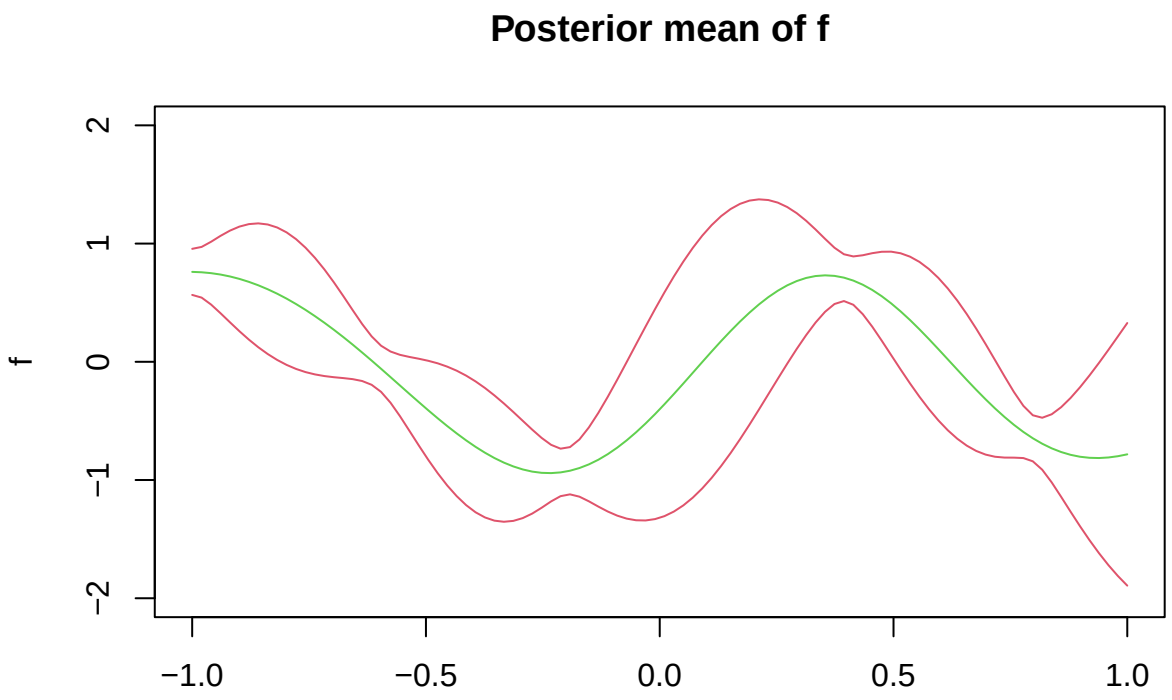


Task (4)

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

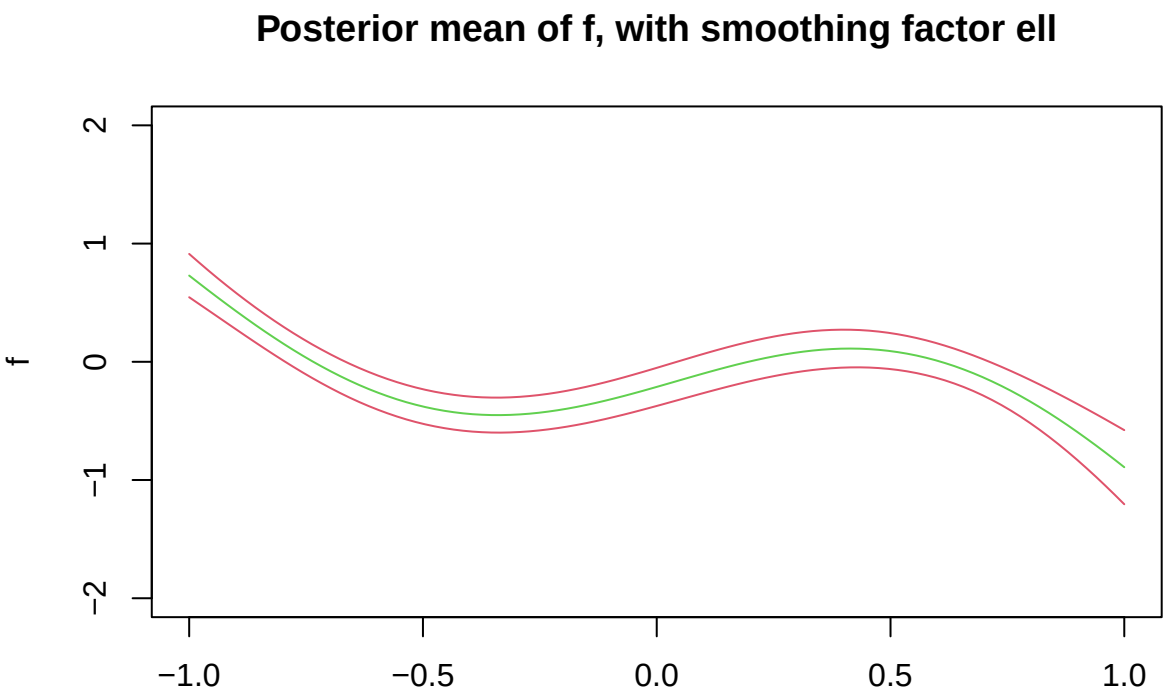
```
# Initialize paramters
x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.94, 0.719, -0.664)

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)
```



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)
```



2.2 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

Create the variable time which records the day number since the start of the dataset (i.e., `time` = 1, 2, . . ., 365Ö6 = 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.

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

data$time = 1:nrow(data)
data$day = rep(1:365, 6)

# Subsample every 5th observation
subsample_idx = seq(1, nrow(data), by = 5)
data_sub = data[subsample_idx, ]
```

Task (1)

Familiarize yourself with the functions `gausspr` and `kernelMatrix` in `kernlab`. Use `?gausspr` to read the input arguments and the output. Also, go through the file `KernLabDemo.R` available on the course website. You will need to understand it.

Now, define your own square exponential kernel function (with parameters ℓ (`ell`) and σ_f (`sigma_f`)), evaluate it at 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 $X^* = (2, 3, 4)^T$.

```
library(kernlab)
#########################################################
###### Inspired by code from Lab instructions ######
```

```
################ KernLabDemo.R ####################
####################################################
SquaredExpKernel <- function(sigmaF=1,ell=3){
  rval <- function(x1, x2) {
    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)
  }
  class(rval) <- "kernel"
  return(rval)
}
####################################################


SE_kernel = SquaredExpKernel(1,2)
print("Evaluation at 1,2")
```

```
## [1] "Evaluation at 1,2"
```

```
print(SE_kernel)
```

```
## function(x1, x2) {
##     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)
##   }
## <bytecode: 0x11a9cbcd8>
## <environment: 0x1296b5bb8>
## attr(,"class")
## [1] "kernel"
```

```
X = c(1,2,3)
XStar = c(2,3,4)
print("kernelMatrix")
```

```
## [1] "kernelMatrix"
```

```
kernelMatrix(SE_kernel,X,XStar)
```

```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.8824969 0.6065307 0.3246525
## [2,] 1.0000000 0.8824969 0.6065307
## [3,] 0.8824969 1.0000000 0.8824969
```

Task (2)

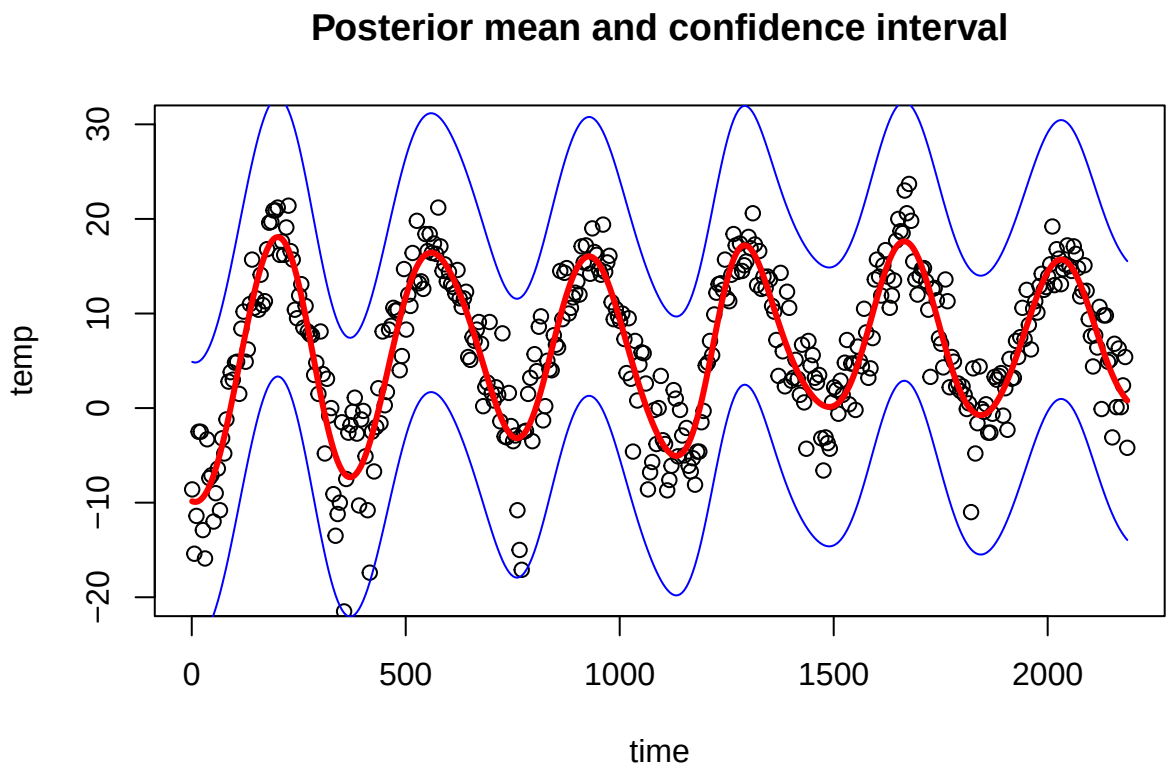
Consider first the following model:

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

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 of σ and ℓ (no need to write this in the report though).

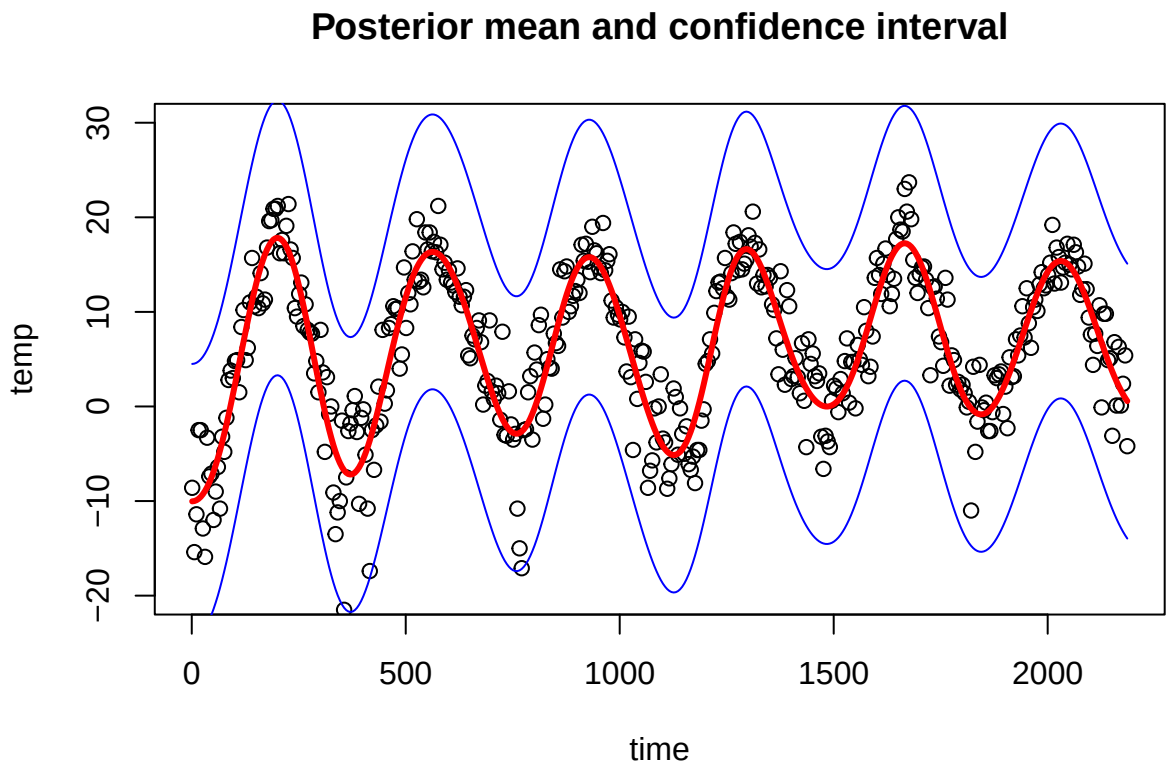
```
temp = data_sub$temp
time = data_sub$time
day = data_sub$day

polyFit = lm(temp ~  time + I(time^2))
sigmaNoise = sd(polyFit$residuals)


GPfit = gausspr(time,temp, kernel = SquaredExpKernel(sigmaF = 20, ell = 100), var = sigmaNoise, scaled =
meanPred = predict(GPfit, time)

plot(time,temp, main = "Posterior mean and confidence interval", ylim = c(-20,30))
lines(time, meanPred, col="red", lwd = 3)
lines(time, meanPred+1.96*sd(meanPred),col="blue")
lines(time, meanPred-1.96*sd(meanPred),col="blue")
```



Task (3)

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.

```
posterior = posteriorGP(X=time, y=temp, sigmaNoise = sigmaNoise, XStar = time, k = SquaredExpKernel(20,
meanPred = posterior$mean

plot(time,temp, main = "Posterior mean and confidence interval", ylim = c(-20,30))
lines(time, meanPred, col="red", lwd = 3)
lines(time, meanPred+1.96*sd(meanPred),col="blue")
lines(time, meanPred-1.96*sd(meanPred),col="blue")
```



As expected, these look exaclty the same since they both use the same kernel.

Task (4)

Consider now the following model:

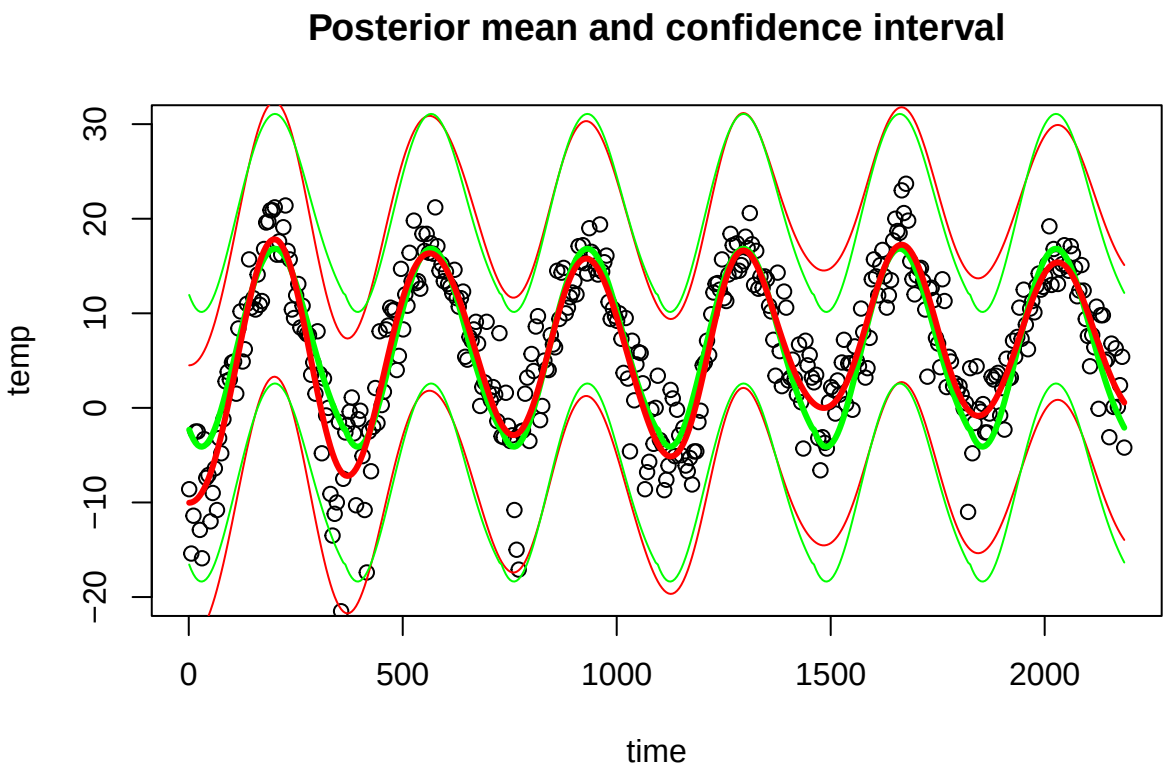
$$\text{temp} = f(\text{day}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_n^2) \quad \text{and} \quad f \sim \mathcal{GP}(0, k(\text{day}, \text{day'}))$$

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

```
GPfit = gausspr(day,temp, kernel = SquaredExpKernel(sigmaF = 20, ell = 100), var = sigmaNoise, scaled =
meanPredDays = predict(GPfit, day)
```

```
plot(time,temp, main = "Posterior mean and confidence interval", ylim = c(-20,30))
lines(time, meanPredDays, col="green", lwd = 3)
lines(time, meanPred, col="red", lwd = 3)
lines(time, meanPred+1.96*sd(meanPred),col="red")
lines(time, meanPred-1.96*sd(meanPred),col="red")
lines(time, meanPredDays+1.96*sd(meanPredDays),col="green")
lines(time, meanPredDays-1.96*sd(meanPredDays),col="green")
```



Here, the posterior fitted with time (red) does not show similar increase in temperature over time. The posterior fitted with days (green) increases slightly more. However, these are very similar averages.

Task (5)

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

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

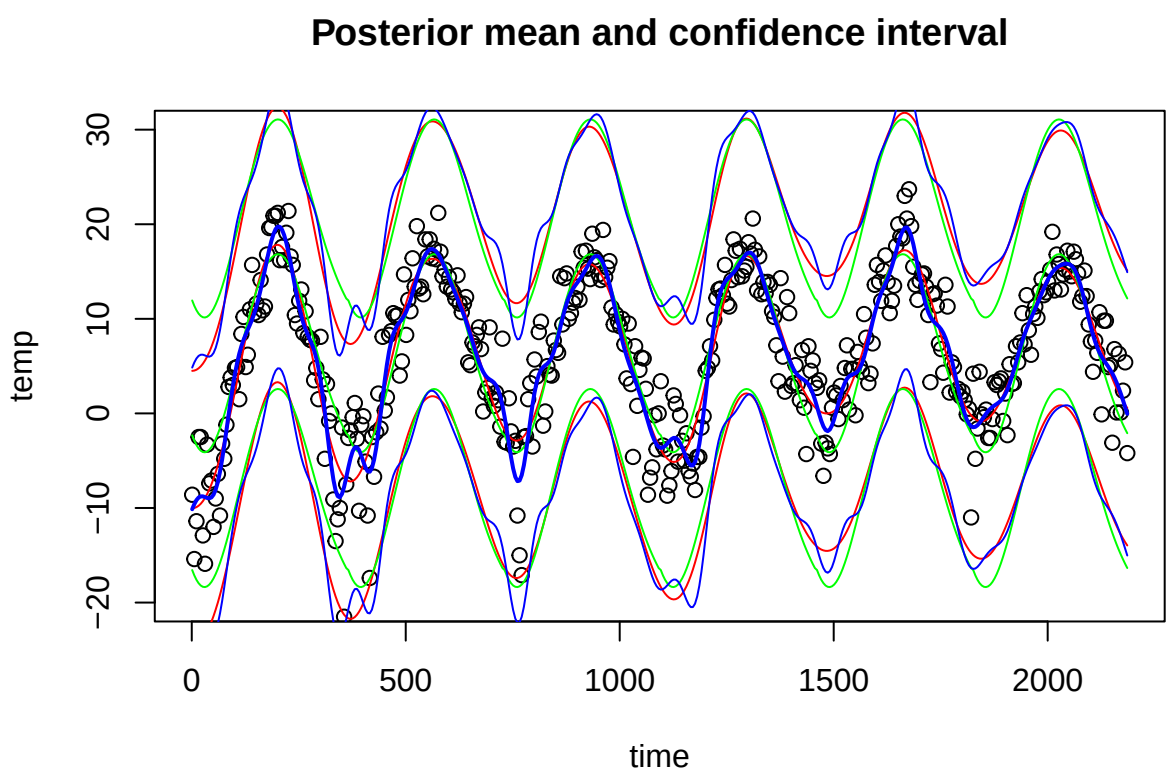
Note that we have two different length scales in the kernel. Intuitively, ℓ_1 controls the correlation between two days in the same year, and ℓ_2 controls the correlation between the same day in different years.

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.

Compare the fit to the previous two models (with $\sigma_f = 20$ and $\ell = 100$). Discuss the results.

```
ExtendedSquaredExpKernel <- function(sigmaF,ell1, ell2, d){
  rval <- function(x1, x2) {
```

```
    n1 <- length(x1)
    n2 <- length(x2)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      diff = abs(x1 - x2[i])
      K[, i] = sigmaF^2*exp(-2*sin(pi*diff/d)^2/ell1^2) * exp(-0.5*diff^2/ell2^2)
    }
    return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}

GPfit = gausspr(time,temp, kernel = ExtendedSquaredExpKernel(sigmaF = 20, ell1 = 1, ell2 = 100, d = 365
meanPredExtended = predict(GPfit, time)

plot(time,temp, main = "Posterior mean and confidence interval", ylim = c(-20,30))
lines(time, meanPred, col="red", lwd = 1)
lines(time, meanPred+1.96*sd(meanPred),col="red")
lines(time, meanPred-1.96*sd(meanPred),col="red")
lines(time, meanPredDays, col="green", lwd = 1)
lines(time, meanPredDays+1.96*sd(meanPredDays),col="green")
lines(time, meanPredDays-1.96*sd(meanPredDays),col="green")
lines(time, meanPredExtended, col="blue", lwd = 2)
lines(time, meanPredExtended+1.96*sd(meanPredExtended),col="blue")
lines(time, meanPredExtended-1.96*sd(meanPredExtended),col="blue")
```



The extended periodic kernel is better at fitting the data because it can find correlation between days in different years. The time-based model might be better for long-term trends, while the day-based and periodic models capture seasonal patterns more effectively.

2.3 GP Classification with `Kernlab`

GP Classification with kernlab. Download the banknote fraud data:

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

You can read about this dataset here. Choose 1000 observations as training data using the following command (i.e., use the vector `SelectTraining` to subset the training observations):

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

Task (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.

```
library(kernlab)
GPfitFraud = gausspr(fraud ~ varWave + skewWave, data = trainingData)
```

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

```
GPfitFraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1.42415004757028
##
## Number of training instances learned : 1000
## Train error : 0.059
```

```
# predict on the training set
predictions = predict(GPfitFraud, trainingData[,1:2])
table(predictions, trainingData[,5]) # confusion matrix
```

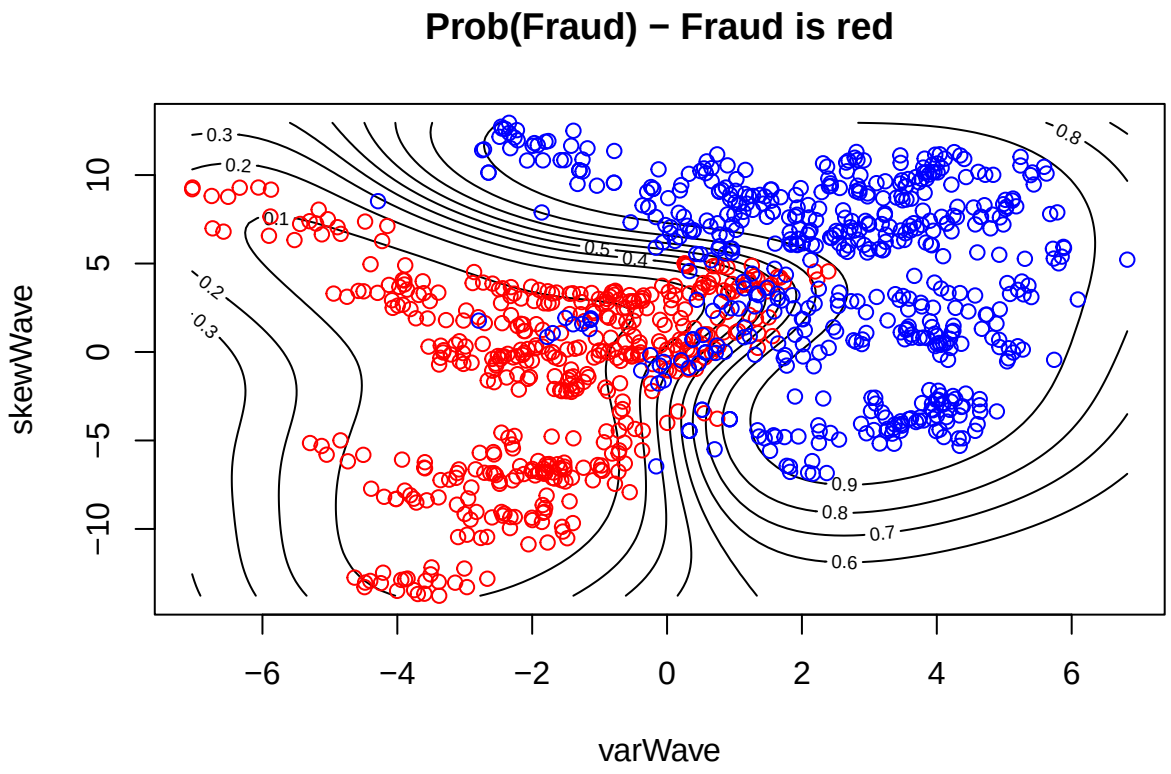
```
##
## predictions   0   1
##           0 503  18
##           1  41 438
```

```
mean(predictions == trainingData[,5]) # accuracy
```

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

```
varWave_seq = seq(min(trainingData$varWave), max(trainingData$varWave), length = 100)
skewWave_seq = seq(min(trainingData$skewWave), max(trainingData$skewWave), length = 100)

grid = expand.grid(varWave = varWave_seq, skewWave = skewWave_seq)
probabilities = predict(GPfitFraud, grid, type = "probabilities")
```

```
# Plot the contour
prob_matrix = matrix(probabilities[, 1], nrow = 100)
contour(varWave_seq, skewWave_seq, prob_matrix, 10, col = "black", xlab = "varWave", ylab = "skewWave",
points(trainingData[trainingData[,5] =='1',1],trainingData[trainingData[,5] =='1',2],col="red")
points(trainingData[trainingData[,5] =='0',1],trainingData[trainingData[,5] =='0',2],col="blue")
```



Task (2)

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

```
predictions = predict(GPfitFraud, testingData[,1:2])
table(predictions, testingData[,5]) # confusion matrix
```

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

```
mean(predictions == testingData[,5]) # accuracy
```

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

The accuracy on the testing data is a little lower than the training data, which is expected.

Task (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 = trainingData)
```

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

```
GPfitFraud
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.438275644976386
##
## Number of training instances learned : 1000
## Train error : 0.003
```

```
# predict on the training set
predictions = predict(GPfitFraud, testingData[,1:4])
table(predictions, testingData[,5]) # confusion matrix
```

```
##
## predictions   0   1
##           0 216   0
##           1   2 154
```

```
mean(predictions == testingData[,5]) # accuracy
```

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

The accuracy is 99.46%, which is higher than when predicting using two variables and is reasonable since the model can use more information.