Advanced Machine Learning (732A96) - Lab 04

Hector Plata (hecpl268)

Contents

Task 2.1 **1**
Task 2.1.1 1
Task 2.1.2 3
Task 2.1.3 4
Task 2.1.4 5
Task 2.1.5 6

Task 2.2 **7**
Task 2.2.1 7
Task 2.2.2 8
Task 2.2.3 10
Task 2.2.4 11
Task 2.2.5 13

Task 2.3 **16**
Task 2.3.1 17
Task 2.3.2 19
Task 2.3.3 20

Task 2.1

Implementing GP Regression. This first exercise will have you writing your own code for the Gaussian process regression model.

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

You must implement Algorithm 2.1 on page 19 of Rasmussen and Willams' book. The algorithm uses the Cholesky decomposition (`chol`in `R`) to attain numerical stability. NOte that L in the algorithm is a lower triangular matrix. So, you need to transpose the output of the `R` function. In the algorithm, the notation $\mathbf{Ab}$ means that the vector $\mathbf{x}$ that solves the equation $\mathbf{Ax} = \mathbf{b}$ (see p. xvii in the book). This is implementedin `R` with the help of the function `solve`. Here is what you need to do:

Task 2.1.1

Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it `posteriorGP`) 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 the following inputs:

- `X`: Vector of training inputs.
- `y`: Vector of training targets/outpus.
- `XStar`: Vector of inputs where the posterior distribution is evaluated, i.e. X_*.
- `hyperParam`: Vector with two elements, σ_f and l.
- `sigmaNoise`: Noise standard deviation σ_n.

Hint: Write a separate function for the kernel (see the file `GaussianProcess.R` on the course web page).

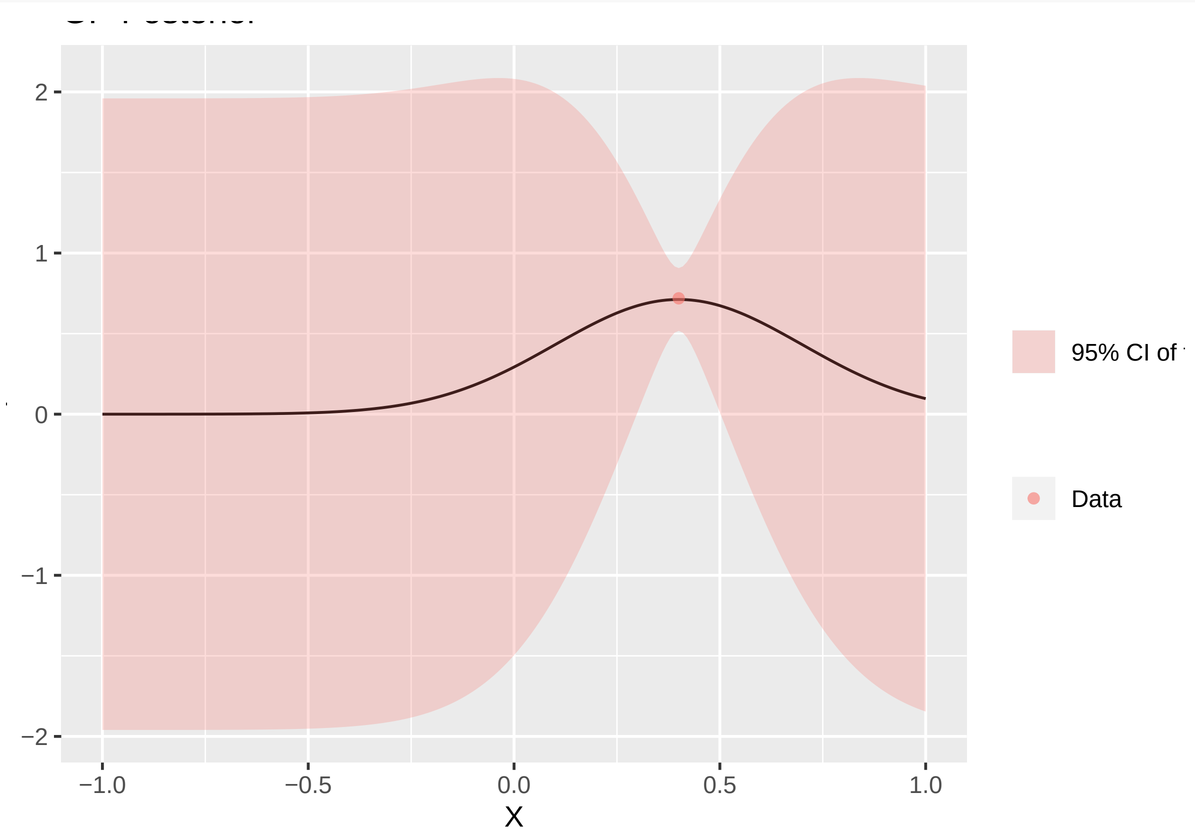
```r
library(ggplot2)

# Auxiliary plotting functions.
plot_me_pls = function(model, X_star, X, y, confidence=0.975, title='GP Posterior')
{
  # Plots the posterior of the GP with confidence intervals.
  z = qnorm(confidence)
  y_min = model$f_mean_star - z * sqrt(diag(model$var_f_star))
  y_max = model$f_mean_star + z * sqrt(diag(model$var_f_star))

  p = ggplot() +
      geom_line(aes(x=X_star, y=model$f_mean_star)) +
      geom_ribbon(aes(x=X_star, ymin=y_min, ymax=y_max, fill='95% CI of f'),
                  alpha=0.25) +
      geom_point(aes(x=X, y=y, colour='Data'), alpha=0.6) +
      labs(x='X', y='Y', fill='', colour='', title=title)

  return(p)
}


# Radial basis function.
rbf = function(x_i, x_j, h=1, factor=1)
{
  # Calculates the radial basis function for two vectors
  # x_i and x_j.
  #
  # Parameters
  # ----------
  # x_i, x_j: vector
  # h : float
  # factor : float
  #
  # Returns
  # -------
  # rbf: scalar

  res = factor * exp(-sum((x_i - x_j) ^ 2) / (2 * (h ^ 2)))
  return(res)
}


# Radial basis function kernel.
rbfk = function(X, X_new, h=1, factor=1)
{
  # Calculates the radial basis function between two matrices
  # X and X_new.
  #
  # Parameters
  # ----------
  # x_i, x_j: vector
  # h : float
  # factor : float
  #
  # Returns
  # -------
  # rbf: scalar
  N1 = dim(X)[1]
  N2 = dim(X_new)[1]
  K = matrix(NA, N1, N2)

  for (i in 1:N1)
  {
    for (j in 1:N2)
    {
      K[i, j] = rbf(X[i,], X_new[j, ], h, factor)
    }
  }

  return(K)
}


posteriorGP = function(X, y, XStar, hyperParam, sigma2Noise)
{
  # I'm lazy, no more docs.
  n = dim(X)[1]
  sigma_f = hyperParam[1]
  l = hyperParam[2]
  K = rbfk(X, X, l, sigma_f)
  K_star = rbfk(X, XStar, l, sigma_f)
  K_test = rbfk(XStar, XStar, l, sigma_f)

  # Calculating the predictive mean.
  L = t(chol(K + diag(sigma2Noise, n)))
  alpha = solve(t(L), solve(L, y))
  f_mean_star = t(K_star) %*% alpha

  # Calculating the predictive variance.
  v = solve(L, K_star)
  var_f_star = K_test - (t(v) %*% v)
  log_marginal_lh = -0.5 * t(y) %*% alpha - sum(log(diag(L))) - (n / 2) * log(2 * pi)

  return(list(f_mean_star=f_mean_star,
              var_f_star=var_f_star,
              log_marginal_lh=log_marginal_lh))
}
```

Task 2.1.2

Now, let the prior hyperparameters be $\sigma_f = 1$ and $l = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Plot also 95% probability (pointwise) bands for f.

```
# Parameters and inputs of the model.
hyperparam = c(1, 0.3)
X = matrix(0.4)
y = matrix(0.719)
```

```
sigma2_noise = 0.1 ^ 2
X_star = matrix(seq(-1, 1, 1/100))

# Create and estimate the model.
model = posteriorGP(X, y, X_star, hyperparam, sigma2_noise)

p = plot_me_pls(model, X_star, X, y)
print(p)
```

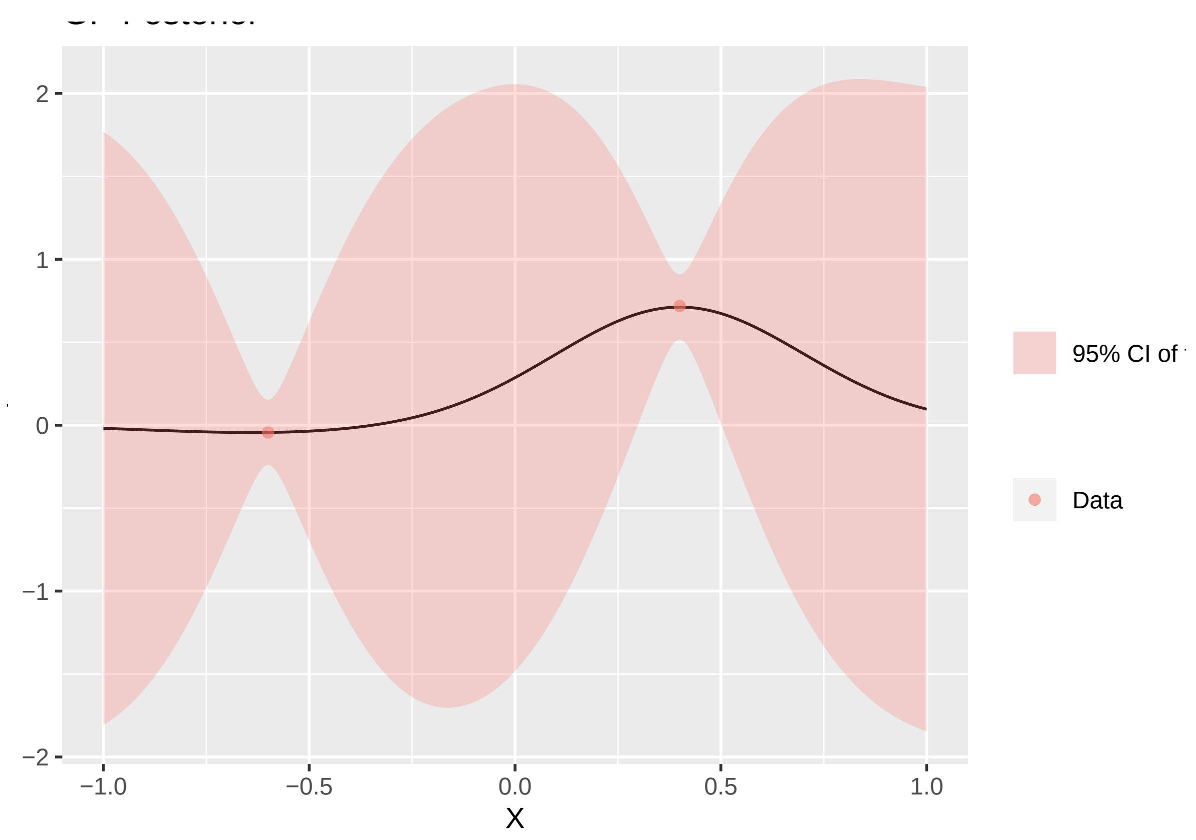


Task 2.1.3

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 (point-wise) bands for f.

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

```
hyperparam = c(1, 0.3)
X = matrix(c(0.4, -0.6))
y = matrix(c(0.719, -0.044))
sigma2_noise = 0.1 ^ 2
X_star = matrix(seq(-1, 1, 1/100))

# Create and estimate the model.
model = posteriorGP(X, y, X_star, hyperparam, sigma2_noise)

p = plot_me_pls(model, X_star, X, y)
print(p)
```

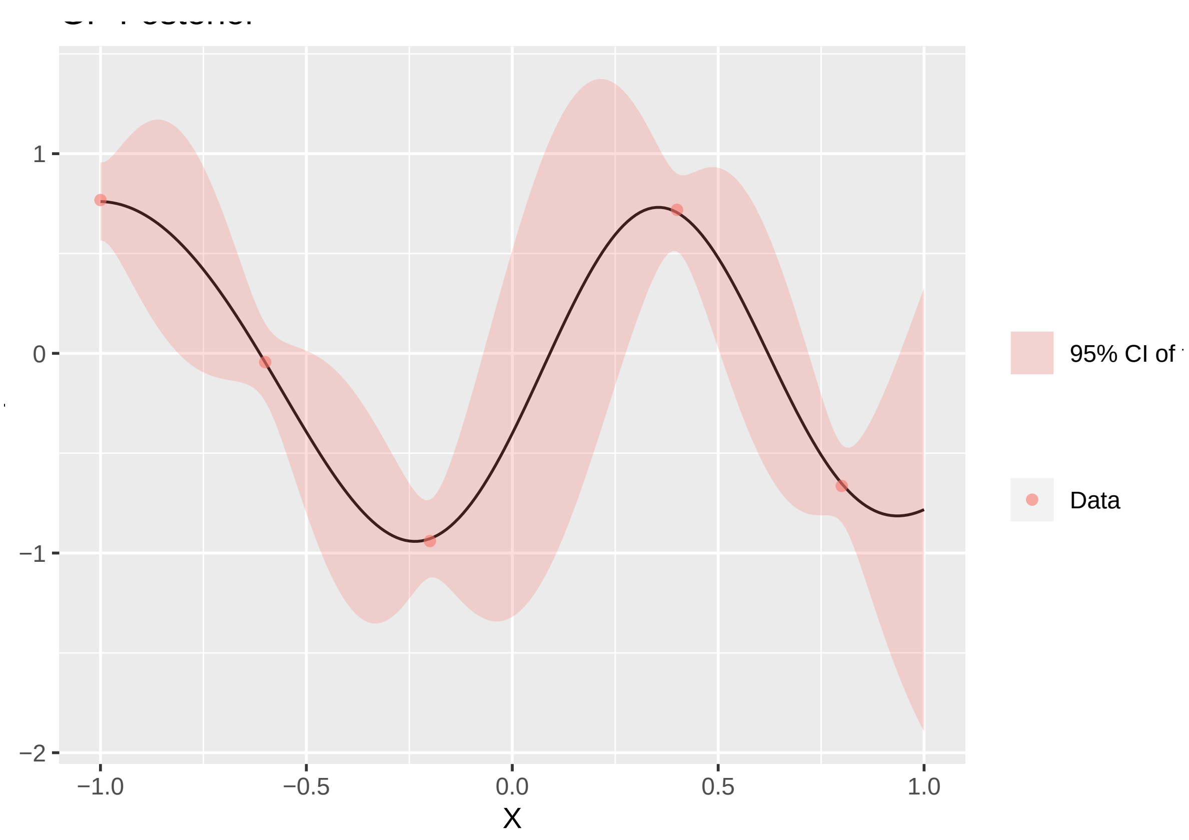


Task 2.1.4

Compute the posterior distribution of f using all the 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

```r
hyperparam = c(1, 0.3)
X = matrix(c(-1, -0.6, -0.2, 0.4, 0.8))
y = matrix(c(0.768, -0.044, -0.94, 0.719, -0.664))
sigma2_noise = 0.1 ^ 2
X_star = matrix(seq(-1, 1, 1/100))

# Create and estimate the model.
model = posteriorGP(X, y, X_star, hyperparam, sigma2_noise)

p = plot_me_pls(model, X_star, X, y)
print(p)
```

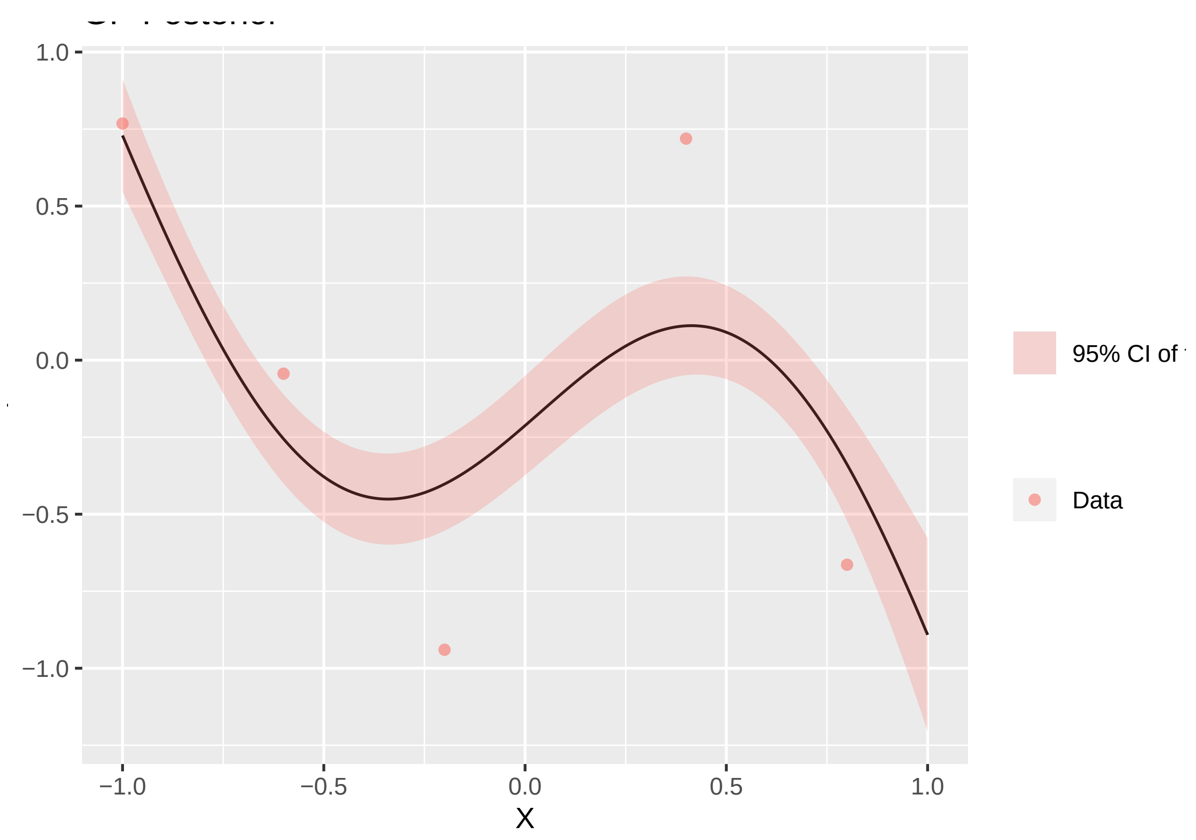


Task 2.1.5

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

Answer Since the distance over where we let the functions vary is higher, the mean of them changes 'slowly' and misses some of the points unlike on task 2.1.4. This parameter also affects the variance of each of the points.

```r
hyperparam = c(1, 1)
X = matrix(c(-1, -0.6, -0.2, 0.4, 0.8))
y = matrix(c(0.768, -0.044, -0.94, 0.719, -0.664))
sigma2_noise = 0.1 ^ 2
X_star = matrix(seq(-1, 1, 1/100))

# Create and estimate the model.
model = posteriorGP(X, y, X_star, hyperparam, sigma2_noise)

p = plot_me_pls(model, X_star, X, y)
print(p)
```



Task 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. 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`$= 1, 2, \ldots,$ 365 x 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`$= 1, 6, 11, \ldots, 361, 1, 6, 11, \ldots, 361$.

Task 2.2.1

Familiarize yourself with the functions `gausspr` and `kernelMatrix` in `kernlab`. Do `?gausspr` and 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, dfine 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 vector $X = (1, 3, 4)^T$ and $X_* = (2, 3, 4)^T$.

```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
```

```
##
##     alpha
```

```
# Loading the data and creating variables.
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c
                header=TRUE,
                sep=";")
data['time'] = seq(1, dim(data)[1])
data['day'] = rep(seq(1, 365), dim(data)[1] / 365)
data = data[data$time %% 5 == 1, ]

# Defining my own square exponential kernel function.
rbfk = function(sigma_f=1, ell=1)
{
  rbf = function(x_i, x_j, h=1, factor=1)
  {
    res = (sigma_f ^ 2) * exp(-sum((x_i - x_j) ^ 2) / (2 * (ell ^ 2)))
    return(res)
  }

  class(rbf) = 'kernel'
  return(rbf)
}

# Evaluating the kernel on one point.
print(rbfk(1, 1)(1, 2))
```

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

```
# Calculating the covariance matrix between X and X_star.
X = matrix(c(1, 3, 4))
X_star = matrix(c(2, 3, 4))
K = kernelMatrix(rbfk(), X, X_star)
print(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
```

Task 2.2.2

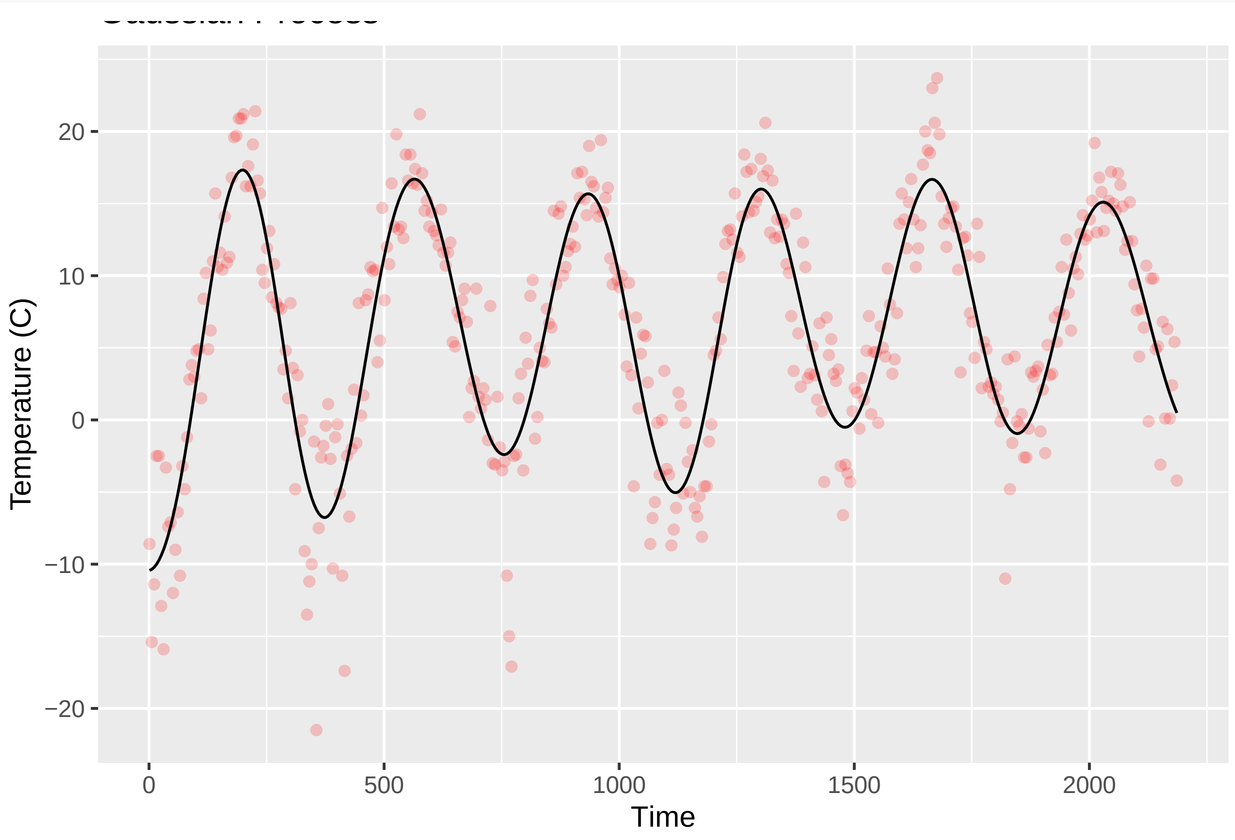
Consider first the following model:

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

Let σ_n^2 be the residual variance from a simple quadratic regression fit (using the `lm` function inR). Estimate the above Gaussian process regression model using the squared exponential function from (1) with $\sigma_f = 20$ and $l = 0.2$. 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 valued on σ_f and l (no need to write this in the report though).

```
# Calculating sigma2_n.
reg = lm(temp ~ time + I(time^2), data)
sigma_noise = sd(reg$residuals)
```

```r
# Parameters for the GP.
sigma_f = 20
ell = 0.2

model = gausspr(x=data$time,
                y=data$temp,
                kernel=rbfk,
                kpar=list(sigma_f=sigma_f, ell=ell),
                var=sigma_noise ^ 2)

y_hat = predict(model, data$time)

p = ggplot() +
    geom_point(aes(x=data$time,
                   y=data$temp,
                   color='Original Temperature'),
               color='red',
               alpha=0.2) +
    geom_line(aes(x=data$time,
                  y=y_hat,
                  color='Mean Temperature'),
              color='black') +
    labs(x='Time', y='Temperature (C)', color='', title='Gaussian Process')

print(p)
```

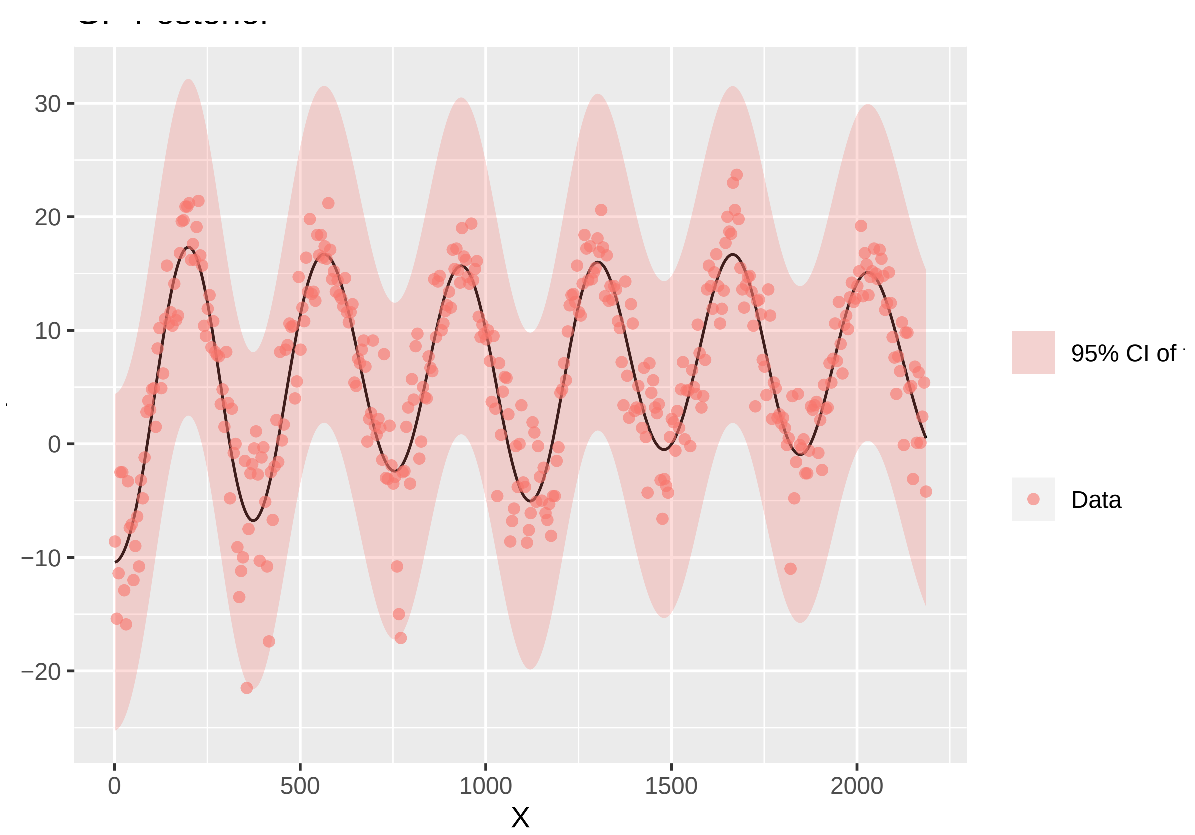


Task 2.2.3

`kernlab` can compute the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f, but it seems to be a bug in the code. So, do your own computations for the posterior variance of f and plot the 95% probability (pointwise) bands for f. Superimpose these bands on the figure with the posterior mean that you obtained in (2).

Hint: Note that Algorithm 2.1 on page 19 of Rasmussen and Willams' book already does the calculations required. Note also that `kernlab` scaled the data by default to have zero mean and standard deviation one. So, the output of your implementation of Algorithm 2.1 will not coincide with the ouput of `kernlab` unless you scale the data first. For this, you may want to use the `R` function `scale`.

```r
get_gp_variance = function(X, y, X_star, sigma_noise, kernel)
{
  # I'm lazy, no more docs.
  n = dim(X)[1]
  K = kernelMatrix(kernel, X, X)
  K_star = kernelMatrix(kernel, X, X_star)
  K_test = kernelMatrix(kernel, X_star, X_star)

  # Calculating the predictive variance.
  var_f_star = K_test - t(K_star) %*% solve(K + diag(sigma_noise ^ 2, n)) %*% K_star

  return(var_f_star)
}

X = matrix(data$time)
y = matrix(data$temp)
var_f_star = get_gp_variance(X,
                             y,
                             X,
                             sigma_noise,
                             rbfk(sigma_f, ell))

mixed_model = list(f_mean_star=y_hat,
                   var_f_star=var_f_star,
                   log_marginal_lh=NA)

p = plot_me_pls(mixed_model, X, X, y)
print(p)
```



Task 2.2.4

Consider now the following model:

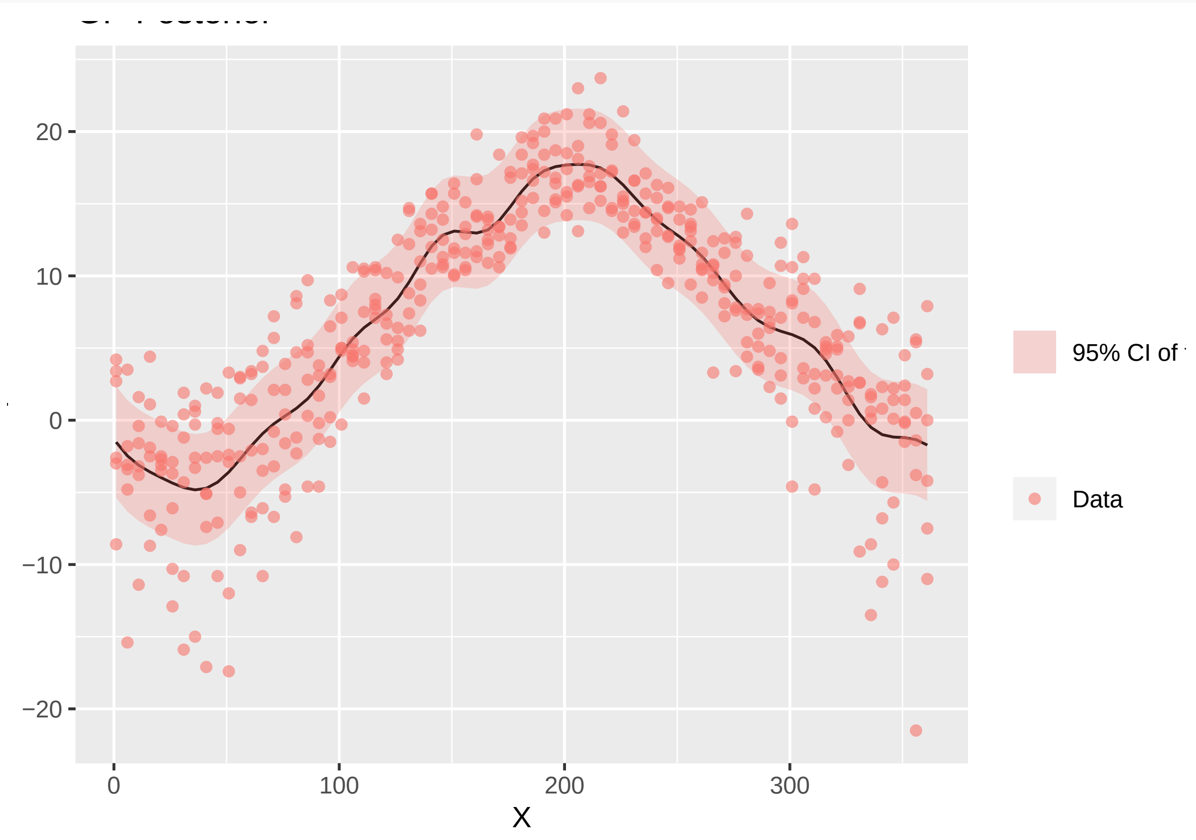
$$temp = f(day) + \epsilon \text{ with } \epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right) \text{ and } f \sim \mathcal{GP}\left(0, k\left(day, day'\right)\right)$$

Estimate the model using the squared exponential function with $\sigma_f = 20$ and $l = 0.2$. Superimpose the posterior mean from this model on the posterio 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?

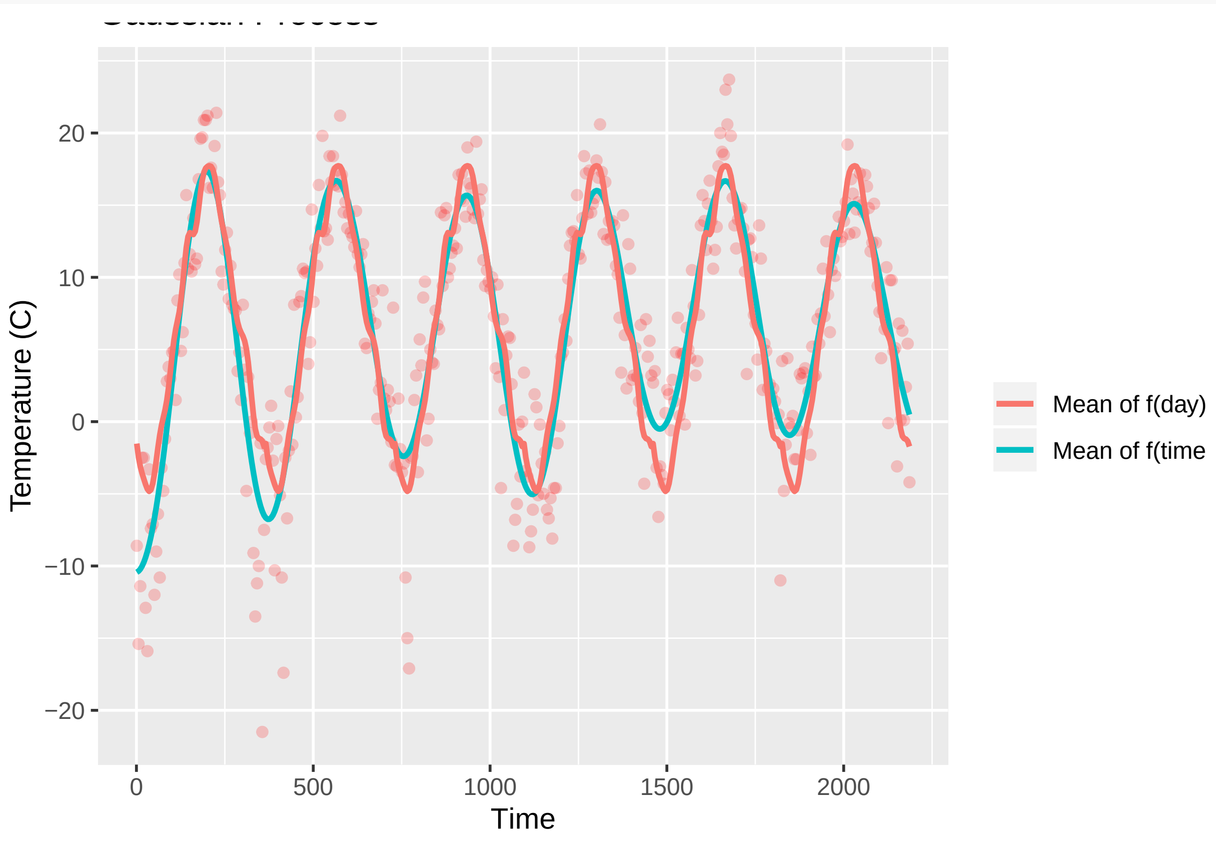
Answer The `day` model is a model that captures the average behaviour of the day of the year and the temperature. This model is able to capture seasonality, however is not able to capture trends over periods longer than 1 year. On the other hand, the `time` model is able to capture temperature trends over years but is not suited to capture seasonality since its just learning a relationship between indexes and temperature.

```r
reg = lm(temp ~ day + I(day^2), data)
sigma_noise = sd(reg$residuals)

# Parameters for the GP.
sigma_f = 20
ell = 0.2

model = gausspr(x=data$day,
                y=data$temp,
                kernel=rbfk,
                kpar=list(sigma_f=sigma_f, ell=ell),
                var=sigma_noise ^ 2)
```

```
y_hat_day = predict(model, data$day)

# Plotting the new standalone model.
X = matrix(data$day)
y = matrix(data$temp)
var_f_star = get_gp_variance(X,
                             y,
                             X,
                             sigma_noise,
                             rbfk(sigma_f, ell))

mixed_model = list(f_mean_star=y_hat_day,
                   var_f_star=var_f_star,
                   log_marginal_lh=NA)

p = plot_me_pls(mixed_model, X, X, y)
print(p)
```



```
# Plotting both models on the same figure.
p = ggplot() +
    geom_point(aes(x=data$time,
                   y=data$temp,
                   color='Original Temperature'),
               color='red',
               alpha=0.2) +
    geom_line(aes(x=data$time,
                  y=y_hat,
                  color='Mean of f(time)'),
              size=1) +
    geom_line(aes(x=data$time,
                  y=y_hat_day,
                  color='Mean of f(day)'),
              size=1) +
    labs(x='Time', y='Temperature (C)', color='', title='Gaussian Process')

print(p)
```



Task 2.2.5

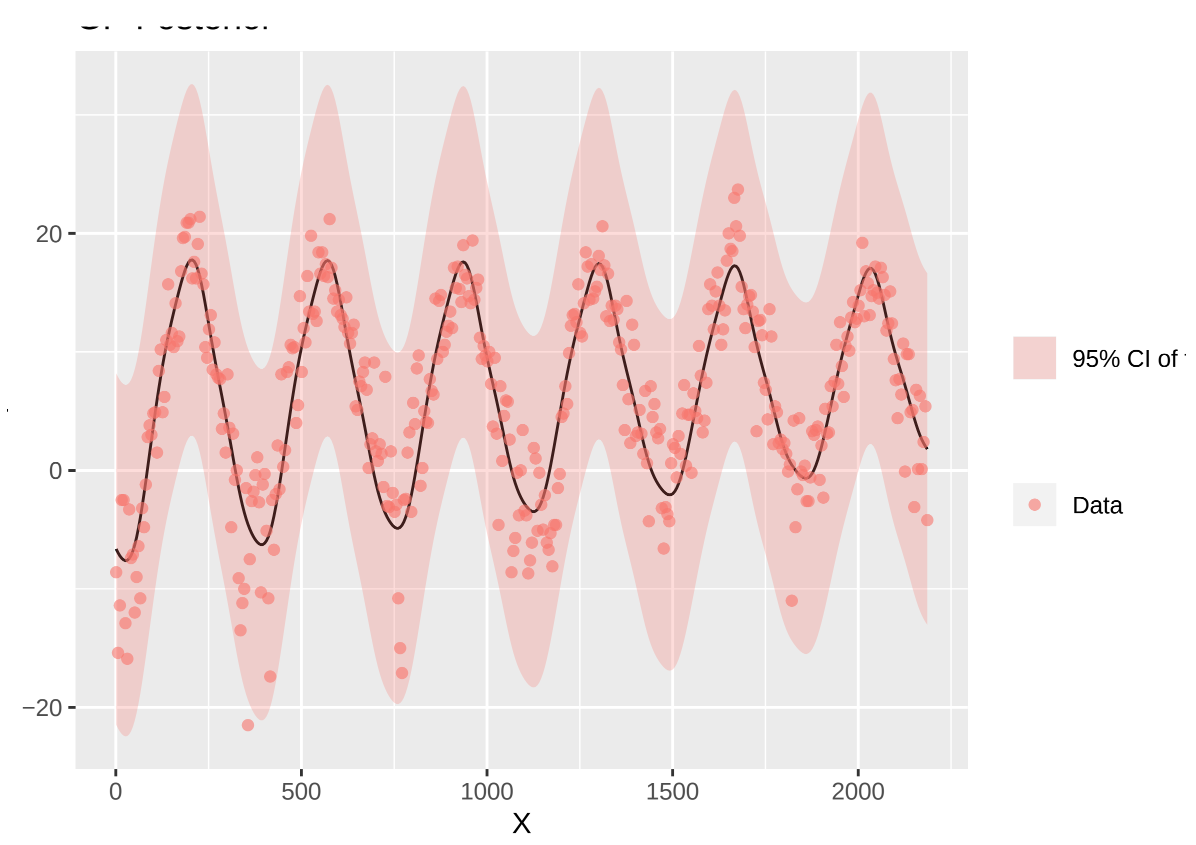
Finally, implement a generalization of the periodic kernel given in the lectures:

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

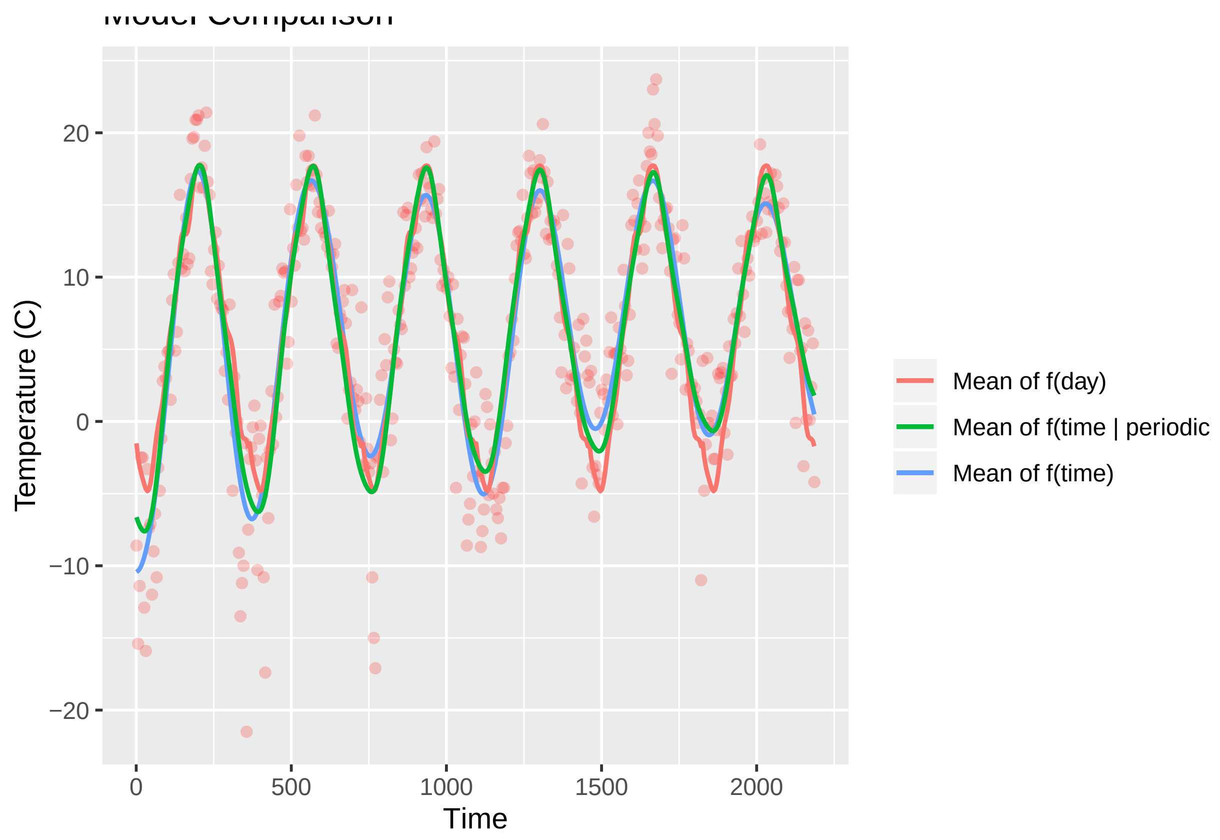
Note that we have two different length scales here, and l_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20$, $l_1 = 1$, $l_2 = 10$ and $d = 365/\sigma_{time}$. The reason for the rather strange period here is that `kernlab` standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20$ and $l = 0.2$). Discuss the results.

Answer From the plots and the results we can see that the model with a preiodic kernel is some what an intermediate step between the `day` model and the `time` model. This model (`periodic`) is able to capture trends and seasonality at the same time thanks to it's kernel. The first part of the kernel that is divided by l_1 controls the seasonality and the second part which is divided by l_2 controls the trends over the years.

```r
library(knitr)

periodic_kernel = function(sigma_f, d, l_1, l_2)
{
  periodic_inception_we_need_to_go_deeper = function(x_i, x_j)
  {
    first_exp = exp(-2 * ((sin(pi * abs(x_i - x_j) / d)) ^ 2) / (l_1 ^ 2))
    second_exp = exp(-0.5 * (abs(x_i - x_j) ^ 2) / (l_2 ^ 2))
    res = (sigma_f ^ 2) * first_exp * second_exp
    return(res)
  }

  class(periodic_inception_we_need_to_go_deeper) = 'kernel'
  return(periodic_inception_we_need_to_go_deeper)
}

# Calculating sigma2_n.
reg = lm(temp ~ time + I(time^2), data)
sigma_noise = sd(reg$residuals)

# Kernel parameters.
kpar = list(sigma_f=sigma_f,
            d=(365 / sd(data$time)),
            l_1=1,
            l_2=10)

model_periodic = gausspr(x=data$time,
                         y=data$temp,
                         kernel=periodic_kernel,
                         kpar=kpar,
                         var=sigma_noise ^ 2)

y_hat_periodic = predict(model_periodic, data$time)

# Plotting the new standalone model.
X = matrix(data$time)
y = matrix(data$temp)
var_f_star = get_gp_variance(X,
                             y,
                             X,
                             sigma_noise,
                             rbfk(sigma_f, ell))

mixed_model = list(f_mean_star=y_hat_periodic,
                   var_f_star=var_f_star,
                   log_marginal_lh=NA)

p = plot_me_pls(mixed_model, X, X, y)
print(p)
```



```
# Plotting all models on the same figure.
p = ggplot() +
    geom_point(aes(x=data$time,
                   y=data$temp,
                   color='Original Temperature'),
               color='red',
               alpha=0.2) +
    geom_line(aes(x=data$time,
                  y=y_hat,
                  color='Mean of f(time)'),
              size=0.8) +
    geom_line(aes(x=data$time,
                  y=y_hat_day,
                  color='Mean of f(day)'),
              size=0.8) +
    geom_line(aes(x=data$time,
                  y=y_hat_periodic,
                  color='Mean of f(time | periodic)'),
              size=0.8) +
    labs(x='Time', y='Temperature (C)', color='', title='Model Comparison')

print(p)
```



```r
# Data.
mse = function(y, y_hat)
{
  return(mean((y - y_hat) ^ 2))
}

MSE = c(mse(y, y_hat),
        mse(y, y_hat_day),
        mse(y, y_hat_periodic))

models = c('f(day)',
           'f(time)',
           'f(periodic)')

results_df = data.frame(models, MSE)
kable(results_df)
```

models	MSE
f(day)	12.50590
f(time)	15.44681
f(periodic)	13.44136

Task 2.3

GP Classification with `kernlab`. Donwload the banknote fraud data:

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

You can read about this dataset here](http://archive.ics.uci.edu/ml/datasets/banknote+authentication). 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)
```

Task 2.3.1

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(caret)
```

```
## Loading required package: lattice
```

```
library(AtmRay)

# Loading and preparing the data.
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud.
                header=FALSE,
                sep=",")
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)

X_train = data[SelectTraining, 1:2]
y_train = data[SelectTraining, 5]

X_test = data[-SelectTraining, 1:2]
y_test = data[-SelectTraining, 5]

model = gausspr(x=X_train, y=y_train)
```

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

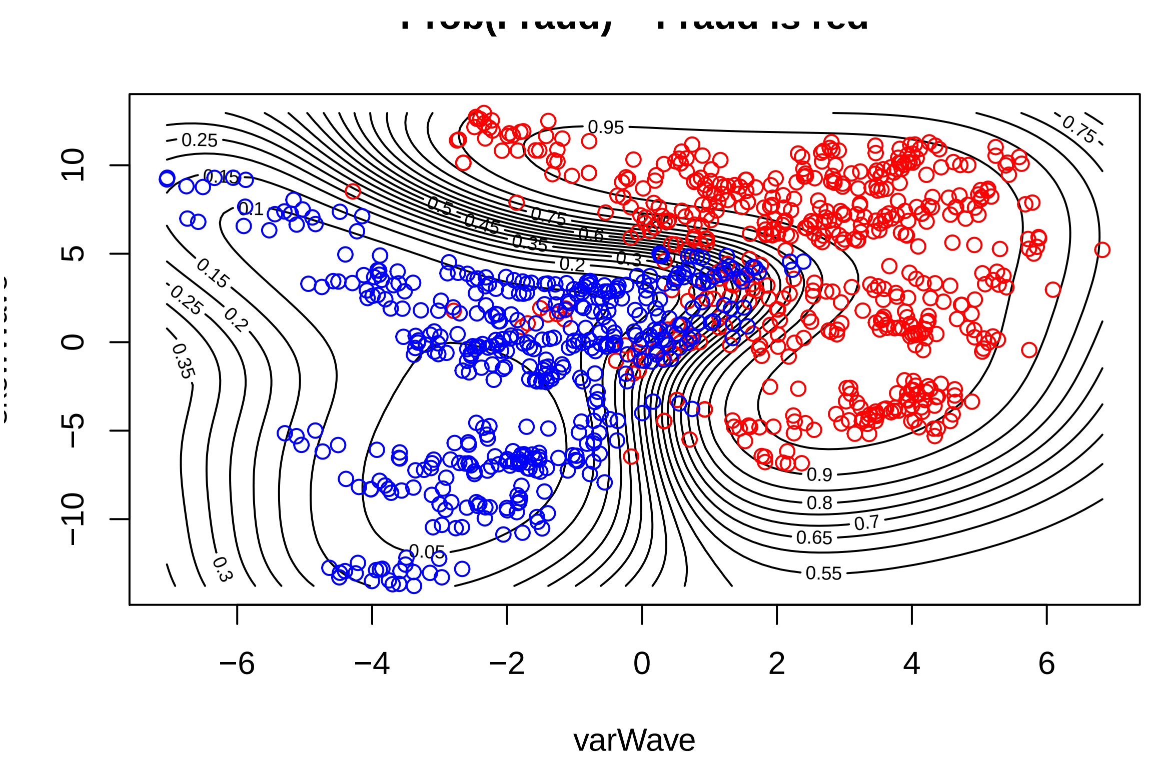
```
y_hat_train = predict(model, X_train)

print("Train confusion matrix")
```

```
## [1] "Train confusion matrix"
```

```
print(confusionMatrix(y_hat_train, y_train))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 503  18
##          1  41 438
##
##                Accuracy : 0.941
##                  95% CI : (0.9246, 0.9548)
##     No Information Rate : 0.544
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8816
##
##  Mcnemar's Test P-Value : 0.004181
##
##             Sensitivity : 0.9246
##             Specificity : 0.9605
##          Pos Pred Value : 0.9655
##          Neg Pred Value : 0.9144
##              Prevalence : 0.5440
##          Detection Rate : 0.5030
##    Detection Prevalence : 0.5210
##       Balanced Accuracy : 0.9426
##
##        'Positive' Class : 0
##
```

```
# Just copy paste code from KernLabDemo.R
# class probabilities
probPreds = predict(model, X_train, type='probabilities')
x1 = seq(min(X_train[, 1]),max(X_train[, 1]),length=100)
x2 = seq(min(X_train[, 2]),max(X_train[, 2]),length=100)
gridPoints = meshgrid(x1, x2)
gridPoints = cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints = data.frame(gridPoints)
names(gridPoints) = names(X_train)[1:2]
probPreds = predict(model, gridPoints, type='probabilities')

# Plotting for Prob(setosa)
contour(x1,
        x2,
        matrix(probPreds[, 1], 100, byrow = TRUE),
        20,
        xlab="varWave",
        ylab="skewWave",
        main='Prob(Fraud) - Fraud is red')
points(X_train[y_train==0, 1],X_train[y_train==0,2],col="red")
points(X_train[y_train==1,1],X_train[y_train==1,2],col="blue")
```



Task 2.3.2

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

```
y_hat_test = predict(model, X_test)

print("Test confusion matrix")
```

```
## [1] "Test confusion matrix"
```

```
print(confusionMatrix(y_hat_test, y_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 199   9
##          1  19 145
##
##                Accuracy : 0.9247
##                  95% CI : (0.8931, 0.9494)
##     No Information Rate : 0.586
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8463
##
##  Mcnemar's Test P-Value : 0.08897
##
##             Sensitivity : 0.9128
##             Specificity : 0.9416
##          Pos Pred Value : 0.9567
##          Neg Pred Value : 0.8841
##              Prevalence : 0.5860
##          Detection Rate : 0.5349
##    Detection Prevalence : 0.5591
##       Balanced Accuracy : 0.9272
##
##        'Positive' Class : 0
##
```

Task 2.3.3

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

Answer The model with four covariates achieve an accuracy of `0.9946` while the model with achieves one of `0.9247`. Overall the model with four covariates outpeforms the one with just two, which indicates that the rest of the variables contain some useful information to categorize a banknote.

```
X_train = data[SelectTraining, 1:4]
y_train = data[SelectTraining, 5]

X_test = data[-SelectTraining, 1:4]
y_test = data[-SelectTraining, 5]

model = gausspr(x=X_train, y=y_train)
```

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

```
y_hat_test = predict(model, X_test)

print("Test confusion matrix for the four covariate model")
```

```
## [1] "Test confusion matrix for the four covariate model"
```

```
print(confusionMatrix(y_hat_test, y_test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 216   0
##          1   2 154
##
##                Accuracy : 0.9946
##                  95% CI : (0.9807, 0.9993)
##     No Information Rate : 0.586
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9889
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.9908
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9872
##              Prevalence : 0.5860
##          Detection Rate : 0.5806
##    Detection Prevalence : 0.5806
##       Balanced Accuracy : 0.9954
##
##        'Positive' Class : 0
##
```