732A96: Advanced Machine Learning

Computer Lab 4: Gaussian Processes

Hariprasath Govindarajan (hargo729)

October 11, 2019

Contents

Question 1: Implementing GP Regression

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

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

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, 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 $\mathbf{A} \backslash \mathbf{b}$ means the vector x that solves the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ (see p. xvii in the book). This is implemented in R with the help of the function `solve`.

1.1

Question: 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 ($X_\star$). 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/outputs.
- `XStar`: Vector of inputs where the posterior distribution is evaluated, i.e. $X_\star$.
- `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).

Answer:

```
# Covariance function
squared_exp_kernel = function(x1, x2, sigma_f = 1, l = 3){
  n1 = length(x1)
  n2 = length(x2)
  K = matrix(NA,n1,n2)

  for (i in 1:n2){
    K[,i] = (sigma_f^2)*exp(-0.5*( (x1-x2[i])/l)^2 )
  }

  return(K)
}
```

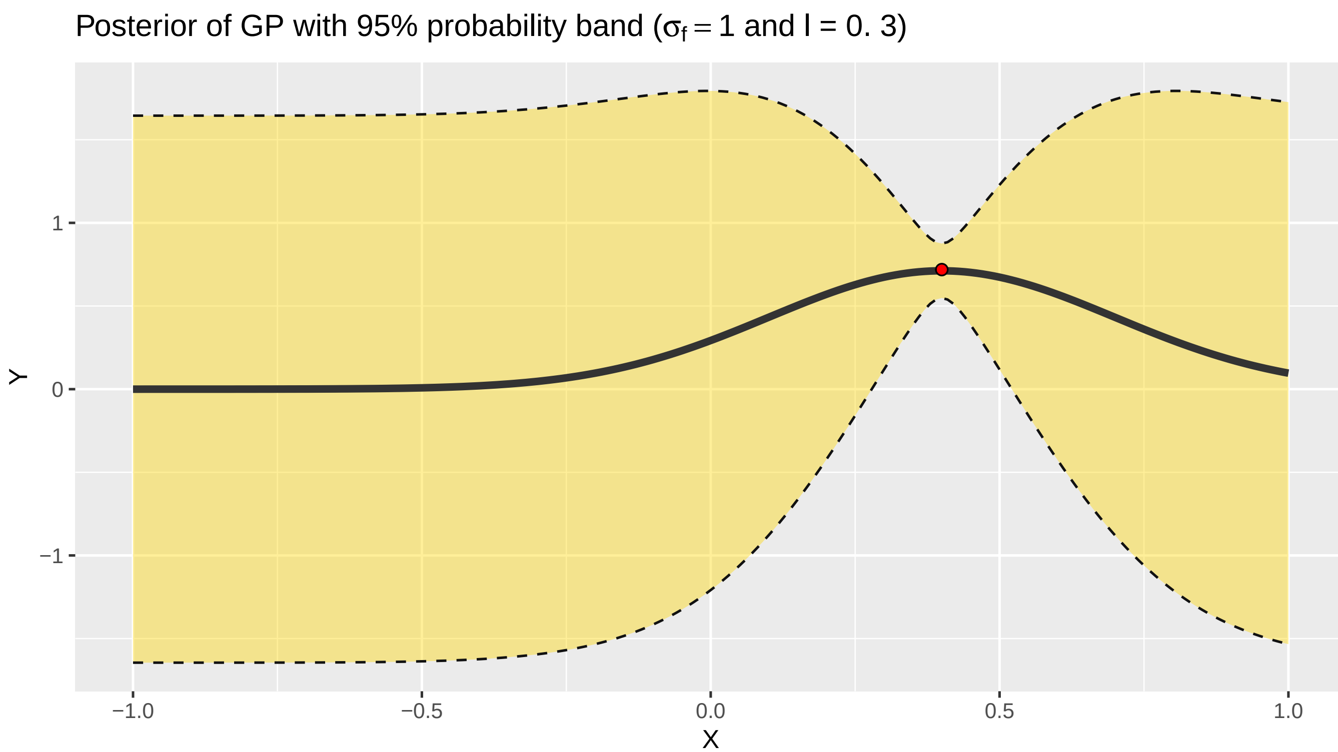
```r
posteriorGP = function(X, y, XStar, hyperParam, sigmaNoise){
  sigma_f = hyperParam[1]
  ell = hyperParam[2]
  n = length(X)

  K = squared_exp_kernel(X, X, sigma_f = sigma_f, l = ell)
  K_star = squared_exp_kernel(X, XStar, sigma_f = sigma_f, l = ell)

  L = t(chol(K + (sigmaNoise^2) * diag(x = 1, nrow = n)))
  alpha = solve(t(L), (solve(L, y, drop = F)), drop = F)
  f_star = t(K_star) %*% alpha
  v = solve(L, K_star)
  Var_f_star = squared_exp_kernel(XStar, XStar, sigma_f = sigma_f, l = ell) - (t(v) %*% v)
  log_ml = -0.5*t(y)%*%alpha - sum(diag(L)) - n*log(2*pi)/2

  return(list(mean = f_star, variance = Var_f_star, log_ml = log_ml))
}
```

1.2

Question: Now, 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 $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.

Answer:

```r
x = c(0.4)
y = c(0.719)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_2 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)
```



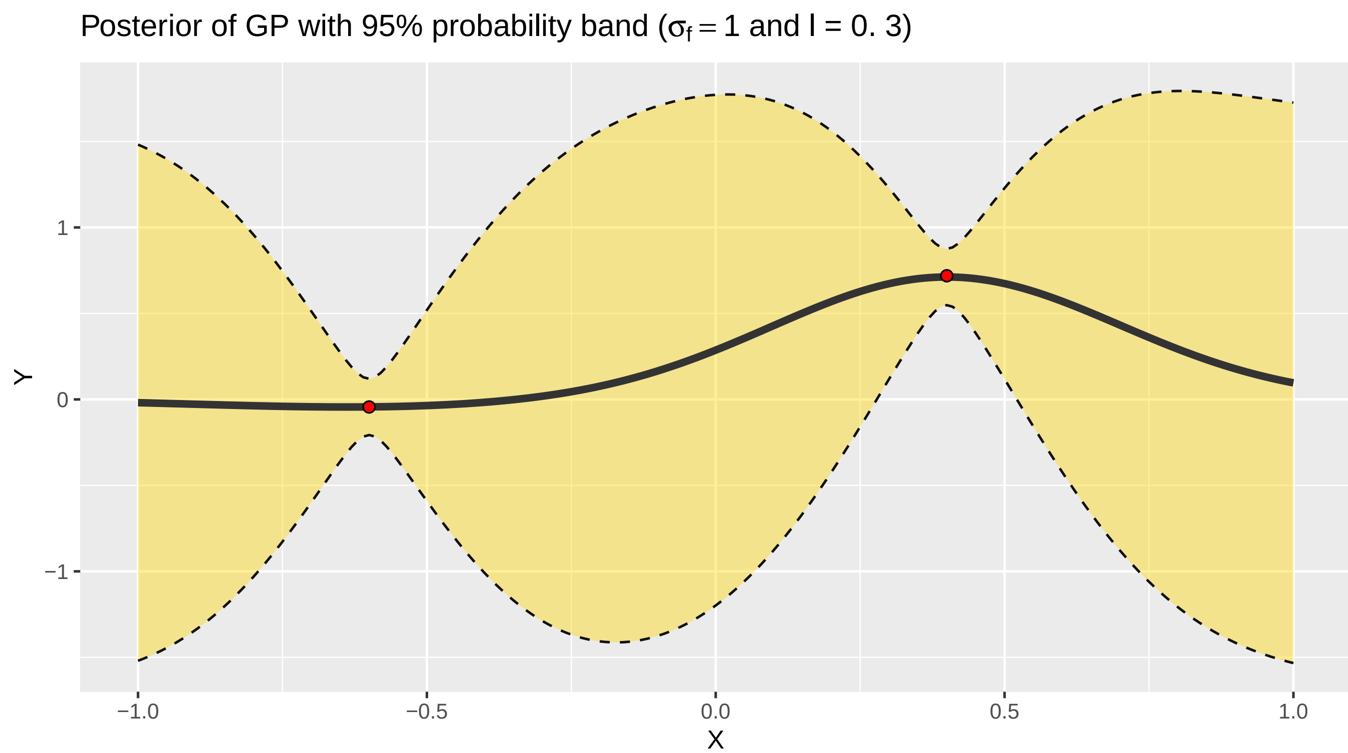
1.3

Question: 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.

Answer:

```
x = c(0.4, -0.6)
y = c(0.719, -0.044)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_3 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)
```



1.4

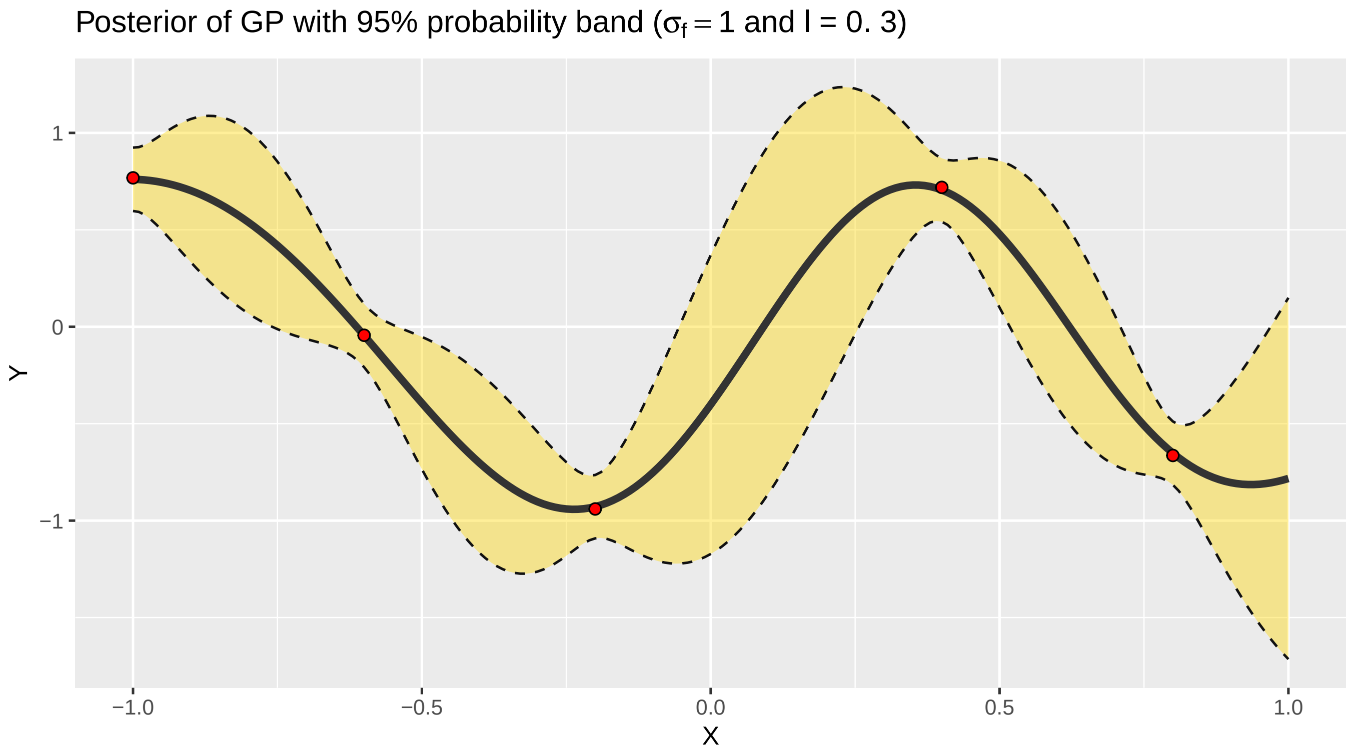
Question: Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval $x \in [-1, 1]$. Plot also 95 % probability (pointwise) bands for f.

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

Answer:

```
x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_4 = posteriorGP(X = x, y = y, XStar = x_star,
```

```
                      hyperParam = c(sigma_f = sigma_f, l = l),
                      sigmaNoise = 0.1)
```

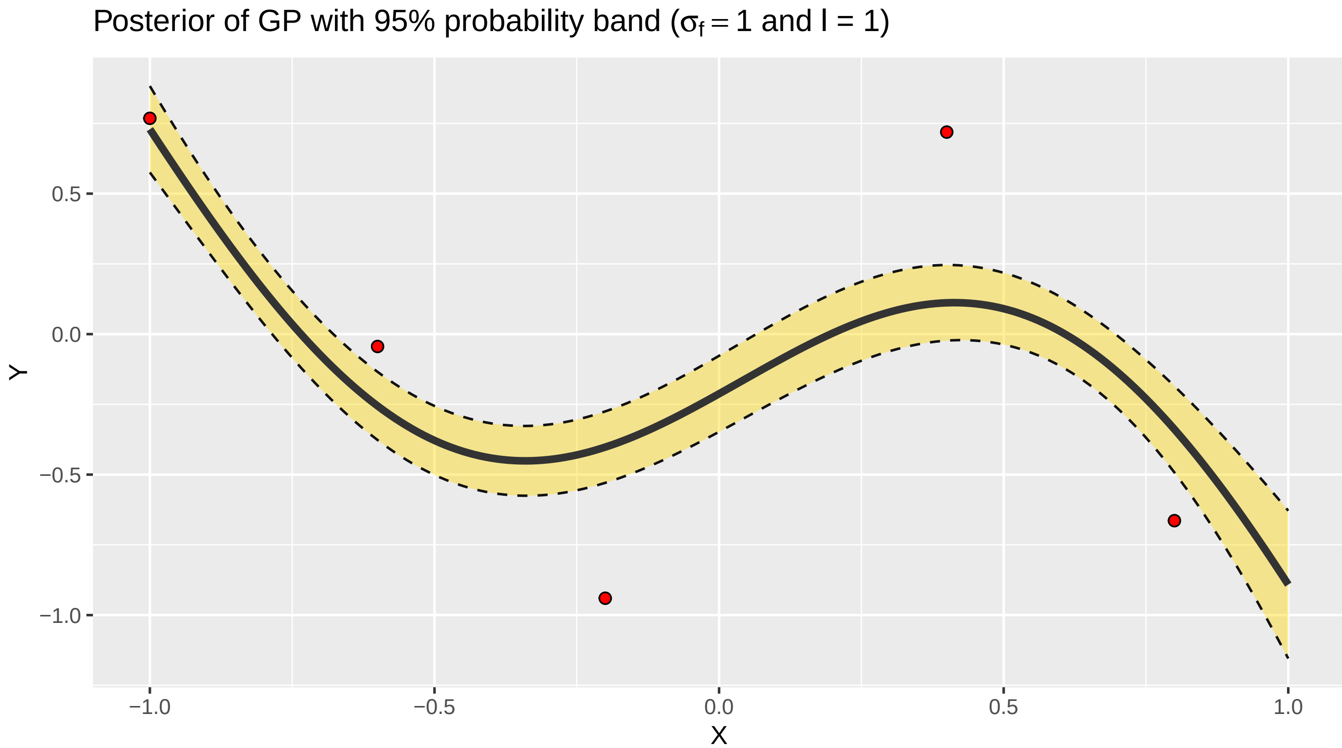


1.5

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

Answer:

```
x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 1

post_gp_5 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)
```



By comparing the plots obtained, we see that the posterior function gets too smooth when we use $\ell = 1$ compared to when we use $\ell = 0.3$. This is an expected behavior of the ℓ parameter of the squared exponential kernel that we have used. When $\ell = 0.3$, the posterior function mean perfectly passes through the data points that we used. Whereas when we used $\ell = 1$, we see that 4 out of the 5 data points fo not even lie within the 95% probability band. So, using $\ell = 1$ leads to worse results compared to using $\ell = 0.3$.

Question 2: GP Regression with `kernlab`

In this exercise, you will work with the daily mean temperature in Stockholm (Tullinge) during the period January 1, 2010 - December 31, 2015. We have removed the leap year day February 29, 2012 to make things simpler. You can read the dataset with the command:

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

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

```r
temp_tullinge = read.csv2("TempTullinge.csv", stringsAsFactors = F)

temp_tullinge$temp = as.numeric(temp_tullinge$temp)
temp_tullinge$time = 1:nrow(temp_tullinge)
temp_tullinge$day = rep(1:365, 6)
temp_tullinge$date = NULL

temp_tullinge = temp_tullinge[temp_tullinge$time %% 5 == 1 , ]

kable(tail(temp_tullinge), booktabs = T) %>%
  kable_styling(latex_option = "striped")
```

	temp	time	day
2161	0.1	2161	336
2166	6.3	2166	341
2171	0.1	2171	346
2176	2.4	2176	351
2181	5.4	2181	356
2186	-4.2	2186	361

2.1

Question: 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, 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 $K(X, X_\star)$ for the input vectors $X = (1, 3, 4)^T$ and $X_\star = (2, 3, 4)^T$.

Answer:

```r
get_SE_kernel = function(sigma_f, ell){
  kernel_func = function(x, x_star){
    (sigma_f^2) * exp(-((x - x_star)^2)/(2*(ell^2)))
  }

  class(kernel_func) = "kernel"

  return(kernel_func)
}

SE_kernel = get_SE_kernel(sigma_f = 1, ell = 1)
```

We evaluate the squared exponential kernel created above with parameters $\sigma_f = 1$ and $\ell = 1$ using $x = 1$ and $x' = 2$.

```
cat("Kernel value = ", SE_kernel(1, 2))
```

```
## Kernel value =  0.6065307
```

```
x = matrix(c(1,3,4), ncol = 1)
x_star = matrix(c(2,3,4), ncol = 1)
K = kernelMatrix(kernel = SE_kernel, x = x, y = x_star)

cat("Kernel matrix: ")
```

```
## Kernel matrix:
```

```
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
```

2.2

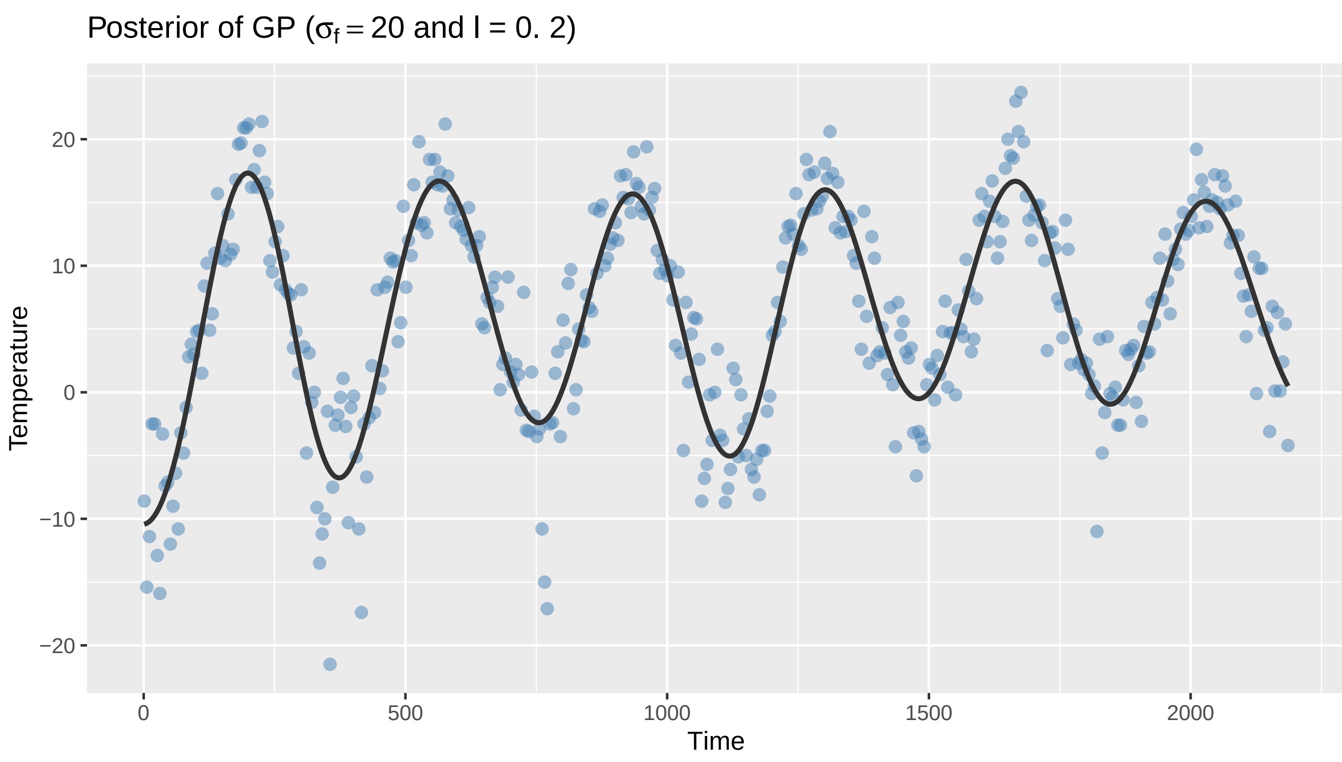
Question: Consider first the following model:

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

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 squared exponential function from (1) with $\sigma_f = 20$ and $\ell = 0.2$. Use the `predict` function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use `type="l"` in the plot function). Play around with different values on σ_f and ℓ (no need to write this in the report though).

Answer:

```r
SE_kernel = get_SE_kernel(sigma_f = 20, ell = 0.2)

lm_temp_time = lm(temp ~ time + I(time^2), data = temp_tullinge)
sigma_n_time = sd(lm_temp_time$residuals)

gp_time_fit = gausspr(temp_tullinge$time, temp_tullinge$temp, kernel = get_SE_kernel, kpar =
gp_time_pred = predict(gp_time_fit, temp_tullinge$time)
```



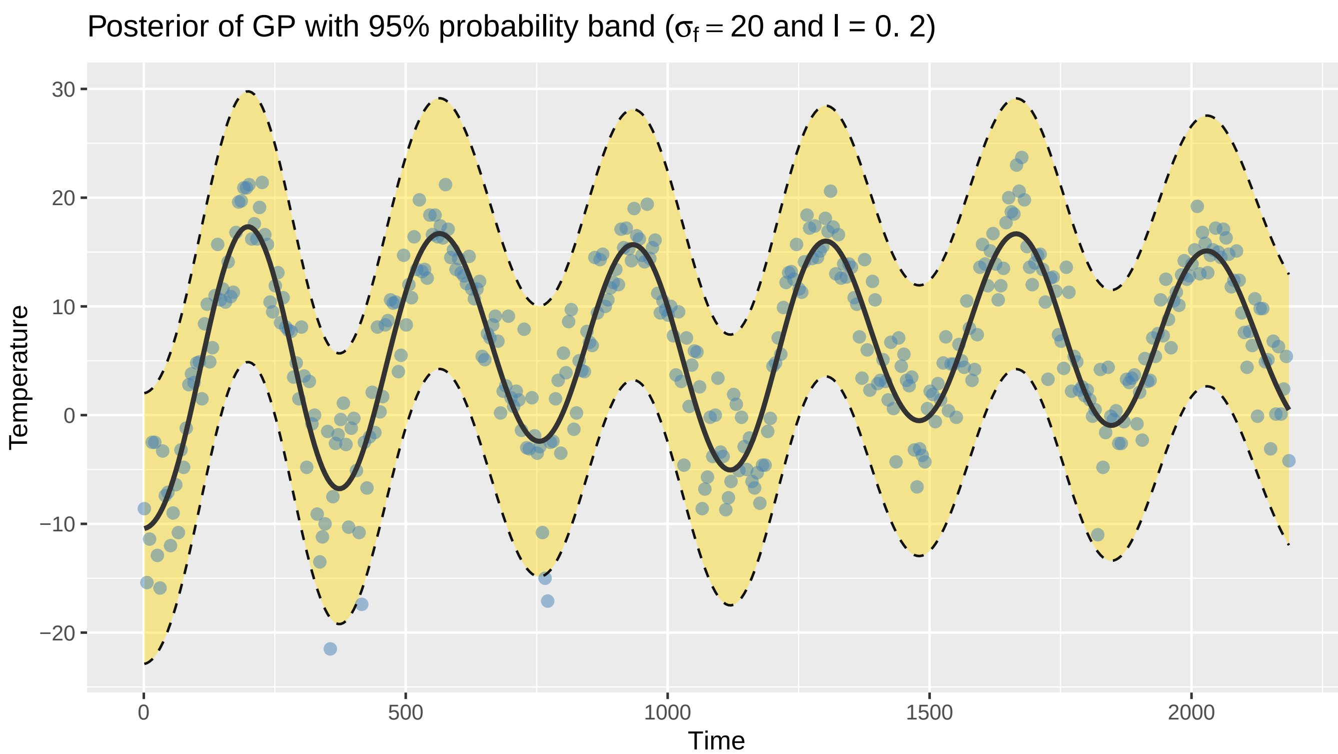
2.3

Question: `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 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` scales 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 output of `kernlab` unless you scale the data first. For this, you may want to use the R function `scale`.

Answer:

```
calc_variance = function(kernel, x, x_star, sigma_n){
  n = length(x)

  Kss = kernelMatrix(kernel = kernel, x = x_star, y = x_star)
  Kxx = kernelMatrix(kernel = kernel, x = x, y = x)
  Kxs = kernelMatrix(kernel = kernel, x = x, y = x_star)
  f_var = Kss-t(Kxs)%*%solve(Kxx + sigma_n^2*diag(n), Kxs)

  return(f_var)
}

gp_time_pred_var = calc_variance(SE_kernel, temp_tullinge$time, temp_tullinge$time, sigma_n
```



2.4

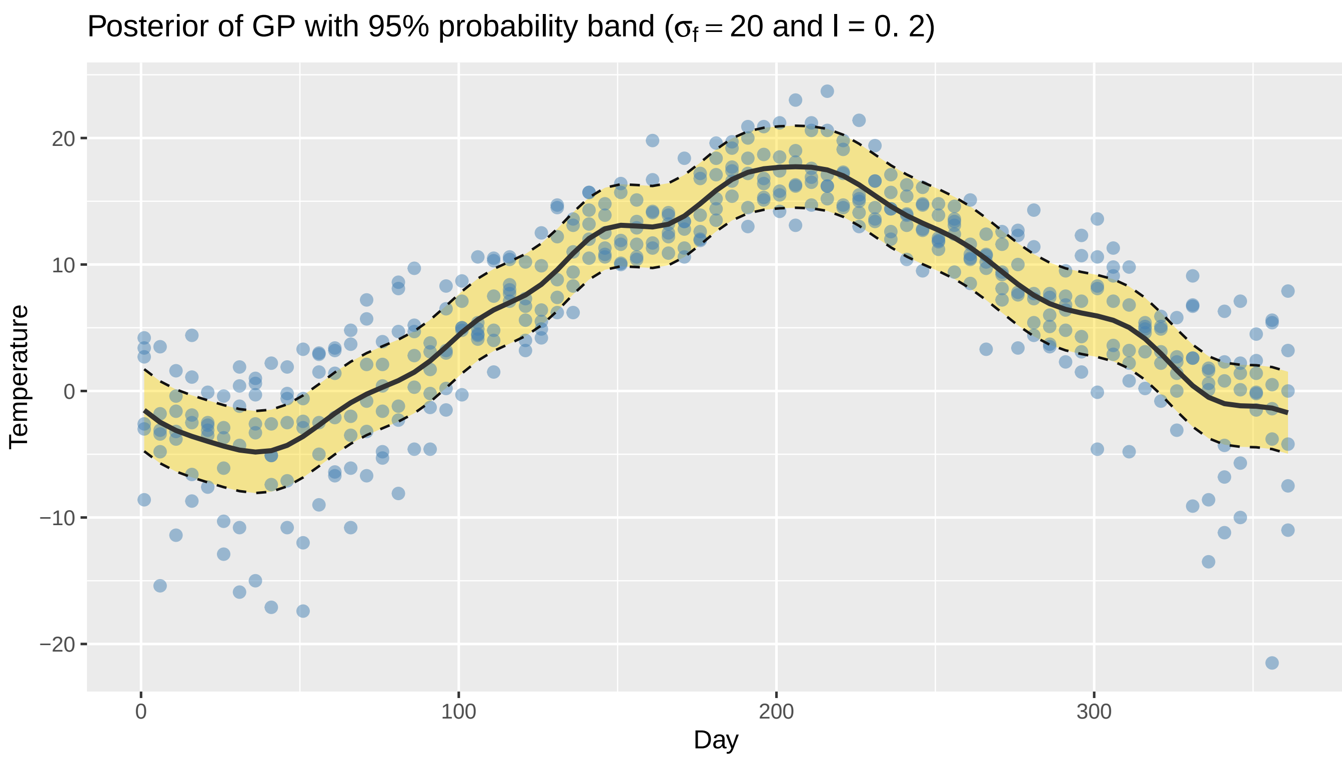
Question: Consider now the following model:

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

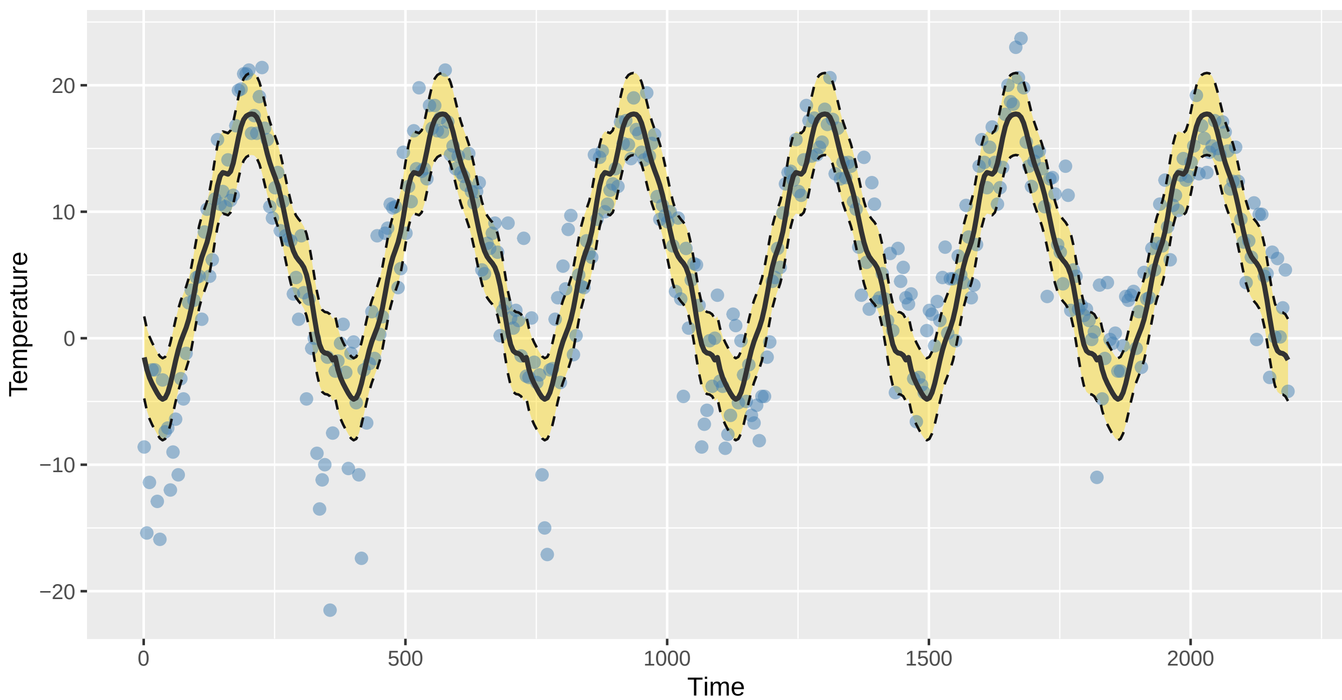
Estimate the model using the squared exponential function with $\sigma_f = 20$ and $\ell = 0.2$. Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis. Compare the results of both models. What are the pros and cons of each model?

Answer:

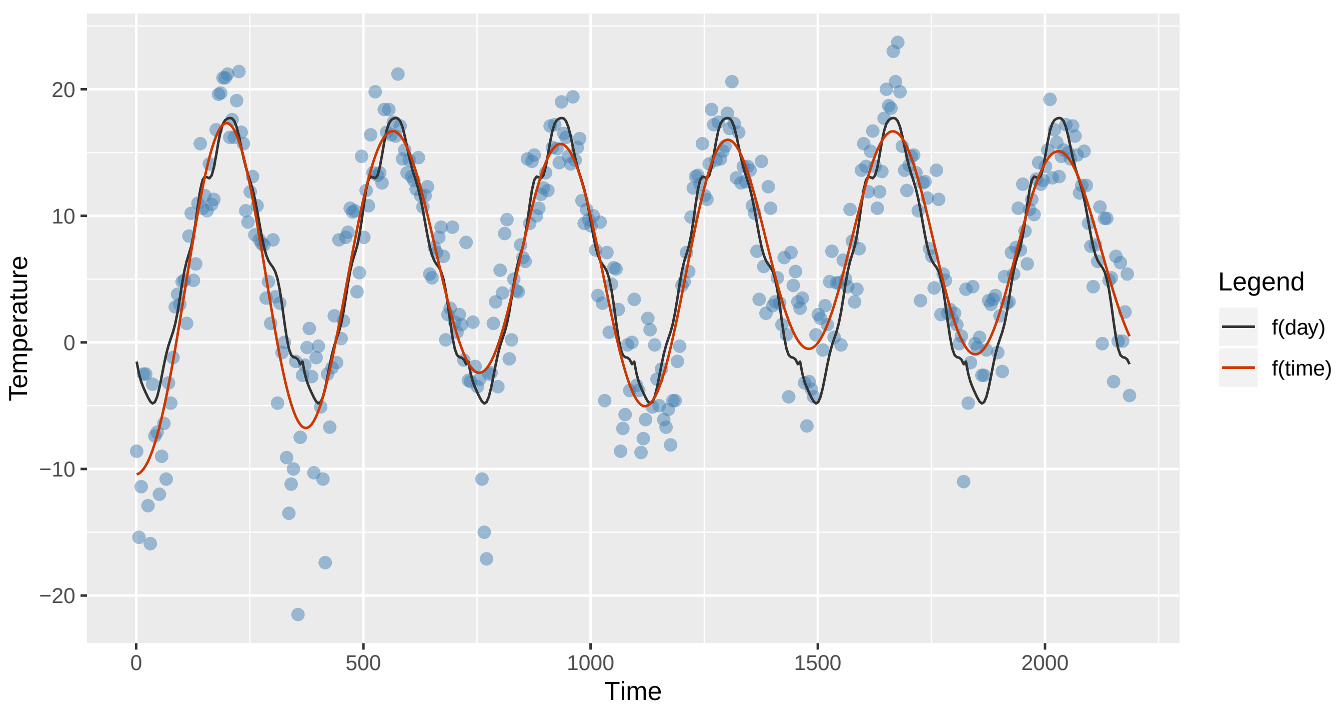
```
lm_temp_day = lm(temp ~ day + I(day^2), data = temp_tullinge)
sigma_n_day = sd(lm_temp_day$residuals)

gp_day_fit = gausspr(temp_tullinge$day, temp_tullinge$temp, kernel = get_SE_kernel, kpar =
gp_day_pred = predict(gp_day_fit, temp_tullinge$day)

gp_day_pred_var = calc_variance(SE_kernel, temp_tullinge$day, temp_tullinge$day, sigma_n_da
```



Posterior of GP with 95% probability band ($\sigma_f = 20$ and $l = 0.2$)



Comparison of posteriors of GP models f(time) and f(day) ($\sigma_f = 1$ and $l = 1$)



2.5

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

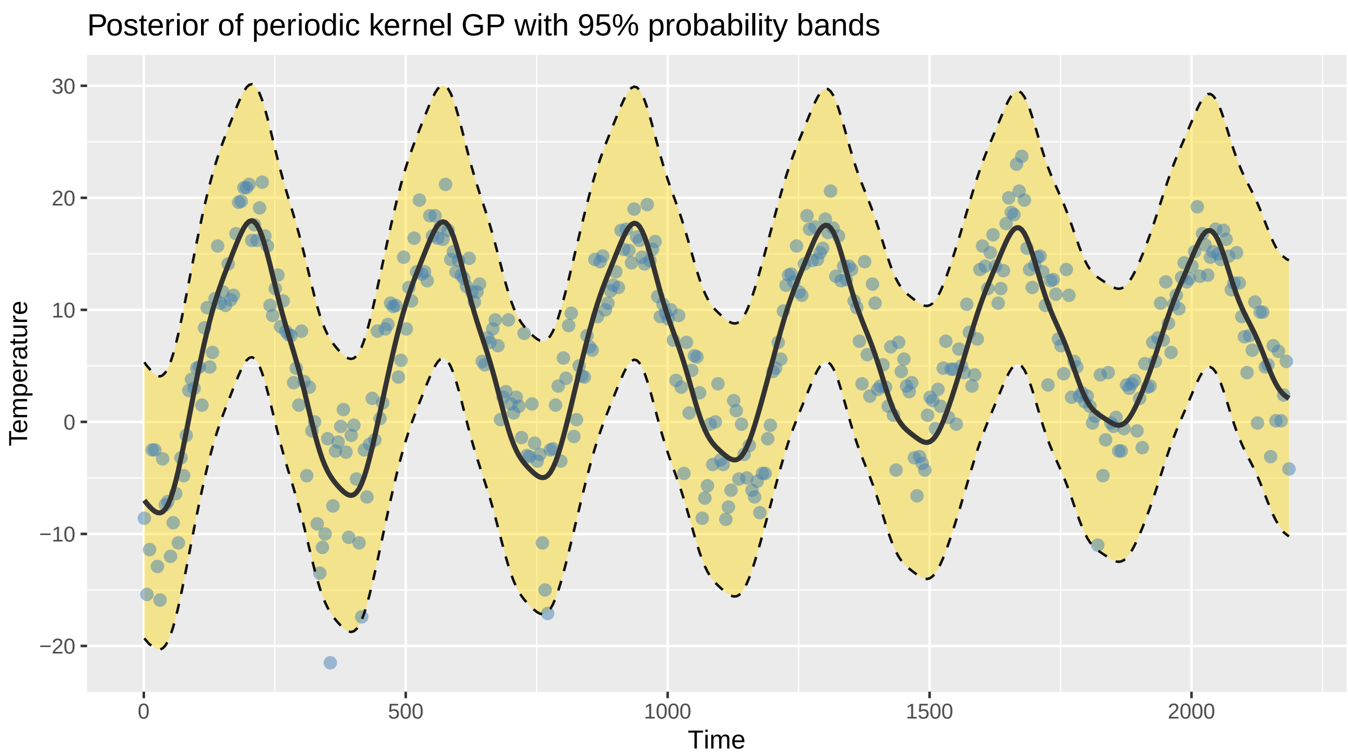
$$k\left(x, x'\right)=\sigma_f^2 \exp \left\{-\frac{2 \sin ^2\left(\pi\left|x-x'\right| / d\right)}{\ell_1^2}\right\} \exp \left\{-\frac{1}{2} \frac{\left|x-x'\right|^2}{\ell_2^2}\right\}$$

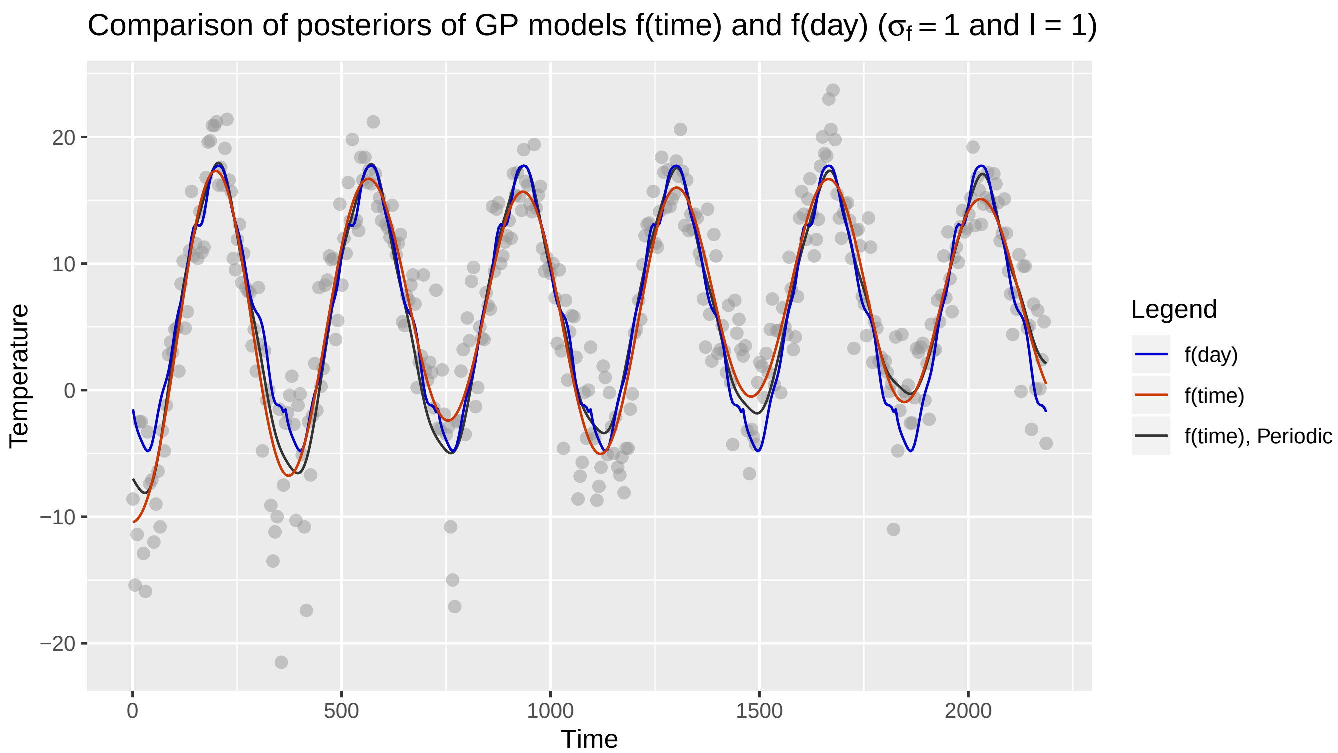
Note that we have two different length scales here, and ℓ_2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters $\sigma_f = 20$, $\ell_1 = 1$, $\ell_2 = 10$ and $d = 365/$`sd(time)`. The reason for the rather strange period here is that kernlab standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with $\sigma_f = 20$ and $\ell = 0.2$). Discuss the results.

Answer:

```
get_period_kernel = function(sigma_f, l1, l2, d){
  kernel_func = function(x, x_star){
    r = abs(x - x_star)
    (sigma_f^2) * exp(-(2*(sin(pi*r/d))^2)/(l1^2)) * exp(-(r^2)/(2*(l2^2)))
  }

  class(kernel_func) = "kernel"

  return(kernel_func)
}
```

```
kernel_params = list(sigma_f = 20, l1 = 1, l2 = 10, d = 365/sd(temp_tullinge$time))

gp_period_fit = gausspr(temp_tullinge$time, temp_tullinge$temp, kernel = get_period_kernel,
gp_period_pred = predict(gp_period_fit, temp_tullinge$time)

period_kernel = get_period_kernel(sigma_f = 20, l1 = 1, l2 = 10, d = 365/sd(temp_tullinge$t
gp_period_pred_var = calc_variance(period_kernel, temp_tullinge$time, temp_tullinge$time, s
```

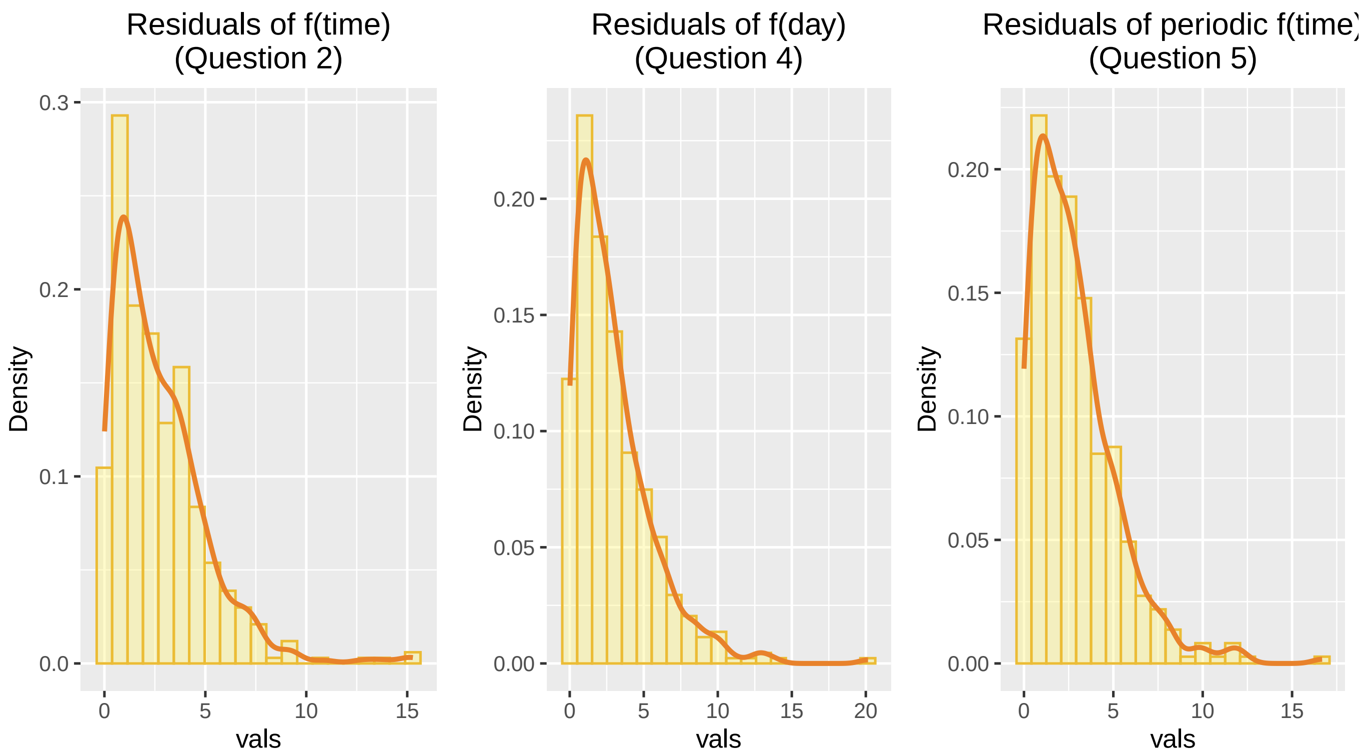





```
res_ftime = abs(temp_tullinge$temp - gp_time_pred)
res_fday = abs(temp_tullinge$temp - gp_day_pred)
res_ftime_period = abs(temp_tullinge$temp - gp_period_pred)
```

Table 1: Model Residuals

Gaussian Process	Residual Mean
f(time)	2.697859
f(day)	2.943640
Periodic f(time)	2.801839



Question 3: GP Classification with `kernlab`

Download the banknote fraud data:

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/ GaussianProcess/Code/b
header=FALSE, sep=",")
```

```
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)
```

```r
bn_data = read.csv("banknoteFraud.csv", header = F, sep=",")
names(bn_data) = c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
bn_data[ , 5] = as.factor(bn_data[ , 5])

set.seed(111)
SelectTraining = sample(1:dim(bn_data)[1], size = 1000, replace = FALSE)
bn_train = bn_data[SelectTraining, ]
bn_test = bn_data[-SelectTraining, ]
```

	varWave	skewWave	kurtWave	entropyWave	fraud
814	-2.1333	1.5685	-0.084261	-1.74530	1
997	-2.3142	2.0838	-0.468130	-1.67670	1
508	4.6014	5.6264	-2.123500	0.19309	0
705	3.7022	6.9942	-1.851100	-0.12889	0
517	-2.3983	12.6060	2.946400	-5.78880	0
572	2.2517	-5.1422	4.291600	-1.24870	0

3.1

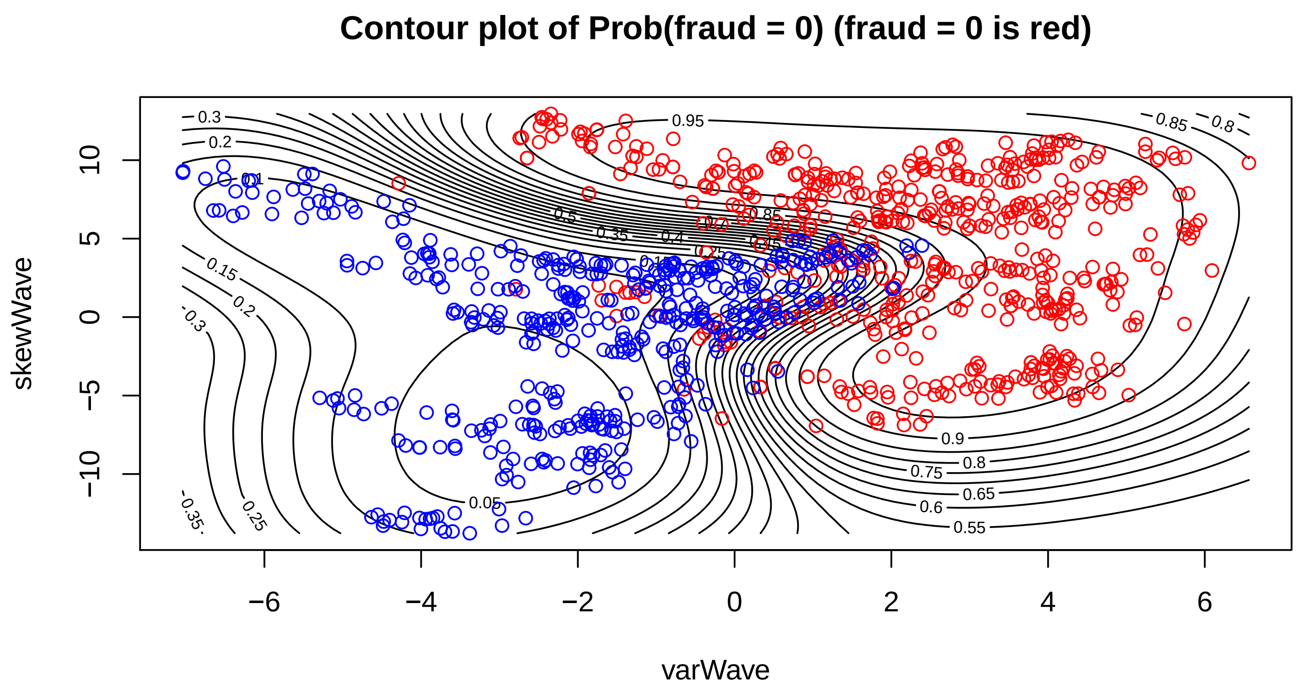
Question: Use the R package `kernlab` to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates `varWave` and `skewWave` in the model. Plot contours of the prediction probabilities over a suitable grid of 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.

Answer:

```r
bn_gp_fit = gausspr(fraud ~ varWave + skewWave, data = bn_train)
```

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

```r
pred_train = predict(bn_gp_fit, bn_train)
```




```r
confusionMatrix(pred_train, bn_train$fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 512  24
##          1  44 420
##
##                Accuracy : 0.932
##                  95% CI : (0.9146, 0.9468)
##     No Information Rate : 0.556
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.8629
##
##  Mcnemar's Test P-Value : 0.02122
##
##             Sensitivity : 0.9209
##             Specificity : 0.9459
##          Pos Pred Value : 0.9552
##          Neg Pred Value : 0.9052
##              Prevalence : 0.5560
##          Detection Rate : 0.5120
##    Detection Prevalence : 0.5360
##       Balanced Accuracy : 0.9334
```

```
##
##        'Positive' Class : 0
##
```

3.2

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

Answer:

```r
pred_test = predict(bn_gp_fit, bn_test)
confusionMatrix(pred_test, bn_test$fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 191   9
##          1  15 157
##
##                Accuracy : 0.9355
##                  95% CI : (0.9055, 0.9582)
##     No Information Rate : 0.5538
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.8699
##
##  Mcnemar's Test P-Value : 0.3074
##
##             Sensitivity : 0.9272
##             Specificity : 0.9458
##          Pos Pred Value : 0.9550
##          Neg Pred Value : 0.9128
##              Prevalence : 0.5538
##          Detection Rate : 0.5134
##    Detection Prevalence : 0.5376
##       Balanced Accuracy : 0.9365
##
##        'Positive' Class : 0
##
```

3.3

Question: 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:

```
bn_gp_all_fit = gausspr(fraud ~ ., data = bn_train)
```

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

```
pred_all_train = predict(bn_gp_all_fit, bn_train)
pred_all_test = predict(bn_gp_all_fit, bn_test)

confusionMatrix(pred_all_test, bn_test$fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 205   0
##          1   1 166
##
##                Accuracy : 0.9973
##                  95% CI : (0.9851, 0.9999)
##     No Information Rate : 0.5538
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.9946
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9951
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.9940
##              Prevalence : 0.5538
##          Detection Rate : 0.5511
##    Detection Prevalence : 0.5511
##       Balanced Accuracy : 0.9976
##
##        'Positive' Class : 0
##
```

```r
acc_2_cov_train = mean(pred_train == bn_train$fraud)
acc_2_cov_test = mean(pred_test == bn_test$fraud)
acc_all_cov_train = mean(pred_all_train == bn_train$fraud)
acc_all_cov_test = mean(pred_all_test == bn_test$fraud)
```

Table 2: Model Accuracies

Covariates	Train	Test
varWave, skewWave	0.932	0.9354839
All	0.996	0.9973118

Appendix

```r
# Set up general options

knitr::opts_chunk$set(echo = FALSE, warning = FALSE, message = FALSE, fig.width=8)

set.seed(123456)

library(ggplot2)
options(kableExtra.latex.load_packages = FALSE)
library(kableExtra)
library(caret)
library(gridExtra)
library(kernlab)
library(mvtnorm)
library(latex2exp)
library(AtmRay)

options(scipen=999)

# Function to plot histogram and density of a sample
hist_plt = function(vals){
  n_bins = ceiling(sqrt(length(vals)))
  plt = qplot(vals, geom = 'blank') +
    geom_histogram(aes(y = ..density..), alpha = 0.4,
                   bins = n_bins, color = "#EBBC36", fill = "#FFF57D") +
    geom_line(aes(y = ..density.., color = 'Empirical Density'),
              stat = 'density', size = 1, color = "#E8822B") +
    theme(plot.title = element_text(hjust = 0.5),
          plot.subtitle = element_text(hjust = 0.5)) + ylab("Density")
```

```
  return(plt)
}

# Function to plot posterior of Gaussian Process
post_gp_plot = function(x, y, x_star, y_star, variance, pred_conf, sigma_f, l){
  bw = qnorm(p = pred_conf)
  y_min = y_star - bw*sqrt(diag(variance))
  y_max = y_star + bw*sqrt(diag(variance))

  ggplot() +
    geom_ribbon(aes(x = x_star, ymin = y_min, ymax = y_max), fill = "gold", alpha = 0.4) +
    geom_line(aes(x = x_star, y = y_star), color = "#333333", size = 1.5) +
    geom_line(aes(x = x_star, y = y_min), color = "#111111", linetype = "dashed") +
    geom_line(aes(x = x_star, y = y_max), color = "#111111", linetype = "dashed") +
    geom_point(aes(x = x, y = y), shape = 21, fill = "red", size = 2) +
    labs(x = "X", y = "Y") +
    ggtitle(TeX(paste0("Posterior of GP with ", round(100*pred_conf, 1),
                       "% probability band ($\\sigma_f = ", sigma_f, "$ and l = ", l, ")")))

}

# Function to plot posterior of Gaussian Process
post_gp_scatterplot = function(x, y, x_star, y_star, variance, pred_conf, sigma_f, l, axis_
  plt = NA

  if(is.na(variance)){
    plt = ggplot() +
      geom_point(aes(x = x, y = y), color = "steelblue", alpha = 0.5, size = 2) +
      geom_line(aes(x = x_star, y = y_star), color = "#333333", size = 1) +
      labs(x = axis_titles[1], y = axis_titles[2]) +
      ggtitle(TeX(paste0("Posterior of GP ($\\sigma_f = ", sigma_f, "$ and l = ", l, ")")))
  }
  else{
    bw = qnorm(p = pred_conf)
    y_min = y_star - bw*sqrt(diag(variance))
    y_max = y_star + bw*sqrt(diag(variance))

    plt = ggplot() +
      geom_ribbon(aes(x = x_star, ymin = y_min, ymax = y_max), fill = "gold", alpha = 0.4)
      geom_point(aes(x = x, y = y), color = "steelblue", alpha = 0.5, size = 2) +
      geom_line(aes(x = x_star, y = y_star), color = "#333333", size = 1) +
      geom_line(aes(x = x_star, y = y_min), color = "#111111", linetype = "dashed") +
      geom_line(aes(x = x_star, y = y_max), color = "#111111", linetype = "dashed") +
      labs(x = axis_titles[1], y = axis_titles[2]) +
      ggtitle(TeX(paste0("Posterior of GP with ", round(100*pred_conf, 1),
                         "% probability band ($\\sigma_f = ", sigma_f, "$ and l = ", l, ")"
  }
```

```r
  return(plt)
}



# --------------------------------------------------------------------------
# Question 1.1
# --------------------------------------------------------------------------

# Covariance function
squared_exp_kernel = function(x1, x2, sigma_f = 1, l = 3){
  n1 = length(x1)
  n2 = length(x2)
  K = matrix(NA,n1,n2)

  for (i in 1:n2){
    K[,i] = (sigma_f^2)*exp(-0.5*( (x1-x2[i])/l)^2 )
  }

  return(K)
}

posteriorGP = function(X, y, XStar, hyperParam, sigmaNoise){
  sigma_f = hyperParam[1]
  ell = hyperParam[2]
  n = length(X)

  K = squared_exp_kernel(X, X, sigma_f = sigma_f, l = ell)
  K_star = squared_exp_kernel(X, XStar, sigma_f = sigma_f, l = ell)

  L = t(chol(K + (sigmaNoise^2) * diag(x = 1, nrow = n)))
  alpha = solve(t(L), (solve(L, y, drop = F)), drop = F)
  f_star = t(K_star) %*% alpha
  v = solve(L, K_star)
  Var_f_star = squared_exp_kernel(XStar, XStar, sigma_f = sigma_f, l = ell) - (t(v) %*% v)
  log_ml = -0.5*t(y)%*%alpha - sum(diag(L)) - n*log(2*pi)/2

  return(list(mean = f_star, variance = Var_f_star, log_ml = log_ml))
}


# --------------------------------------------------------------------------
# Question 1.2
# --------------------------------------------------------------------------

x = c(0.4)
y = c(0.719)
```

```r
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_2 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)

post_gp_plot(x, y, x_star, post_gp_2$mean, post_gp_2$variance, 0.95, sigma_f, l)


# ------------------------------------------------------------------------------
# Question 1.3
# ------------------------------------------------------------------------------

x = c(0.4, -0.6)
y = c(0.719, -0.044)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_3 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)

post_gp_plot(x, y, x_star, post_gp_3$mean, post_gp_3$variance, 0.95, sigma_f, l)

kable(data.frame(x = seq(-1, 0.8, 0.4),
                 y = c(0.768, -0.044, -0.940, 0.719, -0.664)))


# ------------------------------------------------------------------------------
# Question 1.4
# ------------------------------------------------------------------------------

x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 0.3

post_gp_4 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)

post_gp_plot(x, y, x_star, post_gp_4$mean, post_gp_4$variance, 0.95, sigma_f, l)
```

```r
# ------------------------------------------------------------------------------
# Question 1.5
# ------------------------------------------------------------------------------

x = c(-1, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
x_star = seq(-1, 1, 0.01)
sigma_f = 1
l = 1

post_gp_5 = posteriorGP(X = x, y = y, XStar = x_star,
                        hyperParam = c(sigma_f = sigma_f, l = l),
                        sigmaNoise = 0.1)

post_gp_plot(x, y, x_star, post_gp_5$mean, post_gp_5$variance, 0.95, sigma_f, l)

temp_tullinge = read.csv2("TempTullinge.csv", stringsAsFactors = F)

temp_tullinge$temp = as.numeric(temp_tullinge$temp)
temp_tullinge$time = 1:nrow(temp_tullinge)
temp_tullinge$day = rep(1:365, 6)
temp_tullinge$date = NULL

temp_tullinge = temp_tullinge[temp_tullinge$time %% 5 == 1 , ]

kable(tail(temp_tullinge), booktabs = T) %>%
  kable_styling(latex_option = "striped")

# ------------------------------------------------------------------------------
# Question 2.1
# ------------------------------------------------------------------------------

get_SE_kernel = function(sigma_f, ell){
  kernel_func = function(x, x_star){
    (sigma_f^2) * exp(-((x - x_star)^2)/(2*(ell^2)))
  }

  class(kernel_func) = "kernel"

  return(kernel_func)
}

SE_kernel = get_SE_kernel(sigma_f = 1, ell = 1)

cat("Kernel value = ", SE_kernel(1, 2))

x = matrix(c(1,3,4), ncol = 1)
```

```
x_star = matrix(c(2,3,4), ncol = 1)
K = kernelMatrix(kernel = SE_kernel, x = x, y = x_star)

cat("Kernel matrix: ")
print(K)


# ----------------------------------------------------------------------------
# Question 2.2
# ----------------------------------------------------------------------------

SE_kernel = get_SE_kernel(sigma_f = 20, ell = 0.2)

lm_temp_time = lm(temp ~ time + I(time^2), data = temp_tullinge)
sigma_n_time = sd(lm_temp_time$residuals)

gp_time_fit = gausspr(temp_tullinge$time, temp_tullinge$temp, kernel = get_SE_kernel, kpar =
gp_time_pred = predict(gp_time_fit, temp_tullinge$time)

post_gp_scatterplot(temp_tullinge$time, temp_tullinge$temp, temp_tullinge$time, gp_time_pre


# ----------------------------------------------------------------------------
# Question 2.3
# ----------------------------------------------------------------------------

calc_variance = function(kernel, x, x_star, sigma_n){
  n = length(x)

  Kss = kernelMatrix(kernel = kernel, x = x_star, y = x_star)
  Kxx = kernelMatrix(kernel = kernel, x = x, y = x)
  Kxs = kernelMatrix(kernel = kernel, x = x, y = x_star)
  f_var = Kss-t(Kxs)%*%solve(Kxx + sigma_n^2*diag(n), Kxs)

  return(f_var)
}

gp_time_pred_var = calc_variance(SE_kernel, temp_tullinge$time, temp_tullinge$time, sigma_n

post_gp_scatterplot(temp_tullinge$time, temp_tullinge$temp, temp_tullinge$time, gp_time_pre


# ----------------------------------------------------------------------------
# Question 2.4
# ----------------------------------------------------------------------------

lm_temp_day = lm(temp ~ day + I(day^2), data = temp_tullinge)
```

```r
sigma_n_day = sd(lm_temp_day$residuals)

gp_day_fit = gausspr(temp_tullinge$day, temp_tullinge$temp, kernel = get_SE_kernel, kpar =
gp_day_pred = predict(gp_day_fit, temp_tullinge$day)

gp_day_pred_var = calc_variance(SE_kernel, temp_tullinge$day, temp_tullinge$day, sigma_n_da

post_gp_scatterplot(temp_tullinge$day, temp_tullinge$temp, temp_tullinge$day, gp_day_pred,

post_gp_scatterplot(temp_tullinge$time, temp_tullinge$temp, temp_tullinge$time, gp_day_pred

ggplot() +
  geom_point(aes(x = time, y = temp), data = temp_tullinge, color = "steelblue", alpha = 0.
  geom_line(aes(x = time, y = gp_day_pred, color = "f(day)"), data = temp_tullinge) +
  geom_line(aes(x = time, y = gp_time_pred, color = "f(time)"), data = temp_tullinge) +
  labs(x = "Time", y = "Temperature", color = "Legend") +
  scale_color_manual(values = c("f(time)" = "orangered3", "f(day)" = "#333333")) +
  ggtitle(TeX(paste0("Comparison of posteriors of GP models f(time) and f(day) ", "($\\sigm


# ------------------------------------------------------------------------------
# Question 2.5
# ------------------------------------------------------------------------------

get_period_kernel = function(sigma_f, l1, l2, d){
  kernel_func = function(x, x_star){
    r = abs(x - x_star)
    (sigma_f^2) * exp(-(2*(sin(pi*r/d))^2)/(l1^2)) * exp(-(r^2)/(2*(l2^2)))
  }

  class(kernel_func) = "kernel"

  return(kernel_func)
}

kernel_params = list(sigma_f = 20, l1 = 1, l2 = 10, d = 365/sd(temp_tullinge$time))

gp_period_fit = gausspr(temp_tullinge$time, temp_tullinge$temp, kernel = get_period_kernel,
gp_period_pred = predict(gp_period_fit, temp_tullinge$time)

period_kernel = get_period_kernel(sigma_f = 20, l1 = 1, l2 = 10, d = 365/sd(temp_tullinge$t
gp_period_pred_var = calc_variance(period_kernel, temp_tullinge$time, temp_tullinge$time, s

post_gp_scatterplot(temp_tullinge$time, temp_tullinge$temp, temp_tullinge$time, gp_period_p

ggplot() +
  geom_point(aes(x = time, y = temp), data = temp_tullinge, color = "#999999", alpha = 0.5,
```

```
  geom_line(aes(x = time, y = gp_period_pred, color = "f(time), Periodic"), data = temp_tul
  geom_line(aes(x = time, y = gp_day_pred, color = "f(day)"), data = temp_tullinge) +
  geom_line(aes(x = time, y = gp_time_pred, color = "f(time)"), data = temp_tullinge) +
  labs(x = "Time", y = "Temperature", color = "Legend") +
  scale_color_manual(values = c("f(time)" = "orangered3", "f(day)" = "mediumblue", "f(time)
  ggtitle(TeX(paste0("Comparison of posteriors of GP models f(time) and f(day) ", "($\\sigm

res_ftime = abs(temp_tullinge$temp - gp_time_pred)
res_fday = abs(temp_tullinge$temp - gp_day_pred)
res_ftime_period = abs(temp_tullinge$temp - gp_period_pred)

hplt_1 = hist_plt(res_ftime) + ggtitle("Residuals of f(time)\n(Question 2)")
hplt_2 = hist_plt(res_fday) + ggtitle("Residuals of f(day)\n(Question 4)")
hplt_3 = hist_plt(res_ftime_period) + ggtitle("Residuals of periodic f(time)\n(Question 5)"

grid.arrange(hplt_1, hplt_2, hplt_3, nrow = 1)

res_df = data.frame(gp = c("f(time)", "f(day)", "Periodic f(time)"), mean_res = c(mean(res_
kable(res_df, booktabs = T, col.names = c("Gaussian Process", "Residual Mean"), caption = "
  kable_styling(latex_option = "striped")

bn_data = read.csv("banknoteFraud.csv", header = F, sep=",")
names(bn_data) = c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
bn_data[ , 5] = as.factor(bn_data[ , 5])

set.seed(111)
SelectTraining = sample(1:dim(bn_data)[1], size = 1000, replace = FALSE)
bn_train = bn_data[SelectTraining, ]
bn_test = bn_data[-SelectTraining, ]

kable(head(bn_train), booktabs = T) %>%
  kable_styling(latex_option = "striped")

# ------------------------------------------------------------------------------
# Question 3.1
# ------------------------------------------------------------------------------

bn_gp_fit = gausspr(fraud ~ varWave + skewWave, data = bn_train)
pred_train = predict(bn_gp_fit, bn_train)

x1 = seq(min(bn_train$varWave), max(bn_train$varWave), length = 100)
x2 = seq(min(bn_train$skewWave), max(bn_train$skewWave), length = 100)
grid_points = meshgrid(x1, x2)
grid_points = cbind(c(grid_points$x), c(grid_points$y))

grid_points = data.frame(grid_points)
```

```r
names(grid_points) = c("varWave", "skewWave")
grid_probs = predict(bn_gp_fit, grid_points, type = "probabilities")

contour(x1, x2, matrix(grid_probs[, 1], 100, byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = "Contour plot of Prob(fraud = 0) (fraud = 0 is red)")
points(bn_train[bn_train$fraud == 0, "varWave"], bn_train[bn_train$fraud == 0, "skewWave"],
points(bn_train[bn_train$fraud == 1, "varWave"], bn_train[bn_train$fraud == 1, "skewWave"],

confusionMatrix(pred_train, bn_train$fraud)


# ------------------------------------------------------------------------------
# Question 3.2
# ------------------------------------------------------------------------------

pred_test = predict(bn_gp_fit, bn_test)
confusionMatrix(pred_test, bn_test$fraud)


# ------------------------------------------------------------------------------
# Question 3.3
# ------------------------------------------------------------------------------

bn_gp_all_fit = gausspr(fraud ~ ., data = bn_train)
pred_all_train = predict(bn_gp_all_fit, bn_train)
pred_all_test = predict(bn_gp_all_fit, bn_test)

confusionMatrix(pred_all_test, bn_test$fraud)

acc_2_cov_train = mean(pred_train == bn_train$fraud)
acc_2_cov_test = mean(pred_test == bn_test$fraud)
acc_all_cov_train = mean(pred_all_train == bn_train$fraud)
acc_all_cov_test = mean(pred_all_test == bn_test$fraud)

acc_df = data.frame(Covariates = c("varWave, skewWave", "All"),
                    Train = c(acc_2_cov_train, acc_all_cov_train),
                    Test = c(acc_2_cov_test, acc_all_cov_test))

kable(acc_df, booktabs = T, longtable = T, caption = "Model Accuracies") %>%
  kable_styling(latex_option = "striped")
```