Lab 4: Gaussian processes

Martynas Lukosevicius (marlu207), Alejo Perez Gomez, Joel Nilsson

09/10/2021

Statement of Contribution

- code: Martynas Lukosevicius
- Answers to questions: Joel Nilsson, Alejo Perez Gomez, Martynas Lukosevicius

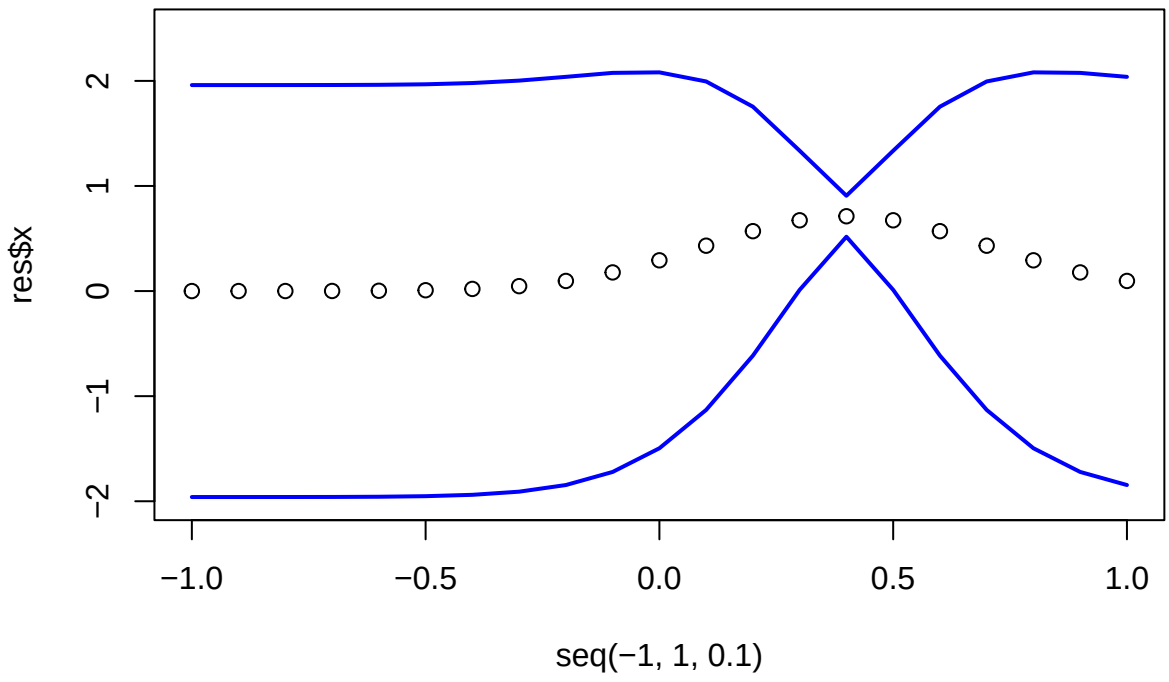
1. Implementing GP Regression.

1. Write your own code for simulating from the posterior distribution of f using the squared exponential kernel.

```
posteriorGP <- function(X,y,XStar,sigmaNoise, k,...){
  Ksigma <- k(X,X,...) + (sigmaNoise^2 * diag(length(X)))
  L <- t(chol(Ksigma))
  alpha <- solve(t(L),solve(L,y))
  kstar <- k(X,XStar,...)
  f_pred <- t(kstar) %*% alpha
  v <- solve(L,kstar)
  V <- k(XStar,XStar,...) - (t(v) %*% v)
  return(list(x = f_pred, var = V))
}
```

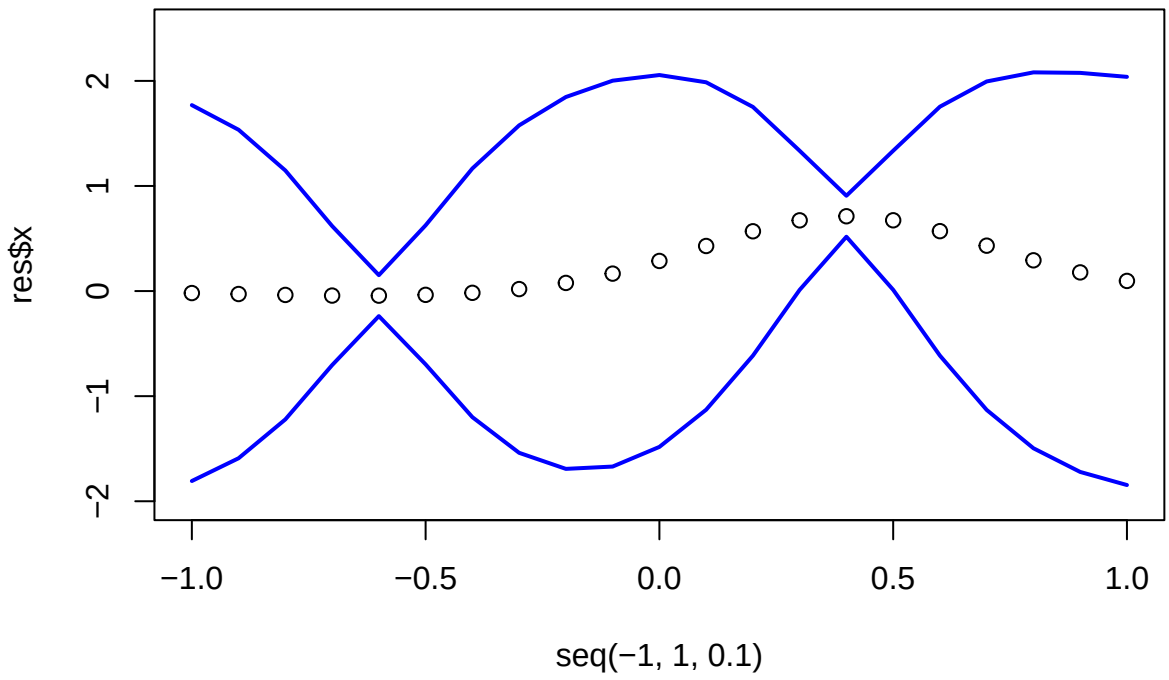
2. Now, let the prior hyperparameters be $\sigma_f = 1$ and $l = 0.3$. Update this prior with a single observation: $(x, y) = (0.4, 0.719)$. Assume that $\sigma_n = 0.1$. Plot the posterior mean of f over the interval x in [-1, 1]. Plot also 95 % probability (pointwise) bands for f.

```
res <- posteriorGP(0.4,0.719, seq(-1,1,0.1), 0.1,
                   SquaredExpKernel, sigmaF=1,l=0.3)
plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
```



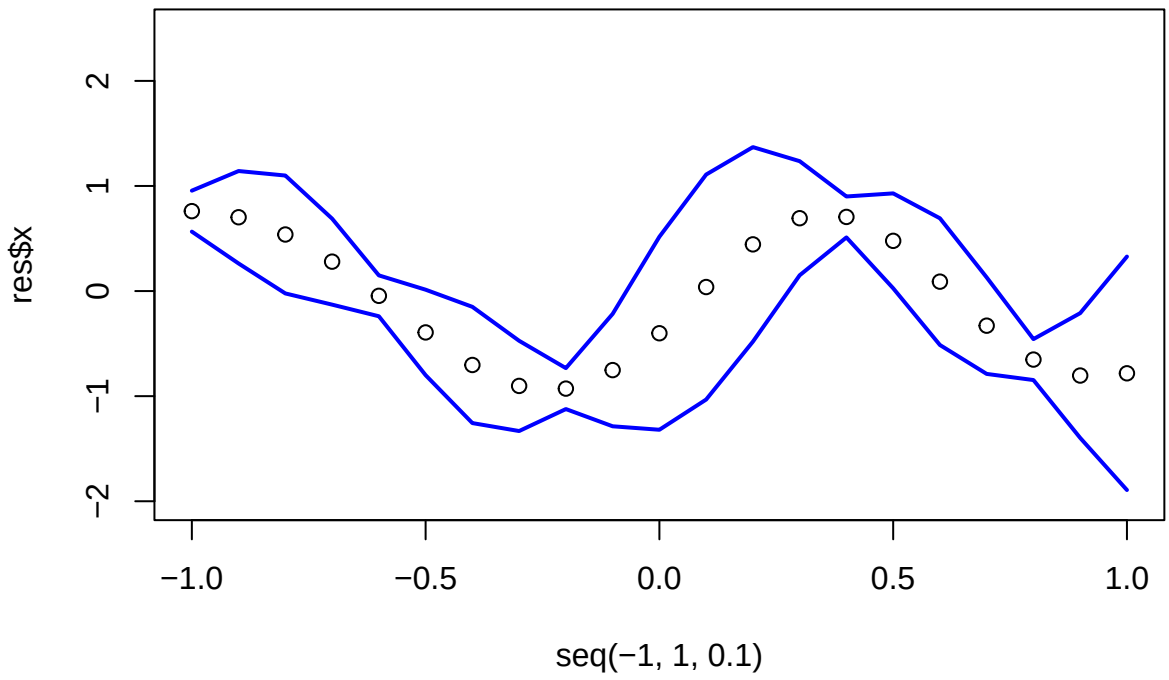
3. Update your posterior from (2) with another observation: $(x, y) = (-0.6, -0.044)$**. Plot the posterior mean of f over the interval** $xin[-1, 1]$**. Plot also 95 % probability (pointwise) bands for f.**

```
res <- posteriorGP(c(0.4, -0.6),c(0.719,-0.044),
                   seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=0.3)

plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
```



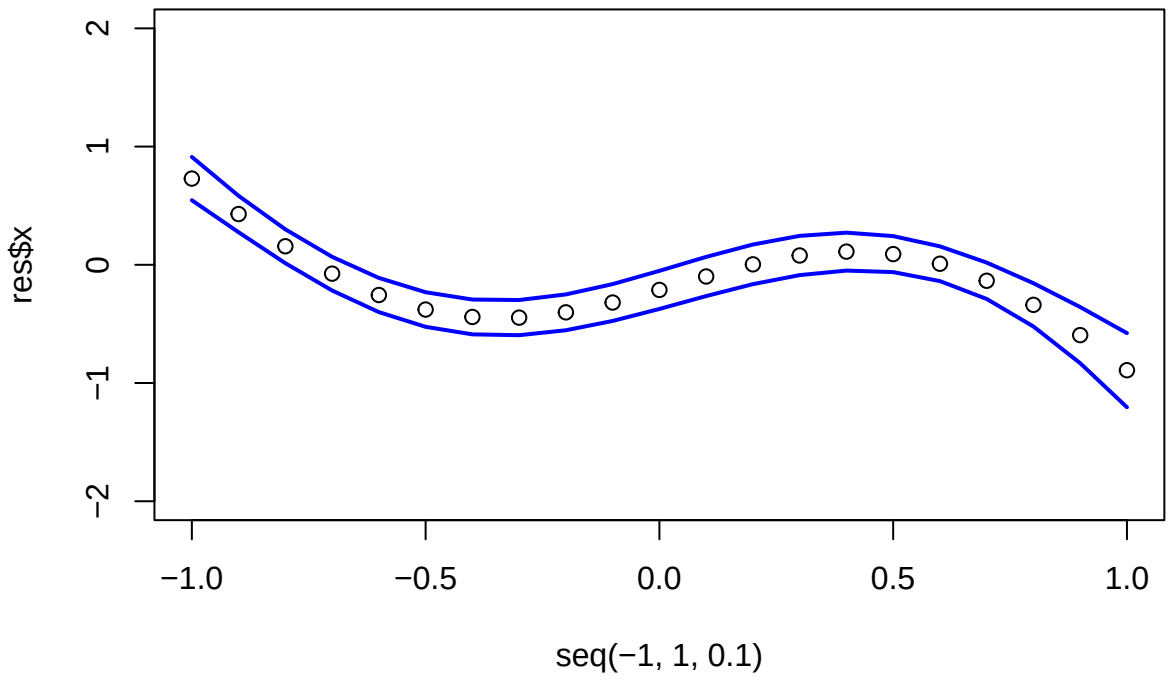
4.

```
res <- posteriorGP(c(-1, -0.6, -0.2, 0.4, 0.8),
                   c(0.768, -0.044, -0.94, 0.719, -0.664),
                   seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=0.3)

plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
```



5.

```
res <- posteriorGP(c(-1, -0.6, -0.2, 0.4, 0.8),
                   c(0.768, -0.044, -0.94, 0.719, -0.664),
                   seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=1)

plot(seq(-1,1,0.1),res$x,ylim = c(-2,2))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
```



Larger l means higher correlation between points. As we can see in the plot, means and variances are different, with l = 1, means and variances look much smoother.

2.

1.

```
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.
#plot(1:length(data$date),data$temp)
time <- 1:length(data$date)
day <- rep(c(1:365),length(data$date)/365)
sub_time <- time[seq(1, length(time), 5)]
sub_day <- day[seq(1, length(time), 5)]
sub_temp <- data$temp[seq(1, length(time), 5)]


sqexpkernel <- function(sigmaf = 1, l = 1)
{
  rval <- function(x,y){
      r <-  sqrt(crossprod(x-y));
      return(exp(-(r^2)/(2*(l^2))) * sigmaf^2)
    }
  class(rval) <- "kernel"
```

```
  return(rval)
}

kern <- sqexpkernel()
print(kern(1,2))
```

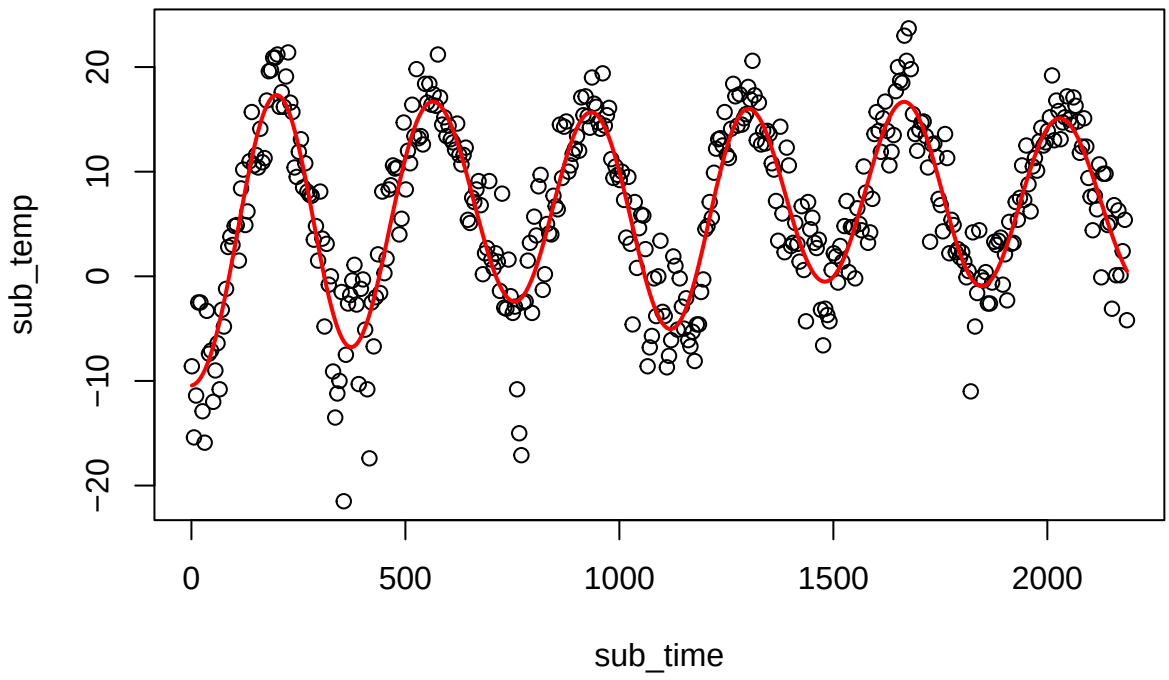
```
##           [,1]
## [1,] 0.6065307
```

```
kern <- sqexpkernel(l = 1, sigmaf = 1)
kernelMatrix(kernel = kern, x = c(1,3,4), y = c(2,3,4))
```

```
## 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.

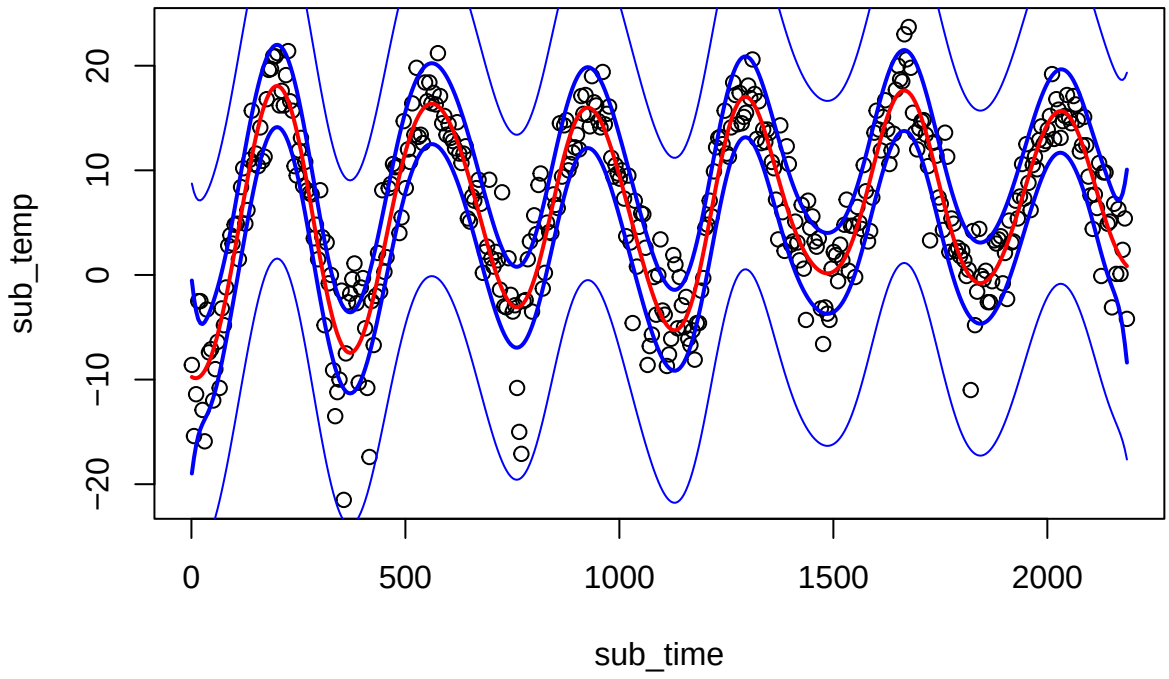
```
polyFit <- lm(sub_temp ~  sub_time + I(sub_time^2))
sigmaNoise = sd(polyFit$residuals)
plot(sub_time,sub_temp)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
SEkernel <- sqexpkernel(sigmaf = 20, l = 0.2) # Note the reparametrization.
GPfit <- gausspr(sub_time,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_time) # Predicting the training data.
lines(sub_time, meanPred, col="red", lwd = 2)
```



3.

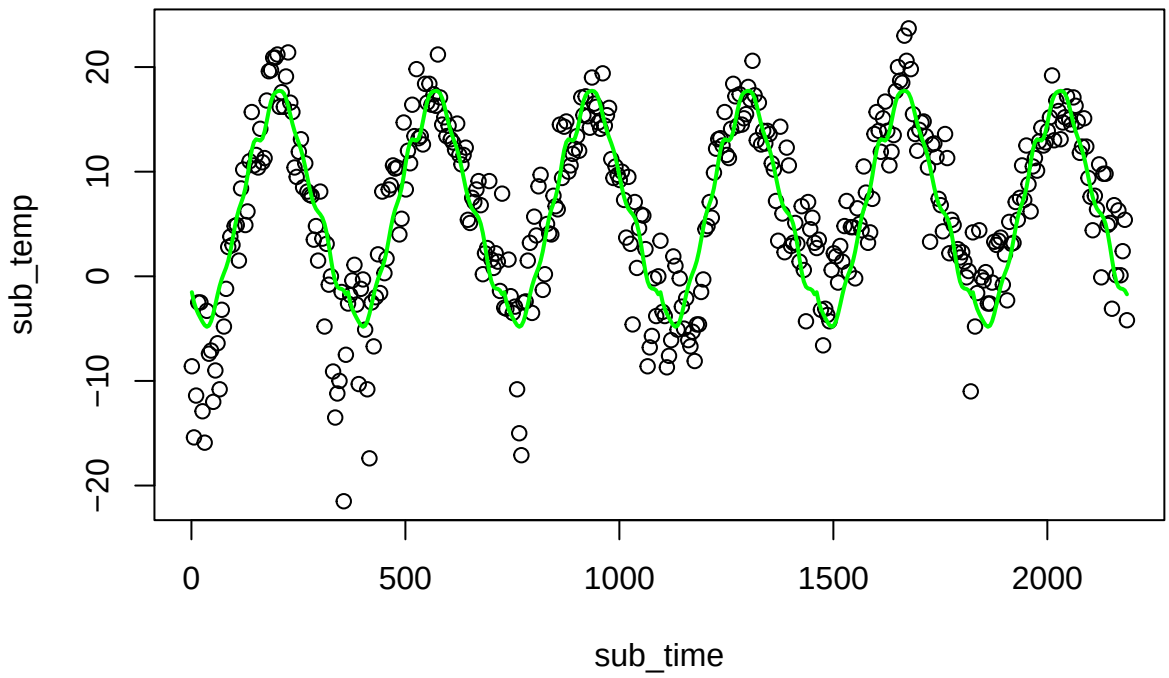
```
res_scale <- scale(sub_temp)
time_scale <- scale(sub_time)

x_star <- time_scale
res <- posteriorGP(as.vector(time_scale),
                   as.vector(res_scale),
                   x_star,
                   1,
                   SquaredExpKernel,
                   sigmaF=20,
                   l=0.2)

x_star <- x_star * sd(sub_time) + mean(sub_time)

meanPred <- res$x * sd(sub_temp) + mean(sub_temp)
CovPred <- res$var * var(sub_temp)


plot(sub_time,sub_temp)
lines(x_star, meanPred, col="red", lwd = 2)
# Probability intervals for fStar.
lines(x_star, (meanPred - 1.96*sqrt(diag(CovPred))), col = "blue", lwd = 2)
lines(x_star, (meanPred + 1.96*sqrt(diag(CovPred))), col = "blue", lwd = 2)

# Prediction intervals for yStar.
```

```
lines(x_star, meanPred - 1.96*sqrt((diag(CovPred) + sigmaNoise^2)), col = "blue")
lines(x_star, meanPred + 1.96*sqrt((diag(CovPred) + sigmaNoise^2)), col = "blue")
```



4.

```
polyFit <- lm(sub_temp ~  sub_day + I(sub_day^2))
sigmaNoise = sd(polyFit$residuals)
#plot(sub_day,sub_temp)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
SEkernel <- sqexpkernel(sigmaf = 20, l = 0.2) # Note the reparametrization.
GPfit <- gausspr(sub_day,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_day) # Predicting the training data.
plot(sub_time,sub_temp)
lines(sub_time, meanPred, col="green", lwd = 2)
```

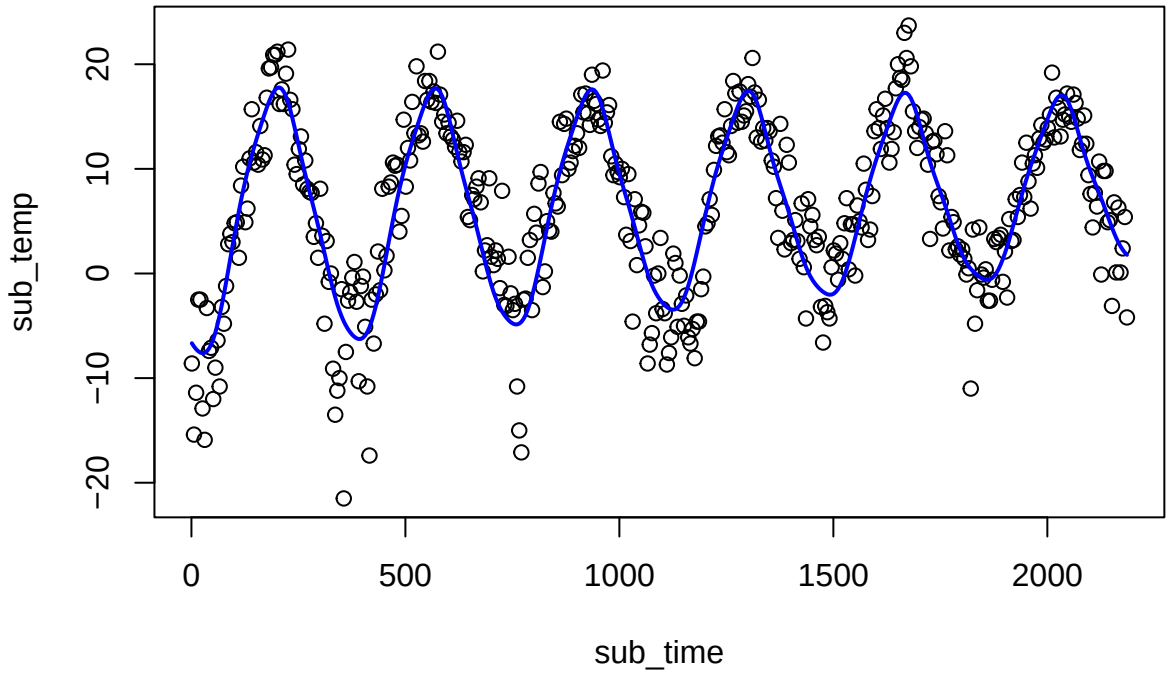


Since the input sequence is repeated periodically during 365 days, we are getting same response in a prediction nth n + 365 for model (4), where in model (2) to predict future values we will need to extrapolate, because model does not have any data in the future. Extrapolation will lead to prior mean which is 0 + scaled mean

5.

```
lpk <- function(sigmaf = 1, l1 = 1, l2 = 1, d = 1)
{
  rval <- function(x,y){
    r <-  sqrt(crossprod(x-y));
    return(exp(-2*sin((pi * r/d))^2/((l1^2))) *
             exp(-(r^2)/(2*(l2^2))) * sigmaf^2)
  }
  class(rval) <- "kernel"
  return(rval)
}


SEkernel <- lpk(sigmaf = 20, l1 = 1, l2 = 10, d = (365/sd(time)))
polyFit <- lm(sub_temp ~  sub_time + I(sub_time^2))
sigmaNoise = sd(polyFit$residuals)
GPfit <- gausspr(sub_time,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_time) # Predicting the training data.
```

```
plot(sub_time,sub_temp)
lines(sub_time, meanPred, col="blue", lwd = 2)
```



(Joel Nilsson answer)

The generalized kernel has a periodic factor, and a factor that is identical to the squared exponential kernel. This gives a trade-off between the behaviors we have seen above: if we were to predict the temperature into the future, the posterior mean will probably not tend to zero as quickly as in the case of the squared exponential kernel, but it will push the predictions towards the prior mean when the prediction points are far away from the training points, in contrast to the model that uses days as input.

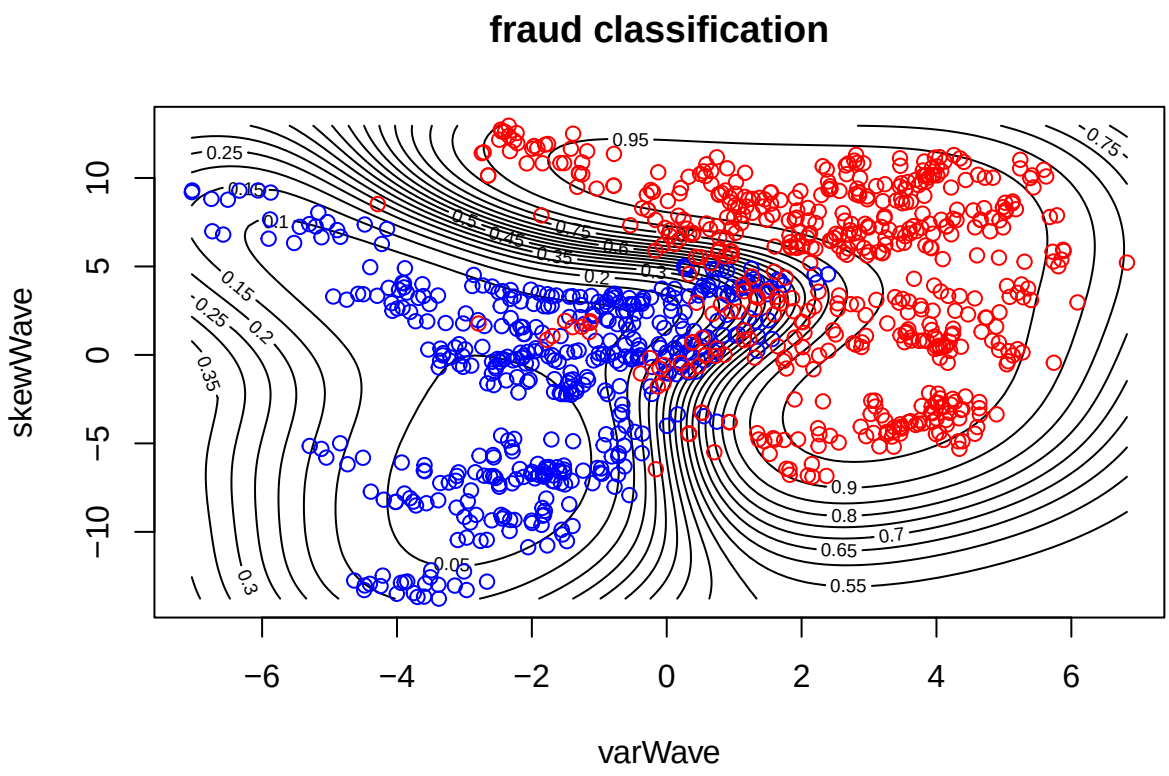
3.

1.

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

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

```
Train <- data[SelectTraining,]

GPfit <- gausspr(fraud ~  varWave + skewWave, data=Train)


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

```
# class probabilities
probPreds <- predict(GPfit, Train[,1:2], type="probabilities")
x1 <- seq(min(Train[,1]),max(Train[,1]),length=100)
x2 <- seq(min(Train[,2]),max(Train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(Train)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")


contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'fraud classification')
points(Train[Train[,5]==1,1], Train[Train[,5]==1,2],col="blue")
points(Train[Train[,5]==0,1], Train[Train[,5]==0,2],col="red")
```



```
confusiontrain <- table(predict(GPfit,Train[,1:2]), Train[,5])

acctrain <- sum(diag(confusiontrain))/sum(confusiontrain)
```

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

	0	1
0	503	18
1	41	438

```
## [1] "accuracy"
```

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

2.

```
Test <- data[-SelectTraining,]
confusiontest <- table(predict(GPfit,Test[,1:2]), Test[,5])
acctest <- sum(diag(confusiontest))/sum(confusiontest)
```

confusion matrix and accuracy on validation data

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

	0	1
0	199	9
1	19	145

```
## [1] "accuracy"
```

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

3.

```
GPfit <- gausspr(fraud ~., data=Train)
```

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

```
confusiontest <- table(predict(GPfit,Test[,1:4]), Test[,5])
acctest <- sum(diag(confusiontest))/sum(confusiontest)
```

confusion matrix and accuracy on validation data of model trained on 4 covariates

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

	0	1
0	216	0
1	2	154

```
## [1] "accuracy"
```

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

We can see that model with all 4 covariates has higher accuracy, than model with 2 covariates.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(kernlab)
library(AtmRay)
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l)^2 )
  }
  return(K)
}
posteriorGP <- function(X,y,XStar,sigmaNoise, k,...){
  Ksigma <- k(X,X,...) + (sigmaNoise^2 * diag(length(X)))
  L <- t(chol(Ksigma))
  alpha <- solve(t(L),solve(L,y))
  kstar <- k(X,XStar,...)
  f_pred <- t(kstar) %*% alpha
  v <- solve(L,kstar)
  V <- k(XStar,XStar,...) - (t(v) %*% v)
  return(list(x = f_pred, var = V))
}
res <- posteriorGP(0.4,0.719, seq(-1,1,0.1), 0.1,
                   SquaredExpKernel, sigmaF=1,l=0.3)
plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
res <- posteriorGP(c(0.4, -0.6),c(0.719,-0.044),
                   seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=0.3)

plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
res <- posteriorGP(c(-1, -0.6, -0.2, 0.4, 0.8),
                   c(0.768, -0.044, -0.94, 0.719, -0.664),
```

```
                  seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=0.3)

plot(seq(-1,1,0.1),res$x, ylim = c(-2,2.5))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
res <- posteriorGP(c(-1, -0.6, -0.2, 0.4, 0.8),
                  c(0.768, -0.044, -0.94, 0.719, -0.664),
                  seq(-1,1,0.1), 0.1, SquaredExpKernel, sigmaF=1,l=1)

plot(seq(-1,1,0.1),res$x,ylim = c(-2,2))
lines(seq(-1,1,0.1), res$x - 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
lines(seq(-1,1,0.1), res$x + 1.96*sqrt(diag(res$var)), col = "blue", lwd = 2)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.
#plot(1:length(data$date),data$temp)
time <- 1:length(data$date)
day <- rep(c(1:365),length(data$date)/365)
sub_time <- time[seq(1, length(time), 5)]
sub_day <- day[seq(1, length(time), 5)]
sub_temp <- data$temp[seq(1, length(time), 5)]


sqexpkernel <- function(sigmaf = 1, l = 1)
{
  rval <- function(x,y){
      r <-  sqrt(crossprod(x-y));
      return(exp(-(r^2)/(2*(l^2))) * sigmaf^2)
    }
  class(rval) <- "kernel"
  return(rval)
}

kern <- sqexpkernel()
print(kern(1,2))
kern <- sqexpkernel(l = 1, sigmaf = 1)
kernelMatrix(kernel = kern, x = c(1,3,4), y = c(2,3,4))
polyFit <- lm(sub_temp ~  sub_time + I(sub_time^2))
sigmaNoise = sd(polyFit$residuals)
plot(sub_time,sub_temp)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
SEkernel <- sqexpkernel(sigmaf = 20, l = 0.2) # Note the reparametrization.
GPfit <- gausspr(sub_time,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_time) # Predicting the training data.
lines(sub_time, meanPred, col="red", lwd = 2)

res_scale <- scale(sub_temp)
time_scale <- scale(sub_time)

x_star <- time_scale
res <- posteriorGP(as.vector(time_scale),
                  as.vector(res_scale),
                  x_star,
                  1,
```

```
                    SquaredExpKernel,
                    sigmaF=20,
                    l=0.2)

x_star <- x_star * sd(sub_time) + mean(sub_time)

meanPred <- res$x * sd(sub_temp) + mean(sub_temp)
CovPred <- res$var * var(sub_temp)


plot(sub_time,sub_temp)
lines(x_star, meanPred, col="red", lwd = 2)
# Probability intervals for fStar.
lines(x_star, (meanPred - 1.96*sqrt(diag(CovPred))), col = "blue", lwd = 2)
lines(x_star, (meanPred + 1.96*sqrt(diag(CovPred))), col = "blue", lwd = 2)

# Prediction intervals for yStar.
lines(x_star, meanPred - 1.96*sqrt((diag(CovPred) + sigmaNoise^2)), col = "blue")
lines(x_star, meanPred + 1.96*sqrt((diag(CovPred) + sigmaNoise^2)), col = "blue")
polyFit <- lm(sub_temp ~  sub_day + I(sub_day^2))
sigmaNoise = sd(polyFit$residuals)
#plot(sub_day,sub_temp)

# Fit the GP with built-in square expontial kernel (called rbfdot in kernlab).
SEkernel <- sqexpkernel(sigmaf = 20, l = 0.2) # Note the reparametrization.
GPfit <- gausspr(sub_day,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_day) # Predicting the training data.
plot(sub_time,sub_temp)
lines(sub_time, meanPred, col="green", lwd = 2)
lpk <- function(sigmaf = 1, l1 = 1, l2 = 1, d = 1)
{
  rval <- function(x,y){
    r <-  sqrt(crossprod(x-y));
    return(exp(-2*sin((pi * r/d))^2/((l1^2))) *
             exp(-(r^2)/(2*(l2^2))) * sigmaf^2)
  }
  class(rval) <- "kernel"
  return(rval)
}


SEkernel <- lpk(sigmaf = 20, l1 = 1, l2 = 10, d = (365/sd(time)))
polyFit <- lm(sub_temp ~  sub_time + I(sub_time^2))
sigmaNoise = sd(polyFit$residuals)
GPfit <- gausspr(sub_time,sub_temp, kernel = SEkernel, var = sigmaNoise^2)
meanPred <- predict(GPfit, sub_time) # Predicting the training data.
plot(sub_time,sub_temp)
lines(sub_time, meanPred, col="blue", lwd = 2)
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave", "skewWave", "kurtWave", "entropyWave", "fraud")
data[,5] <- as.factor(data[,5])

set.seed(111)
```

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


Train <- data[SelectTraining,]

GPfit <- gausspr(fraud ~  varWave + skewWave, data=Train)

# class probabilities
probPreds <- predict(GPfit, Train[,1:2], type="probabilities")
x1 <- seq(min(Train[,1]),max(Train[,1]),length=100)
x2 <- seq(min(Train[,2]),max(Train[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(Train)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")


contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 20, xlab = "varWave",
        ylab = "skewWave", main = 'fraud classification')
points(Train[Train[,5]==1,1], Train[Train[,5]==1,2],col="blue")
points(Train[Train[,5]==0,1], Train[Train[,5]==0,2],col="red")

confusiontrain <- table(predict(GPfit,Train[,1:2]), Train[,5])

acctrain <- sum(diag(confusiontrain))/sum(confusiontrain)
print("confusion matrix")
knitr::kable(confusiontrain)
print("accuracy")
print(acctrain)
Test <- data[-SelectTraining,]
confusiontest <- table(predict(GPfit,Test[,1:2]), Test[,5])
acctest <- sum(diag(confusiontest))/sum(confusiontest)
print("confusion matrix")
knitr::kable(confusiontest)
print("accuracy")
print(acctest)
GPfit <- gausspr(fraud ~., data=Train)
confusiontest <- table(predict(GPfit,Test[,1:4]), Test[,5])
acctest <- sum(diag(confusiontest))/sum(confusiontest)
print("confusion matrix")
knitr::kable(confusiontest)
print("accuracy")
print(acctest)
```