732A96 Lab 3

Emil K Svensson

28 September 2017

Assignment 1

a)

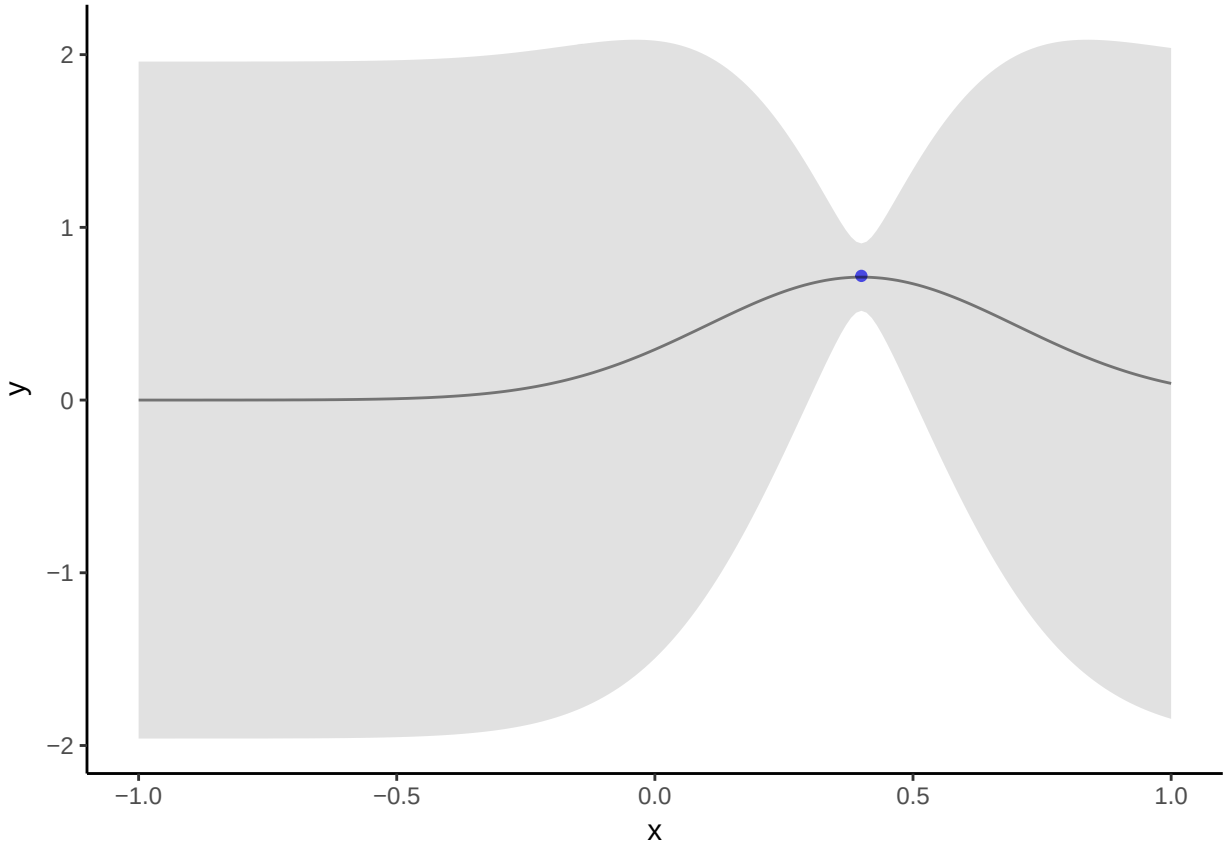
```
# The kernel function

exp_kern <- function(x,xi,l, sigmaf ){

  return((sigmaf^2)*exp(-0.5*( (x - xi) / l )^2))

}

# The implementation, can take a custom kernel of any class.

linear_gp <- function(x,y,xStar,hyperParam,sigmaNoise,kernel){

n <- length(x)
kernel_f <- kernel


# K = Covariance matrix calculation
  K <- function(X, XI,...){

    kov <- matrix(0,nrow = length(X), ncol = length (XI))

    for(i in 1:length(XI)){

      kov[,i]<- kernel_f(X,XI[i],...)

    }
    return(kov)
  }

l <-hyperParam[1]
sigmaf <- hyperParam[2]

#K(X,X)
K_xx   <- K(x,x, l = l, sigmaf = sigmaf) #, kernel = exp_kern

#K(X*,X*)
K_xsxs <- K(xStar,xStar, l = l, sigmaf = sigmaf) # kernel = exp_kern,

#K(X,X*)
K_xxs  <- K(x,xStar, l = l, sigmaf = sigmaf)  #kernel = exp_kern,
```

```r
# Algorithm in page 19 of the Rasmus/Williams book


sI <- sigmaNoise^2 * diag(dim(as.matrix(K_xx))[1])
# L is transposed according to a definition in the R & W book
L_transposed <- chol(K_xx + sI)
L <- t(L_transposed)

alpha <- solve(t(L), solve(L,y))

f_bar_star <- t(K_xxs) %*% alpha

v <- solve(L,K_xxs)

V_fs <- K_xsxs - t(v) %*% v

log_mlike <- -0.5 %*% t(y) %*% alpha - sum( diag(L) - n/2 * log(2*pi) )

return(list(fbar = f_bar_star, vf = V_fs, log_post= log_mlike))

}
```

```r
# The data given
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)

# The noise
sn <- 0.1

# The training grid
xs <- seq(-1,1,0.01)

# Hyperparameters l an sigma
hyperParam <- c(0.3, 1)

# Another utility function

repeter <- function(x,y,xs,sn,hyperParam,kernel){

res <- linear_gp(x,y,xs,hyperParam,sn,kernel)

# If you want the prediction band just add the noise variance (ie the sigma_n)

upp <- res$fbar + 1.96*sqrt(diag(res$vf))
low <- res$fbar - 1.96*sqrt(diag(res$vf))

plot_it <- data.frame(x = x, y = y)
band_it <- data.frame(xGrid = xs, fbar = res$fbar,  upp = upp, low = low)

plot_gp(plot_it,band_it)
}
```

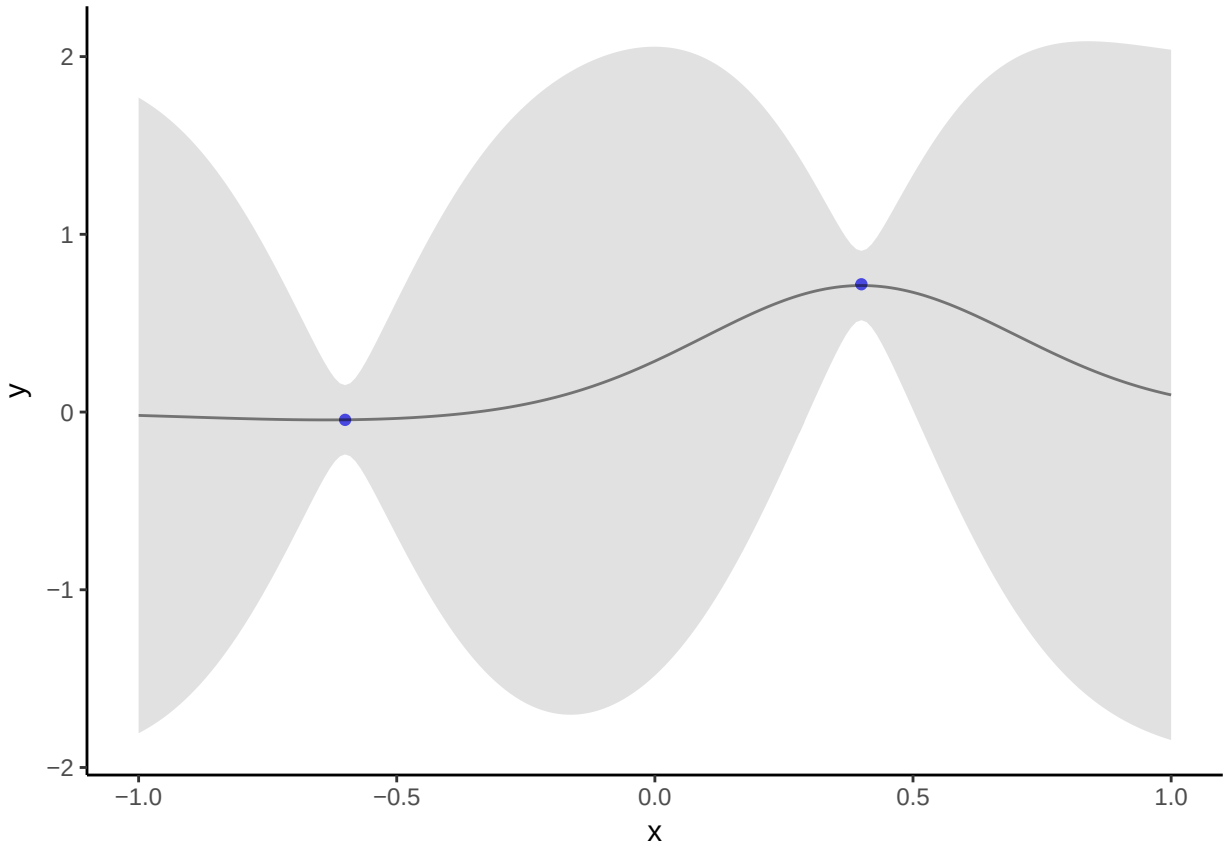
b)

```
repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```



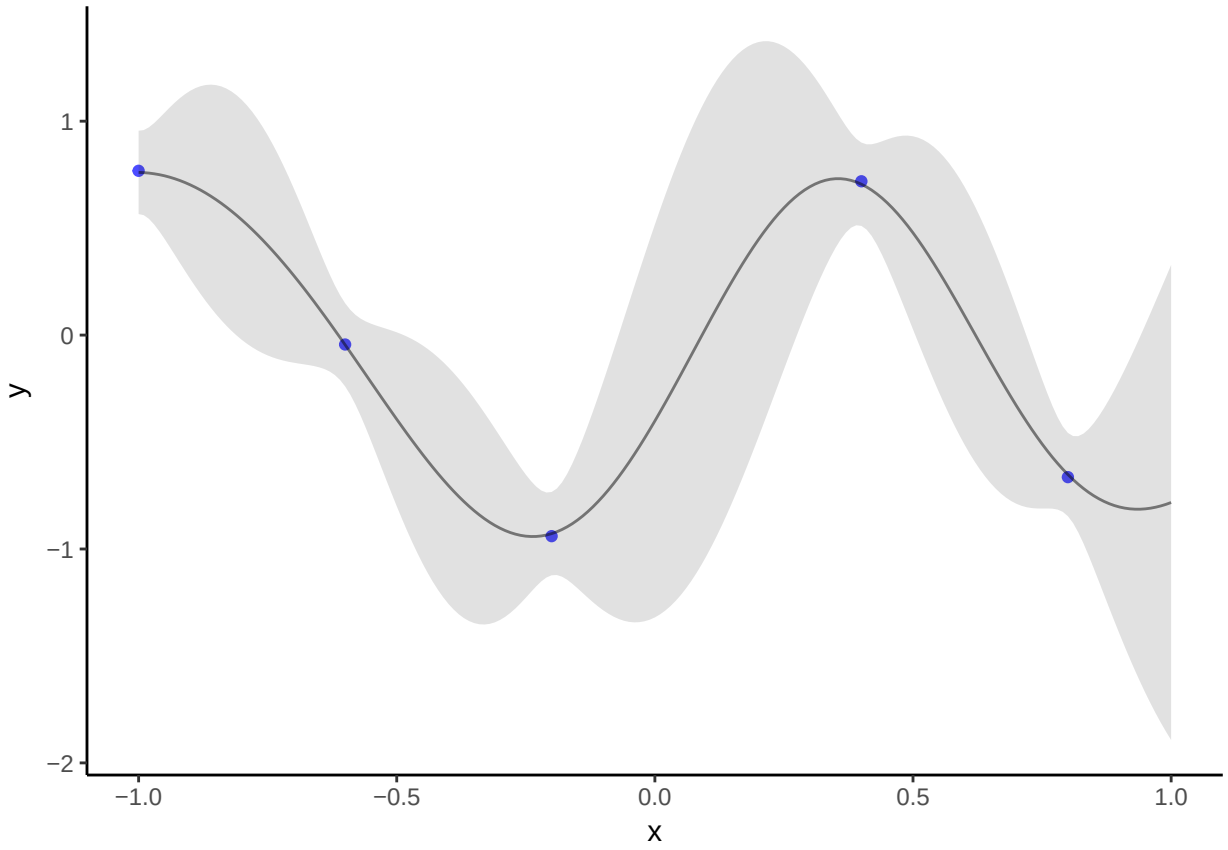
c)

```
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```



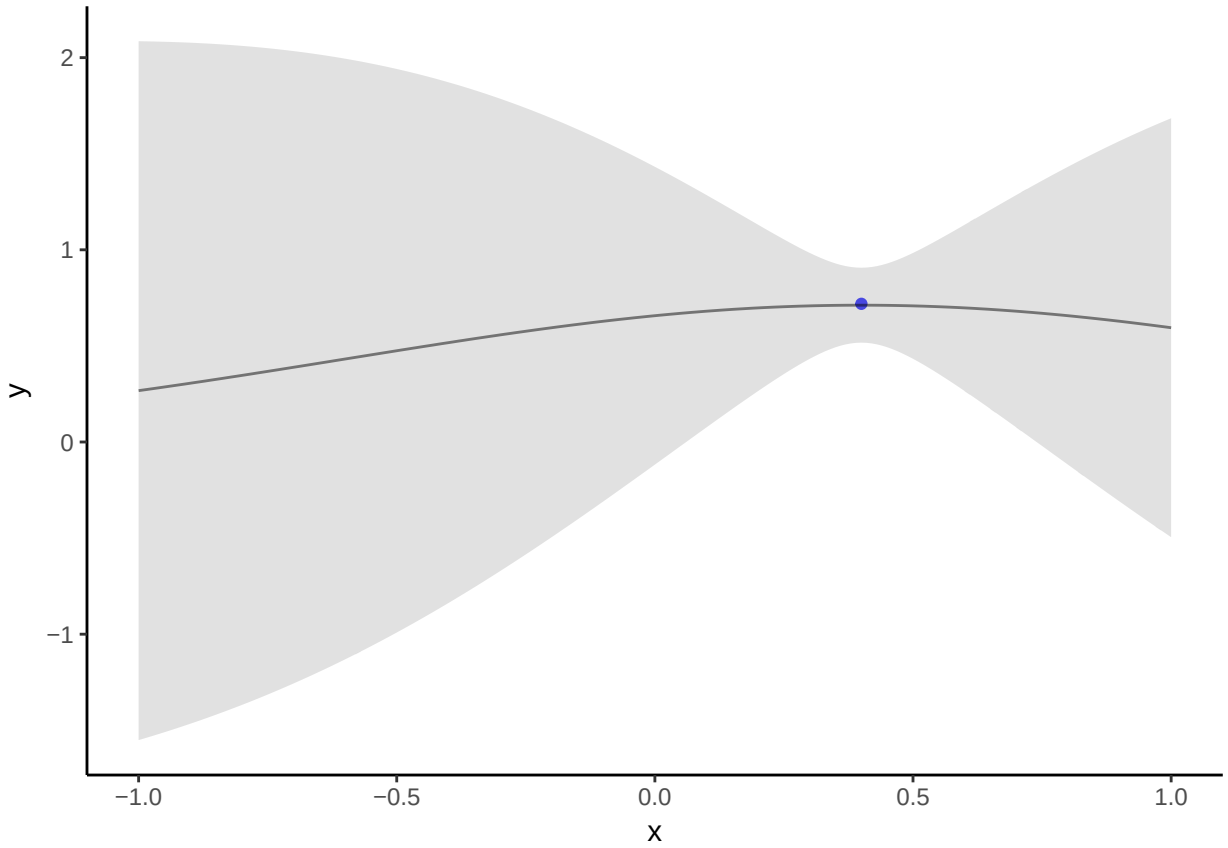
d)

```
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```

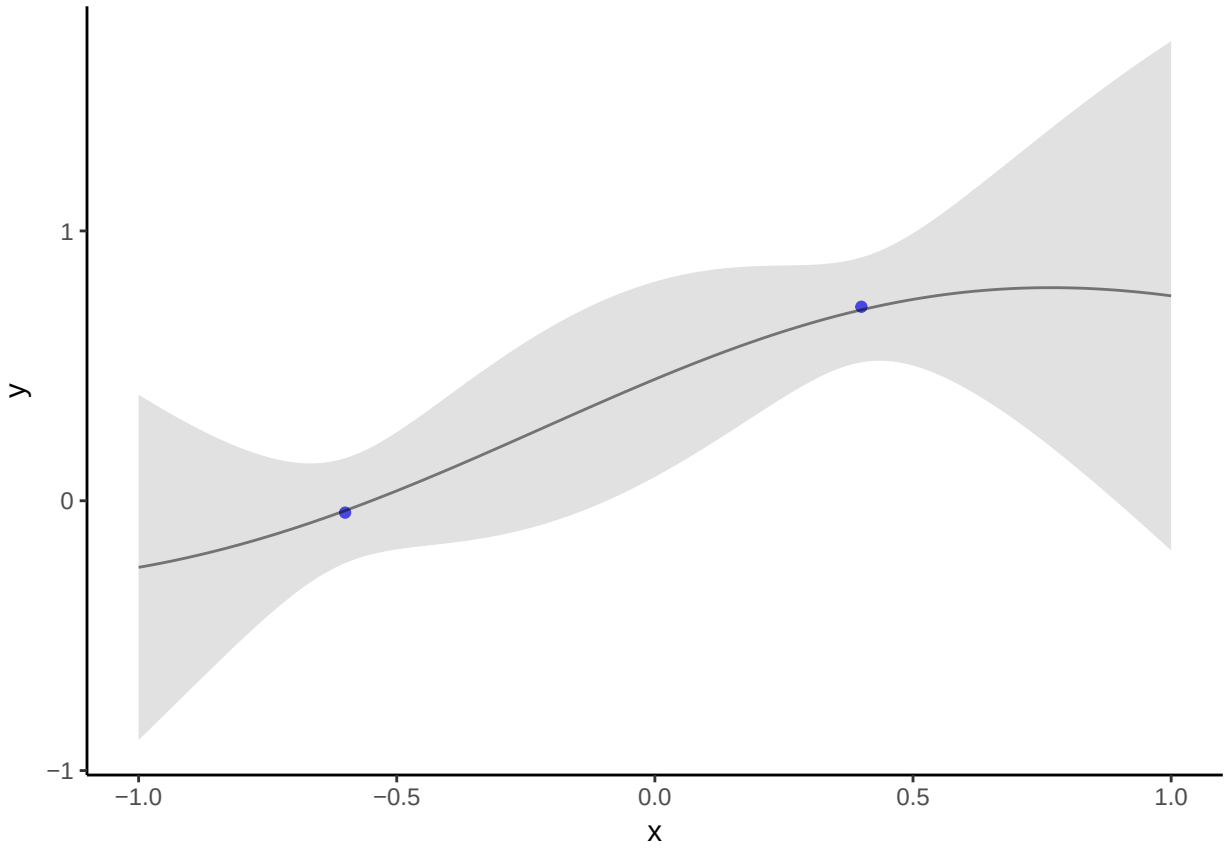


e)

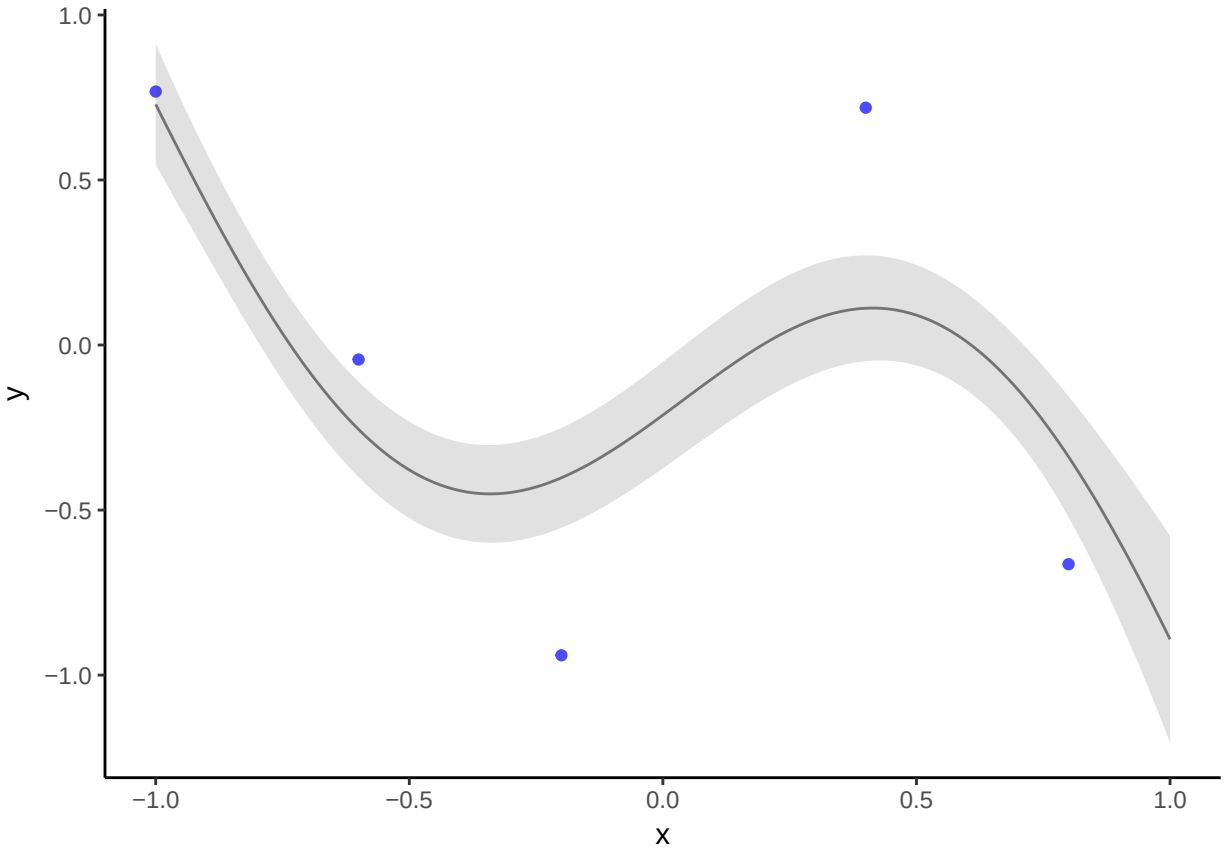
```
x <- c(-1.0, -0.6, -0.2, 0.4, 0.8)
y <- c(0.768, -0.044, -0.940, 0.719 ,-0.664)
sn <- 0.1
xs <- seq(-1,1,0.01)
hyperParam <- c(1, 1)

repeter(x = x[4], y = y[4],xs,sn,hyperParam, kernel = exp_kern)
```




```
repeter(x = x[c(2,4)], y = y[c(2,4)],xs,sn,hyperParam, kernel = exp_kern)
```




```
repeter(x = x, y = y,xs,sn,hyperParam, kernel = exp_kern)
```



Assignment 2

Data preparations

```
tullinge$time <- 1:nrow(tullinge)
tullinge$day <- rep(1:365,6)

time_sub <- tullinge$time %in% seq(1,2190,5)
tullinge <- tullinge[time_sub,]
```

a)

```
kern_maker <- function(l,sigmaf){

  exp_k <- function(x,y = NULL){

   return((sigmaf^2)*exp(-0.5*( (x - y) / l )^2))
  }

  class(exp_k) <- "kernel"
  return(exp_k)
}
```

```
# gausspr()
# kernelMatrix()


ell <- 1
# SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdo (which is the SE kernel)
# SEkernel(1,2)

my_exp <- kern_maker(l = 10, sigmaf =20)

x <- c(1,3,4)
x_star <- c(2,3,4)
#my_exp(x,x_star)
kernelMatrix(my_exp,x,x_star)
```

```
## An object of class "kernelMatrix"
##          [,1]     [,2]    [,3]
## [1,] 398.0050 392.0795 382.399
## [2,] 398.0050 400.0000 398.005
## [3,] 392.0795 398.0050 400.000
```

b)

```
lm_tull <- lm(temp ~ time + I(time^2), data = tullinge)

sigma_2n <- var(resid(lm_tull))


a2b_kern <- kern_maker(l = 0.2, sigmaf = 20 )
gp_tullinge <- gausspr(x = tullinge$time,
                       y = tullinge$temp,
                       kernel = a2b_kern,
                       var = sigma_2n)
```

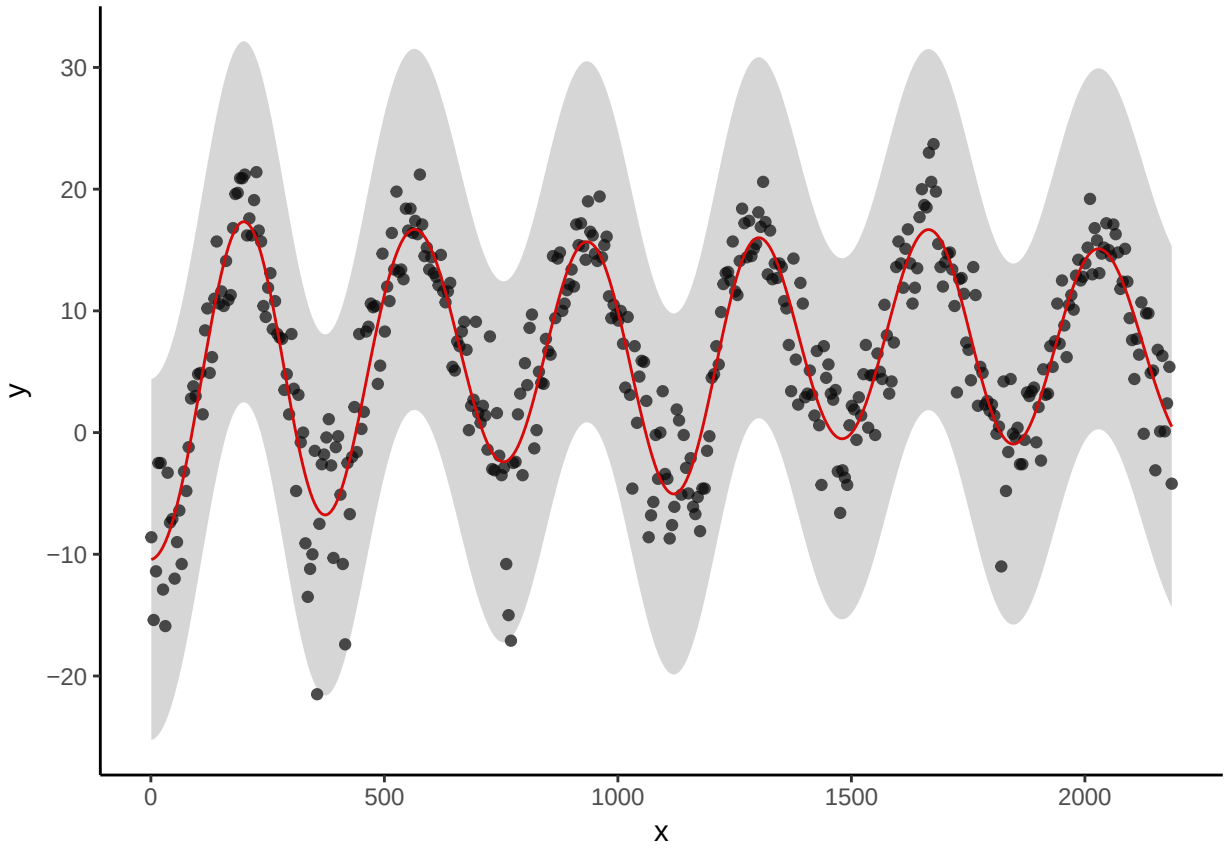
See task c) for the plot.

c)

```
sn_2c <- sqrt(sigma_2n)
xs_2c <- tullinge$time
hyperParam_2c <- c(0.2, 20)


res_2c<- linear_gp(x = tullinge$time,
        y = tullinge$temp,
        xStar = xs_2c,
        sigmaNoise  = sn_2c,
        hyperParam = hyperParam_2c,
        kernel = exp_kern)

upp2c <- predict(gp_tullinge) + 1.96*sqrt(diag(res_2c$vf))
```

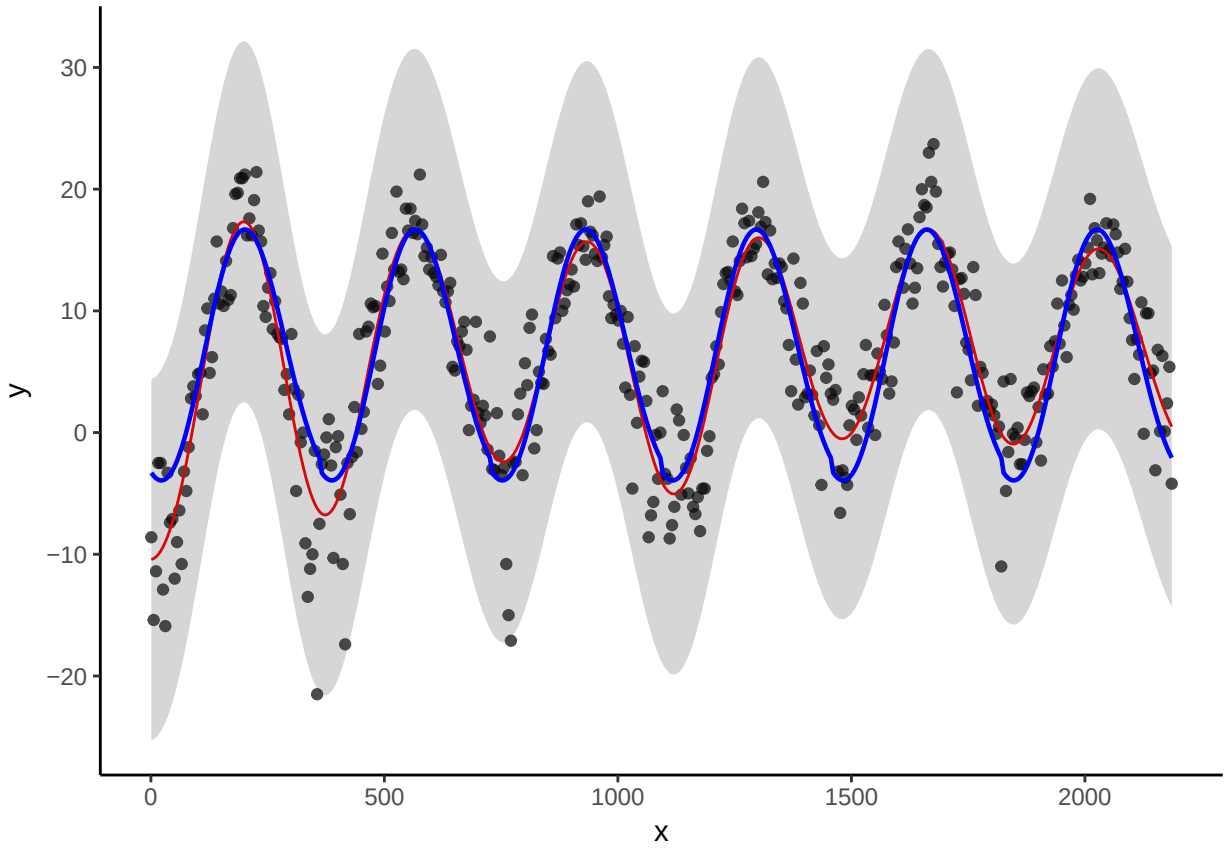
```r
low2c <- predict(gp_tullinge) - 1.96*sqrt(diag(res_2c$vf))

plot_it1 <- data.frame(x = tullinge$time, y = tullinge$temp)
band_it1 <- data.frame(xGrid = xs_2c, fbar = res_2c$fbar,  upp = upp2c, low = low2c)
```

```r
C2 <- ggplot() +

   geom_point(
     aes(x = x, y = y),
     data = plot_it1,
     col = "black",
     alpha = 0.7) +

  geom_line(
    aes(x = xGrid, y = predict(gp_tullinge)),
    data = band_it1,
    alpha = 1,
    col = "red") +

   geom_ribbon(
     aes(ymin = low2c, ymax = upp2c, x = xGrid),
     data = band_it1,
     alpha = 0.2) +

   theme_classic()

  #plot(x=band_it1$xGrid, y=band_it1$fbar, type = "l")

C2
```



d)

```
a2d_kern <- kern_maker(l = 1.2, sigmaf = 20 )
gp_tullinge_d <- gausspr(x = tullinge$day,
                         y = tullinge$temp,
                         kernel = a2d_kern,
                         var = sigma_2n)


C23 <- C2 + geom_line(aes(x= tullinge$time,y = predict(gp_tullinge_d)), col = "blue", size = 0.8)
  #geom_point(data = tullinge, aes(x= time, y = temp)) +


C23
```




```
# plot(y = tullinge$temp, x = tullinge$day)
# #lines(x = tullinge$time, y = fitted(lm_tull), col = "red")
# lines(x = tullinge$day, y = predict(gp_tullinge_d), col = "red" , lwd = 1)
```

The process model after time has an advantage in that sense that you can capture a trend isolated to a specific time since your modeling using the closest observations in time rather than the day model that assumes that the clostes related temperature point is the one on the same day previous years.

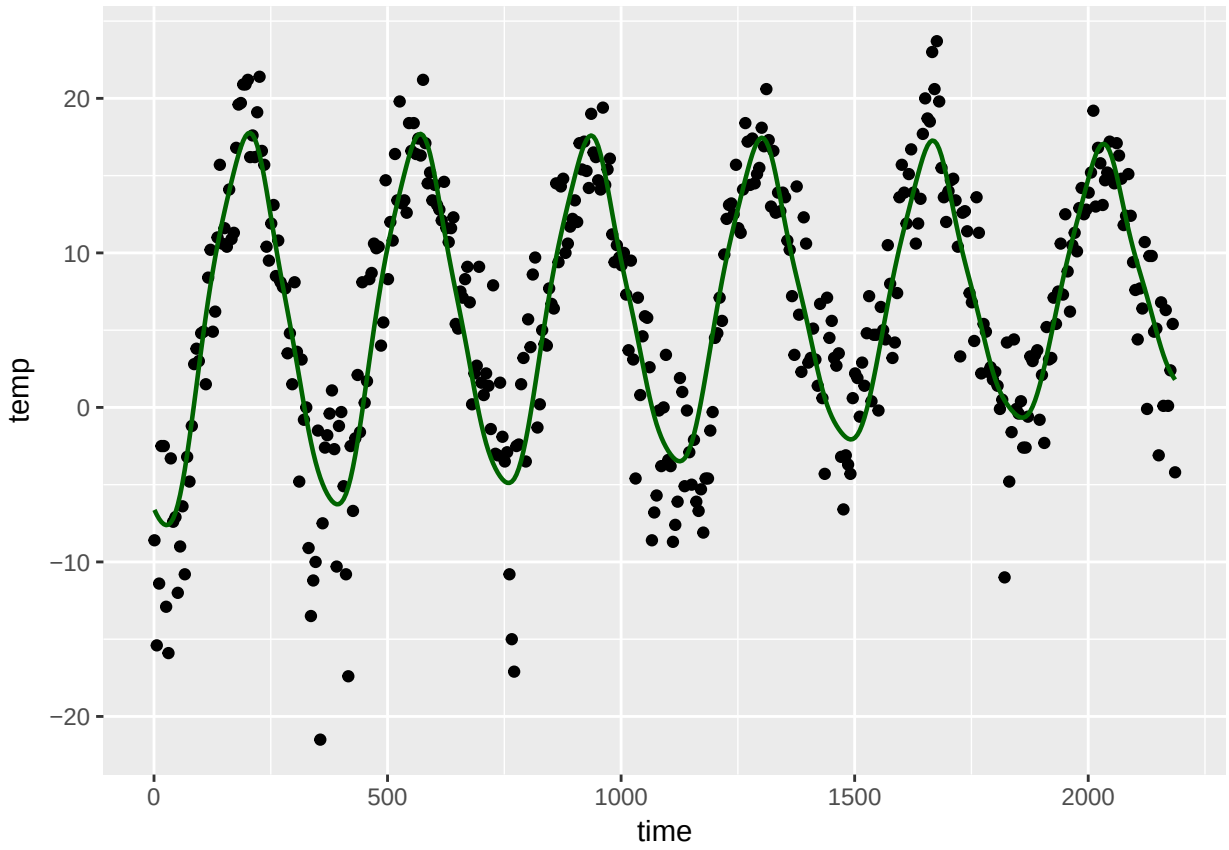
e)

```
# periodic_kernel <- function(x,xi,sigmaf,d, l_1, l_2){
#
#   part1 <- exp(2 * sin(pi * abs(x - xi) / d)^2 / l_1^2 )
#   part2 <- exp(-0.5 * abs(x - xi)^2 / l_2)
#
#   sigmaf^2 * part1 * part2
#
# }



kern_maker2 <- function(sigmaf,d, l_1, l_2){

 periodic_kernel <- function(x,y = NULL){

  part1 <- exp(-2 * sin(pi * abs(x - y) / d)^2 / l_1^2 )
```

```
  part2 <- exp(-0.5 * abs(x - y)^2 / l_2^2)

  sigmaf^2 * part1 * part2

}


  class(periodic_kernel) <- "kernel"
  return(periodic_kernel)
}
```

```
sigmaff <- 20
l1 <- 1
l2 <- 10
d_est <- 365 / sd(tullinge$time)

periodic_kernel <- kern_maker2(sigmaf = sigmaff,
                               d = d_est ,
                               l_1 = l1,
                               l_2 = l2)

gp_tullinge_et <- gausspr(x = tullinge$time,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)


gp_tullinge_ed <- gausspr(x = tullinge$day,
                          y = tullinge$temp,
                          kernel = periodic_kernel,
                          var = sigma_2n)
```

```
ggplot(data = tullinge, aes(x= time, y = temp)) +
  geom_point() +
   geom_line(aes(y = predict(gp_tullinge_et)), col = "darkgreen", size = 0.8)
```



This kernel seems to catch both variation in day and over time.

Assignment 3

```
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,]
test <- data[-SelectTraining,]
```

a)

```
colnames(data)
```

```
## [1] "varWave"     "skewWave"    "kurtWave"    "entropyWave" "fraud"
```

```
GPfitFraud <- gausspr(fraud ~  varWave + skewWave, data = train)
```

```
## 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 =  1.2043047635594
##
## Number of training instances learned : 1000
## Train error : 0.068
```

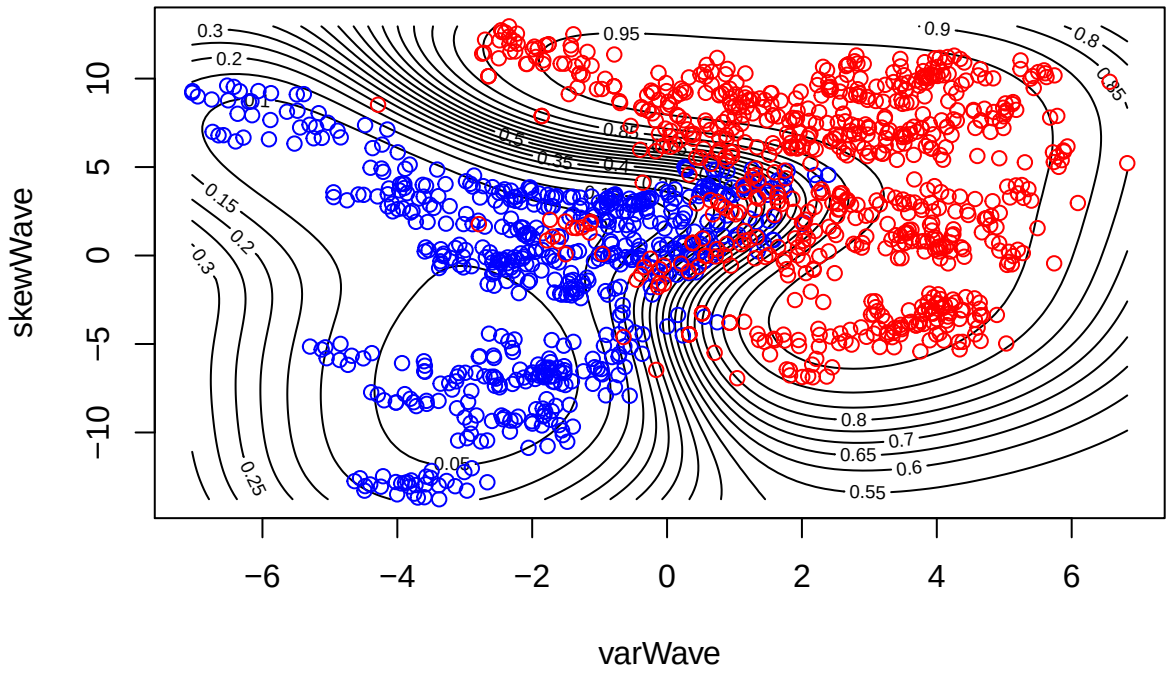
```
# predict on the test set
fit_train<- predict(GPfitFraud,train[,c("varWave","skewWave")])
table(fit_train, train$fraud) # confusion matrix
```

```
##
## fit_train   0   1
##         0 512  24
##         1  44 420
```

```
mean(fit_train == train$fraud)
```

```
## [1] 0.932
```

```
# probPreds <- predict(GPfitIris, iris[,3:4], type="probabilities")
x1 <- seq(min(data[,"varWave"]),max(data[,"varWave"]),length=100)
x2 <- seq(min(data[,"skewWave"]),max(data[,"skewWave"]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))

gridPoints <- data.frame(gridPoints)
names(gridPoints) <- c("varWave","skewWave")
probPreds <- predict(GPfitFraud, gridPoints, type="probabilities")

contour(x1,x2,t(matrix(probPreds[,1],100)), 20,
        xlab = "varWave", ylab = "skewWave",
        main = 'Prob(Fraud) - Fraud is red')

points(data[data[,5]== 1,"varWave"],data[data[,5]== 1,"skewWave"],col="blue")
points(data[data[,5]== 0,"varWave"],data[data[,5]== 0,"skewWave"],col="red")
```



b)

```
# predict on the test set
fit_test<- predict(GPfitFraud,test[,c("varWave","skewWave")])
table(fit_test, test$fraud) # confusion matrix
```

```
##
## fit_test   0   1
##        0 191   9
##        1  15 157
```

```
mean(fit_test == test$fraud)
```

```
## [1] 0.9354839
```

c)

```
GPfitFraudFull <- gausspr(fraud ~ ., data = train)
```

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

```
GPfitFraudFull
```

```
## Gaussian Processes object of class "gausspr"
## Problem type: classification
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.399933221120042
##
```

```
## Number of training instances learned : 1000
## Train error : 0.004
```

```
# predict on the test set
fit_Full<- predict(GPfitFraudFull,test[,-ncol(test)])
table(fit_Full, test$fraud) # confusion matrix
```

```
##
## fit_Full   0   1
##        0 205   0
##        1   1 166
```

```
mean(fit_Full == test$fraud)
```

```
## [1] 0.9973118
```