Lab4

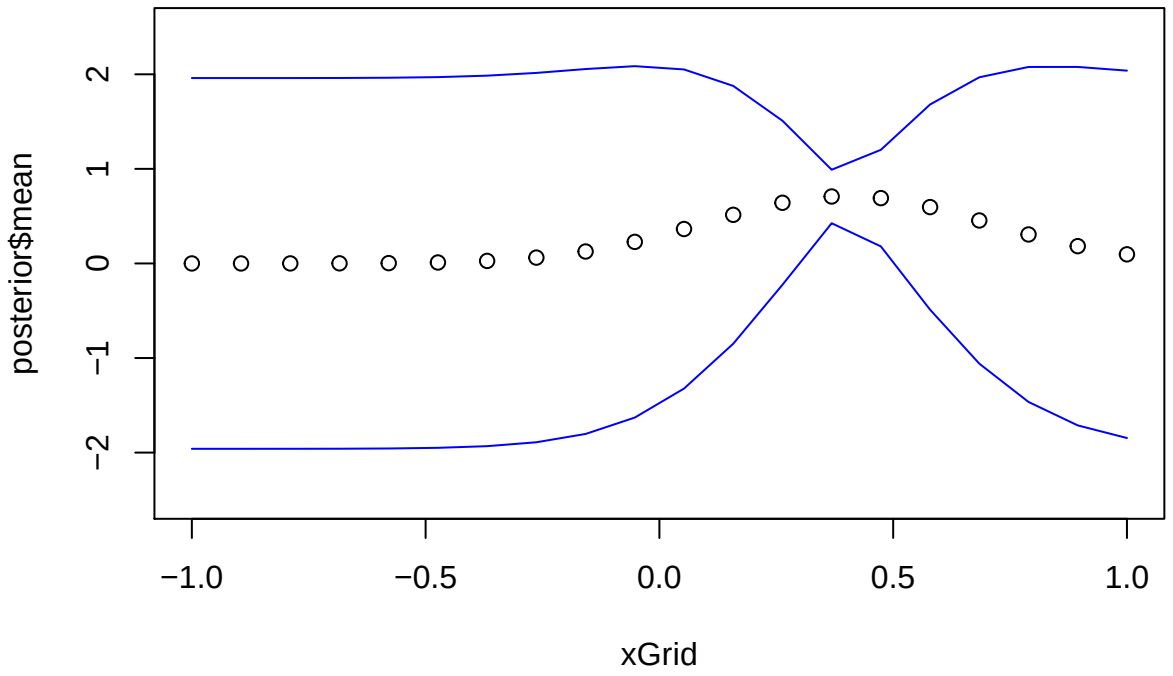
Oskar Hidén - oskhi827

10/18/2020

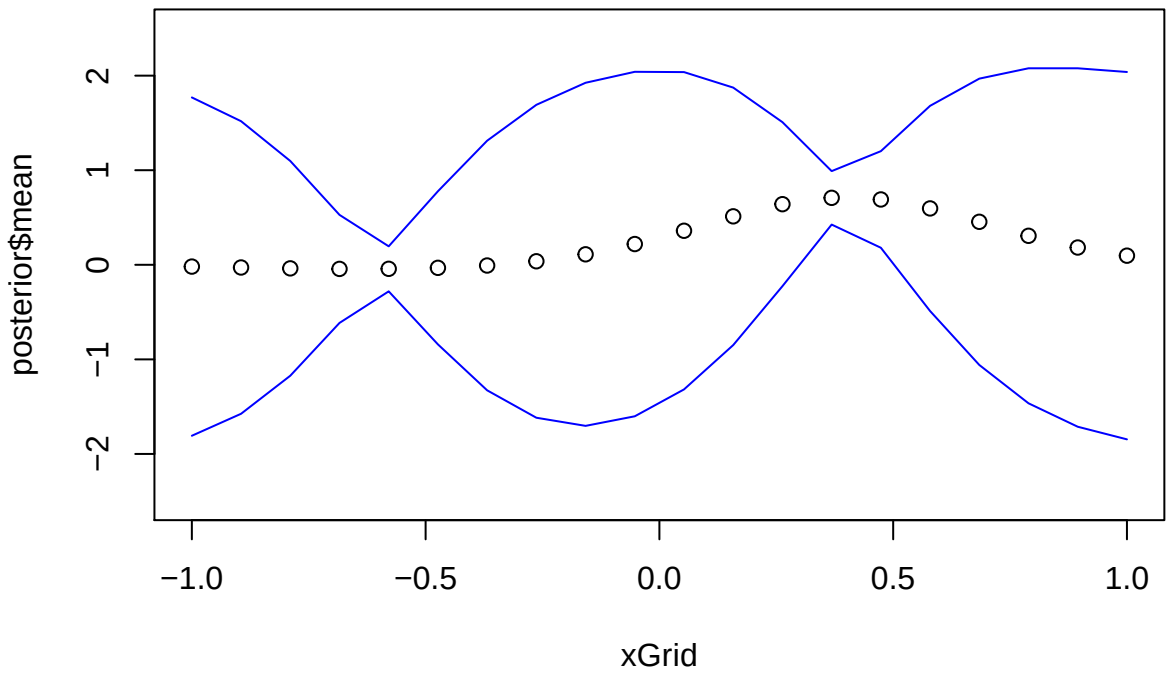
Part 1 - Implementing GP Regression

```
#install.packages('kernlab')
library(kernlab)

# Computing the whole covariance matrix K from the kernel.
cov_function = function(X, Xstar){
  return(kernelMatrix(kernel = SEkernel, x=X, y=Xstar))
}

# Algorithm 2.1 sunns in n^3/6 instead of O(n^3). GP can be estimated faster, O(n) with eg. KISS-GP
posteriorGP = function(X_input, y_targets, k_cov_function, sigmaNoise=1, XStar){
  A = k_cov_function(X_input, X_input)
  A = A + diag(length(X_input))*sigmaNoise^2
  L = t(chol(A)) # chol Returns t(L)
  L_y = solve(L, y_targets)
  alpha = solve(t(L), L_y)

  k_star = k_cov_function(X_input, XStar)
  f_star = t(k_star)%*%alpha

  v = solve(L,k_star)
  V_f_star = k_cov_function(XStar, XStar) - t(v)%*%v
  return(list("mean"=f_star, "cov" = V_f_star))
}

sigma_f = 1
ell = 0.3
SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Reparametrize the rbfdot (which is the SE kernel) in kernlab.
sigma_n = 0.1

# 2 - Update prior with a signle observation
x=0.4
y=0.719
xGrid <- seq(-1,1,length=20) # x_star

posterior = posteriorGP(x, y, cov_function, sigma_n, xGrid)

plot(xGrid, posterior$mean, ylim = c(-2.5,2.5)) # posterior mean
std_dev = sqrt(diag(posterior$cov))
```

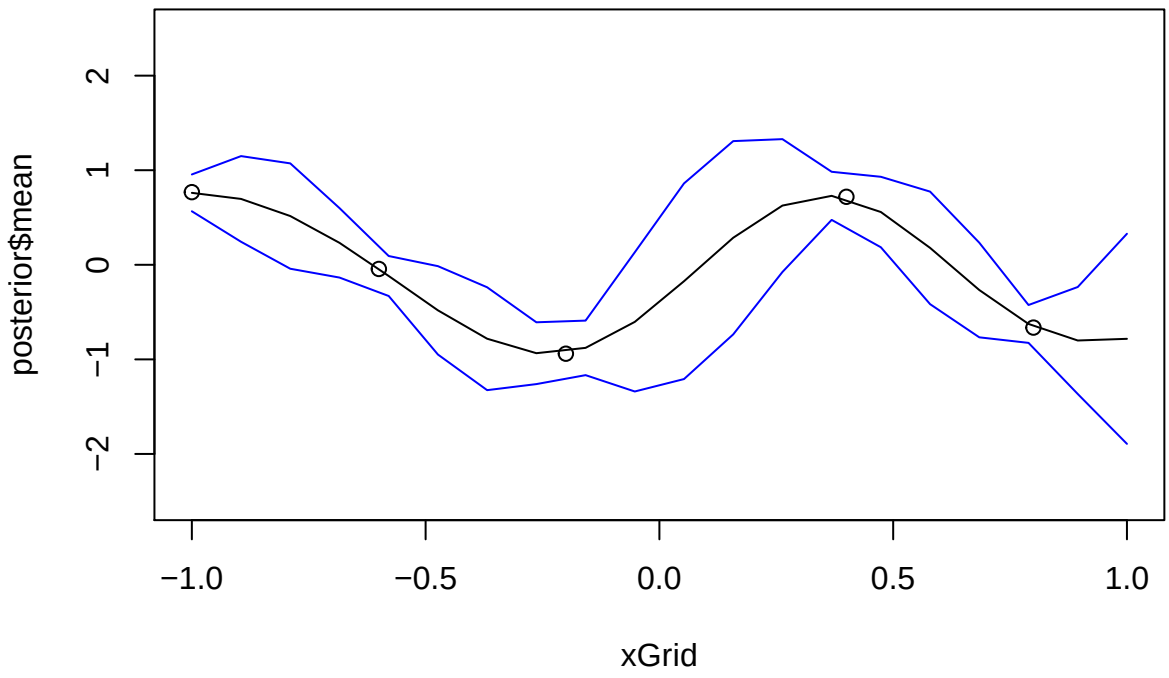
```r
points(xGrid, posterior$mean + 1.96*std_dev, col="blue", type="l")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue", type="l")
```



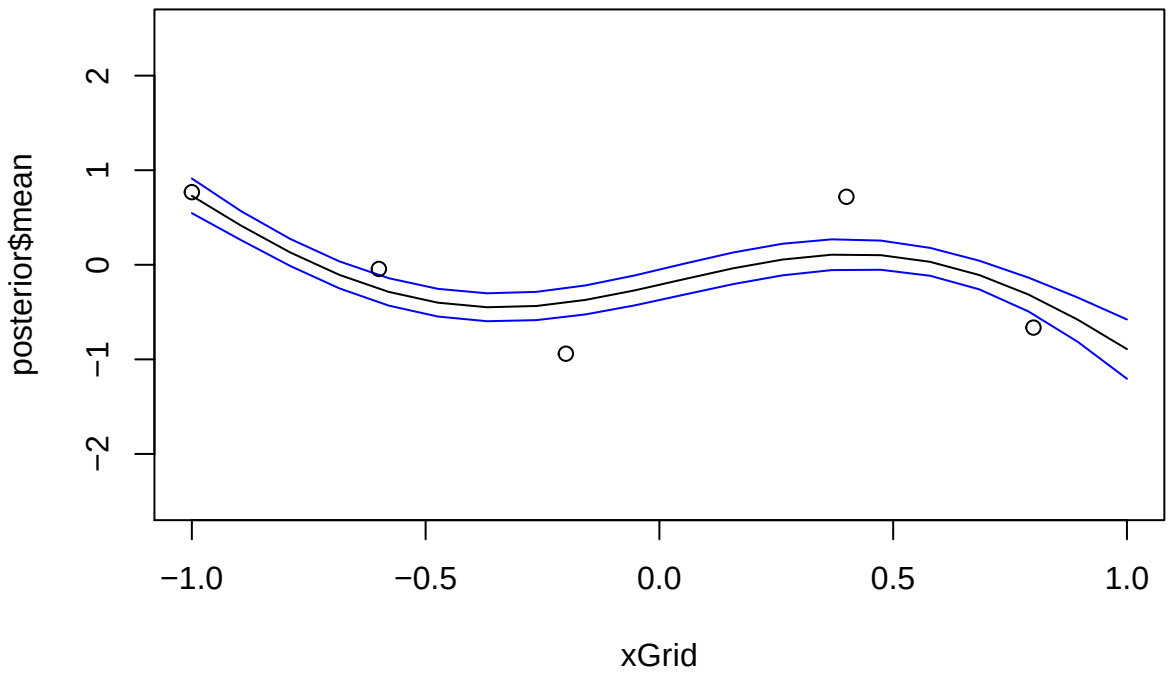

```r
# 3 - Update posterior with another point. (by recalculating posterior wiht Algorithm 2.1)
x = c(0.4, -0.6)
y = c(0.719 , -0.044)
posterior = posteriorGP(x, y, cov_function, sigma_n, xGrid)

plot(xGrid, posterior$mean, ylim = c(-2.5,2.5)) # posterior mean
std_dev = sqrt(diag(posterior$cov))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue", type="l")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue", type="l")
```




```
# 4 - Calculate the posterior with five observations.
x = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
posterior = posteriorGP(x, y, cov_function, sigma_n, xGrid)

plot(xGrid, posterior$mean, ylim = c(-2.5,2.5), type="l") # posterior mean
std_dev = sqrt((diag(posterior$cov)))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue", type="l")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue", type="l")
points(x,y)
```




```
# 5 - Redo (4) with the same sigma_f but with a higher ell.
sigma_f = 1
ell = 1
SEkernel <- rbfdot(sigma = 1/(2*ell^2))

x = c(-1.0, -0.6, -0.2, 0.4, 0.8)
y = c(0.768, -0.044, -0.940, 0.719, -0.664)
posterior = posteriorGP(x, y, cov_function, sigma_n, xGrid)

plot(xGrid, posterior$mean, ylim = c(-2.5,2.5), type="l") # posterior mean
std_dev = sqrt((diag(posterior$cov)))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue", type="l")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue", type="l")
points(x,y)
```



The two plots has the same sigma_f but different ell-values. The last plot has a smoother mean prediction, with a higher ell value. A higher ell gives a higher covariance for two points that are further apart. When we used more training data, we get a narrower confidence intervall. We are also more confident in our posterior distribution around our training points.

Part 2 - GP Regression with kernlab

```
temprature = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTull
time = (1:2190)
nr_year = 2190/365
day = rep((1:365), nr_year)

reduce_data = function(array , nth){
  array[seq(1, length(array), nth)]
}

temp = reduce_data(temprature$temp,5)
time = reduce_data(time, 5)
day = reduce_data(day, 5)

set_se_kernel = function(ell, sigma_f){
  se_kernel = function(x , y){
    r_square = sum((x-y)*(x-y)) # Euclidian distance^2
    return(sigma_f^2*exp(-r_square/(2*ell^2)))
```

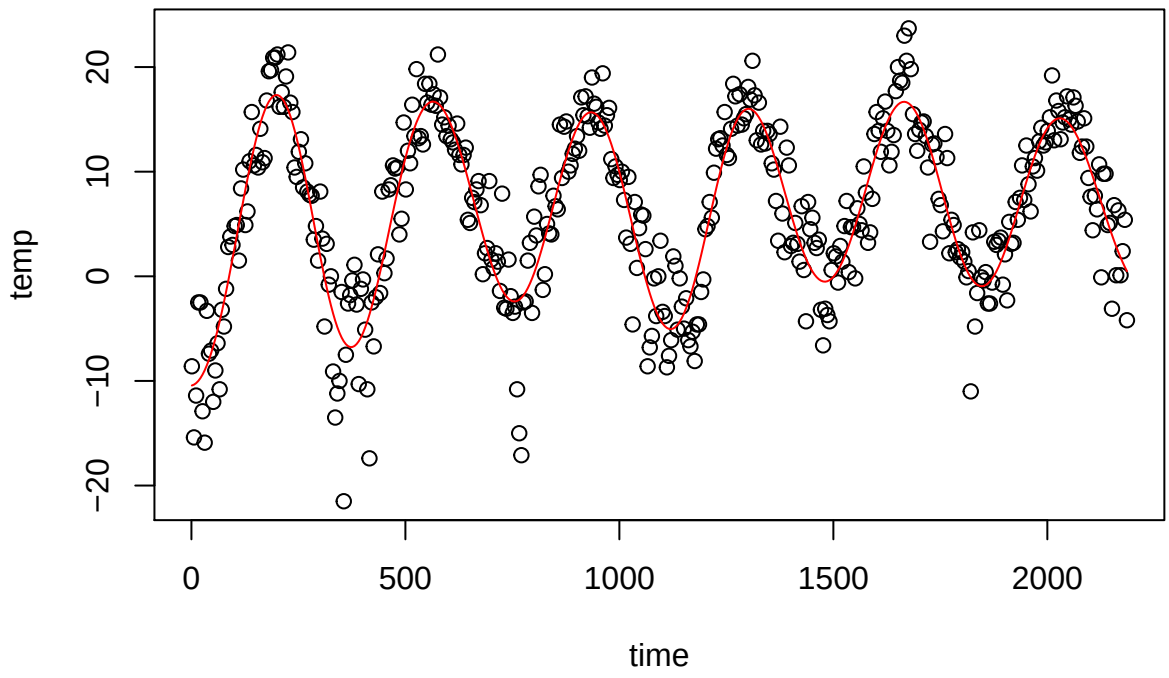
```
  }
  class(se_kernel) <- "kernel"
  return(se_kernel)
}

se_kernel = set_se_kernel(ell=1, sigma_f=1)

# Evaluate kernel in x=1 and x'=2
se_kernel(1,2)
```

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

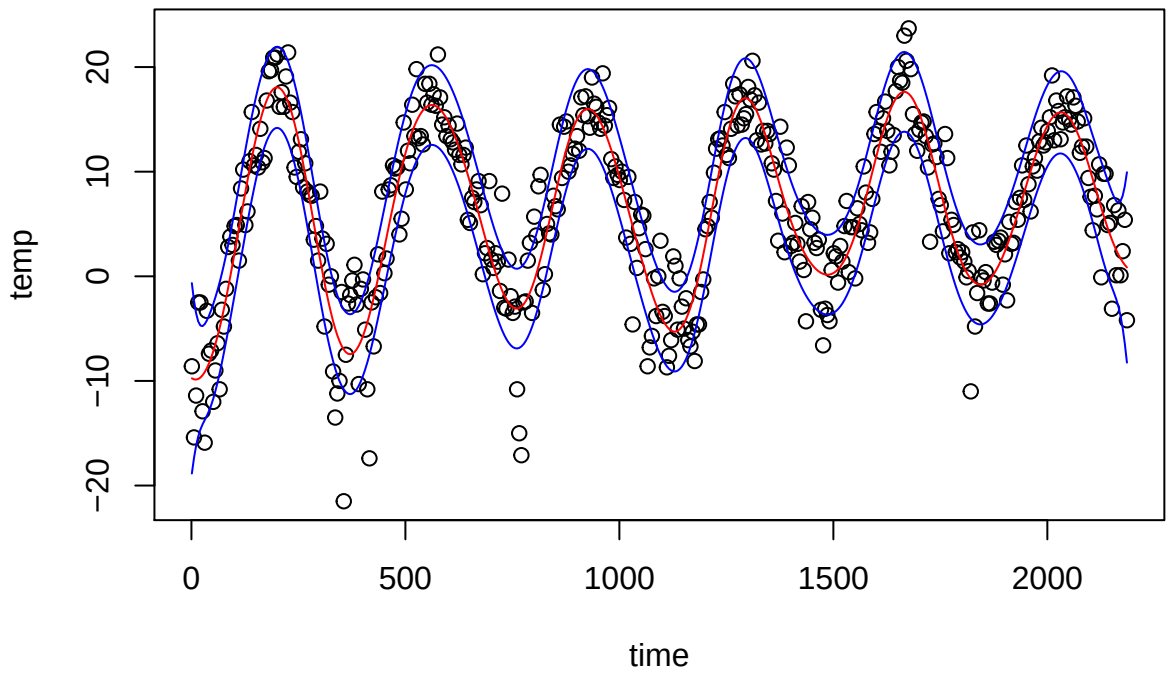
```
k = function(x, x_star){
  return(kernelMatrix(kernel = se_kernel, x=x, y=x_star))
}

# Evaluate the covariance matrix, k
x = c(1, 3, 4)
x_star = c(2, 3, 4)
k(x, x_star)
```

```
## 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
```

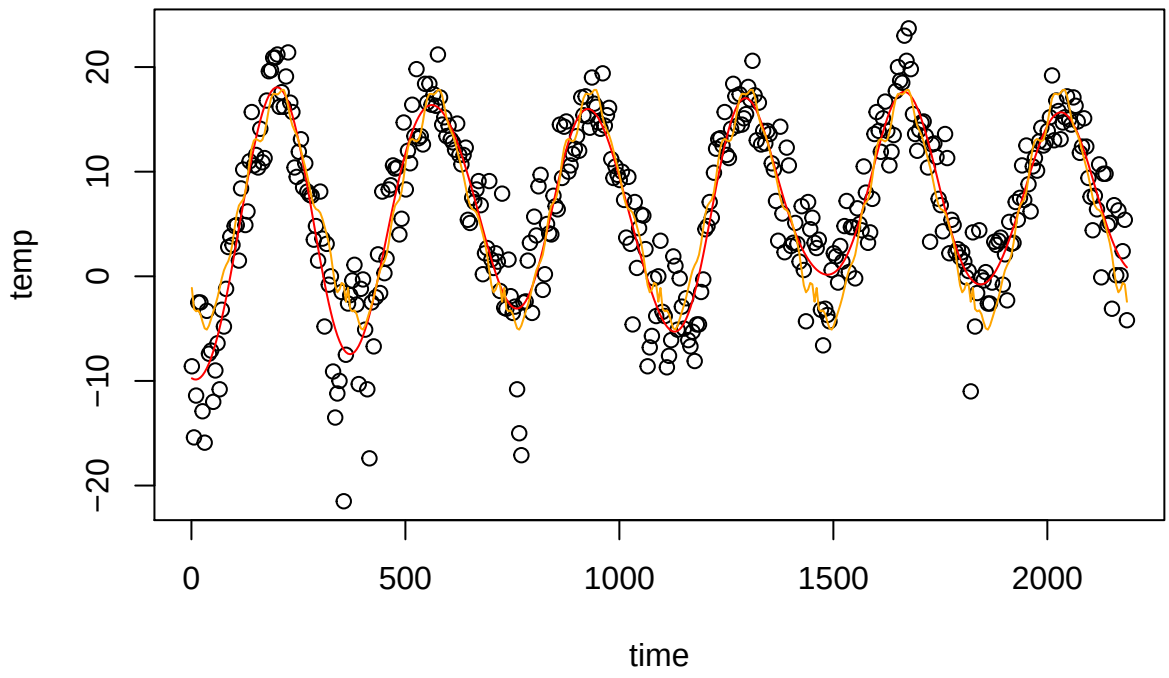
```
# 2 - Find f* with function: gausspr(kernlab)
sigma_f = 20
ell = 0.2
se_kernel = set_se_kernel(ell=ell, sigma_f=sigma_f)

# Find sigma_n
quadratic_reg = lm(temp ~ time + I(time^2))
sigma_n = sd(quadratic_reg$residuals)

gp_model = gausspr(x = time, y = temp, type="regression", kernel=se_kernel, var = sigma_n^2, variance.m

mean_pred = predict(gp_model, time)
plot(time, temp)
points(time, mean_pred, type = "l", col="red")
```




```
#lines(time, mean_pred + 1.96 * predict(gp_model, time, type="sdeviation")) # ---- wrong due to scalein


# 3 - Scale and calculate with Algorithm 2.1
mean_temp = mean(temp)
sd_temp = sd(temp)
time_scale = scale(time)
temp_scale = scale(temp)

# Find sigma_n
quadratic_reg = lm(temp_scale ~ time_scale + I(time_scale^2))
sigma_n = sd(quadratic_reg$residuals)

post_gp = posteriorGP(time_scale, temp_scale, k, sigma_n, time_scale)
st_dev_new = sqrt(diag(post_gp$cov))*sd_temp
mean_pred_new = post_gp$mean*sd_temp+mean_temp
plot(time, temp)
lines(time, mean_pred_new, col ="red")
lines(time, mean_pred_new+1.96*st_dev_new, col="blue")
lines(time, mean_pred_new-1.96*st_dev_new, col="blue")
```




```
# 4 - use of Day
day_scale = scale(day)

#Find sigma_n
quadratic_reg = lm(temp_scale ~ day_scale + I(day_scale^2))
sigma_n = sd(quadratic_reg$residuals)

post_gp = posteriorGP(day_scale, temp_scale, k, sigma_n, day_scale)
st_dev_day = sqrt(diag(post_gp$cov))*sd_temp
mean_pred_day = post_gp$mean*sd_temp+mean_temp

# old plot
plot(time, temp)
lines(time, mean_pred_new, col ="red")
#lines(time, mean_pred_new+1.96*st_dev_new, col="blue")
#lines(time, mean_pred_new-1.96*st_dev_new, col="blue")
# new lines from day
lines(time, mean_pred_day, col ="orange")
```

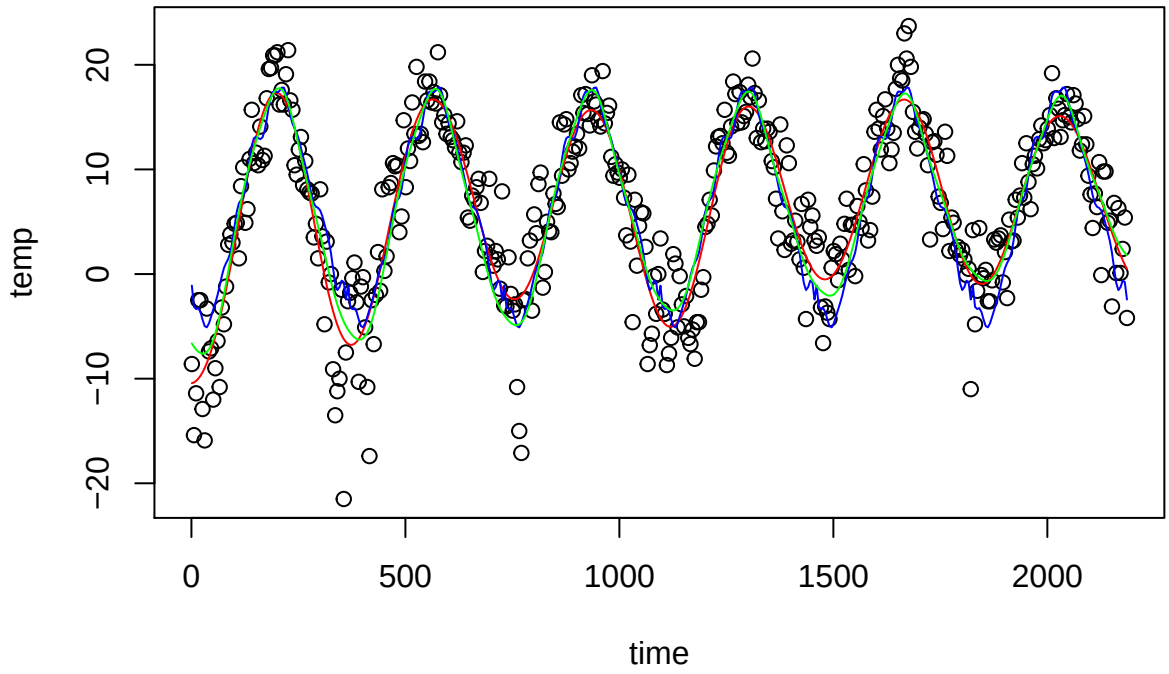



```
#lines(time, mean_pred_day+1.96*st_dev_day, col="green")
#lines(time, mean_pred_day-1.96*st_dev_day, col="green")
```

When predicting using time and sqare exponential kernel we have a low correlation between the same date between two consecutive years. But when using day and square exponential kernel we assume the same correlation between the same date, no mather which year that date is from. And we get a prediction that does not take the long term change (ex. warmer winters) into account. Day is not that smooth when we change year, because day=1 and day=365 will not get a strong correlation, even though they are close.

```
# 5 - Generalization periodic kernel
sigma_f = 20
ell_1 = 1
ell_2 = 10
d = 365/sd(time)

set_gen_kernel = function(ell_1, ell_2, sigma_f, d){
  gen_kernel = function(x , y){
    r_square = sum((x-y)*(x-y)) #euclidian distance ^2
    return(sigma_f^2* exp(-(2*sin(pi*sqrt(r_square)/d)^2)/(ell_1^2)) * exp(-r_square/(2*ell_2^2)))
  }
  class(gen_kernel) <- "kernel"
  return(gen_kernel)
}

k_gen = function(x, x_star){
```

```
  return(kernelMatrix(kernel = gen_kernel, x=x, y=x_star))
}

# Set sigma_n
quadratic_reg = lm(temp ~ time + I(time^2))
sigma_n = sd(quadratic_reg$residuals)

gen_kernel = set_gen_kernel(ell_1, ell_2, sigma_f, d)
gp_model = gausspr(x = time, y = temp, type="regression", kernel=gen_kernel, var = sigma_n^2)

mean_pred_gen = predict(gp_model, time)
plot(time, temp)
points(time, mean_pred, type="l", col="red")
points(time, mean_pred_day, type = "l", col="blue")
points(time, mean_pred_gen, type = "l", col="green")
```

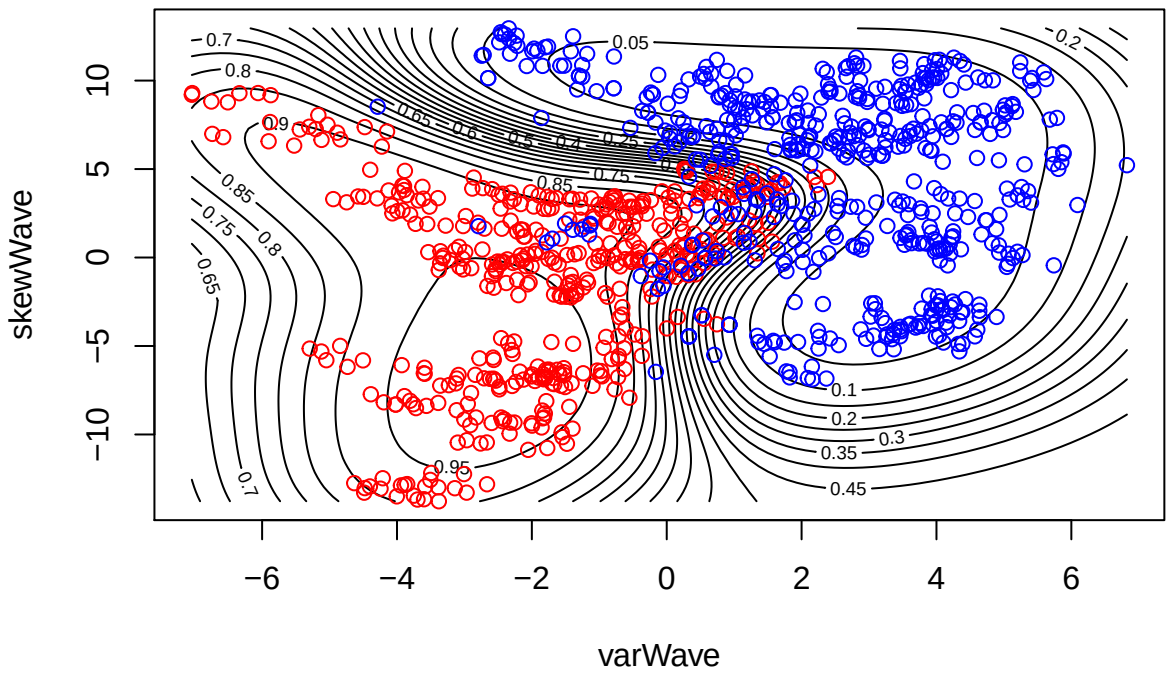


The periodic kernel will take both time and periodisation into accout. Two equal dates will have a higher covariance if they are from two consecutive years. This means that we mannage to capture the long term trend better, because of the second "exp()" in the kernel.

Part 3 - GP Classification with kernlab

```
#install.packages("AtmRay") # To make 2D grid like in Matlab's meshgrid.
library(AtmRay)
```

```
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,]

# 1 - prediciton with two covariats.
gp_fraud = gausspr(fraud ~ varWave + skewWave, data=train, type="classification")


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

fraud_pred = predict(gp_fraud, train[,1:4])
# Confusion matrix
table = table(fraud_pred, train[,5])
missclass = 1 - sum(diag(table))/sum(table)
# Create the basis of the plot
x1 = seq(min(train[,1]),max(train[,1]),length=100)
x2 = seq(min(train[,2]),max(train[,2]),length=100)
grid_points = meshgrid(x1, x2)
grid_points = cbind(c(grid_points$x), c(grid_points$y))

grid_points = data.frame(grid_points)
names(grid_points) = names(train)[1:2]

prob_pred = predict(gp_fraud, grid_points, type="probabilities")

# Plot the plot
contour(x1,x2,matrix(prob_pred[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = '
points(train[train[,5]==1,1],train[train[,5]==1,2],col="red")
points(train[train[,5]==0,1],train[train[,5]==0,2],col="blue")
```




```
cat("Confusion matrix: ")
```

```
## Confusion matrix:
```

```
table
```

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

```
cat("missclassification rate: ", missclass)
```

```
## missclassification rate:  0.059
```

```
# 2 apply model on test data.
test = data[-SelectTraining,]
test_pred = predict(gp_fraud, test[,1:4])
table = table(test_pred, test[,5])
missclass = 1 - sum(diag(table))/sum(table)

cat("Confusion matrix:")
```

```
## Confusion matrix:
```

```
table
```

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

```
cat("missclassification rate: ", missclass)
```

```
## missclassification rate:  0.07526882
```

```
# 3 training model with all 4 covariats

gp_fraud = gausspr(fraud ~ varWave + skewWave + kurtWave + entropyWave, data=train, type="classificatio
```

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

```
test_pred = predict(gp_fraud, test[,1:4])
table = table(test_pred, test[,5])
missclass = 1 - sum(diag(table))/sum(table)

cat("Confusion matrix:")
```

```
## Confusion matrix:
```

```
table
```

```
##
## test_pred   0   1
##         0 216   0
##         1   2 154
```

```
cat("missclassification rate: ", missclass)
```

```
## missclassification rate:  0.005376344
```

The model with all four covariats is performing better while predicting fraud in the test data. One reason could be that the model now uses more input data (more information), while predicting the data. This is not true for all input data, becaus there is a risk of overfitting the moddel to the training data.