Lab4

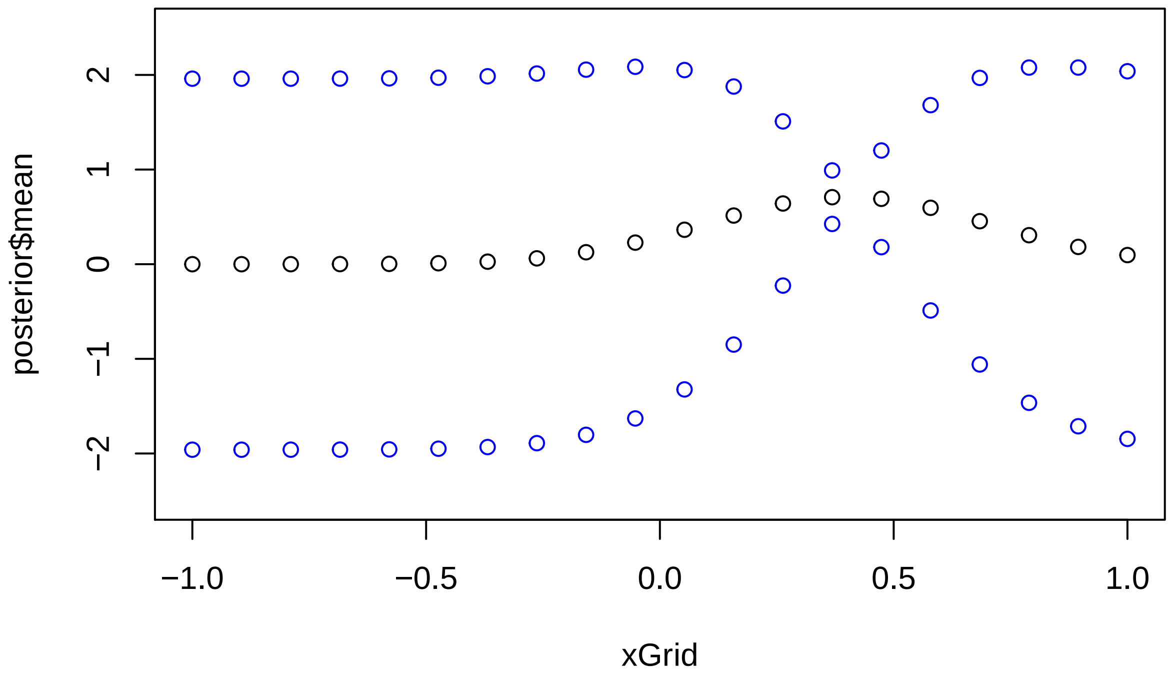
Oskar Hidén - oskhi827

10/18/2020

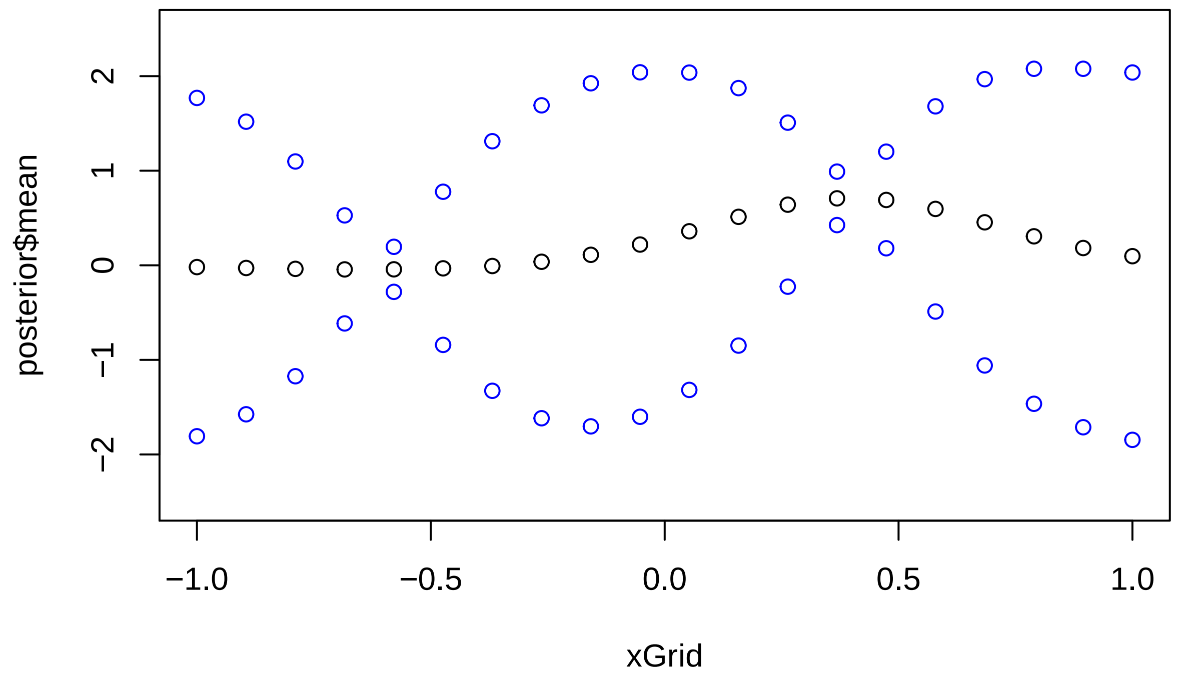
2.1 - Implementing GP Regression

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

# ------- TEST ----
#ell <- 1
#SEkernel <- rbfdot(sigma = 1/(2*ell^2)) # Note how I reparametrize the rbfdot (which is the SE kernel)
#SEkernel(1,2) # Just a test - evaluating the kernel in the points x=1 and x'=2.
# Computing the whole covariance matrix K from the kernel. Just a test.
#kernelMatrix(kernel = SEkernel, x = X, y = Xstar) # So this is K(X,Xstar).
# ------End TEST--

cov_function = function(X, Xstar){
  return(kernelMatrix(kernel = SEkernel, x=X, y=Xstar))
}

# cov_function(1,2)

posteriorGP = function(X_input, y_targets, k_cov_function, sigmaNoise=1, XStar){

  # wehre to inplement noise(sigmaNoise)?
  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
```

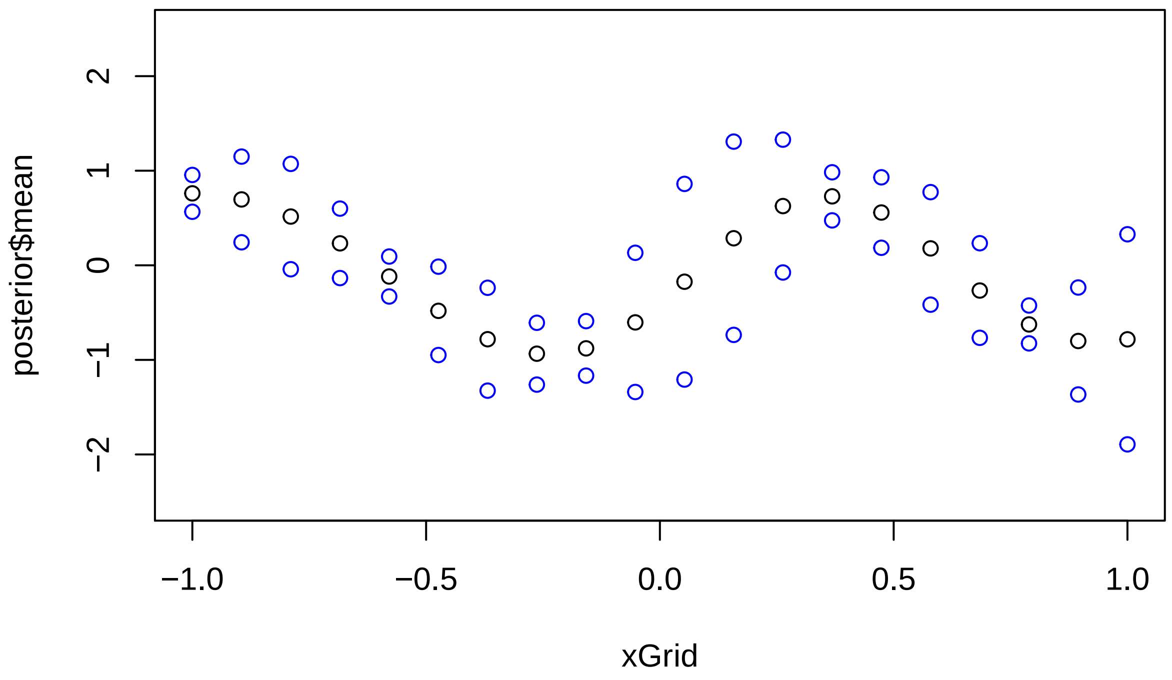
```r
SEkernel <- rbfdot(sigma = 1/(2*ell^2))

sigma_n = 0.1
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))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue")
```



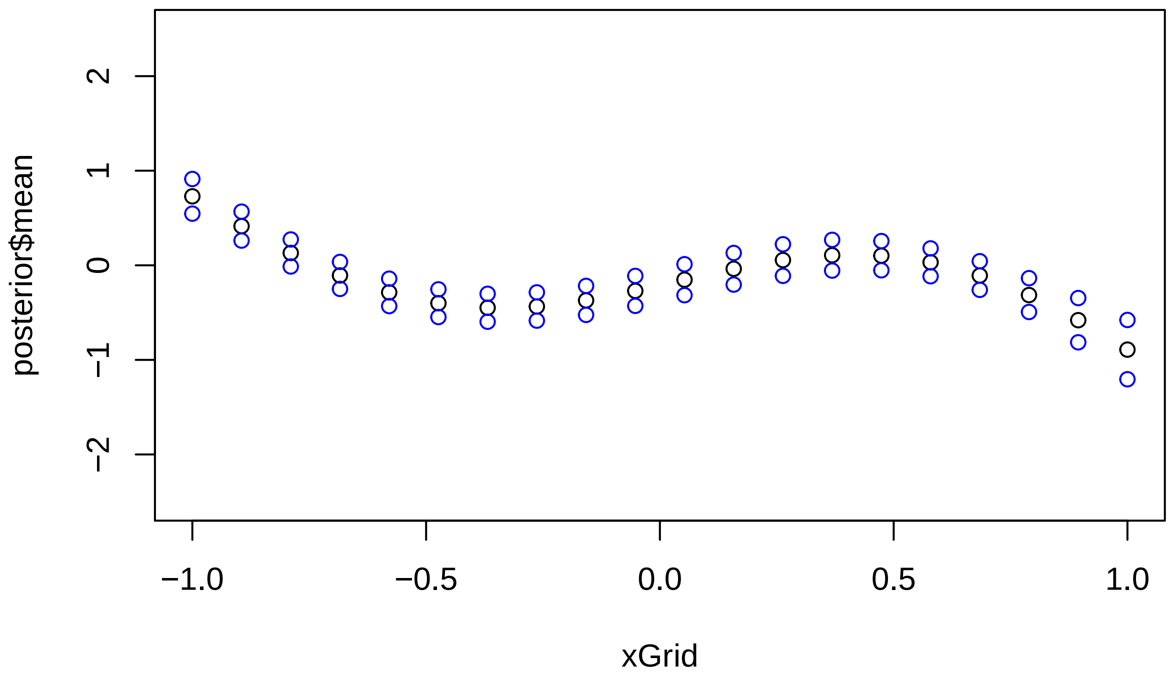
```r
#max(posterior$mean)

# 3
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$mean + 1.96*std_dev, col="blue")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue")
```



```r
# 4
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$mean, ylim = c(-2.5,2.5)) # posterior mean
std_dev = sqrt((diag(posterior$cov)))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue")
```



```r
# 5
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$mean, ylim = c(-2.5,2.5)) # posterior mean
std_dev = sqrt((diag(posterior$cov)))
points(xGrid, posterior$mean + 1.96*std_dev, col="blue")
points(xGrid, posterior$mean - 1.96*std_dev, col="blue")
```



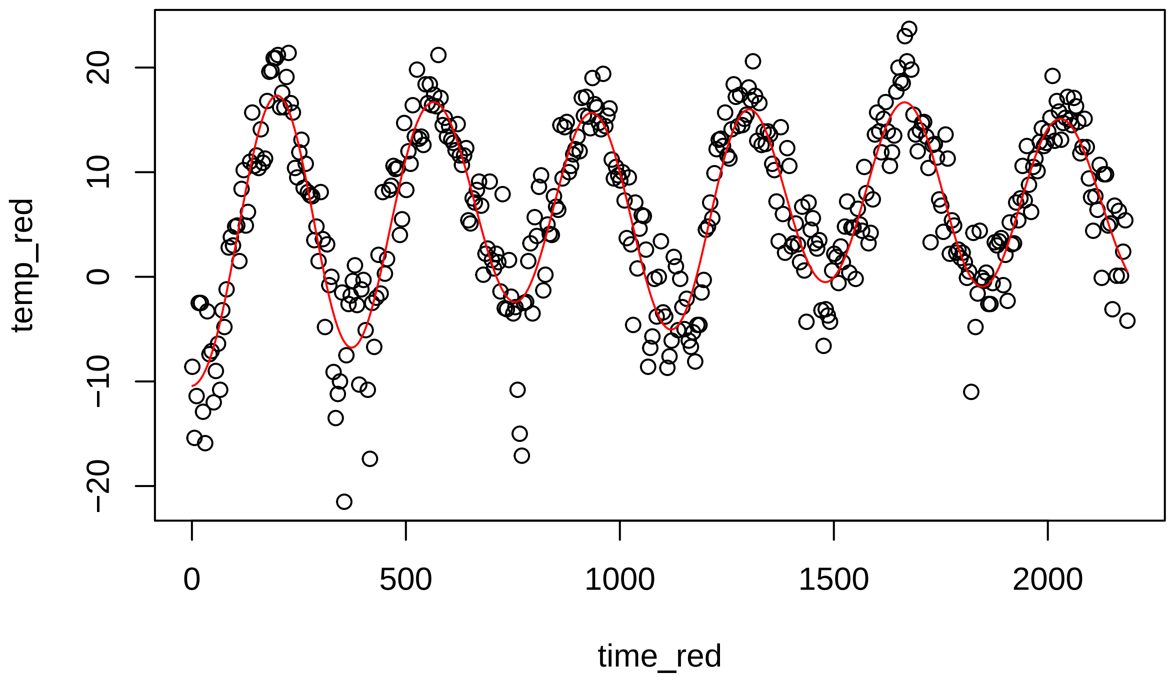
The two plots has the same sigma_f but different l-values. The last plot is smoother, with a higher l value. And the 95% confidence intervall is closer to posterior mean.

2.2 - GP Regression with kernlab

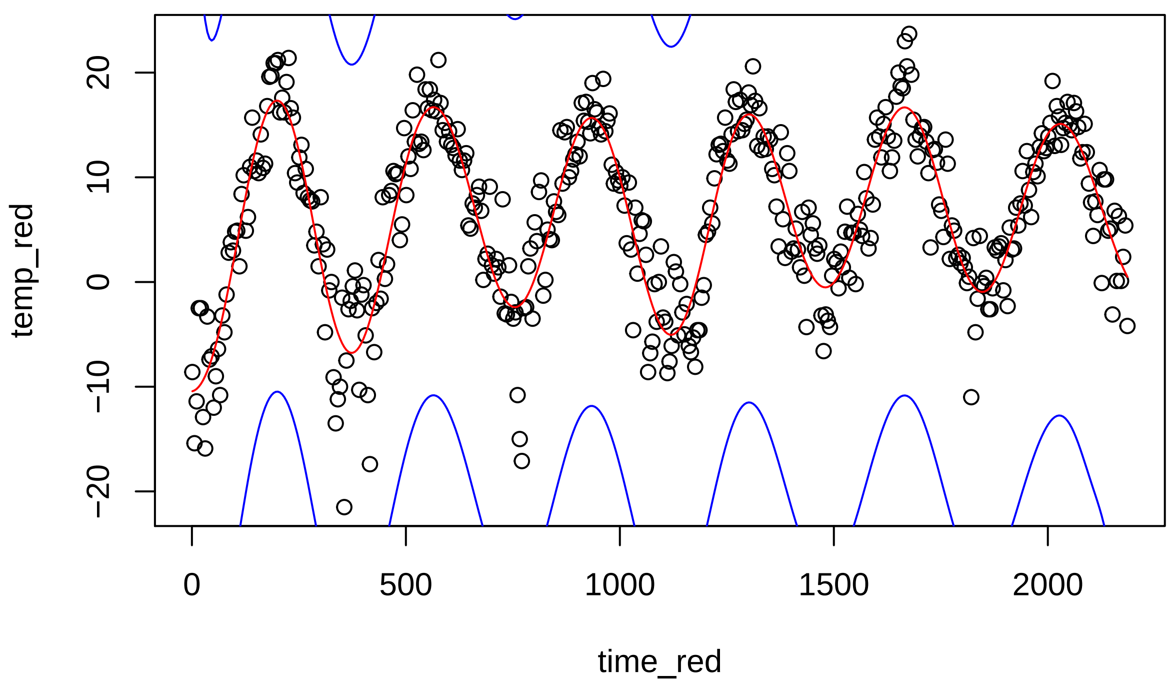
```r
temp = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c

time = (1:2190)
nr_year = 2190/365
day = rep((1:365), nr_year)

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

time_red = reduce_data(time, 5)
day_red = reduce_data(day, 5)

?kernelMatrix
# ------OLD ----------
# Set sqare exponential kernel function
ell = 1
sigma_f = 1
se_kernel = function(ell, sigma_f){
  SEkernel <- sigma_f*rbfdot(sigma = 1/(2*ell^2))
}
k = function(x, x_star, ell, sigma_f){
  SEkernel <- rbfdot(sigma = 1/(2*ell^2))
  return(kernelMatrix(kernel = SEkernel, x=x, y=x_star)*sigma_f^2)
}
# k(x, x_star, 1, 1) #Old k
# ------End OLD------------

set_se_kernel = function(ell, sigma_f){
  se_kernel = function(x , y){
    r_square = sum((x-y)*(x-y)) #euclidian distance
    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)
x = c(1, 3, 4)
x_star = c(2, 3, 4)
#kernelMatrix(kernel = se_kernel, x=x, y=x_star)

k = function(x, x_star){
  return(kernelMatrix(kernel = se_kernel, x=x, y=x_star))
}
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 gausspr
sigma_f = 20
ell = 0.2
se_kernel = set_se_kernel(ell=ell, sigma_f=sigma_f)

# Find sigma_n
temp_red = reduce_data(temp$temp,5)
quadratic_reg = lm(temp_red ~ time_red + I(time_red^2))
sigma_n = sd(quadratic_reg$residuals)

gp_model = gausspr(x = time_red, y = temp_red, type="regression", kernel=se_kernel, var = sigma_n^2, va

mean_pred= predict(gp_model, time_red)
plot(time_red, temp_red)
points(time_red, mean_pred, type = "l", col="red")
lines(time_red, mean_pred+1.96*predict(gp_model, time_red, type="sdeviation")) # wrong due to scaleing
```



```
# Scale data -> GP -> Scale back. To obtain correct.
mean = mean(temp_red)
st_dev = sd(temp_red)
temp_red_scale = scale(temp_red)

gp_model_scaled = gausspr(x = time_red, y = temp_red_scale, type="regression", kernel=se_kernel, var = s

# Scale back mean and standard deviation
mean_pred_scaled = predict(gp_model_scaled, time_red)
mean_pred_new = mean_pred_scaled*st_dev+mean
st_dev_scaled = predict(gp_model_scaled, time_red, type="sdeviation")
st_dev_new = st_dev_scaled*st_dev

# Plot mean and std
plot(time_red, temp_red)
lines(time_red, mean_pred_new, col ="red")
lines(time_red, mean_pred_new+1.96*st_dev_new, col="blue")
lines(time_red, mean_pred_new-1.96*st_dev_new, col="blue")
```



```
mean_pred[100]
```

```
## [1] 10.65205
```

```
mean_pred_new[100]
```

```
## [1] 10.65205
```

```
# 3
```