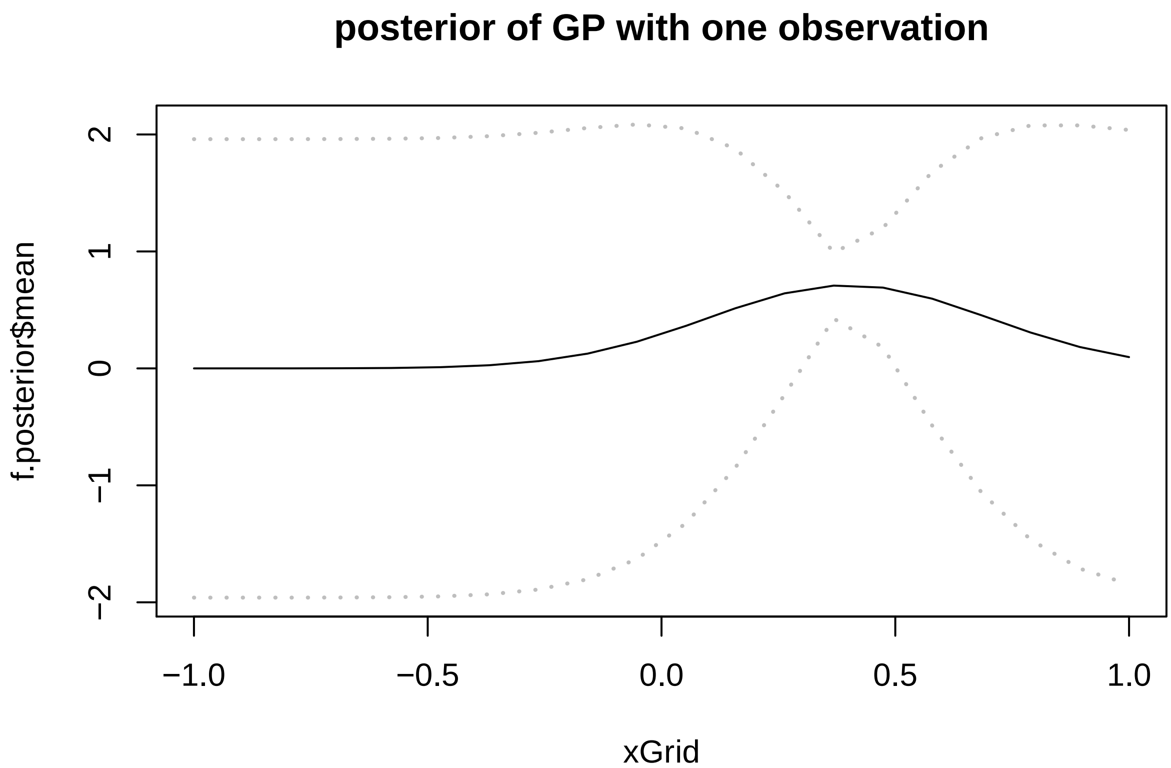
lab4_william_anzen

2.1. Implementing GP Regression.

```r
# Covariance function
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)
}

# Calculate the Posterior GP of f
posteriorGP = function (X, y, XStar, sigmaNoise, k, ...){
  n = length(X)
  K = k(X,X, ...)
  l = t(chol(K+(sigmaNoise^2)*diag(n)))
  alpha = solve(t(l),solve(l, y))
  kStar = k(X,XStar, ...)
  fMean = t(kStar)%*%alpha

  v = solve(l,kStar)
  fVar = diag(k(XStar, XStar, ...) - t(v)%*%v)

  return(list(mean=fMean, var=fVar))
}
```

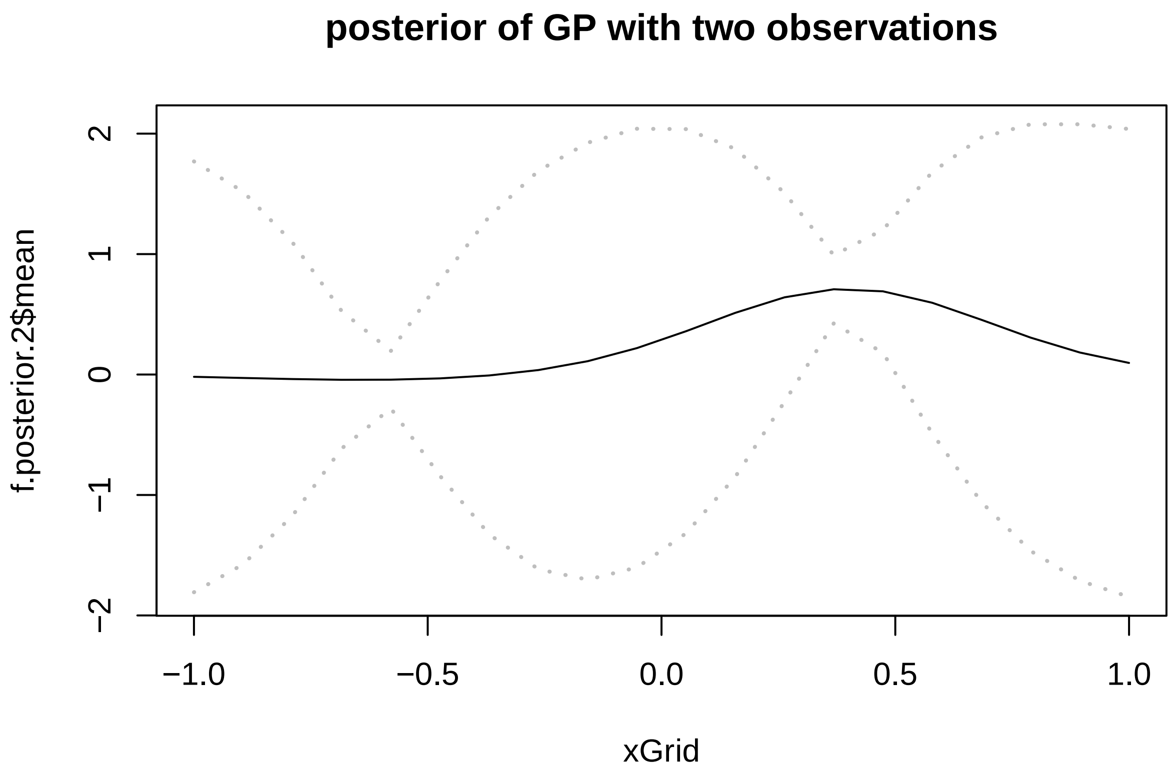
(1)

```r
sigma.f = 1
l = 0.3
xGrid = seq(-1,1,length=20)
obs = data.frame(x=0.4, y=0.719)
sigma.n = 0.1

f.posterior <- posteriorGP(obs$x,obs$y,xGrid,sigma.n,SquaredExpKernel, sigma.f, l)

plot(xGrid, f.posterior$mean, type="l",
     ylim=c(min(f.posterior$mean - 1.96*sqrt(f.posterior$var)),
            max(f.posterior$mean + 1.96*sqrt(f.posterior$var))),
     main="posterior of GP with one observation")
lines(xGrid, f.posterior$mean - 1.96*sqrt(f.posterior$var), col = "gray", lty=21,lwd = 2)
lines(xGrid, f.posterior$mean + 1.96*sqrt(f.posterior$var), col = "gray", lty=21, lwd = 2)
```



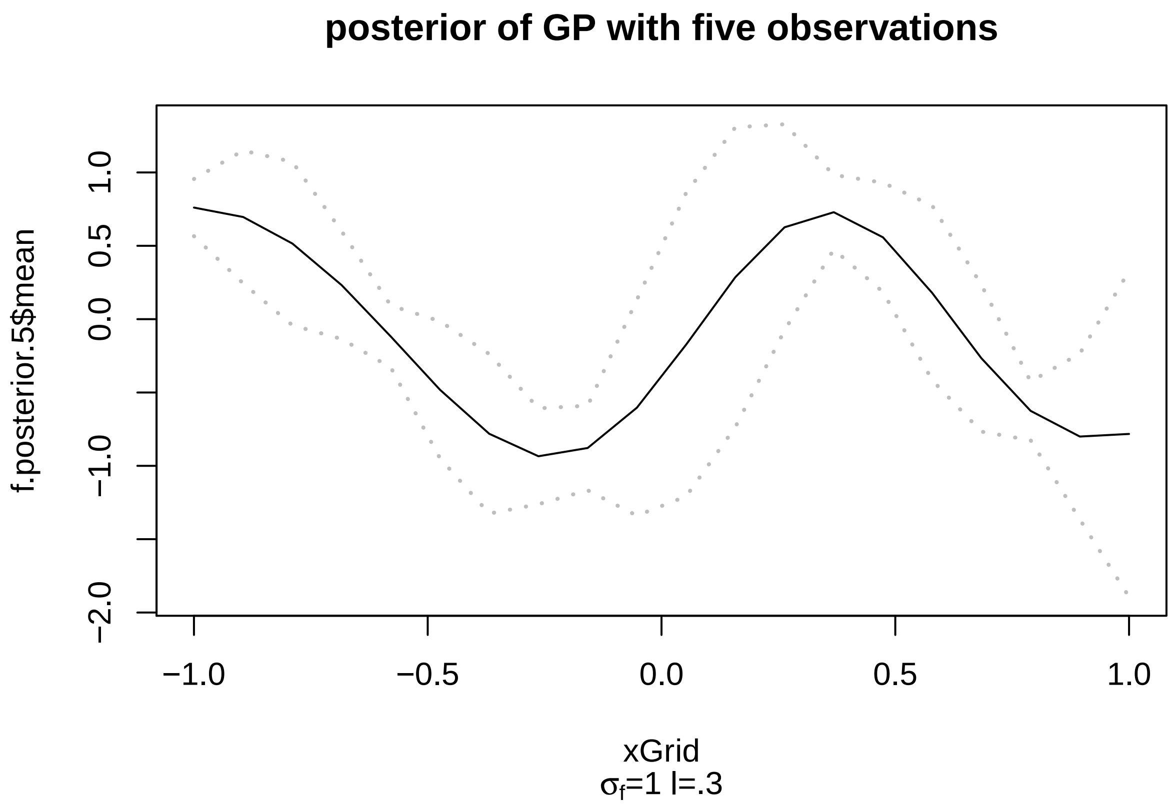
(2)

```
obs.2 = data.frame(x=c(0.4,-0.6),y=c(0.719,-0.044))

f.posterior.2 = posteriorGP(obs.2$x,obs.2$y,xGrid,sigma.n,SquaredExpKernel, sigma.f, l)

plot(xGrid, f.posterior.2$mean, type="l",
     ylim=c(min(f.posterior.2$mean - 1.96*sqrt(f.posterior.2$var)),
            max(f.posterior.2$mean + 1.96*sqrt(f.posterior.2$var))),
     main="posterior of GP with two observations")
lines(xGrid, f.posterior.2$mean - 1.96*sqrt(f.posterior.2$var), col = "gray", lty=21,lwd = 2)
lines(xGrid, f.posterior.2$mean + 1.96*sqrt(f.posterior.2$var), col = "gray", lty=21, lwd = 2)
```



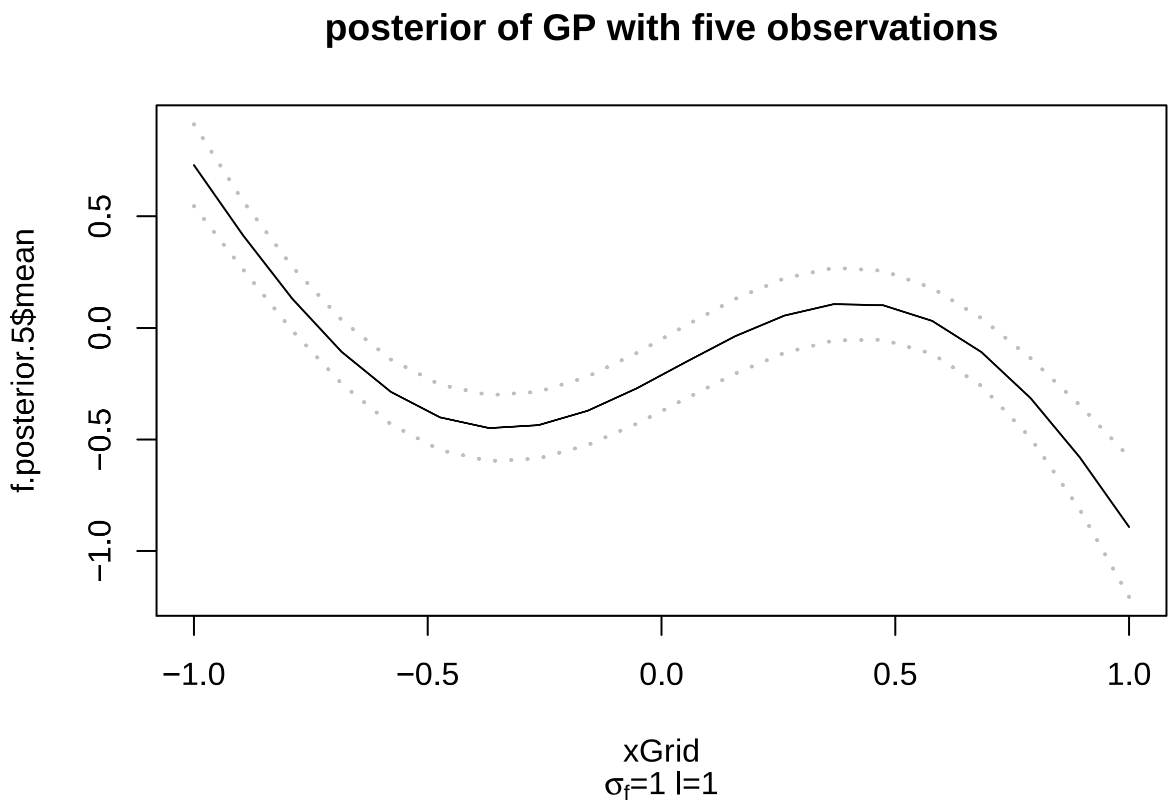
(3)

```
obs.5 = data.frame(x=c(-1,-0.6, -0.2, 0.4, 0.8),y=c(0.768,-0.044, -0.940,0.719,-0.664))


f.posterior.5 = posteriorGP(obs.5$x,obs.5$y,xGrid,sigma.n,SquaredExpKernel, sigma.f, l)

plot(xGrid, f.posterior.5$mean, type="l",
     ylim=c(min(f.posterior.5$mean - 1.96*sqrt(f.posterior.5$var)),
            max(f.posterior.5$mean + 1.96*sqrt(f.posterior.5$var))),
     main="posterior of GP with five observations", sub=expression(sigma[f]*'=1 l=.3'))
lines(xGrid, f.posterior.5$mean - 1.96*sqrt(f.posterior.5$var), col = "gray", lty=21,lwd = 2)
lines(xGrid, f.posterior.5$mean + 1.96*sqrt(f.posterior.5$var), col = "gray", lty=21, lwd = 2)
```



(4)

```
sigma.f = 1
l = 1

f.posterior.5 = posteriorGP(obs.5$x,obs.5$y,xGrid,sigma.n,SquaredExpKernel, sigma.f, l)

plot(xGrid, f.posterior.5$mean, type="l",
     ylim=c(min(f.posterior.5$mean - 1.96*sqrt(f.posterior.5$var)),
            max(f.posterior.5$mean + 1.96*sqrt(f.posterior.5$var))),
     main="posterior of GP with five observations", sub=expression(sigma[f]*'=1 l=1'))
lines(xGrid, f.posterior.5$mean - 1.96*sqrt(f.posterior.5$var), col = "gray", lty=21,lwd = 2)
lines(xGrid, f.posterior.5$mean + 1.96*sqrt(f.posterior.5$var), col = "gray", lty=21, lwd = 2)
```



(5)

GP Regression with kernlab

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

# install.packages("kernlab")
library(kernlab)

time = seq(1,2186,5)
day = rep(seq(1,361,5), times=6)
tempTime = temps$temp[time]


KernelFunc <- function(sigmaF = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    n1 <- length(x)
    n2 <- length(y)
    K <- matrix(NA,n1,n2)
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x-y[i])/ell)^2 )
    }
    return(K)
  }
  class(rval) <- "kernel"
```

```r
  return(rval)
}


X = c(1,3,4)
XStar = c(2,3,4)
KernelFunct = KernelFunc(sigmaF = 1, ell = 2) #This is a kernelfunction
KernelFunct(1,2) # Evaluating the kernel in x=1, x'=2
```

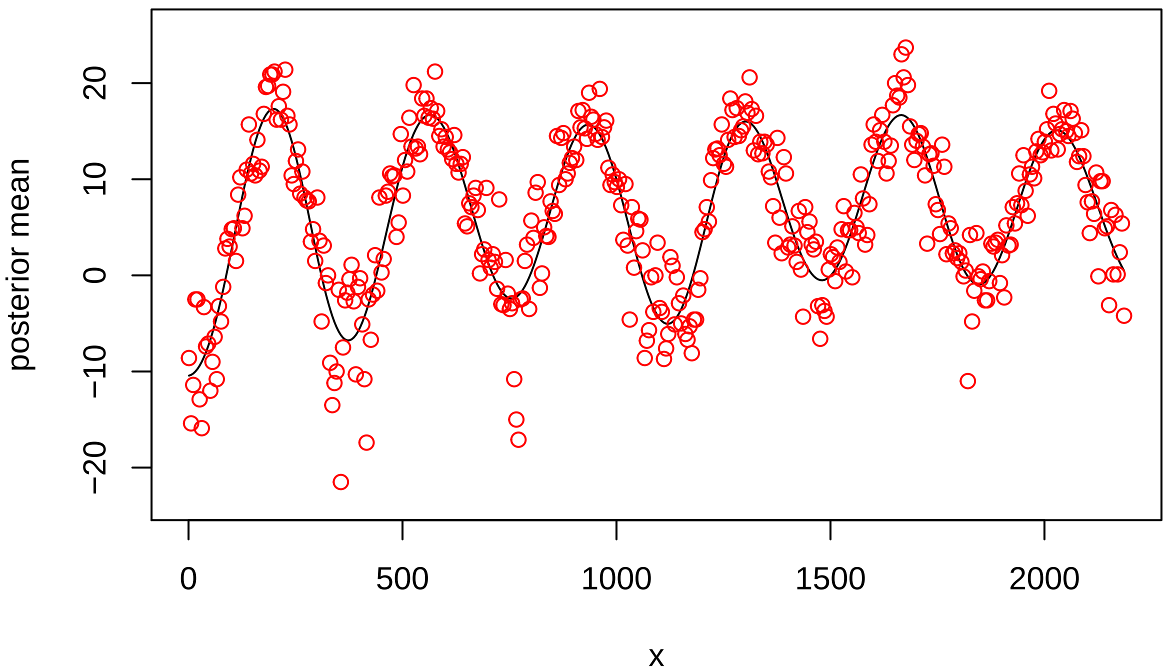
(1)

```
##           [,1]
## [1,] 0.8824969
```

```r
## computing the covariance matrix over the whole input set
kernelMatrix(kernel = KernelFunct, x= X, y= XStar) # K(X,Xstar)
```

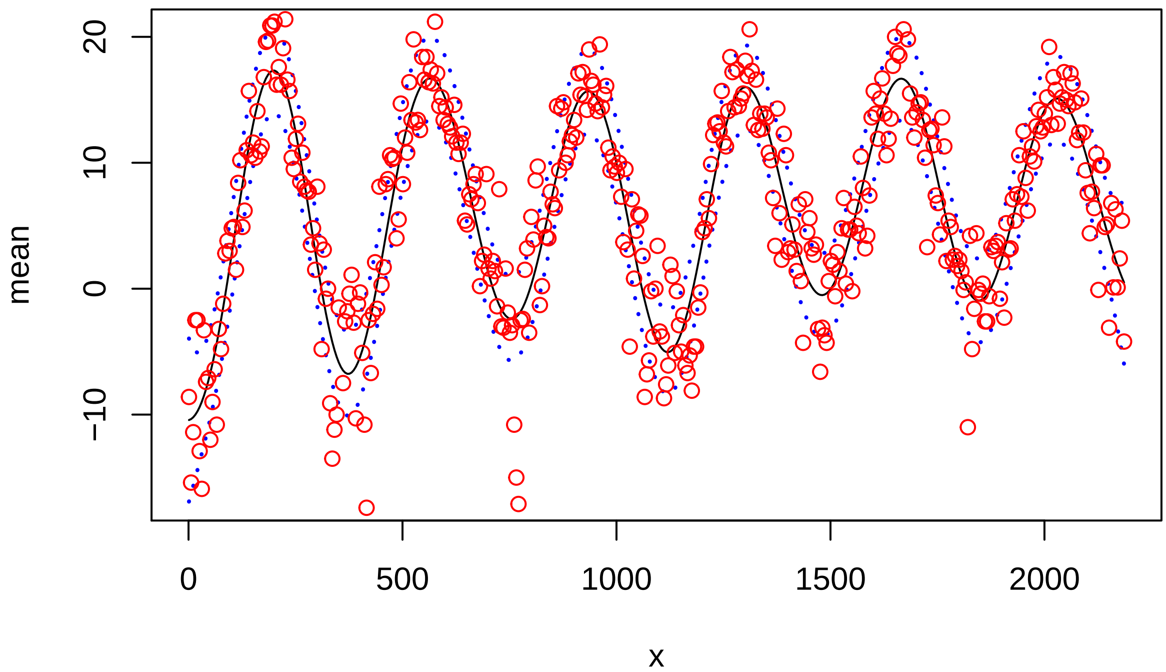
```
## An object of class "kernelMatrix"
##           [,1]      [,2]      [,3]
## [1,] 0.8824969 0.6065307 0.3246525
## [2,] 0.8824969 1.0000000 0.8824969
## [3,] 0.6065307 0.8824969 1.0000000
```

```r
plotGP = function(x, mean, var = NULL , obs) {
  if( !is.null(var)){
    plot(x, mean, type="l", ylim=c(min(mean - 1.96*sqrt(var)),
                max(mean + 1.96*sqrt(var))))
    lines(x, mean - 1.96*sqrt(var), col = "blue", lty=21,lwd = 2)
    lines(x, mean + 1.96*sqrt(var), col = "blue", lty=21, lwd = 2)
    points(obs$x, obs$y, col='red', pch=1)
  } else {
    plot(x, mean, type= "l", ylab="posterior mean", ylim=c(min(obs$y)-2, max(obs$y)+2))
    points(obs$x, obs$y, col='red', pch=1)
  }

}

fit = lm(tempTime ~ time+ time^2)
sigma.n = sd(fit$residuals)

sigma.f = 20
l = .2

fit.GP = gausspr(time,tempTime,kernel=KernelFunc, kpar=list(sigmaF = sigma.f, ell = l), var=sigma.n^2)

meanPred = predict(fit.GP, time)
plotGP(time,meanPred, obs = data.frame(x=time, y=tempTime))
```



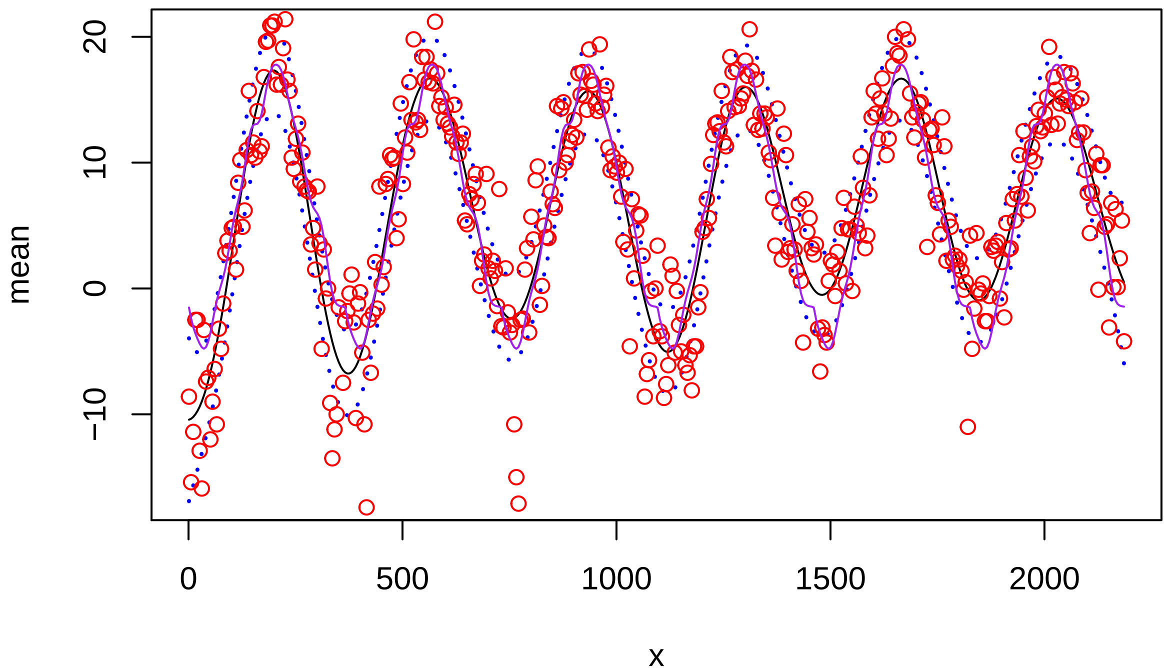
(2)

```
interval = seq(1,365*6,1)
KernelFunct = KernelFunc(sigmaF = 20, ell = 0.2)
pred.GP = posteriorGP(X = scale(time), y=scale(tempTime), XStar = scale(time),sigmaNoise = sigma.n, k=Ke

stdTemp = sqrt(var(tempTime))

meanTemp = mean(tempTime)
meanPred2 = pred.GP$mean*stdTemp+meanTemp ##Scale back the mean

plotGP(time, meanPred2, pred.GP$var, obs=data.frame(x=time, y=tempTime))
```



(3)

```
fit.GP.daily = gausspr(day, tempTime, kernel=KernelFunc, kpar=list(sigmaF = sigma.f, ell = l), var=sigma
meanPred.daily = predict(fit.GP.daily, day)

plotGP(time, meanPred, var= pred.GP$var, obs=data.frame(x=time, y=tempTime))
lines(time,meanPred.daily, col="purple" )
```

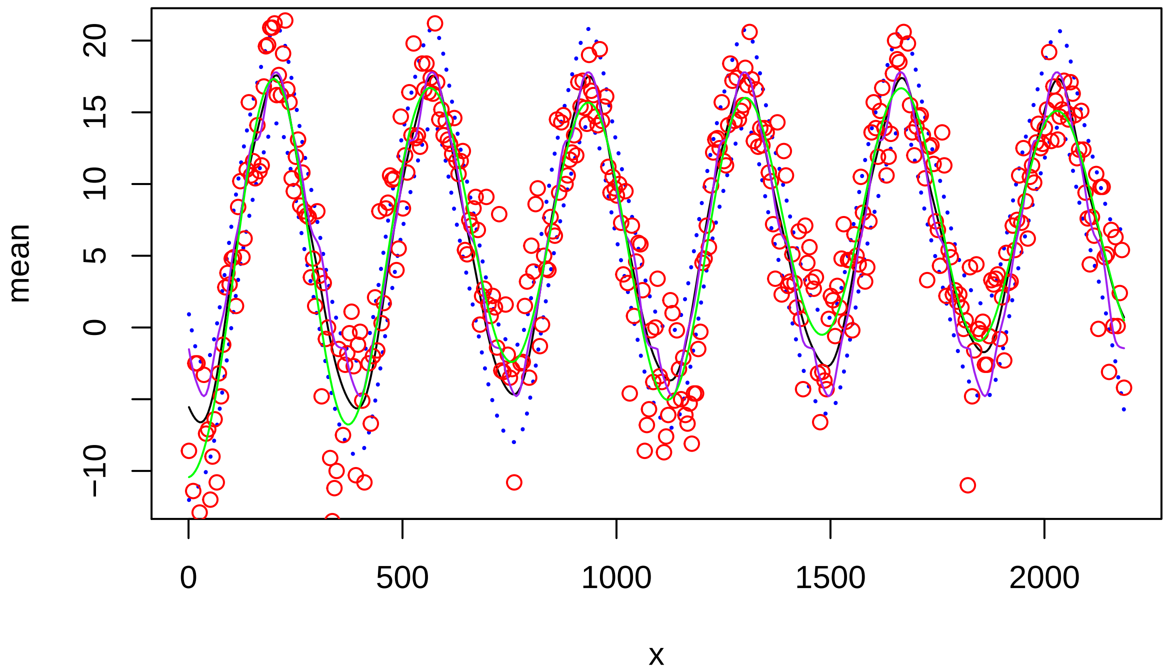


(4)

Since we count every day as equal we get 6 observations per day instead of 1 observation per "time", this allows us to create a more general modeling of the temperature at different timepoints of the year meanwhile it might be a bit more off-target when it comes to modeling outlying temperature-waves a certain year.

```
generalPeriodicKernel = function(sigma.f, l.1, l.2, d){
  rval <- function(x, y = NULL) {
    diff = abs(x-y)
    kern = (sigma.f^2)*exp(-2*(sin(pi*diff/d)^2)/(l.1^2))*exp(-(1/2)*(diff/l.2)^2)
    return(kern)
  }
  class(rval) <- "kernel"
  return(rval)
}

sigma.f = 20
l.1 = 1
l.2 = 20
d = 365/sd(time)

fit.GP = gausspr(x=time,y=tempTime, kernel = generalPeriodicKernel,
                 kpar=list(sigma.f = sigma.f, l.1 = l.1, l.2 = l.2, d= d),
                 var=sigma.n^2)

meanGP.pred = predict(fit.GP, time)
```

```r
plotGP(time,meanGP.pred, obs=data.frame(x=time,y=tempTime), var = pred.GP$var)
lines(time, meanPred.daily, col="purple")
lines(time,meanPred, col="green")
```



(5)

The periodic version seems like a balanced kernel with results somewhere between the days / time versions.

GP Classification with kernlab

```r
library(AtmRay)
```

```
## Warning: package 'AtmRay' was built under R version 4.0.2
```

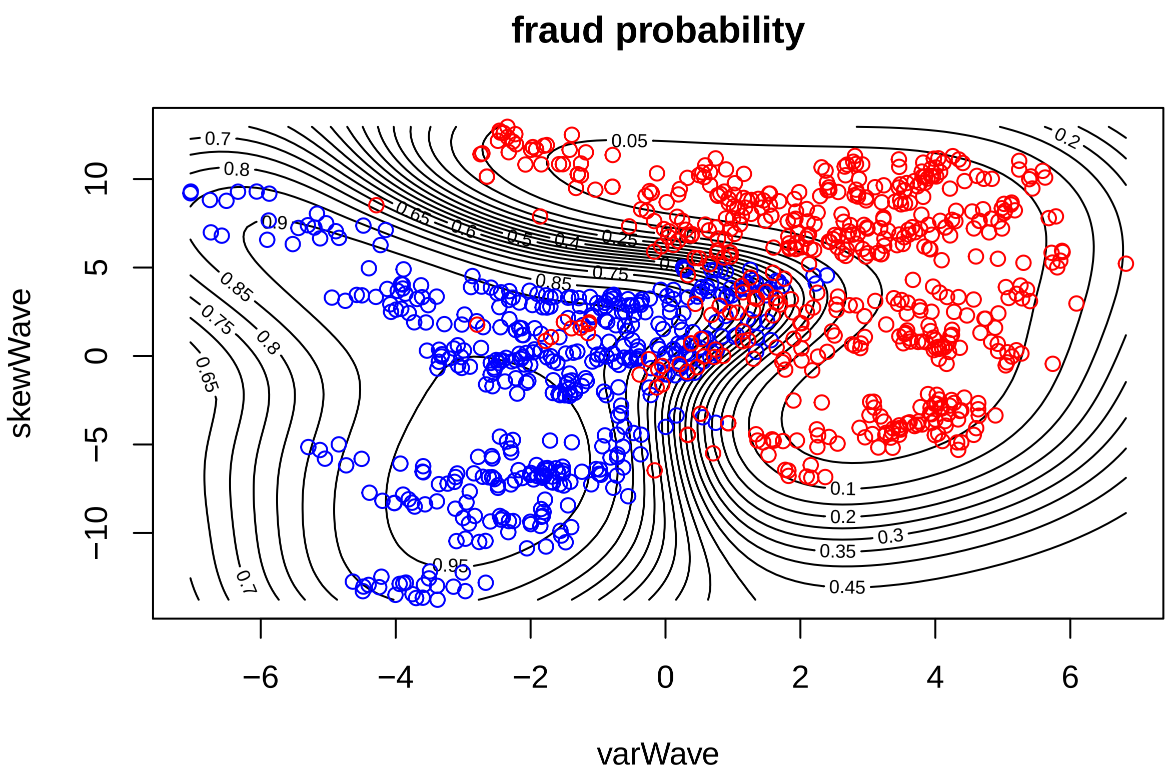
```r
data <- read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/banknoteFraud
names(data) <- c("varWave","skewWave","kurtWave","entropyWave","fraud")
data[,5] <- as.factor(data[,5])

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

```r
fraud.GP.fit = gausspr(fraud~varWave+skewWave, data=trainingData)
```

(1)

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

```
x1 <- seq(min(trainingData$varWave),max(trainingData$varWave),length=100)
x2 <- seq(min(trainingData$skewWave),max(trainingData$skewWave),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- c("varWave", "skewWave")


GP.gridPred = predict( fraud.GP.fit, gridPoints, type="probabilities")

frauds = which(trainingData$fraud == 1)

contour(x = x1, y= x2, z = matrix(GP.gridPred[,2], 100, byrow=TRUE), 20,
        xlab = "varWave", ylab="skewWave", main= 'fraud probability')

points( x= trainingData$varWave[frauds], y=trainingData$skewWave[frauds], col="blue")
points(x=trainingData$varWave[-frauds], y=trainingData$skewWave[-frauds], col="red")
```




```
training.GP.pred = predict(fraud.GP.fit, trainingData)

confusion_train = table(training.GP.pred, trainingData$fraud)
confusion_train
```

```
##
## training.GP.pred   0   1
```

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

```r
accuracy = sum(diag(confusion_train))/sum(confusion_train)
accuracy
```

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

```r
test.GP.pred = predict( fraud.GP.fit, testData)

confusion_test = table( test.GP.pred, testData$fraud)
confusion_test
```

(2)

```
##
## test.GP.pred   0   1
##            0 199   9
##            1  19 145
```

```r
accuracy_test = sum(diag(confusion_test))/sum(confusion_test)
accuracy_test
```

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

```r
allVariables.GP.fit = gausspr( fraud ~ . , data = trainingData)
```

(3)

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

```r
allVariables.GP.pred = predict(allVariables.GP.fit, testData)

confusion_test_all = table(allVariables.GP.pred, testData$fraud)
confusion_test_all
```

```
##
## allVariables.GP.pred   0   1
##                    0 216   0
##                    1   2 154
```

```r
accuracy_all = sum(diag(confusion_test_all))/sum(confusion_test_all)
accuracy_all
```

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