Lab4_report

Gustav Wahlquist

2020-10-16

TASK 1.1

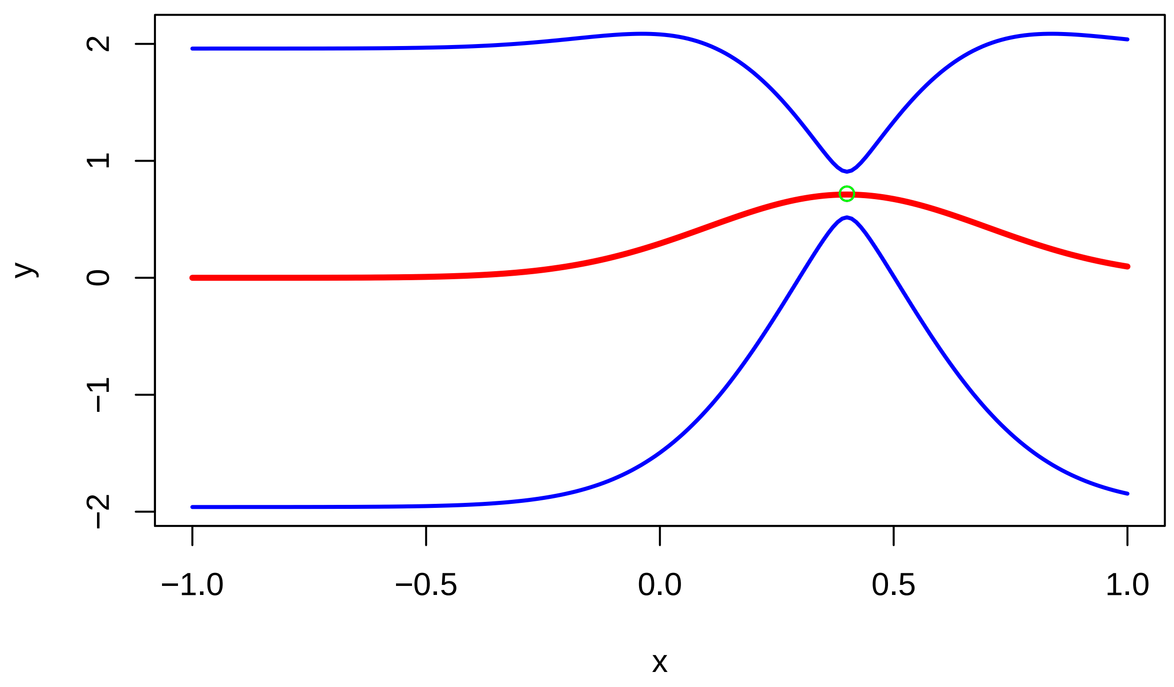
*Write your own code for simulating from the posterior distribution of f using the squared exponential kernel. The function (name it posteriorGP) should return a vector with the posterior mean and variance of f, both evaluated at a set of x-values. You can assume that the prior mean of f is zero for all x. The function should have the following inputs:

- X: Vector of training inputs.
- y: Vector of training targets/outputs.
- XStar: Vector of inputs where the posterior distribution is evaluated.
- sigmaNoise: Noise standard deviation.
- k: Covariance function or kernel. That is, the kernel should be a separate function.*

```r
# Gaussian Regression Model
# INPUTS:
#   X: Vector of training inputs.
#   y: Vector of training targets/outputs.
#   XStar: Vector of inputs where the posterior distribution is evaluated.
#   sigmaNoise: Noise standard deviation.
#   k: Covariance function or kernel. That is, the kernel should be a separate function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, sigmaf, ell){

  # Compute covariance matrix K(X,X)
  covM = k(X, X, sigmaf, ell)

  # Identity matrix
  I = diag(dim(covM)[1])

  # Transposing the output from the cholesky-function since the implementation in R returns
  # an upper triangular matrix and we want a lower triangular matrix.
  L = t(chol(covM + (sigmaNoise**2)*I))

  # Compute covariance matrix K(X,XStar)
  KStar = k(X, XStar, sigmaf, ell)
  # Predictive mean
  alpha = solve(t(L), solve(L, y))
  FStarMean = t(KStar)%*%alpha

  # Compute covariance matrix K(XStar,XStar)
  KStar2 = k(XStar, XStar, sigmaf, ell)
  # Predictive variance
  v = solve(L, KStar)
  V = diag(KStar2 - t(v)%*%v)

  return(list('posteriorMean' = FStarMean, 'posteriorVariance' = V))

}
```

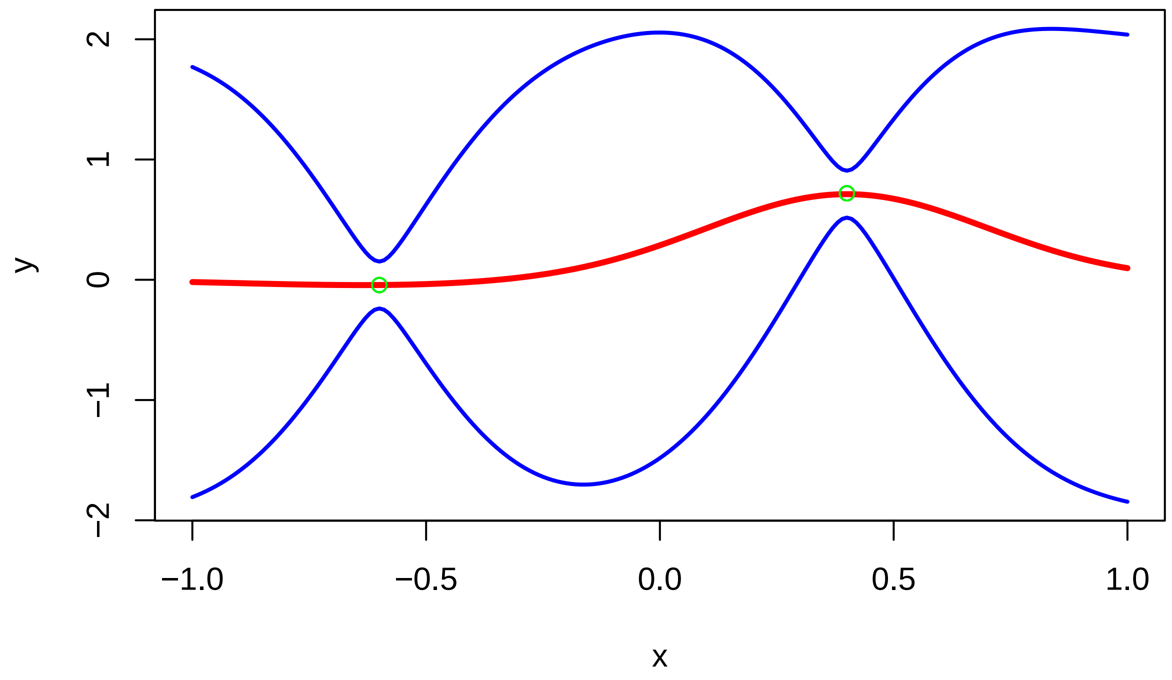
TASK 1.2

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



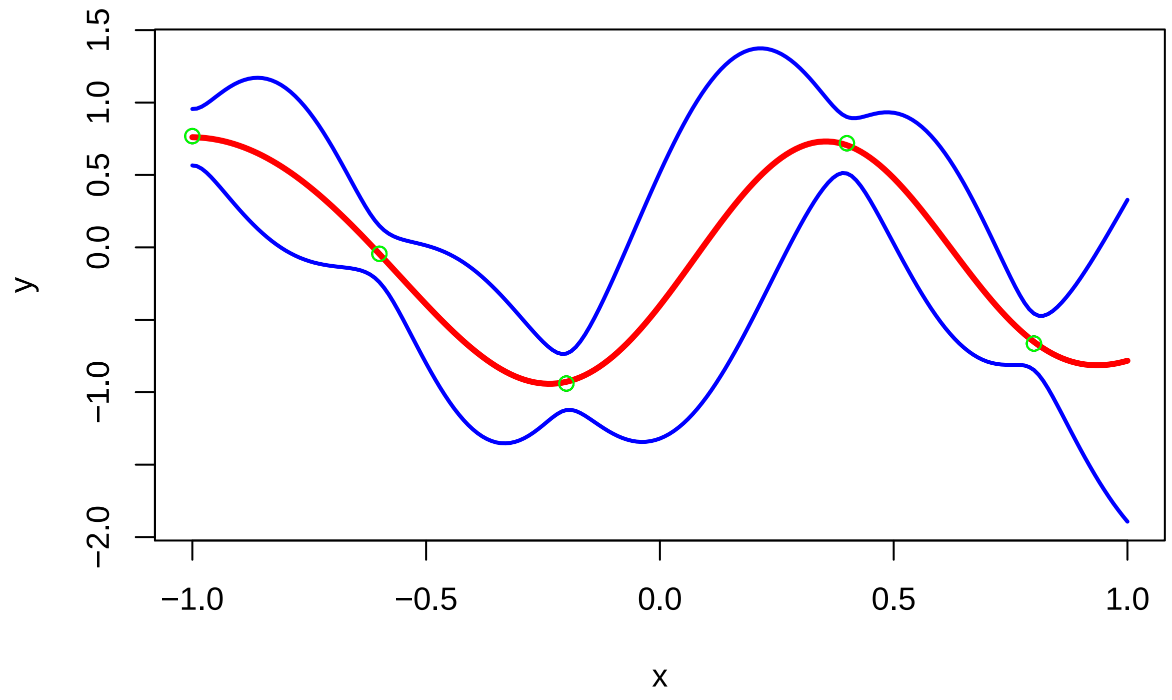
TASK 1.3

Update your posterior from (1.2) with another observation: (x, y) = (-0.6,-0.044). Plot the posterior mean of f over the interval x E [-1, 1]. Plot also 95 % probability (pointwise) bands for f. Hint: Updating the posterior after one observation with a new observation gives the same result as updating the prior directly with the two observations



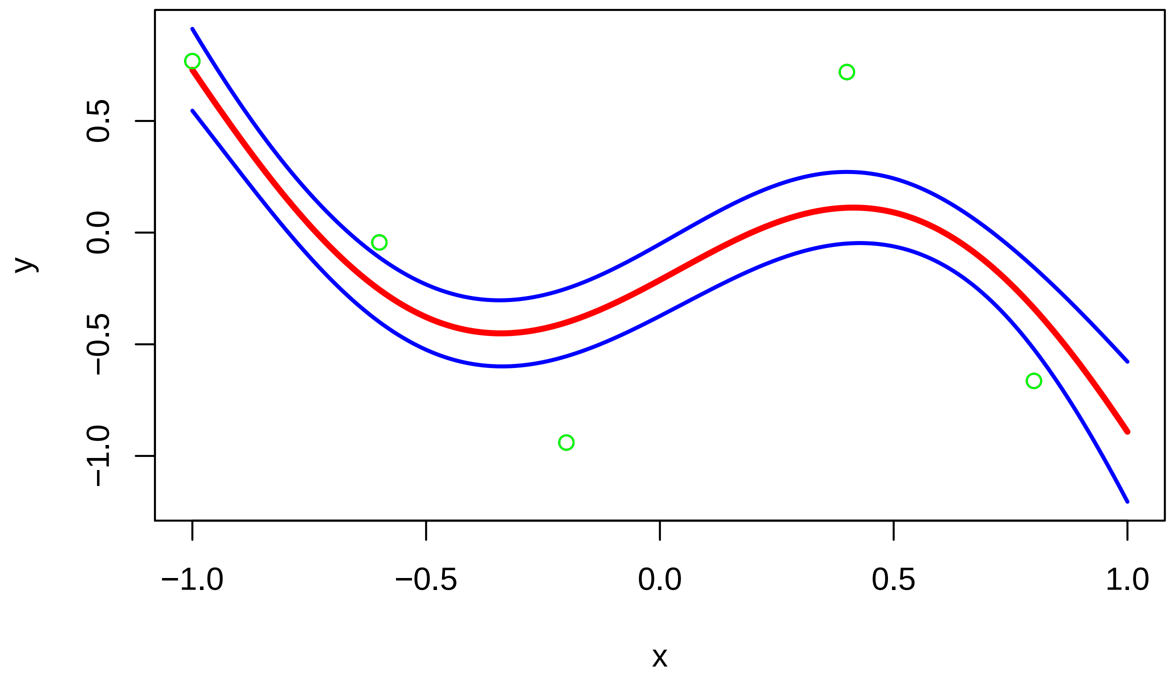
TASK 1.4

Compute the posterior distribution of f using all the five data points in the table below (note that the two previous observations are included in the table). Plot the posterior mean of f over the interval x E [-1, 1]. Plot also 95 % probability (pointwise) bands for f. x -1.0 -0.6 -0.2 0.4 0.8 y 0.768 -0.044 -0.940 0.719 -0.664



TASK 1.5

Repeat (4), this time with hyperparameters sigmaf = 1 and ell = 1. Compare the results.



As expected, we see that this posterior over f is much smoother than the one in the task before. This is since we use a ell = 1 instead of ell = 0.3. The posterior with ell = 0.3 seems to resemble the few data points we have, however, depending on how the real distribution looks like, this model could be overfitted. The posterior with ell = 1 looks like it is underfitted when looking at the data points we have.

TASK 2.1

Create the variable time which records the day number since the start of the dataset (i.e., time= 1, 2, . . ., 365 × 6 = 2190). Also, create the variable day that records the day number since the start of each year (i.e., day= 1, 2, . . ., 365, 1, 2, . . ., 365). Estimating a GP on 2190 observations can take some time on slower computers, so let us subsample the data and use only every fifth observation. This means that your time and day variables are now time= 1, 6, 11, . . ., 2186 and day= 1, 6, 11, . . ., 361, 1, 6, 11, . . ., 361.

Define your own square exponential kernel function (with parameters ell and sigmaf (sigmaf)), evaluate it in the point x = 1, x' = 2, and use the kernelMatrix function to compute the covariance matrix K(X, Xstar) for the input vectors X = T(1, 3, 4) and Xstar = T(2, 3, 4).

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

# Creating variables with index for every fifth reading
tempTime = 1:2190
time = tempTime[seq(1,2190,5)]
dayNr = 1:365
tempDay = c()
```

```r
for(i in 1:6){
  tempDay = c(tempDay, dayNr)
}
day = tempDay[seq(1,2190,5)]
temperature = unlist(data['temp'])
tempSubset = temperature[time]

# Returns a square exponential kernel function
# INPUTS:
#   sigmaf: standard deviation for f
#   ell: Smoothing factor
squareExpKernel <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    #return((sigmaf**2)*exp(-r/2*(ell**2)))
    return(sigmaf^2 * exp(-0.5 * r / ell^2))
  }
  class(rval) <- "kernel"
  return(rval)

}
```

```
## 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
```

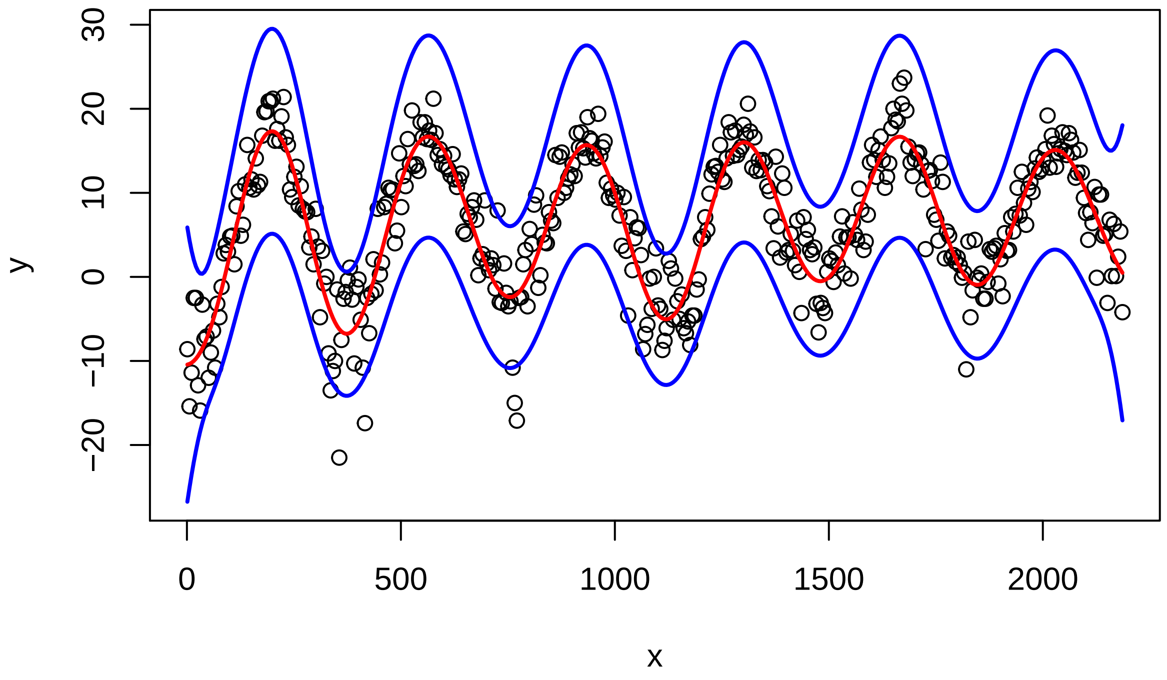
TASK 2.2 & 2.3

Consider first the following model:

$$temp = f(time) + \epsilon \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2) \text{ and } f \sim \mathcal{GP}(0, k(time, time'))$$

Figure 1: Model

Estimate the above Gaussian process regression model using the squared exponential function from (1) with sigmaf = 20 and ell = 0.2. Use the predict function in R to compute the posterior mean at every data point in the training dataset. Make a scatterplot of the data and superimpose the posterior mean of f as a curve (use type="l" in the plot function).



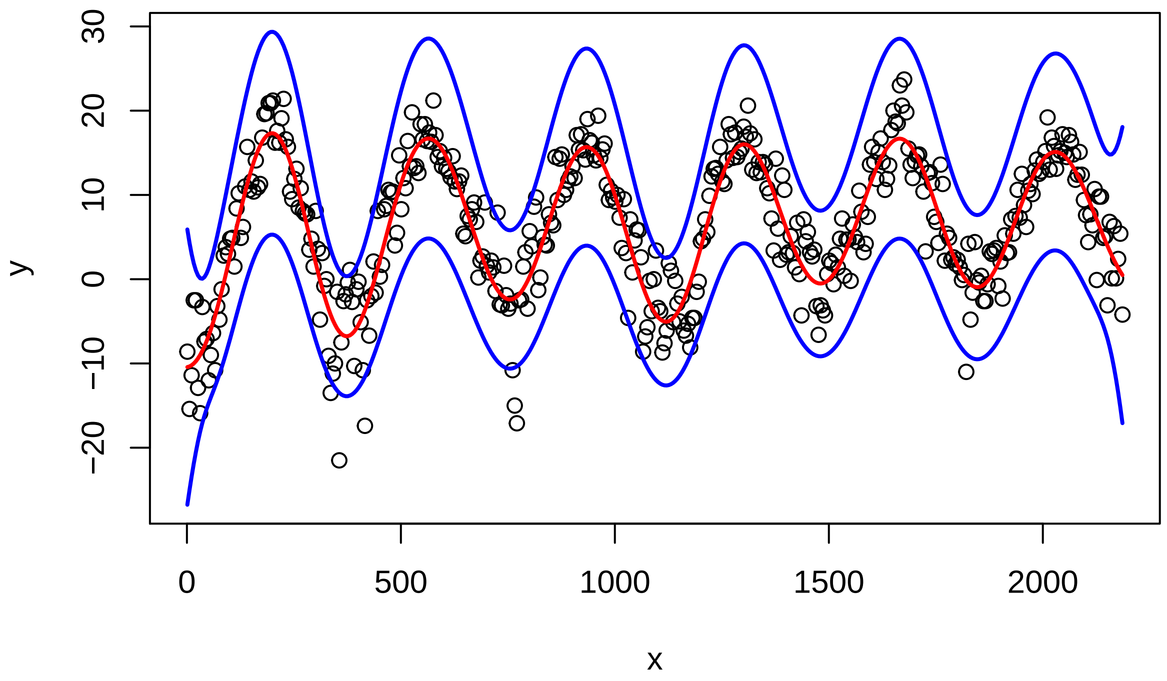
TASK 2.4

Consider first the following model:

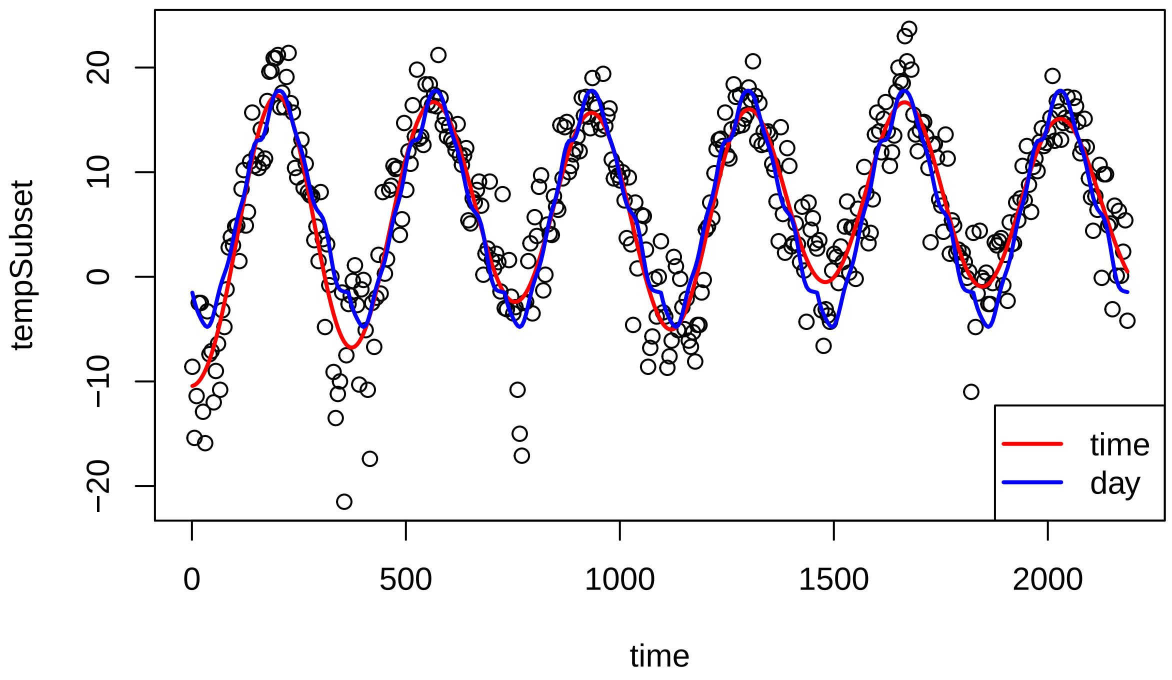
$$temp = f(day) + \epsilon \text{ with } \epsilon \sim \mathcal{N}(0, \sigma_n^2) \text{ and } f \sim \mathcal{GP}(0, k(day, day'))$$

Figure 2: Model

Estimate the model using the squared exponential function with sigmaf = 20 and ell = 0.2. Superimpose the posterior mean from this model on the posterior mean from the model in (2). Note that this plot should also have the time variable on the horizontal axis. Compare the results of both models. What are the pros and cons of each model?



y
30
20
10
0
−10
−20
0
500
1000
1500
2000
x



The pros with the model from (2) is that it takes temperature readings in close time proximity into account. This leads to the model being able to catch time periods that are particularly cold or warm compared to the same time period other years. The model from (4), which is shown by the blue line could however be better when predicting for years in the future if we assume that there is no trend in temperature that covers multiple years (what is global warming even??). If we have this assumption, the red line would wrongly predict cold temperatures for a coming year if the year previously was cold. In conclusion, the model that is best for future predictions depends on what assumptions we have on the data.

The variances for both the models seems to be somewhat similiar.

TASK 2.5

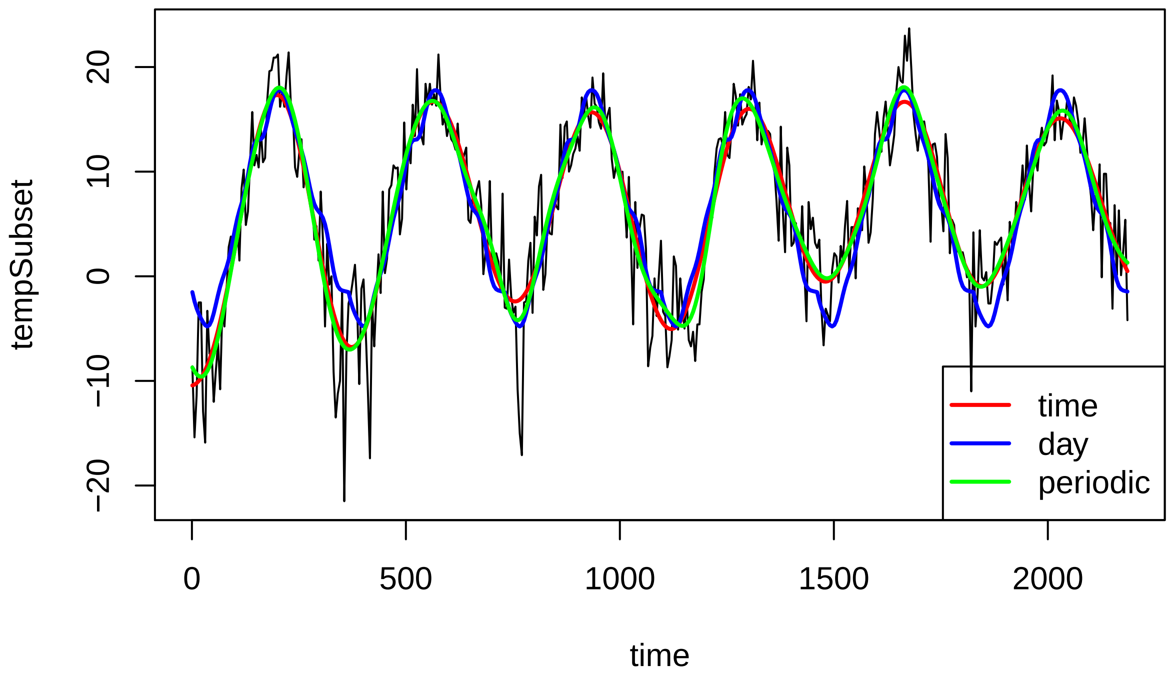
- Finally, implement a generalization of the periodic kernel given in the lectures:*

$$k(x,x') = \sigma_f^2 \exp\left\{-\frac{2\sin^2(\pi|x-x'|/d)}{\ell_1^2}\right\} \exp\left\{-\frac{1}{2}\frac{|x-x'|^2}{\ell_2^2}\right\}$$

Figure 3: Model

Note that we have two different length scales here, and ell2 controls the correlation between the same day in different years. Estimate the GP model using the time variable with this kernel and hyperparameters sigmaf = 20, ell1 = 1, ell2 = 10 and d = 365/sd(time). The reason for the rather strange period here is that kernlab standardizes the inputs to have standard deviation of 1. Compare the fit to the previous two models (with sigmaf = 20 and ell = 0.2). Discuss the results.

```
periodicKernel <- function(sigmaf = 1, ell1 = 1, ell2 = 1, d)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    exp1 = exp(-2*(sin(pi*sqrt(r)/d)**2)/ell1**2)
    exp2 = exp(-0.5*r*(ell2**2))
    return((sigmaf**2)*exp1*exp2)
  }
  class(rval) <- "kernel"
  return(rval)

}
```

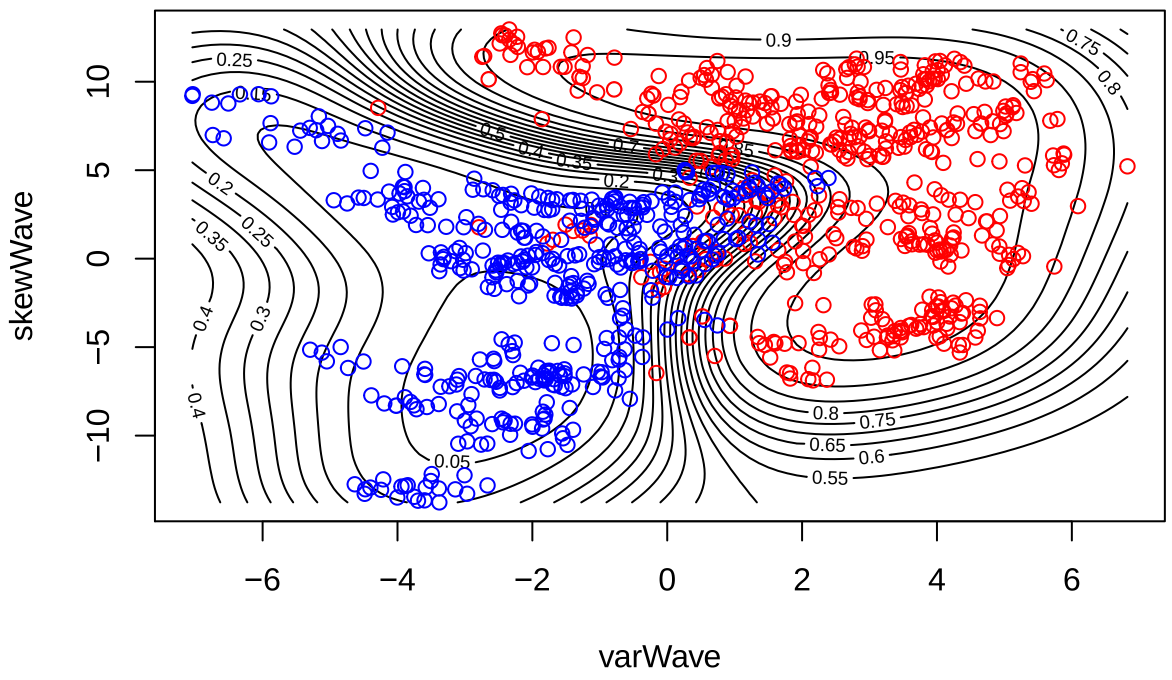


The perodic model, shown by the green line, seems to be fitting the data in the best way. One reason for this could be that this model, unlike the model in (4) and (2), takes into account that the data should be periodic but stills looks at readings in close time proximity.

TASK 3.1

Use the R package kernlab to fit a Gaussian process classification model for fraud on the training data. Use the default kernel and hyperparameters. Start using only the covariates varWave and skewWave in the model. Plot contours of the prediction probabilities over a suitable grid of values for varWave and skewWave. Overlay the training data for fraud = 1 (as blue points) and fraud = 0 (as red points). You can reuse code from the file KernLabDemo.R available on the course website. Compute the confusion matrix for the classifier and its accuracy.

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



```
##
##       0   1
##   0 503  17
##   1  41 439
```

```
## [1] 0.942
```

TASK 3.2

- Using the estimated model from (1), make predictions for the test set. Compute the accuracy.*

```
##
## prediction   0   1
##          0 503  18
##          1  40 439
```

```
## [1] 0.942
```

The resulting accuracy on the test set is the same as on the training set, however we can see that the confusion matrix has changed.

TASK 3.3

Train a model using all four covariates. Make predictions on the test set and compare the accuracy to the model with only two covariates.

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

```
##
## prediction   0   1
##          0 541   0
##          1   2 457
```

```
## [1] 0.998
```

Here we see that we get almost a hundred percent accuarcy when using all four covariates to make the predictions.

Appendix for code

```r
## File Lab4_implement_GPR ##
library(kernlab)

# Code from earlier part of lab
library("mvtnorm")

# 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)
}

# Gaussian Regression Model
# INPUTS:
#   X: Vector of training inputs.
#   y: Vector of training targets/outputs.
#   XStar: Vector of inputs where the posterior distribution is evaluated.
#   sigmaNoise: Noise standard deviation.
#   k: Covariance function or kernel. That is, the kernel should be a separate function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, sigmaf, ell){

  # Compute covariance matrix K(X,X)
  covM = k(X, X, sigmaf, ell)

  # Identity matrix
  I = diag(dim(covM)[1])

  # Transposing the output from the cholesky-function since the implementation in R returns
  # an upper triangular matrix and we want a lower triangular matrix.
  L = t(chol(covM + (sigmaNoise**2)*I))

  # Compute covariance matrix K(X,XStar)
  KStar = k(X, XStar, sigmaf, ell)
  # Predictive mean
  alpha = solve(t(L), solve(L, y))
  FStarMean = t(KStar)%*%alpha

  # Compute covariance matrix K(XStar,XStar)
  KStar2 = k(XStar, XStar, sigmaf, ell)
  # Predictive variance
  v = solve(L, KStar)
  V = diag(KStar2 - t(v)%*%v)

  # Compute log marginal likelihood
  #n = length(y)
  #logSum = 0
  #for(i in (1:n)) {
  #  logSum = logSum + log(L[i,i])
  #}

  # logProbYX = 0.5*t(y)*alpha - logSum - n/2*log(2*pi) # Not needed in this lab? Is a part of algorith
  # Rasmussen and Williams' book
  logProbYX = 1

  return(list('posteriorMean' = FStarMean, 'posteriorVariance' = V, 'logProb' = logProbYX))

}

# Function for plotting posterior distributions with 95% probability band.
# INPUTS:
#   means: Computed means for a number of points
#   variances: Computed variances for a number of points
#   x: x-values for observations
#   y: y-values for observations
#   XStar: Vector of inputs where the posterior distribution is evaluated.
plotPosterior <- function(means, variances, x, y, XStar){
  var = sd(means) * variances + means
  upperBand = means + sqrt(var)*1.96
  lowerBand = means - sqrt(var)*1.96

  plot(x, y, type="p",
       xlim = c(min(x), max(x)),
       ylim = c((min(min(lowerBand), min(y))), max(max(upperBand), max(y))))
  lines(XStar, means, col = "red", lwd = 2)
  lines(XStar, upperBand, col = "blue", lwd = 2)
  lines(XStar, lowerBand, col = "blue", lwd = 2)

}

# New code

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

# Creating variables with index for every fifth reading
tempTime = 1:2190
time = tempTime[seq(1,2190,5)]
dayNr = 1:365
tempDay = c()
for(i in 1:6){
  tempDay = c(tempDay, dayNr)
}
day = tempDay[seq(1,2190,5)]
temperature = unlist(data['temp'])
tempSubset = temperature[time]

# TASK 2.2.1

# Returns a square exponential kernel function
```

```r
# INPUTS:
#   sigmaf: standard deviation for f
#   ell: Smoothing factor
squareExpKernel <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    #return((sigmaf**2)*exp(-r/2*(ell**2)))
    return(sigmaf^2 * exp(-0.5 * r / ell^2))
  }
  class(rval) <- "kernel"
  return(rval)

}

# Creating kernel with sigmaf = 1, ell = 1
kernel = squareExpKernel(1,1)
# Evaluating in point x = 1, x' = 2
covM = kernel(1,2)

# Computing kernel matrix for created kernel and vectors X, XStar
X = c(1, 3, 4)
XStar = c(2, 3 ,4)
kernelMatrix = kernelMatrix(kernel = kernel, X, XStar)

# TASK 2.2.2 & 2.2.3

#
lmFit = lm(tempSubset ~ time + time**2) # vad är detta exakt?
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
sigmaf = 20
ell = 0.2
kernel = squareExpKernel(sigmaf = sigmaf, ell = ell)

#
fitGP = gausspr(x = time,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredict = predict(fitGP, time)

#
posterior = posteriorGP(X = scale(time),
                        y = scale(tempSubset),
                        XStar = scale(time),
                        sigmaNoise = noise,
                        k = SquaredExpKernel,
                        sigmaf = sigmaf,
                        ell = ell)

# Plotting results
```

```r
plotPosterior(means = meanPredict,
              variances = posterior$posteriorVariance,
              x = time,
              y = tempSubset,
              XStar = time)

# TASK 2.2.4

#
lmFit = lm(tempSubset ~ day + day**2)
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
sigmaf = 20
ell = 0.2
kernel = squareExpKernel(sigmaf = sigmaf, ell = ell)

#
fitGP = gausspr(x = day,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredictDay = predict(fitGP, day)

#
posteriorDay = posteriorGP(X = scale(day),
                        y = scale(tempSubset),
                        XStar = scale(time),
                        sigmaNoise = noise,
                        k = SquaredExpKernel,
                        sigmaf = sigmaf,
                        ell = ell)

# Plotting results
plotPosterior(means = meanPredict,
              variances = posteriorDay$posteriorVariance,
              x = time,
              y = tempSubset,
              XStar = time)

plot(time, tempSubset, type = "l")
lines(time, meanPredict, col = "red", lwd = 2)
lines(time, meanPredictDay, col = "blue", lwd = 2)

# TASK 2.2.5

periodicKernel <- function(sigmaf = 1, ell1 = 1, ell2 = 1, d)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    exp1 = exp(-2*(sin(pi*sqrt(r)/d)**2)/ell1**2)
    exp2 = exp(-0.5*r*(ell2**2))
    return((sigmaf**2)*exp1*exp2)
  }
  class(rval) <- "kernel"
  return(rval)

}

lmFit = lm(tempSubset ~ time + time**2)
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
sigmaf = 20
ell1 = 1
ell2 = 10
d = sd(time)
kernel = periodicKernel(sigmaf = sigmaf, ell1 = ell1, ell2 = ell2, d)

#
fitGP = gausspr(x = time,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredictPer = predict(fitGP, time)

#
#posteriorPer = posteriorGP1(X = scale(time),
#                            y = scale(tempSubset),
#                            XStar = scale(time),
#                            sigmaNoise = noise,
#                            k = periodicKernel,
#                            sigmaf = sigmaf,
#                            ell1 = ell1,
#                            ell2 = ell2)

# Plotting results
#plotPosterior(means = meanPredictPer,
#              variances = posteriorPer$posteriorVariance,
#              x = time,
#              y = tempSubset,
#              XStar = time)

plot(time, tempSubset, type = "l")
lines(time, meanPredict, col = "red", lwd = 2)
lines(time, meanPredictDay, col = "blue", lwd = 2)
lines(time, meanPredictPer, col = "green", lwd = 2)

## File Lab4_reg_with_kernlab ##

library(kernlab)

# Code from earlier part of lab
library("mvtnorm")

# Covariance function
```

```r
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)
}

# Gaussian Regression Model
# INPUTS:
#   X: Vector of training inputs.
#   y: Vector of training targets/outputs.
#   XStar: Vector of inputs where the posterior distribution is evaluated.
#   sigmaNoise: Noise standard deviation.
#   k: Covariance function or kernel. That is, the kernel should be a separate function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, sigmaf, ell){

  # Compute covariance matrix K(X,X)
  covM = k(X, X, sigmaf, ell)

  # Identity matrix
  I = diag(dim(covM)[1])

  # Transposing the output from the cholesky-function since the implementation in R returns
  # an upper triangular matrix and we want a lower triangular matrix.
  L = t(chol(covM + (sigmaNoise**2)*I))

  # Compute covariance matrix K(X,XStar)
  KStar = k(X, XStar, sigmaf, ell)
  # Predictive mean
  alpha = solve(t(L), solve(L, y))
  FStarMean = t(KStar)%*%alpha

  # Compute covariance matrix K(XStar,XStar)
  KStar2 = k(XStar, XStar, sigmaf, ell)
  # Predictive variance
  v = solve(L, KStar)
  V = diag(KStar2 - t(v)%*%v)

  # Compute log marginal likelihood
  #n = length(y)
  #logSum = 0
  #for(i in (1:n)) {
  #  logSum = logSum + log(L[i,i])
  #}

  # logProbYX = 0.5*t(y)*alpha - logSum - n/2*log(2*pi) # Not needed in this lab? Is a part of algorith
  # Rasmussen and Williams' book
  logProbYX = 1

  return(list('posteriorMean' = FStarMean, 'posteriorVariance' = V, 'logProb' = logProbYX))
```

```
}

# Function for plotting posterior distributions with 95% probability band.
# INPUTS:
#   means: Computed means for a number of points
#   variances: Computed variances for a number of points
#   x: x-values for observations
#   y: y-values for observations
#   XStar: Vector of inputs where the posterior distribution is evaluated.
plotPosterior <- function(means, variances, x, y, XStar){
  var = sd(means) * variances + means
  upperBand = means + sqrt(var)*1.96
  lowerBand = means - sqrt(var)*1.96

  plot(x, y, type="p",
       xlim = c(min(x), max(x)),
       ylim = c((min(min(lowerBand), min(y))), max(max(upperBand), max(y))))
  lines(XStar, means, col = "red", lwd = 2)
  lines(XStar, upperBand, col = "blue", lwd = 2)
  lines(XStar, lowerBand, col = "blue", lwd = 2)

}

# New code

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

# Creating variables with index for every fifth reading
tempTime = 1:2190
time = tempTime[seq(1,2190,5)]
dayNr = 1:365
tempDay = c()
for(i in 1:6){
  tempDay = c(tempDay, dayNr)
}
day = tempDay[seq(1,2190,5)]
temperature = unlist(data['temp'])
tempSubset = temperature[time]

# TASK 2.2.1

# Returns a square exponential kernel function
# INPUTS:
#   sigmaf: standard deviation for f
#   ell: Smoothing factor
squareExpKernel <- function(sigmaf = 1, ell = 1)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    #return((sigmaf**2)*exp(-r/2*(ell**2)))
    return(sigmaf^2 * exp(-0.5 * r / ell^2))
  }
```

```r
  class(rval) <- "kernel"
  return(rval)

}

# Creating kernel with sigmaf = 1, ell = 1
kernel = squareExpKernel(1,1)
# Evaluating in point x = 1, x' = 2
covM = kernel(1,2)

# Computing kernel matrix for created kernel and vectors X, XStar
X = c(1, 3, 4)
XStar = c(2, 3 ,4)
kernelMatrix = kernelMatrix(kernel = kernel, X, XStar)

# TASK 2.2.2 & 2.2.3

#
lmFit = lm(tempSubset ~ time + time**2) # vad är detta exakt?
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
sigmaf = 20
ell = 0.2
kernel = squareExpKernel(sigmaf = sigmaf, ell = ell)

#
fitGP = gausspr(x = time,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredict = predict(fitGP, time)

#
posterior = posteriorGP(X = scale(time),
                        y = scale(tempSubset),
                        XStar = scale(time),
                        sigmaNoise = noise,
                        k = SquaredExpKernel,
                        sigmaf = sigmaf,
                        ell = ell)

# Plotting results
plotPosterior(means = meanPredict,
              variances = posterior$posteriorVariance,
              x = time,
              y = tempSubset,
              XStar = time)

# TASK 2.2.4

#
lmFit = lm(tempSubset ~ day + day**2)
```

```r
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
sigmaf = 20
ell = 0.2
kernel = squareExpKernel(sigmaf = sigmaf, ell = ell)

#
fitGP = gausspr(x = day,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredictDay = predict(fitGP, day)

#
posteriorDay = posteriorGP(X = scale(day),
                        y = scale(tempSubset),
                        XStar = scale(time),
                        sigmaNoise = noise,
                        k = SquaredExpKernel,
                        sigmaf = sigmaf,
                        ell = ell)

# Plotting results
plotPosterior(means = meanPredict,
              variances = posteriorDay$posteriorVariance,
              x = time,
              y = tempSubset,
              XStar = time)

plot(time, tempSubset, type = "l")
lines(time, meanPredict, col = "red", lwd = 2)
lines(time, meanPredictDay, col = "blue", lwd = 2)

# TASK 2.2.5

periodicKernel <- function(sigmaf = 1, ell1 = 1, ell2 = 1, d)
{
  rval <- function(x, y = NULL) {
    r = crossprod(x-y);
    exp1 = exp(-2*(sin(pi*sqrt(r)/d)**2)/ell1**2)
    exp2 = exp(-0.5*r*(ell2**2))
    return((sigmaf**2)*exp1*exp2)
  }
  class(rval) <- "kernel"
  return(rval)

}

lmFit = lm(tempSubset ~ time + time**2) # vad är detta exakt?
noise = sd(lmFit$residuals)

# Creating kernel with sigmaf = 20, ell = 0.2
```

```r
sigmaf = 20
ell1 = 1
ell2 = 10
d = sd(time)
kernel = periodicKernel(sigmaf = sigmaf, ell1 = ell1, ell2 = ell2, d)

#
fitGP = gausspr(x = time,
                y = tempSubset,
                kernel = kernel,
                var = noise**2)
meanPredictPer = predict(fitGP, time)

#
#posteriorPer = posteriorGP1(X = scale(time),
#                            y = scale(tempSubset),
#                            XStar = scale(time),
#                            sigmaNoise = noise,
#                            k = periodicKernel,
#                            sigmaf = sigmaf,
#                            ell1 = ell1,
#                            ell2 = ell2)

# Plotting results
#plotPosterior(means = meanPredictPer,
#              variances = posteriorPer$posteriorVariance,
#              x = time,
#              y = tempSubset,
#              XStar = time)

plot(time, tempSubset, type = "l")
lines(time, meanPredict, col = "red", lwd = 2)
lines(time, meanPredictDay, col = "blue", lwd = 2)
lines(time, meanPredictPer, col = "green", lwd = 2)
```