Lab4

Oscar Hoffmann

2024-10-08

2.1 Implement GP Regression

Libraries

```
library(kernlab)
library(AtmRay)
rm(list =ls())
```

Task 1: Write Function for Posterior GP Simulation Implement a function named `posteriorGP` to simulate from the posterior distribution using the squared exponential kernel.

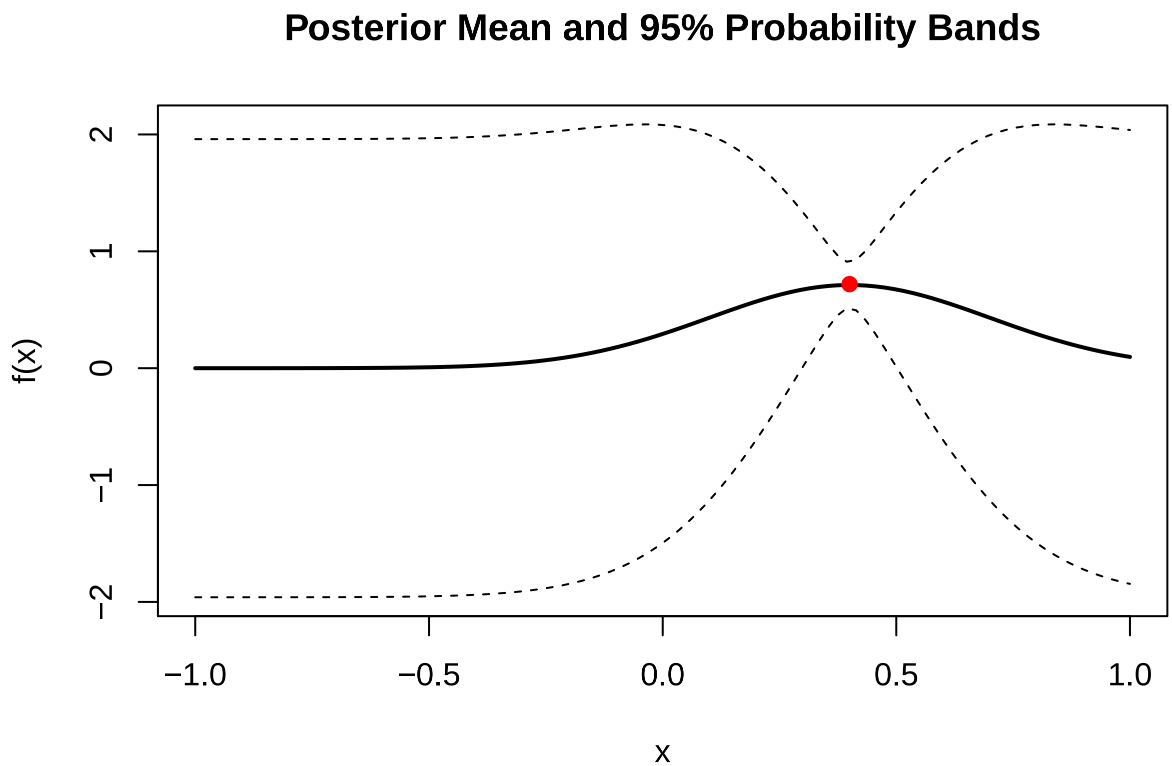
```
# Squared Exponential Kernel Function
SquaredExpKernel <- function(x1,x2,sigmaF=1,ell=3){
  n1 <- length(x1)
  n2 <- length(x2)
  K <- matrix(NA,n1,n2)
  for (i in 1:n2){
    K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/ell)^2 )
  }
  return(K)
}

# Posterior GP Function
posteriorGP <- function(X, y, XStar, sigmaNoise, k, ...) {

  K <- k(X, X, ...)  # Compute the covariance matrix
  kStar <- k(X, XStar, ...) # Compute covariance

  # Step 2 in algo
  #--------------------
  K_y <- K + sigmaNoise^2 * diag(length(X)) # Add noise variance to diagonal
  L <- t(chol(K_y))   # Compute Cholesky decomposition, to get lower triangular L we take t()
  alpha <- solve(t(L), solve(L, y))   # Solve for alpha
  #---------------------

  # Step 4 in algo
  #--------------------
  fStar_mean <- t(kStar) %*% alpha   # Compute posterior mean
  v <- solve(L, kStar)   # Compute v = solve(L, kStar)
  #-------------------
```

```r
  # Step 6 in algo
  #-------------------
  V_fStar <- k(XStar, XStar, ...) - t(v) %*% v # pred variance (cov matrix)
  #-------------------

  # Return posterior mean and variance
  return(list(mean = fStar_mean, variance = V_fStar))
}
```

Task 2: Update Posterior with Single Observation Let prior hyperparameters be sigmaf = 1 and l = 0.3, update with (x, y) = (0.4, 0.719), and plot the posterior mean with 95% bands.

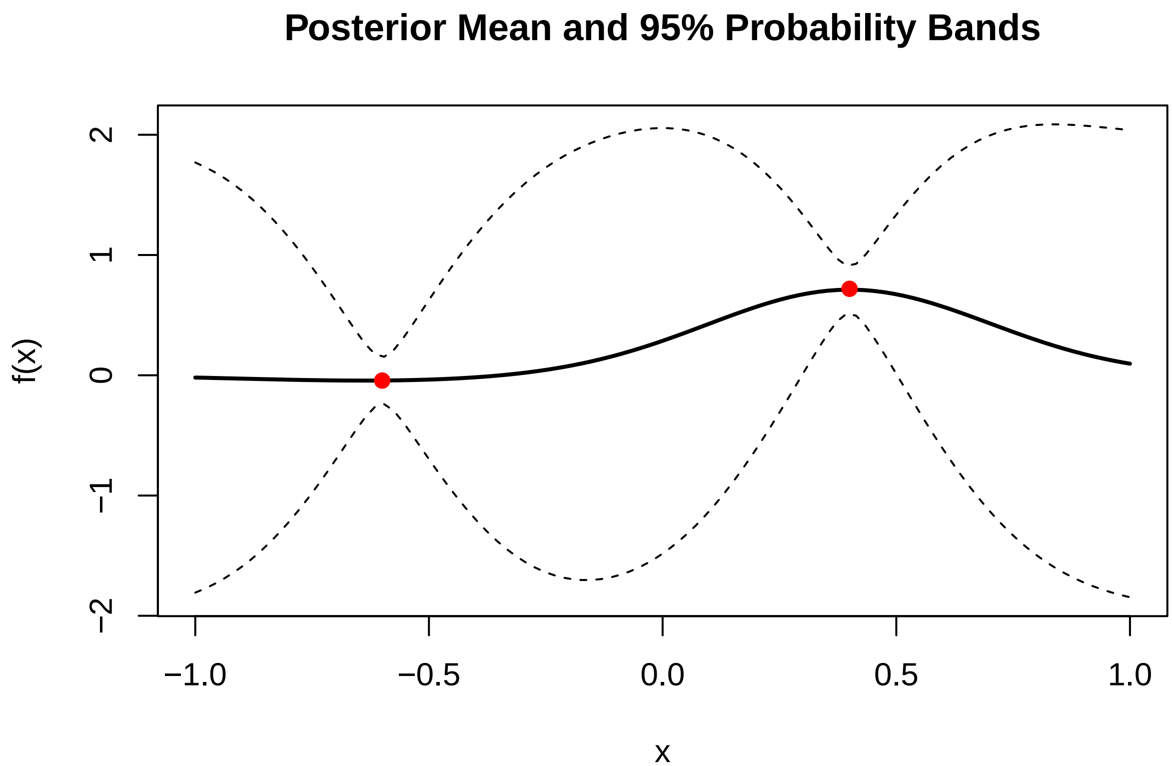
```r
# Plotting Function
plotGP <- function(XStar, res, X_train, y_train, title) {

  # Extract posterior mean and variance
  pos_mean <- res$mean
  pos_var <- diag(res$variance)

  # Compute 95% confidence intervals
  lower_bound <- pos_mean - 1.96 * sqrt(pos_var)
  upper_bound <- pos_mean + 1.96 * sqrt(pos_var)

  # Plot the posterior mean and 95% probability bands
  plot(XStar, pos_mean, type = "l", lwd = 2,
    ylim = range(c(lower_bound, upper_bound, y_train)),
    ylab = "f(x)", xlab = "x", main = title)

  # Add the confidence intervals
  lines(XStar, lower_bound, lty = 2)
  lines(XStar, upper_bound, lty = 2)
  # Plot the training data points
  points(X_train, y_train, pch = 19, col = "red")
}
```

```r
# Single observation
X <- c(0.4)
y <- c(0.719)

# Test inputs over the interval [-1, 1]
XStar <- seq(-1, 1, length.out = 100)

# Hyperparameters
sigmaF <- 1         # sigma_f
ell <- 0.3          # length-scale l
sigmaNoise <- 0.1   # sigma_n

# Call posteriorGP
res <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = sigmaF, ell = ell)

plotGP(XStar, res, X, y, "Posterior Mean and 95% Probability Bands")
```



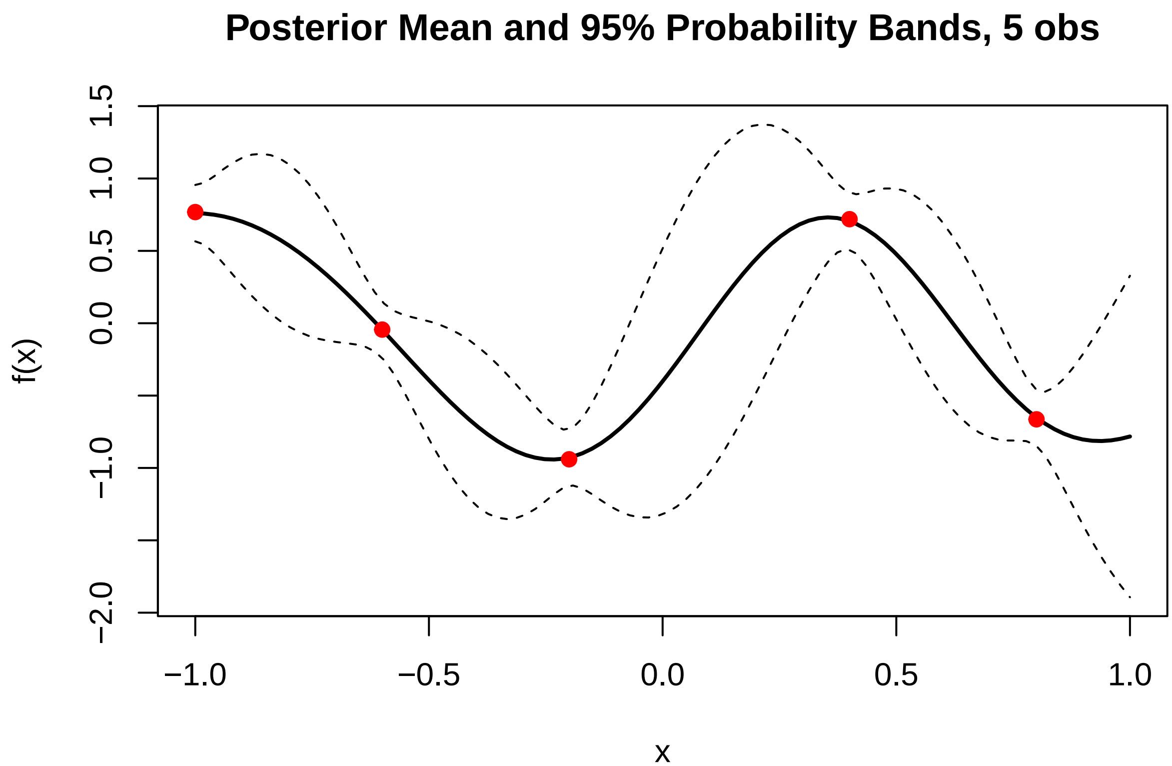
Task 3: Update with Another Observation Update posterior from Task 2 with (x, y) = (-0.6, -0.044), plot the posterior mean, and include 95% probability bands.

```r
# Updated training data
X <- c(0.4, -0.6)
y <- c(0.719, -0.044)

# Same XStar and hyperparameters as in task2
res <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = sigmaF, ell = ell)

plotGP(XStar, res, X, y, "Posterior Mean and 95% Probability Bands")
```



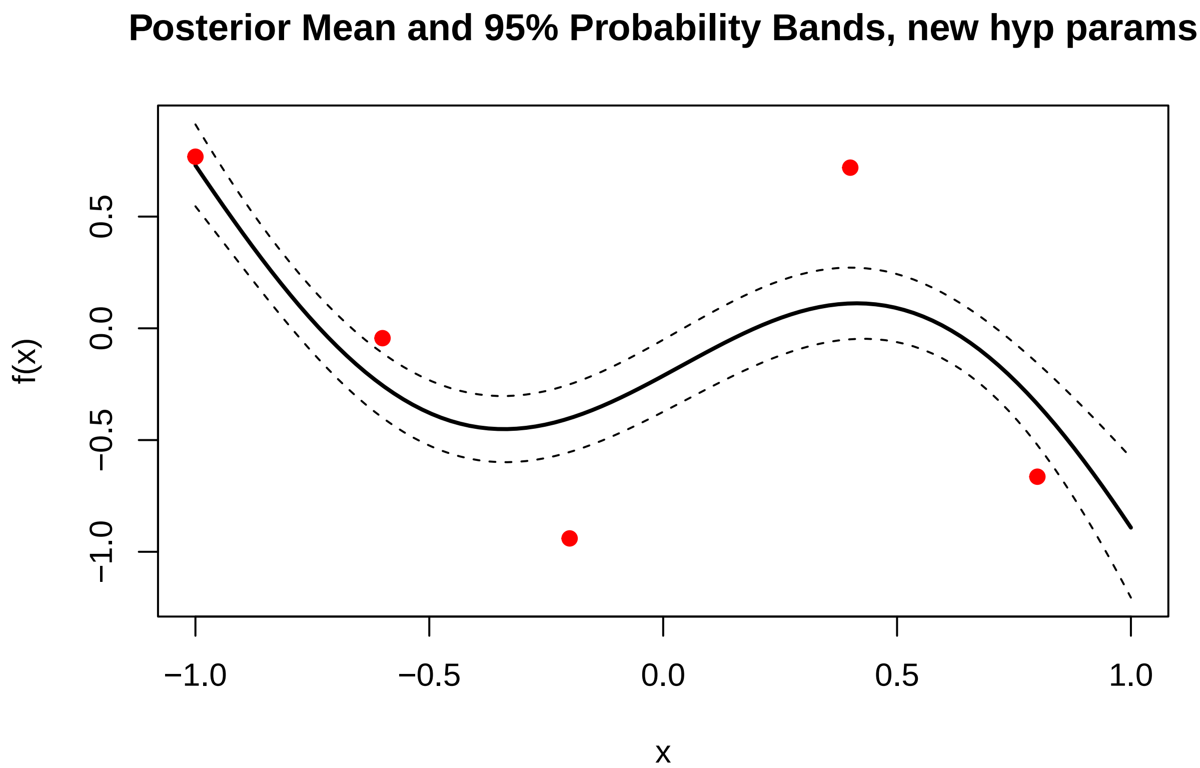
Task 4: Compute Posterior with All Observations Compute the posterior distribution of f using all five data points and plot the posterior mean with 95% bands.

```r
# Step 4: Posterior with all 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)

res <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = sigmaF, ell = ell)
plotGP(XStar, res, X, y, "Posterior Mean and 95% Probability Bands, 5 obs")
```



Task 5: Compare Hyperparameter Settings Repeat Task 4 using different hyperparameters: sigmaf = 1 and l = 1, compare the results.

```
# Hyperparameters
sigmaF <- 1      # sigma_f
ell <- 1         # length-scale l

res <- posteriorGP(X, y, XStar, sigmaNoise, k = SquaredExpKernel, sigmaF = sigmaF, ell = ell)
plotGP(XStar, res, X, y, "Posterior Mean and 95% Probability Bands, new hyp params")
```



When increasing ell from 0.3 to 1 we get more smoothness which is expected.

2.2 GP Regression with kernlab

Task 1: Define Kernel and Compute Covariance Matrix Define own squared exponential kernel and use the `kernelMatrix` function to compute the covariance matrix for given vectors.

```
# Preparing the data
data = read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/TempTullinge.c

time = seq(1,2190, 5)
day = seq(1,365, 5)

data_sampled = data[time,]
temps = data_sampled$temp
```

```
# Squared Exponential Kernel Function
SEKernel = function(ell, sigmaF) {
  calc_K = function (X, XStar) {
    K = matrix(NA, length(X), length(XStar))
    for (i in 1:length(X)) {
      K[, i] = sigmaF ^ 2 * exp(-0.5 * ((X - XStar[i]) / ell) ^ 2)
    }
    return(K)
  }
```

```
  class(calc_K) = 'kernel' # Return as class kernel
  return (calc_K)
}

kernel = SEKernel(1,1)
# eval in points 1,2
kernel(1,2)
```

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

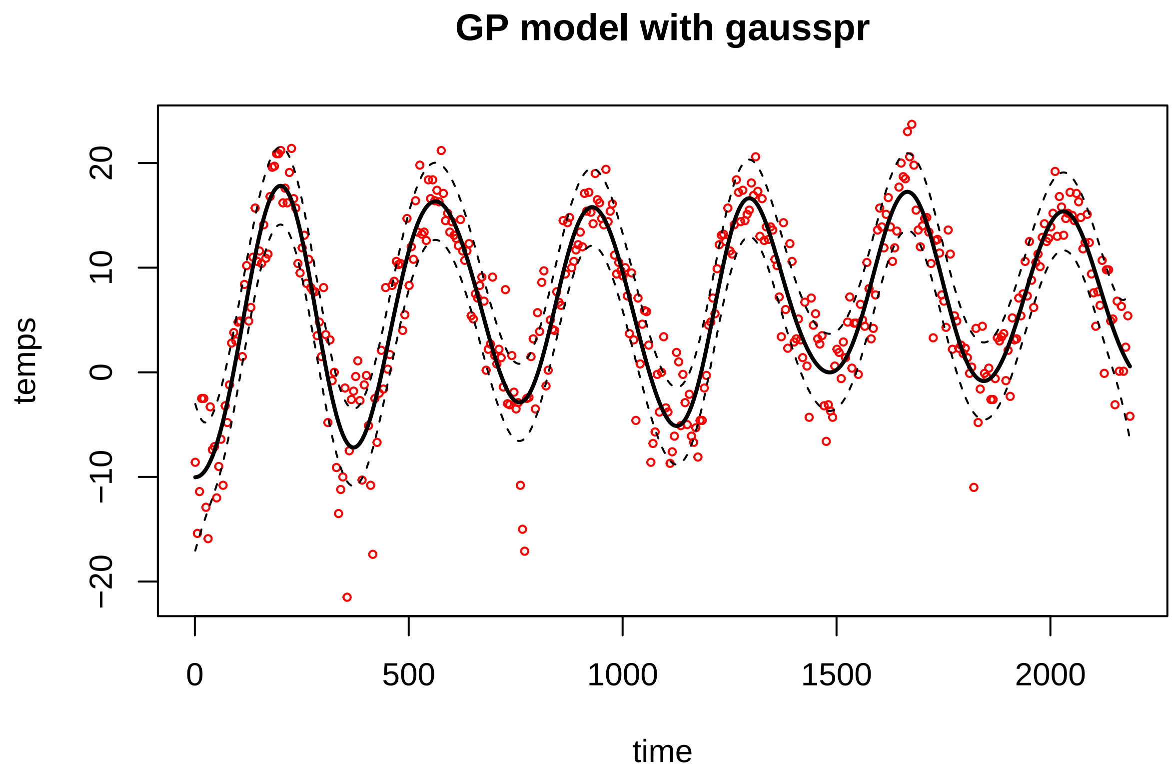
```
X = c(1,3,4)
XStar = c(2,3,4)
# gives matrix for X and XStar
kernelMatrix(kernel, X, XStar) #K(X,XStar)
```

```
## 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: Estimate GP Model with gausspr Use the gausspr function with sigmaf = 20 and l = 100, estimate the Gaussian process regression model, and plot the posterior mean.

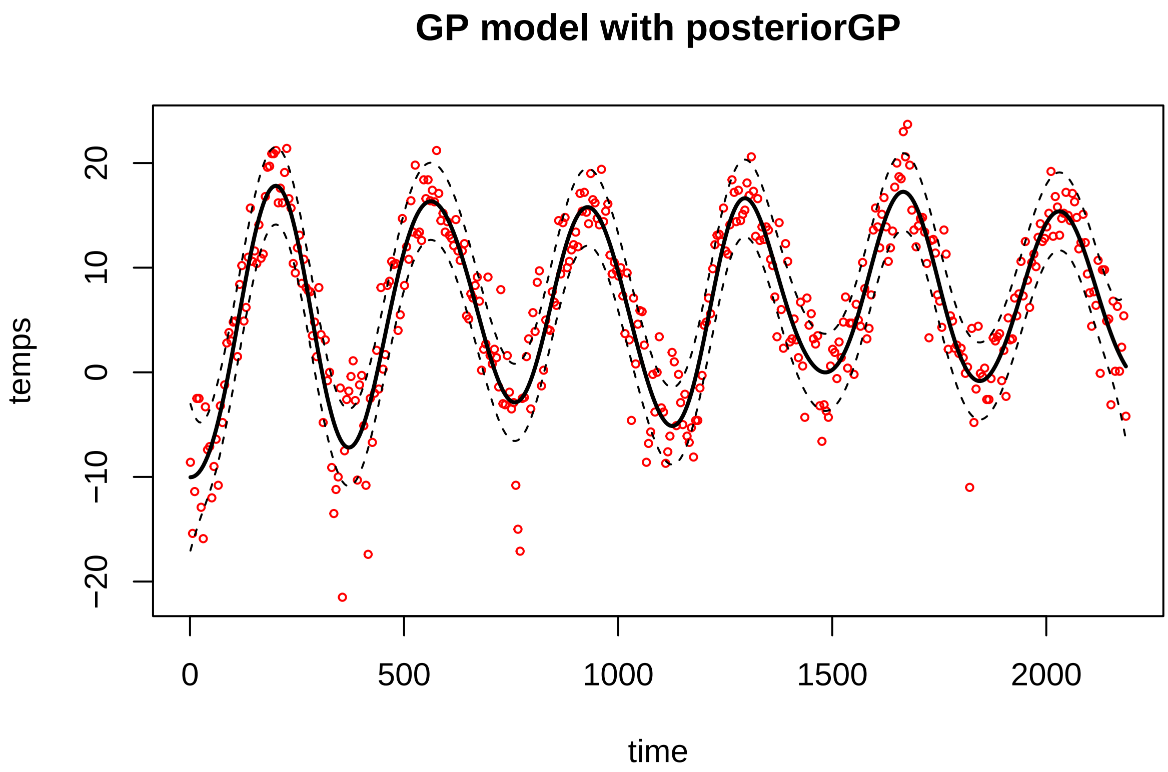
```
# new plot function
GP_plot_new = function(time_mean_pred, upper, lower, title){
plot(time, temps, pch = 1, cex = 0.5, col = "red", main = title)
lines(time, time_mean_pred, lwd = 2)
lines(time, upper,lty = 2)
lines(time, lower ,lty = 2)
}
```

```
quad_model = lm(temps~time + I(time^2), data = data_sampled)

fit = gausspr(time, temps, kernel = SEKernel(ell = 100, sigmaF = 20), var = var(quad_model$residuals), 
time_mean_pred <- predict(fit, time)

upper = time_mean_pred+1.96*predict(fit,time, type="sdeviation")
lower = time_mean_pred-1.96*predict(fit,time, type="sdeviation")
#Plot
GP_plot_new(time_mean_pred, upper, lower, "GP model with gausspr")
```



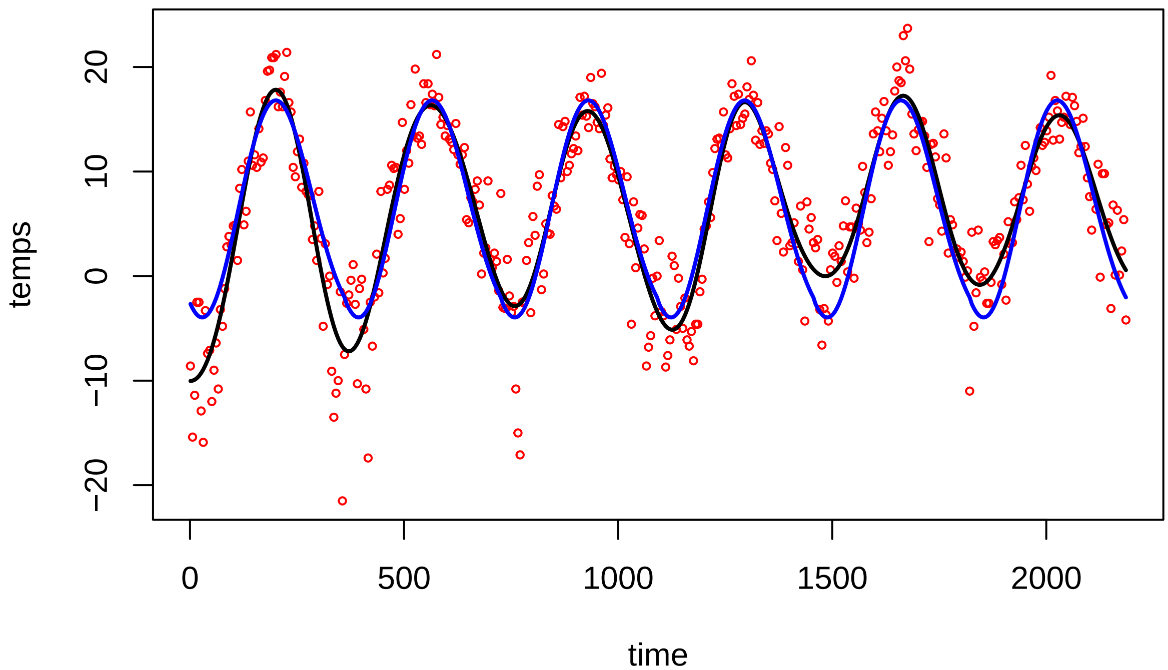
Task 3: Implement Algorithm 2.1 for GP Regression Implement Algorithm 2.1 from Rasmussen and Williams' book to compute the posterior mean and variance.

```
sigmaNoise = sqrt(var(quad_model$residuals))
res = posteriorGP(time, temps, time, sigmaNoise, k = SEKernel(ell = 100, sigmaF = 20))

  upper = res$mean + 1.96 * sqrt(diag(res$variance))
  lower = res$mean - 1.96 * sqrt(diag(res$variance))

#Plot
GP_plot_new(res$mean, upper, lower, "GP model with posteriorGP")
```



Task 4: Estimate Model with day Variable Estimate the model using `gausspr` with the day variable, superimpose the posterior mean on the previous model. Compare the results of both models. What are the pros and cons of each model?

```
# constructing day vector for 6 years
days = rep(day, 6)

quad_model_day = lm(temps~days + I(days^2), data = data_sampled)

fit_day = gausspr(days, temps, kernel = SEKernel(ell = 100, sigmaF = 20), var = var(quad_model_day$resi

day_mean_pred <- predict(fit_day, days)

upper2 = day_mean_pred+1.96*predict(fit_day,days, type="sdeviation")
lower2 = day_mean_pred-1.96*predict(fit_day,days, type="sdeviation")

plot(time, temps, pch = 1, cex = 0.5, col = "red")
lines(time, time_mean_pred, lwd = 2)
#lines(time, upper,lty = 2)
#lines(time, lower ,lty = 2)
lines(time, day_mean_pred, lwd = 2, col = "blue")
```



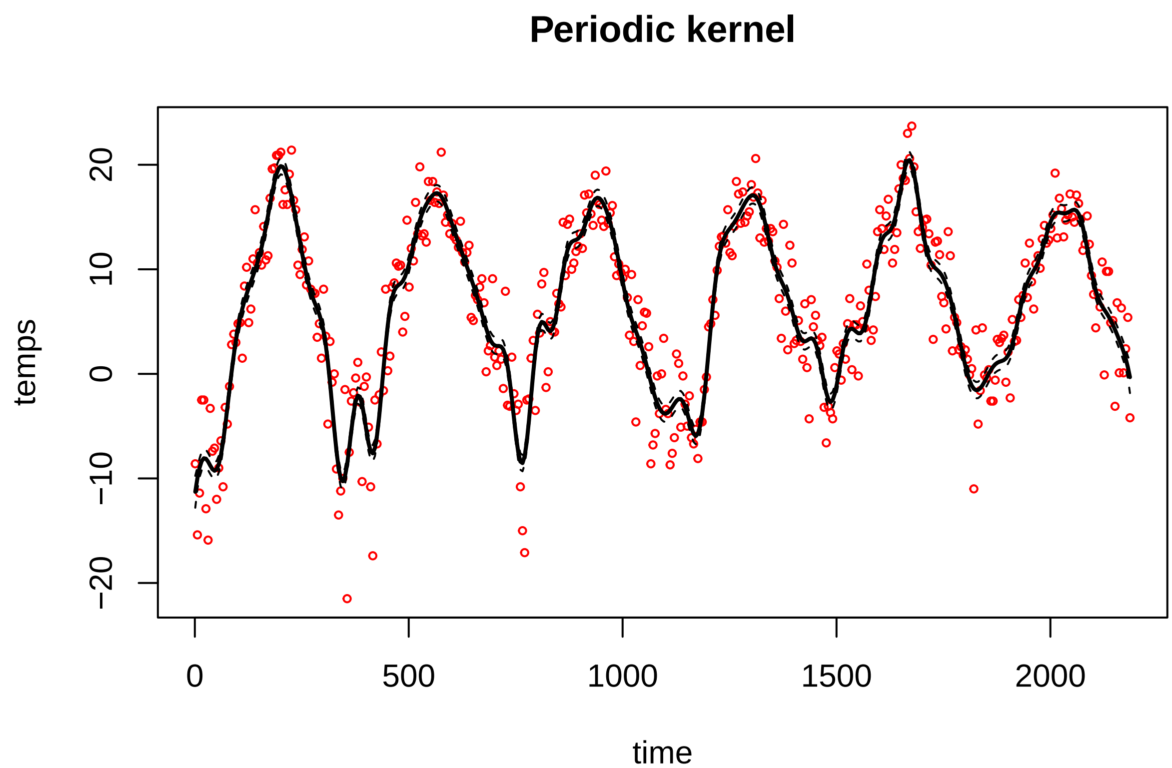

```
# bands for days
#lines(time, upper2,lty = 2, col = "green")
#lines(time, lower2,lty = 2, col = "green")
```

A: Quite similar performance. f(day) seems to have the same height for every year (which is logic since it repeat itself) while f(time) almost has a slight up-going trend, varying more from year to year.

Task 5: Implement Locally Periodic Kernel Implement the extended squared exponential kernel with a periodic kernel and compare the results.

```
# Periodic Kernel Function
PeriodicKernel <- function(sigmaF, ell1, ell2, d){

  calc_K = function (X, XStar) {
    temp1 = exp(-(2*sin(pi*abs(X - XStar)/d)^2)/ell1^2)
    temp2 = exp(- (0.5*abs(X - XStar)^2)/ell2^2)
    K = matrix(NA, length(X), length(XStar))
    for (i in 1:length(X)) {
      K[, i] = (sigmaF^2)*temp1*temp2
    }
    return(K)
  }
  class(calc_K) = 'kernel' # Return as class kernel
  return (calc_K)
}
```

```
fit_periodic = gausspr(time, temps, kernel = PeriodicKernel(ell1 = 1, ell2 = 100, sigmaF = 20, d = 365)

time_mean_periodic_pred = predict(fit_periodic, time)

upper = time_mean_periodic_pred+1.96*predict(fit_periodic, time, type="sdeviation")
lower = time_mean_periodic_pred-1.96*predict(fit_periodic, time, type="sdeviation")

GP_plot_new(time_mean_periodic_pred, upper, lower, "Periodic kernel")
```



Q: Compare the fit to the previous two models (with Sigmaf = 20 and l = 100). Discuss the results.

A: The periodic GP fits the data more tightly with much narrower confidence bands compared to previous models. This could indicate over-fitting but at the same time we don't have any test data so we cant decide which generalizes better

2.3 GP Classification with kernlab

Task 1: Fit GP Classification Model Use the `kernlab` package to fit a Gaussian process classification model for fraud, and plot contours of prediction probabilities.

```
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)
```

```r
SelectTraining <- sample(1:dim(data)[1], size = 1000, replace = FALSE)
train_data = data[SelectTraining, ]
test_data = data[-SelectTraining, ]

GPfit <- gausspr(fraud ~ varWave + skewWave, kernel = "rbfdot", data=train_data)
```

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

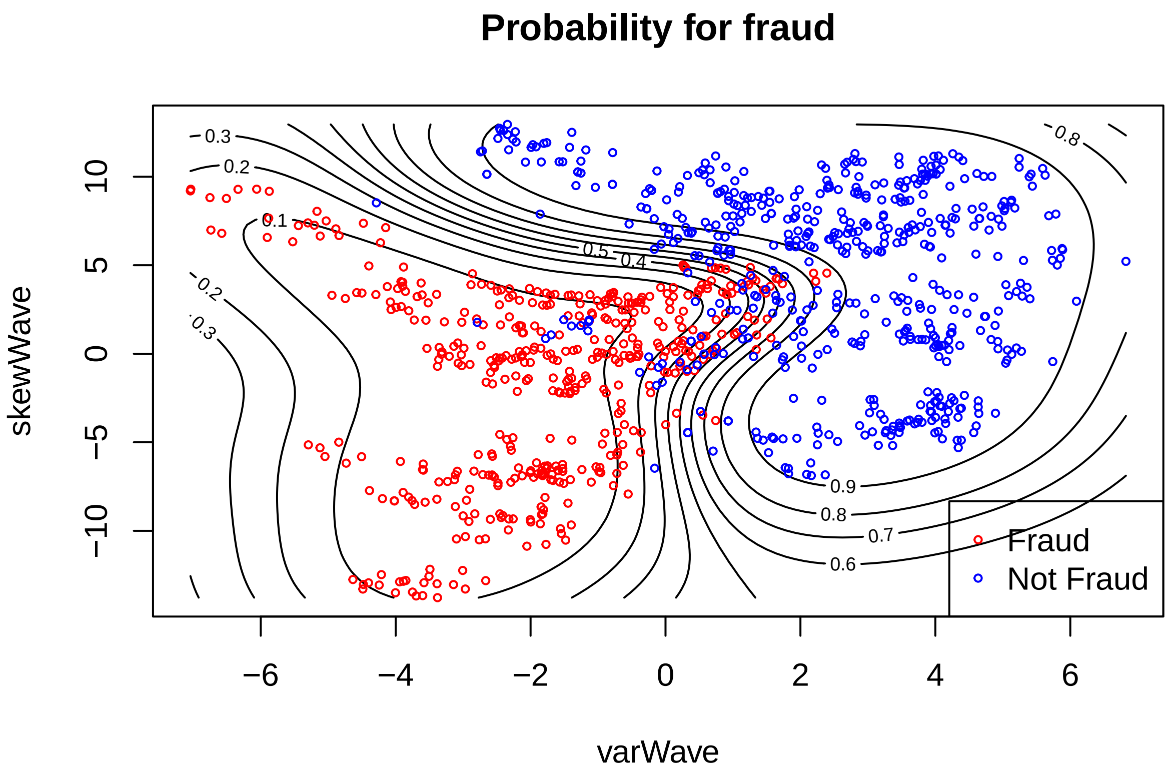
```r
pred = predict(GPfit, train_data)
cm = table(pred, train_data$fraud)
cm
```

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

```r
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set:", accuracy*100, "%\n")
```

```
## Accuracy on the training set: 94.1 %
```

```r
x1 <- seq(min(train_data[,1]),max(train_data[,1]),length=100)
x2 <- seq(min(train_data[,2]),max(train_data[,2]),length=100)
gridPoints <- meshgrid(x1, x2)
gridPoints <- cbind(c(gridPoints$x), c(gridPoints$y))
gridPoints <- data.frame(gridPoints)
names(gridPoints) <- names(train_data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")

# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,1],100,byrow = TRUE), 10, xlab = "varWave", ylab = "skewWave", main = 'I
points(train_data[train_data[,5]==1,1],train_data[train_data[,5]==1,2],col="red", cex = 0.5)
points(train_data[train_data[,5]==0,1],train_data[train_data[,5]==0,2],col="blue", cex = 0.5)
legend("bottomright", legend = c("Fraud", "Not Fraud"), col = c("red", "blue"), pch = 1, pt.cex = 0.5)
```



Task 2: Make Predictions for Test Set Make predictions for the test set and compute the accuracy.

```
pred_test = predict(GPfit, test_data)
cm = table(pred_test, test_data$fraud)
cm
```

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

```
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set using two covariates:", round(accuracy*100, 1), "%\n")
```

```
## Accuracy on the training set using two covariates: 92.5 %
```

Task 3: Train Model with All Covariates Train a model using all four covariates and compare the accuracy to the model with only two covariates.

```
GPfit <- gausspr(fraud ~., kernel = "rbfdot", data=train_data)
```

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

```r
pred_test_all = predict(GPfit, test_data)
cm = table(pred_test_all, test_data$fraud)
cm
```

```
##
## pred_test_all   0   1
##             0 216   0
##             1   2 154
```

```r
accuracy <- sum(diag(cm)) / sum(cm)
cat("Accuracy on the training set using all covariates:", round(accuracy*100, 1), "%\n")
```

```
## Accuracy on the training set using all covariates: 99.5 %
```