lab4

Shipeng Liu

2023-10-07

2.1. Implementing GP Regression

(1)Simulating from the posterior distribution of f using the squared exponential kernel

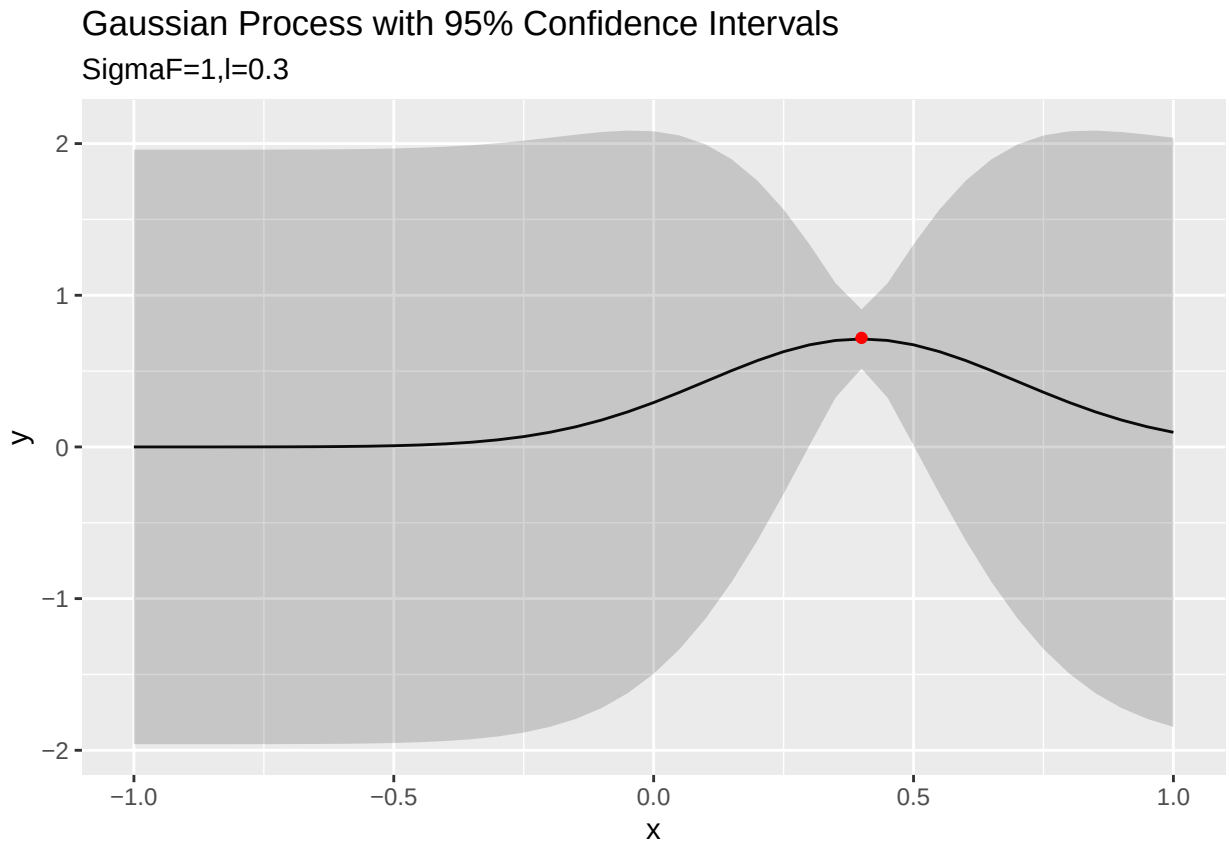
```
# posteriorGP function
posteriorGP=function(train_X,train_target,XStar,sigmaNoise,k){
  train_sample_number=length(train_target)
  L=t(chol(k(train_X,train_X)+(sigmaNoise**2)*diag(train_sample_number)))
  alpha=solve(t(L),solve(L,train_target))

  # Mean of fStar
  kStar=k(train_X,XStar)
  fStar_mean=t(kStar)%*%alpha

  # Variance of fStar
  v=solve(L,kStar)
  fStar_variance=k(XStar,XStar)-t(v)%*%v

  # Log marginal likelihood
  loglik=-(1/2)*t(train_target)%*%alpha-sum(diag(log(L)))-(train_sample_number/2)*log(2*pi)

  f_solution <- list(fStar_mean = fStar_mean, fStar_variance = fStar_variance, loglik = loglik)
  return(f_solution)
}
```

##(2)Update the prior with a single observation, plot the posterior mean over [-1,1]

prior: sigmaF=1, l=0.3 observation: (x,y)=(0.4,0.719)

```
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=0.3){
  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)
}
```

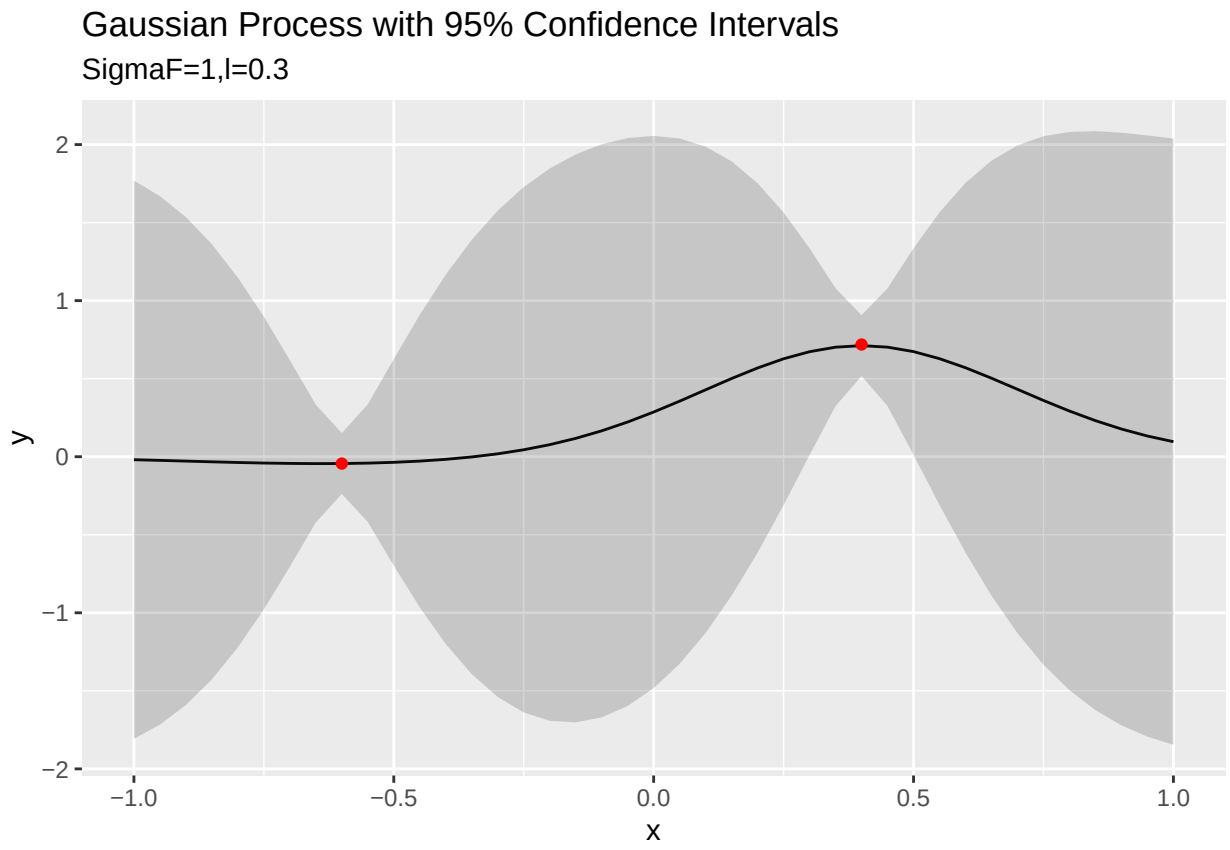
```r
X=0.4
y=0.719

XStar=seq(-1,1,0.05)
plot_matrix=matrix(0,nrow=length(XStar),ncol=4)
colnames(plot_matrix)=c("X","Y","Upper_bound","Lower_bound")

# Make prediction
fStar_GP_res=posteriorGP(X,y,XStar,sigmaNoise=0.1,SquaredExpKernel)

# Plot
plot_matrix[,1]=c(XStar)
plot_matrix[,2]=c(fStar_GP_res$fStar_mean)
plot_matrix[,3]=fStar_GP_res$fStar_mean+sqrt(diag(fStar_GP_res$fStar_variance))*1.96
plot_matrix[,4]=fStar_GP_res$fStar_mean-sqrt(diag(fStar_GP_res$fStar_variance))*1.96

plot_df=as.data.frame(plot_matrix)
observation_df=data.frame(x=X,y=y)

ggplot(data=plot_df,aes(x=X,y=Y))+
  geom_line() +
  geom_ribbon(aes(ymin = Lower_bound, ymax = Upper_bound), alpha = 0.2) +
  geom_point(data=observation_df,aes(x = x, y = y), color='red') +
  labs(title = "Gaussian Process with 95% Confidence Intervals",
       subtitle = "SigmaF=1,l=0.3",
       x = "x",
       y = "y")
```



(3) Update the prior with another observation (-0.6,-0.044)

```
X=c(0.4,-0.6)
y=c(0.719,-0.044)

XStar=seq(-1,1,0.05)
plot_matrix=matrix(0,nrow=length(XStar),ncol=4)
colnames(plot_matrix)=c("X","Y","Upper_bound","Lower_bound")

# Make prediction
fStar_GP_res=posteriorGP(X,y,XStar,sigmaNoise=0.1,SquaredExpKernel)

# Plot
plot_matrix[,1]=c(XStar)
plot_matrix[,2]=c(fStar_GP_res$fStar_mean)
plot_matrix[,3]=fStar_GP_res$fStar_mean+sqrt(diag(fStar_GP_res$fStar_variance))*1.96
plot_matrix[,4]=fStar_GP_res$fStar_mean-sqrt(diag(fStar_GP_res$fStar_variance))*1.96

plot_df=as.data.frame(plot_matrix)
observation_df=data.frame(x=X,y=y)

ggplot(data=plot_df,aes(x=X,y=Y))+
  geom_line() +
  geom_ribbon(aes(ymin = Lower_bound, ymax = Upper_bound), alpha = 0.2) +
```

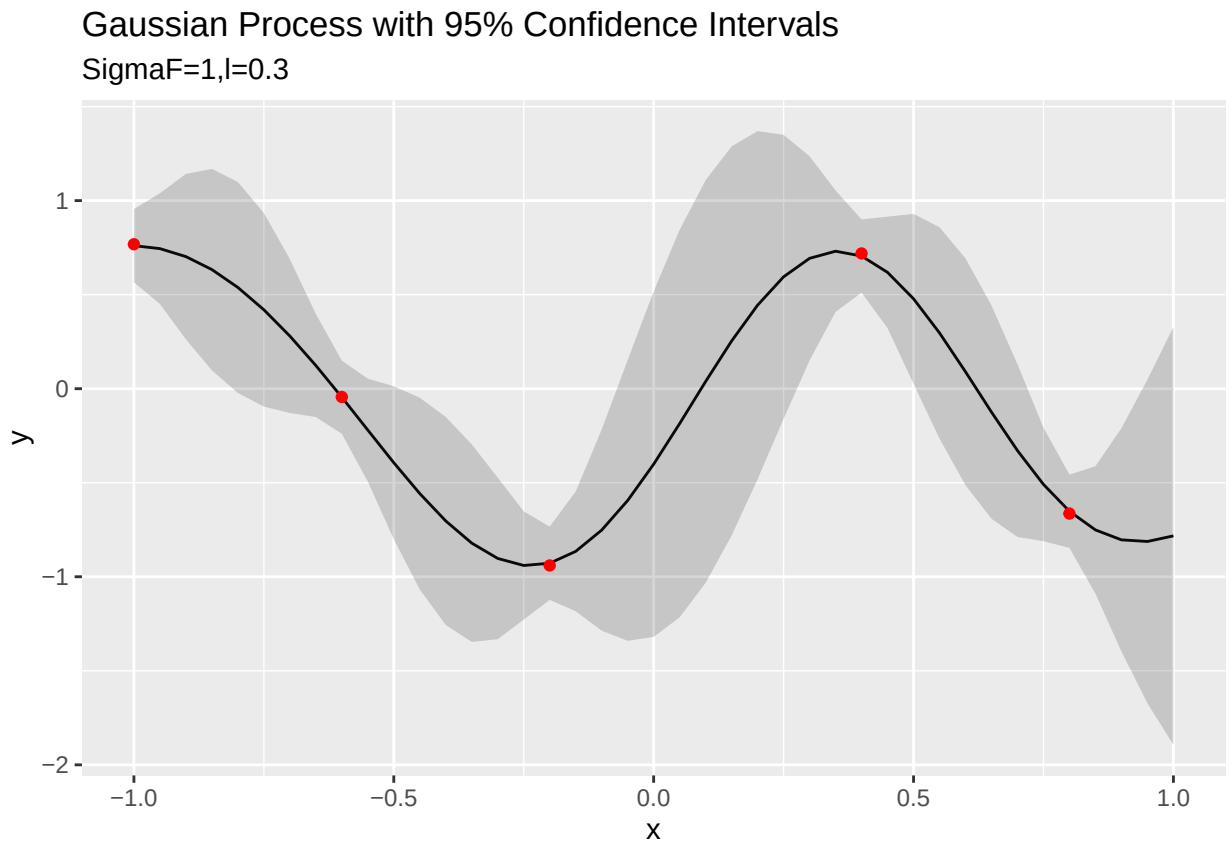
```
  geom_point(data=observation_df,aes(x = x, y = y), color='red') +
  labs(title = "Gaussian Process with 95% Confidence Intervals",
       subtitle = "SigmaF=1,l=0.3",
       x = "x",
       y = "y")
```



(4)Compute the posterior distribution of f using all the five data points

```
X=c(-1.0,0.4,-0.6,-0.2,0.8)
y=c(0.768,0.719,-0.044,-0.940,-0.664)

XStar=seq(-1,1,0.05)
plot_matrix=matrix(0,nrow=length(XStar),ncol=4)
colnames(plot_matrix)=c("X","Y","Upper_bound","Lower_bound")

# Make prediction
fStar_GP_res=posteriorGP(X,y,XStar,sigmaNoise=0.1,SquaredExpKernel)

# Plot
plot_matrix[,1]=c(XStar)
plot_matrix[,2]=c(fStar_GP_res$fStar_mean)
plot_matrix[,3]=fStar_GP_res$fStar_mean+sqrt(diag(fStar_GP_res$fStar_variance))*1.96
plot_matrix[,4]=fStar_GP_res$fStar_mean-sqrt(diag(fStar_GP_res$fStar_variance))*1.96
```

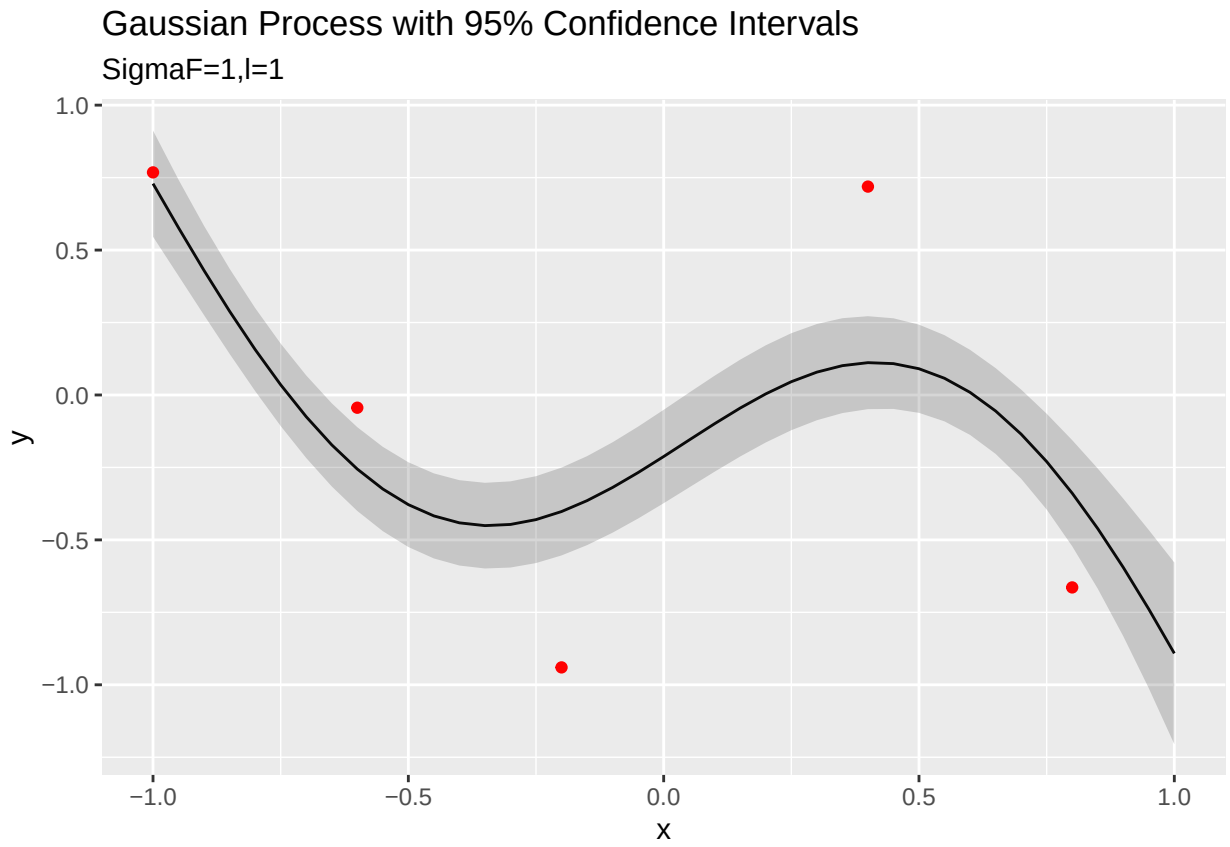
```
plot_df=as.data.frame(plot_matrix)
observation_df=data.frame(x=X,y=y)

ggplot(data=plot_df,aes(x=X,y=Y))+
  geom_line() +
  geom_ribbon(aes(ymin = Lower_bound, ymax = Upper_bound), alpha = 0.2) +
  geom_point(data=observation_df,aes(x = x, y = y), color='red') +
  labs(title = "Gaussian Process with 95% Confidence Intervals",
       subtitle = "SigmaF=1,l=0.3",
       x = "x",
       y = "y")
```



(5) Repeat (4), this time with hyperparameters SigmaF = 1 and l = 1. Compare the results.

```
# Covariance function
SquaredExpKernel <- function(x1,x2,sigmaF=1,l=1){
  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)
}

X=c(-1.0,0.4,-0.6,-0.2,0.8)
y=c(0.768,0.719,-0.044,-0.940,-0.664)

XStar=seq(-1,1,0.05)
plot_matrix=matrix(0,nrow=length(XStar),ncol=4)
colnames(plot_matrix)=c("X","Y","Upper_bound","Lower_bound")

# Make prediction
fStar_GP_res=posteriorGP(X,y,XStar,sigmaNoise=0.1,SquaredExpKernel)

# Plot
plot_matrix[,1]=c(XStar)
plot_matrix[,2]=c(fStar_GP_res$fStar_mean)
plot_matrix[,3]=fStar_GP_res$fStar_mean+sqrt(diag(fStar_GP_res$fStar_variance))*1.96
plot_matrix[,4]=fStar_GP_res$fStar_mean-sqrt(diag(fStar_GP_res$fStar_variance))*1.96

plot_df=as.data.frame(plot_matrix)
observation_df=data.frame(x=X,y=y)

ggplot(data=plot_df,aes(x=X,y=Y))+
  geom_line() +
  geom_ribbon(aes(ymin = Lower_bound, ymax = Upper_bound), alpha = 0.2) +
  geom_point(data=observation_df,aes(x = x, y = y), color='red') +
  labs(title = "Gaussian Process with 95% Confidence Intervals",
       subtitle = "SigmaF=1,l=1",
       x = "x",
       y = "y")
```



We have the Gaussian kernel function:

$$K(X, X') = Cov(f(X), f(X')) = \sigma_f^2 exp\{-\frac{||X - X'||^2}{2l^2}\}$$

The equation $K(X, X') = Cov(f(X), f(X'))$ derived from $Cov(f(X), f(X'))$ when we apply a nonlinear transformation $\phi \in \mathcal{H}$ to the input space s.t. $K(X, X') =< \phi(X), \phi(X') >$, where $\mathcal{H}$ is Hilbert Space. i.e. $K(X, X')$ is the inner product of the two point $\phi(X)$ and $\phi(X')$ in the feature space.

The transformation ϕ can be arbitrary,with different kernel function. We choose the Gaussian kernel function here, whose corresponding transformation can map the input space to an infinite-dimensional feature space.

This can explain why we can fit a non-linear prediction on the chart, and the covariance function is the same as the kernel function we choose.

From two chart, when l is larger, the predicted line is smoother, with larger square error. When l is smaller, the predicted line is sharper, with lower square error. This is because as l is smaller, the covariance reduce faster when the inner product $||X - X'||^2$ in the input space grow larger, vice versa.

Considering covariance can be used to measure whether there is a linear relationship between two random variables, when l is smaller, there are lower linear relationship for each point, hence the predicted line will be sharper. i.e. The l value control the complexity of the model. Smaller l correspond to complex model, it may cause overfitting problem, and larger l may cause underfitting.

2.2. GP Regression with kernlab

```
rm(list = ls())
temperature_data=read.csv("https://github.com/STIMALiU/AdvMLCourse/raw/master/GaussianProcess/Code/Temp
time=1:length(temperature_data$date)
day=rep(1:365,length(temperature_data$date)/365)
time=time[seq(1, length(time), by = 5)]
day=day[seq(1, length(day), by = 5)]
temperature=temperature_data$temp[seq(1, length(temperature_data$temp), by = 5)]
```

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

```
# Covariance function
selfdef_SquaredExpKernel <- function(sigmaF = 1, l = 1)
{
  rval <- function(x1,x2) {
    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)
  }
  class(rval) <- "kernel"
  return(rval)
}


custom_kernel=selfdef_SquaredExpKernel()
custom_kernel(1,2)
```

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

```
X=c(1,3,4)
XStar=c(2,3,4)

kernelMatrix(kernel = custom_kernel, x = X, y = 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
```

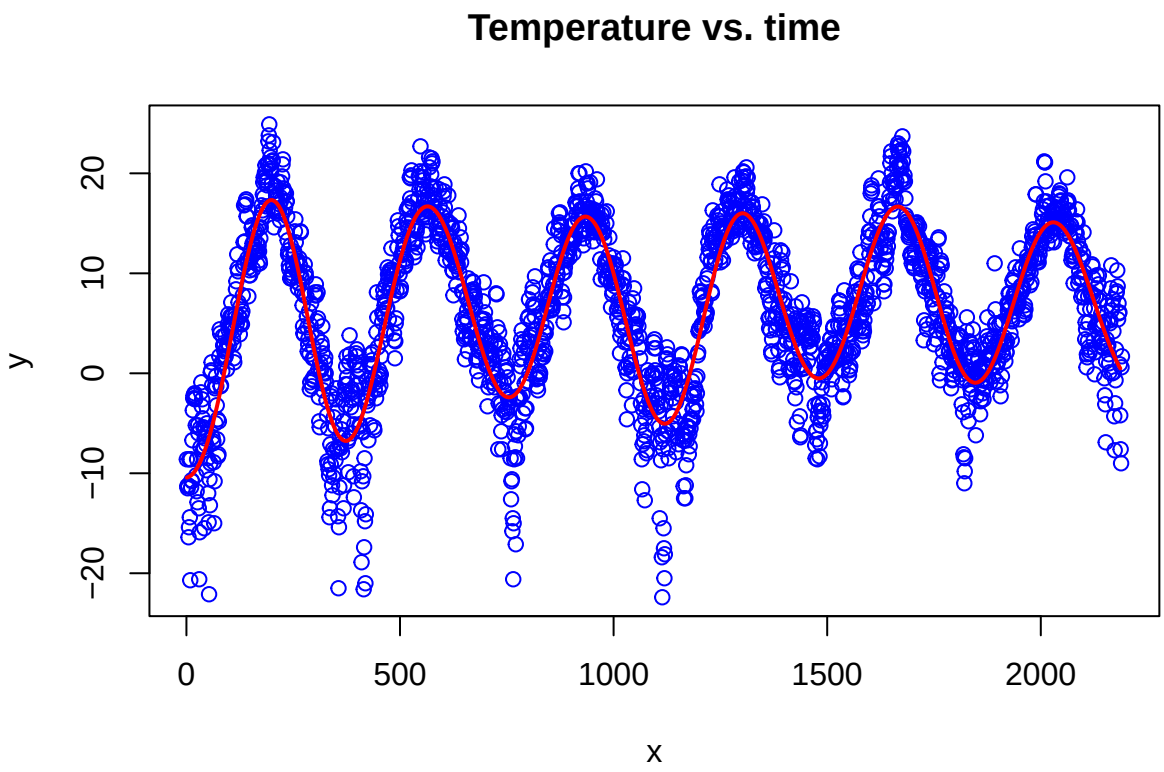
(2)

Predict

```
# Using quadratic regression to fit the temperature with time and get residual variance
quadFit <- lm(temperature ~ time+time^2)
sigmaNoise = sd(quadFit$residuals)

# Fit the Gaussian Process
sigmaF=20
l=0.2
custom_kernel=selfdef_SquaredExpKernel(sigmaF,l)
GPfit <- gausspr(x=as.matrix(time),
                 y=as.matrix(temperature),
                 data=NULL,
                 kernel = custom_kernel,
                 var = sigmaNoise^2)
meanPred <- predict(GPfit, time) # Predicting the training data.
```

Plot

```
plot( 1:2190,temperature_data$temp, type = "p", col = "blue", xlab = "x", ylab = "y", main = "Temperatu
lines( time,meanPred, type = "l", col = "red", lwd=2)
```



(3) Compute the posterior variance of f and plot the 95 % probability (pointwise) bands for f

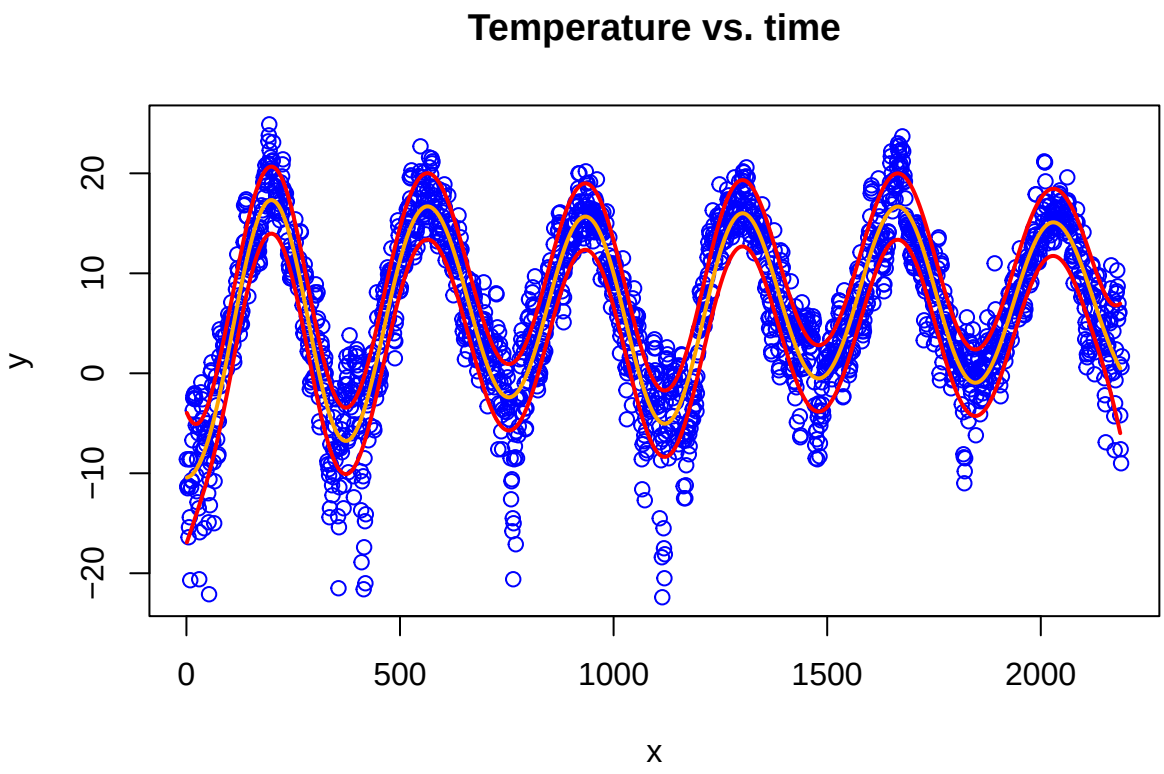
```
# posteriorGP function
posteriorGP=function(train_X,train_target,XStar,sigmaNoise,k){
  train_sample_number=length(train_target)
  L=t(chol(k(train_X,train_X)+(sigmaNoise**2)*diag(train_sample_number)))
  alpha=solve(t(L),solve(L,train_target))

  # Mean of fStar
  kStar=k(train_X,XStar)
  fStar_mean=t(kStar)%*%alpha

  # Variance of fStar
  v=solve(L,kStar)
  fStar_variance=k(XStar,XStar)-t(v)%*%v

  # Log marginal likelihood
  loglik=-(1/2)*t(train_target)%*%alpha-sum(diag(log(L)))-(train_sample_number/2)*log(2*pi)

  f_solution <- list(fStar_mean = fStar_mean, fStar_variance = fStar_variance, loglik = loglik)
  return(f_solution)
}
```

Compute the posterior Variance of f

```
# We need to normalized the time scale
GP_res=posteriorGP(scale(time),temperature,scale(time),sigmaNoise=sigmaNoise,custom_kernel)
```

Plot

```
plot(1:2190,temperature_data$temp, type = "p", col = "blue", xlab = "x", ylab = "y", main = "Temperatur
lines(time,meanPred, type = "l", col = "orange", lwd=2)
lines(time,meanPred+1.96*sqrt(diag(GP_res$fStar_variance)), type = "l", col = "red", lwd=2)
lines(time,meanPred-1.96*sqrt(diag(GP_res$fStar_variance)), type = "l", col = "red", lwd=2)
```



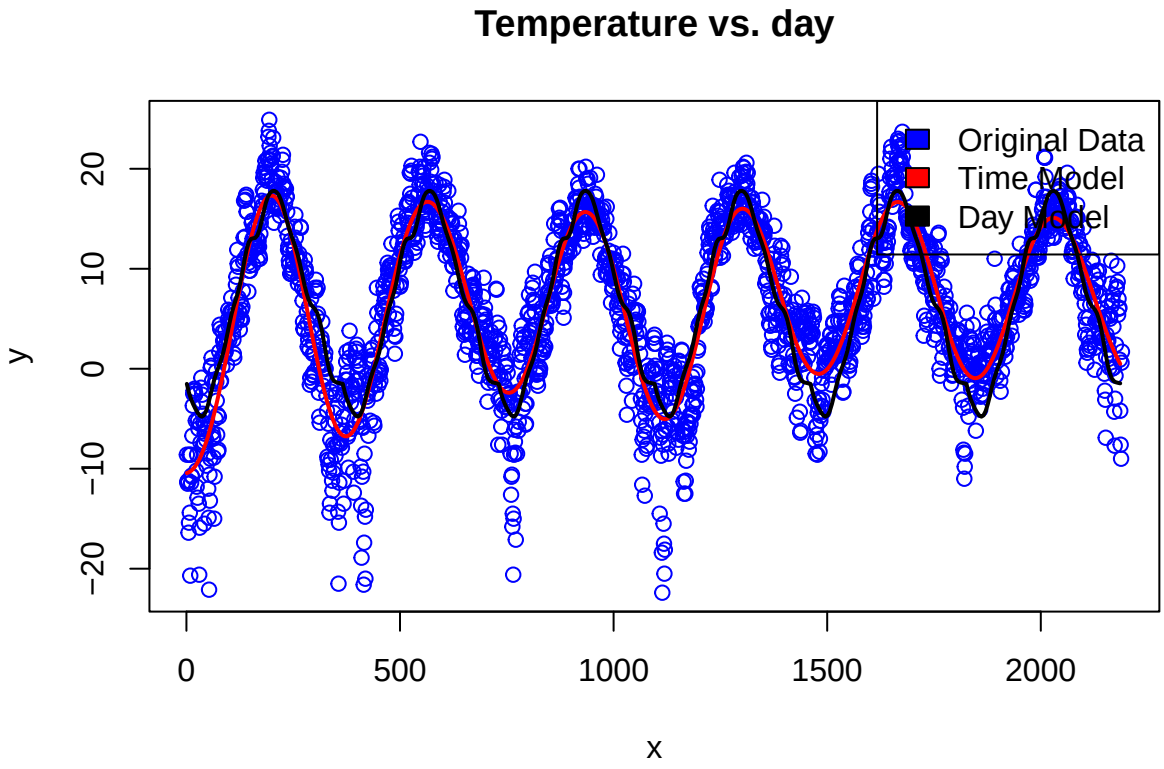
(4)Consider Day Model

```
# Using quadratic regression to fit the temperature with time and get residual variance
quadFit_day <- lm(temperature ~ day+day^2)
sigmaNoise_day = sd(quadFit_day$residuals)

# Fit the Gaussian Process
sigmaF=20
l=0.2
custom_kernel=selfdef_SquaredExpKernel(sigmaF,l)
GPfit_day <- gausspr(x=as.matrix(day),
                     y=as.matrix(temperature),
                     data=NULL,
                     kernel = custom_kernel,
                     var = sigmaNoise_day^2)
meanPred_day <- predict(GPfit_day, day) # Predicting the training data.
```

Plot

```
plot( 1:2190,temperature_data$temp, type = "p", col = "blue", xlab = "x", ylab = "y", main = "Temperatu
lines(time,meanPred, type = "l", col = "red", lwd=2)
lines(time,meanPred_day, type = "l", col = "black", lwd=2)
```

```
legend('topright',legend = c("Original Data","Time Model","Day Model"),
       fill=c('blue','red','black'))
```



Time Model Time Model uses time instead of day to sample from the function space, which means that Time Model ignores periodicity and makes more accurate predictions for each time point of the training data. However, once the Time Model is used to predict future data, the variance will become larger and larger, and it will become increasingly inaccurate.

Day Model Since the independent variable x is a cyclic array, the covariance K(X,X') is roughly the same for each x, the prediction shows obvious periodicity, and the prediction of future data will be better. But because this model only considers periodicity, its prediction of data from 2010 to 2015 is not as good as the Time model, and it is slightly underfitting in this period of time .

(5)Extension of the squared exponential kernel

$$k(x,x') = \sigma_f^2 exp\{-\frac{2sin^2(\pi|x-x'|/d)}{l_1^2}\}exp\{-\frac{1}{2}\frac{|x-x'|^2}{l_2^2}\}$$

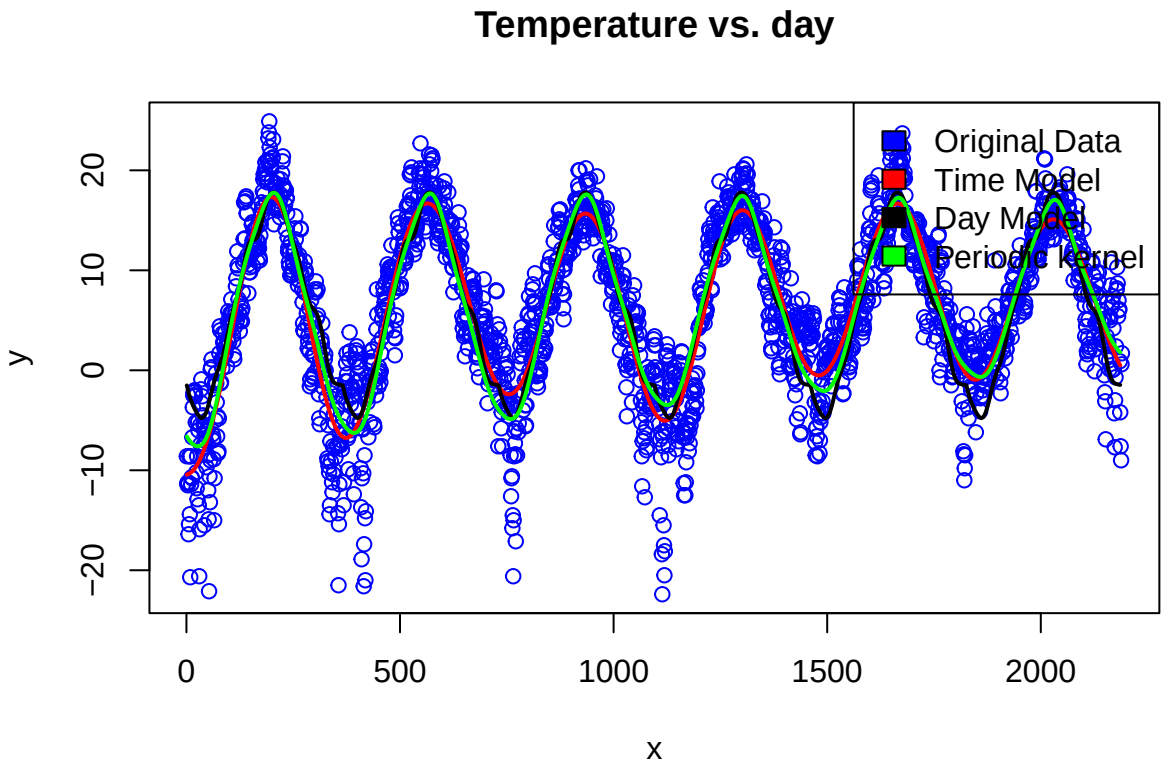
```
# Covariance function
selfdef_SquaredExpKernel <- function(sigmaF = 1, l_2 = 1, l_1 = 1, d)
{
  rval <- function(x1,x2) {
    n1 <- length(x1)
    n2 <- length(x2)
    K <- matrix(NA,n1,n2)
```

```
    for (i in 1:n2){
      K[,i] <- sigmaF^2*exp(-0.5*( (x1-x2[i])/l_2)^2 )*exp(-2*(sin(pi*abs(x1-x2[i])/d)^2)/(l_1^2))
  }
  return(K)
  }
  class(rval) <- "kernel"
  return(rval)
}
```

```
d=365/sd(time)
sigmaF=20
l_1=1
l_2=10

custom_kernel_extension=selfdef_SquaredExpKernel(sigmaF,l_2,l_1,d)

# Fit the Gaussian Process
GPfit_periodic <- gausspr(x=as.matrix(time),
                  y=as.matrix(temperature),
                  data=NULL,
                  kernel = custom_kernel_extension,
                  var = sigmaNoise^2)
meanPred_periodic <- predict(GPfit_periodic, time) # Predicting the training data.
```

Plot

```
plot( 1:2190,temperature_data$temp, type = "p", col = "blue", xlab = "x", ylab = "y", main = "Temperatu
lines(time,meanPred, type = "l", col = "red", lwd=2)
lines(time,meanPred_day, type = "l", col = "black", lwd=2)
lines(time,meanPred_periodic, type = "l", col = "green", lwd=2)
legend('topright',legend = c("Original Data","Time Model","Day Model",'Periodic kernel'),
       fill=c('blue','red','black','green'))
```



We have two different length scales in the kernel. Intuitively, l_1 controls the correlation between two days in the same year, and l_2 controls the correlation between the same day in different years.

Compared with the first two models, the covariance of this model takes into account both different times in the same year and the same day in different years, ensuring periodicity while fitting the data from 2010 to 2015 well, avoiding the shortcomings of the first two models.

2.3. GP Classification with kernlab

```
rm(list = ls())
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)
```

(1)Fit a Gaussian process classification model for fraud on the training data

Prediction

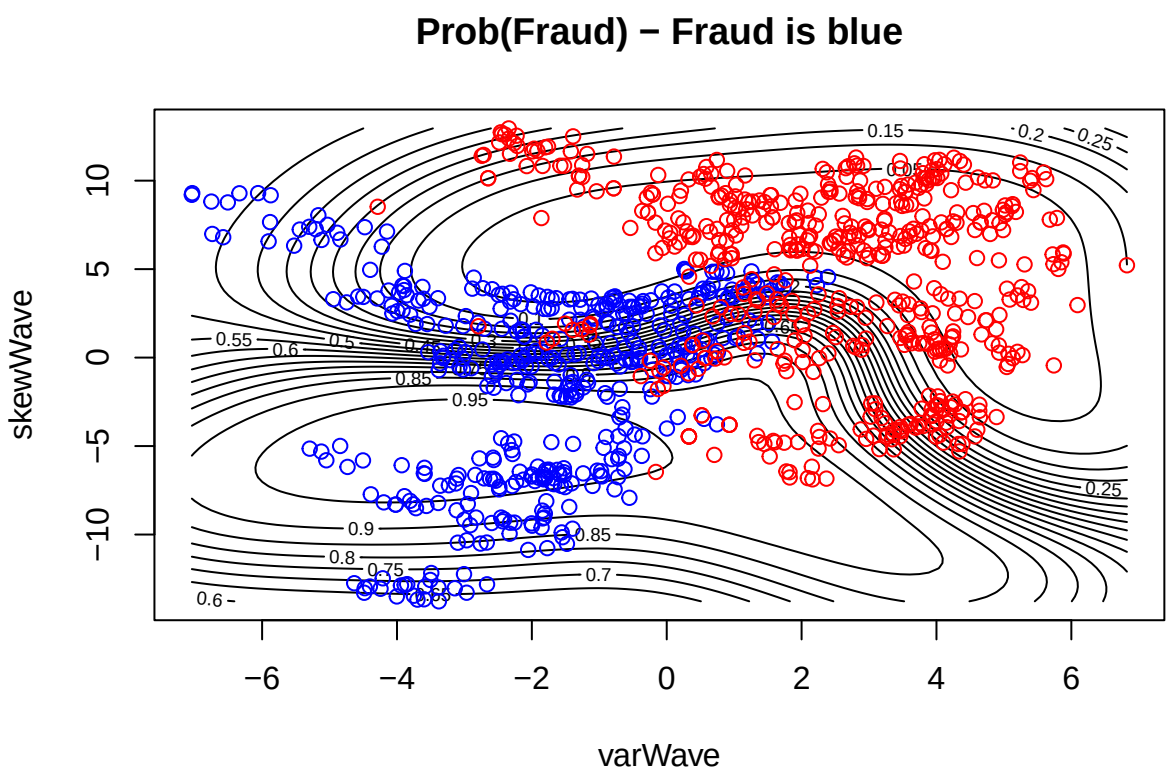
```
training_data=data[SelectTraining,]
GPfit <- gausspr(fraud ~  varWave + skewWave, data=training_data)
```

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

```
# class probabilities

x1 <- seq(min(training_data[,1]),max(training_data[,1]),length=100)
x2 <- seq(min(training_data[,2]),max(training_data[,2]),length=100)

gridPoints <- expand.grid(x1 = x1, x2 = x2)
names(gridPoints) <- names(training_data)[1:2]
probPreds <- predict(GPfit, gridPoints, type="probabilities")
```

Plot

```
# Plotting for Prob(fraud)
contour(x1,x2,matrix(probPreds[,2],100,byrow = TRUE), 20, xlab = "varWave", ylab = "skewWave", main = '
points(training_data[training_data[,5]==1,1],training_data[training_data[,5]==1,2],col="blue")
points(training_data[training_data[,5]==0,1],training_data[training_data[,5]==0,2],col="red")
```



Confusion matrix & Accuracy

```
# predict on the training set
confusion_matrix=table(predict(GPfit,training_data[,1:2]), training_data[,5]) # confusion matrix
confusion_matrix
```

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

```
accuracy=(confusion_matrix[1,1]+confusion_matrix[2,2])/length(training_data[,1])
cat("The Gaussian Process Classification accuracy is:",accuracy)
```

```
## The Gaussian Process Classification accuracy is: 0.941
```

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

```
test_data=data[-SelectTraining,]
test_pred=predict(GPfit,test_data[,1:2])
confusion_matrix_test=table(test_pred,test_data[,5])
accuracy=(confusion_matrix_test[1,1]+confusion_matrix_test[2,2])/length(test_data[,1])
cat("The Gaussian Process Classification accuracy on the test set is:",accuracy)
```

```
## The Gaussian Process Classification accuracy on the test set is: 0.9247312
```

(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

```
GPfit <- gausspr(fraud ~ ., data=training_data)
```

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

```
test_data=data[-SelectTraining,]
test_pred=predict(GPfit,test_data[,1:4])
confusion_matrix_test=table(test_pred,test_data[,5])
accuracy=(confusion_matrix_test[1,1]+confusion_matrix_test[2,2])/length(test_data[,1])
cat("The Gaussian Process Classification accuracy on the test set is:",accuracy)
```

```
## The Gaussian Process Classification accuracy on the test set is: 0.9946237
```

The result is much better than when we use only 2 feature to fit the Gaussian Process Classification model.Considering the model in (1) is underfitting.