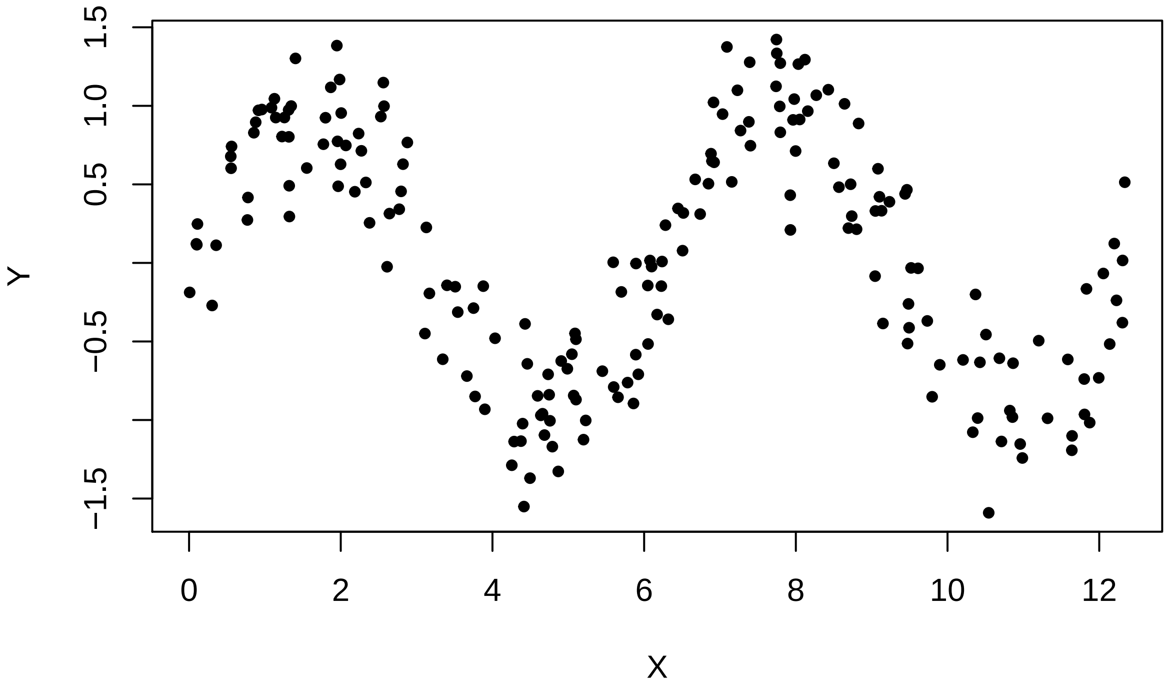
[ESC 21FALL] Homework 1

Sooyon Kim

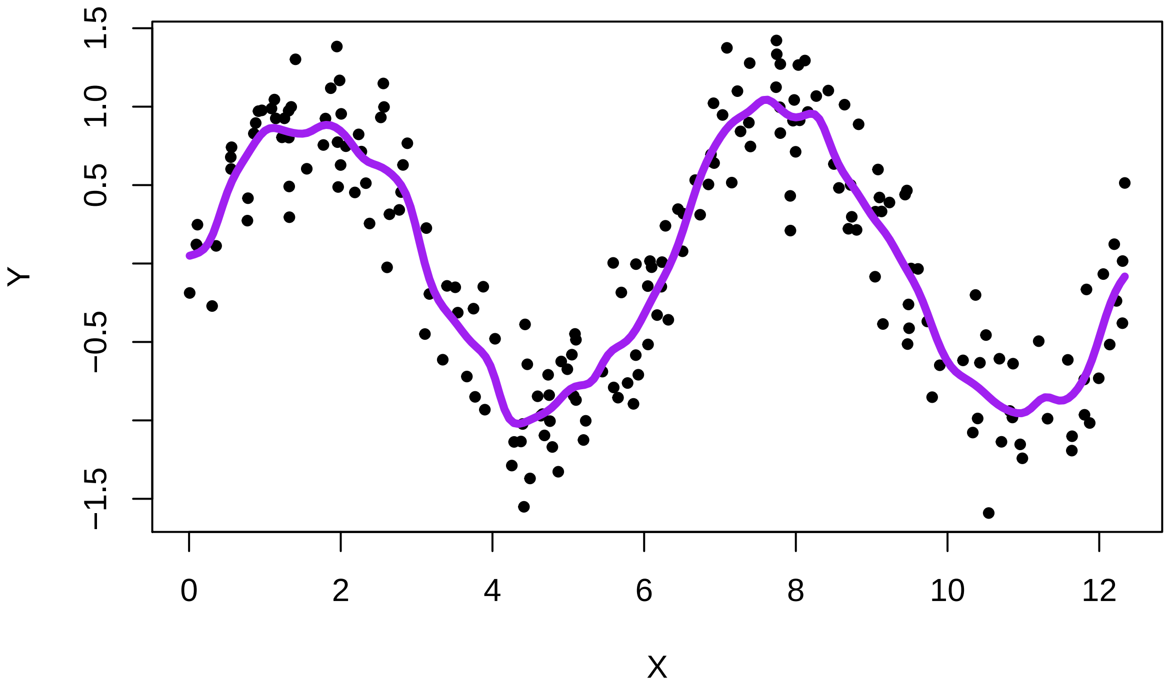
Part 1. Implement and experience KDE on your own!

```
X = sort(runif(200, min=0, max=4*pi)) # generate random number btw 0~4*pi
Y = sin(X) + rnorm(200, sd=0.3)        # add noise to sin function
plot(X, Y, pch=20)                     # draw scatterplot
```



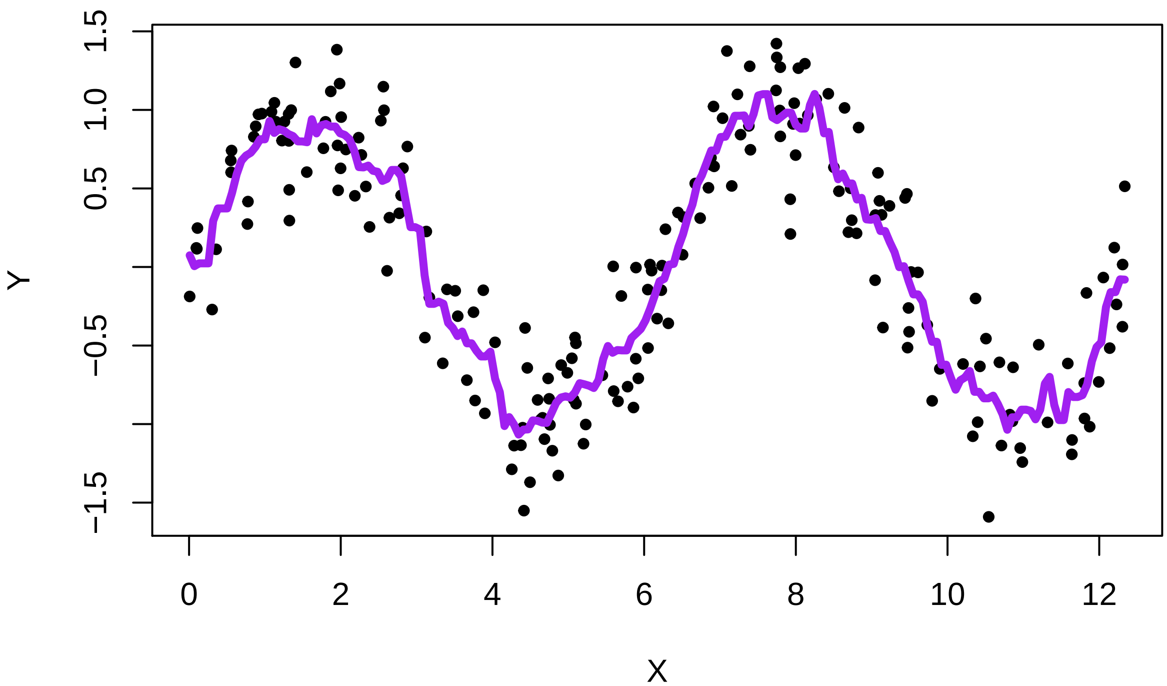
First, let's see how `ksmooth` function works in default R.

```
Kreg = ksmooth(x=X, y=Y, kernel="normal", bandwidth=0.5)
plot(X, Y, pch=20)
lines(Kreg, lwd=4, col="purple")
```



(a) Check how it is different from above when you use box kernel with same bandwith.

```
Kreg = ksmooth(x=X, y=Y, kernel="box", bandwidth=0.5)
plot(X, Y, pch=20)
lines(Kreg, lwd=4, col="purple")
```



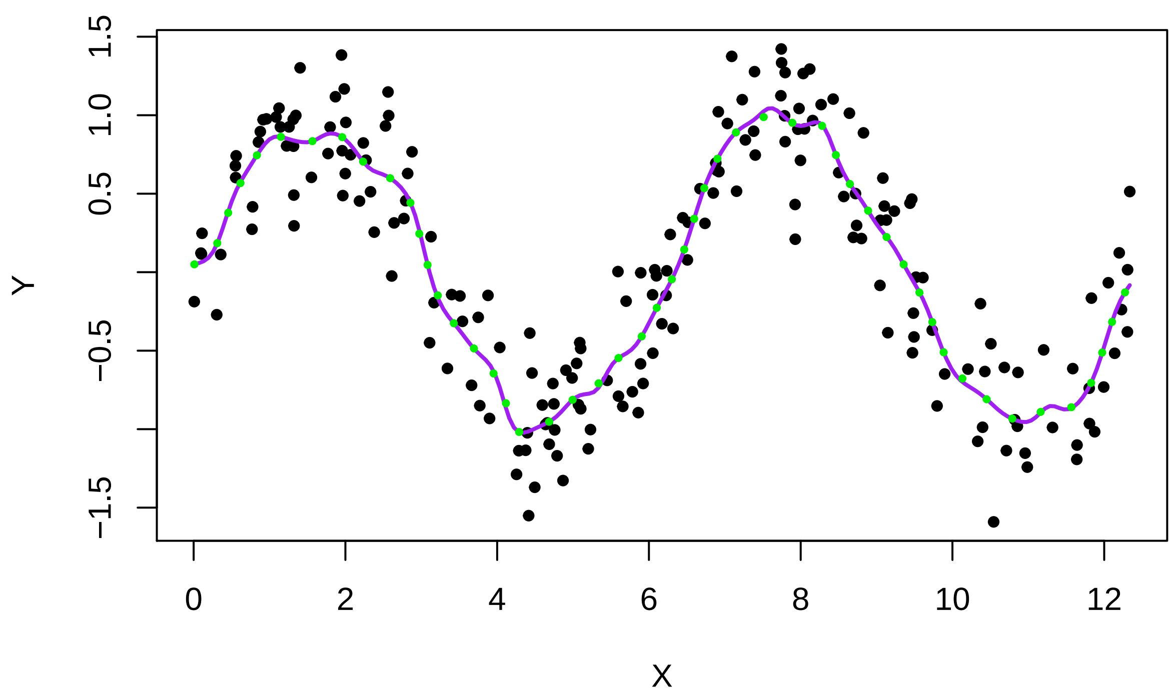
(b) Implement your own kernel function from scratch!

```
ksmooth.train <- function(x.train, y.train, bandwidth = 0.5) {
  # kernel should be scaled so that their quartiles
  # (viewed as probability densities) are at +/- 0.25*bandwidth
```

```
  sigma = 0.25*bandwidth/qnorm(0.75, 0, 1)
  # define Gaussian kernel
  kern <- function(x) dnorm(x, 0, sigma)

  # empty list to store yhat (f hat) values
  yhat.train = numeric(length(x.train))
  for (i in 1:length(x.train)) {
      yhat.train[i]=sum(y.train*kern(x.train[i]-x.train))/sum(kern(x.train[i]-x.train))
  }
  ksmooth.train.out = cbind(x.train, yhat.train)

  return(ksmooth.train.out)
}
```

(c) Check if you did well :)

```
Kreg = ksmooth(x=X, y=Y, kernel="normal", bandwidth=0.5)
myKreg = ksmooth.train(x.train=X, y.train=Y, bandwidth=0.5)
plot(X, Y, pch=20)
lines(Kreg, lwd=2, col="purple")
lines(myKreg, lty=3, lwd=4, col="green2")
```



(d) Let's do it on more realistic dataset.

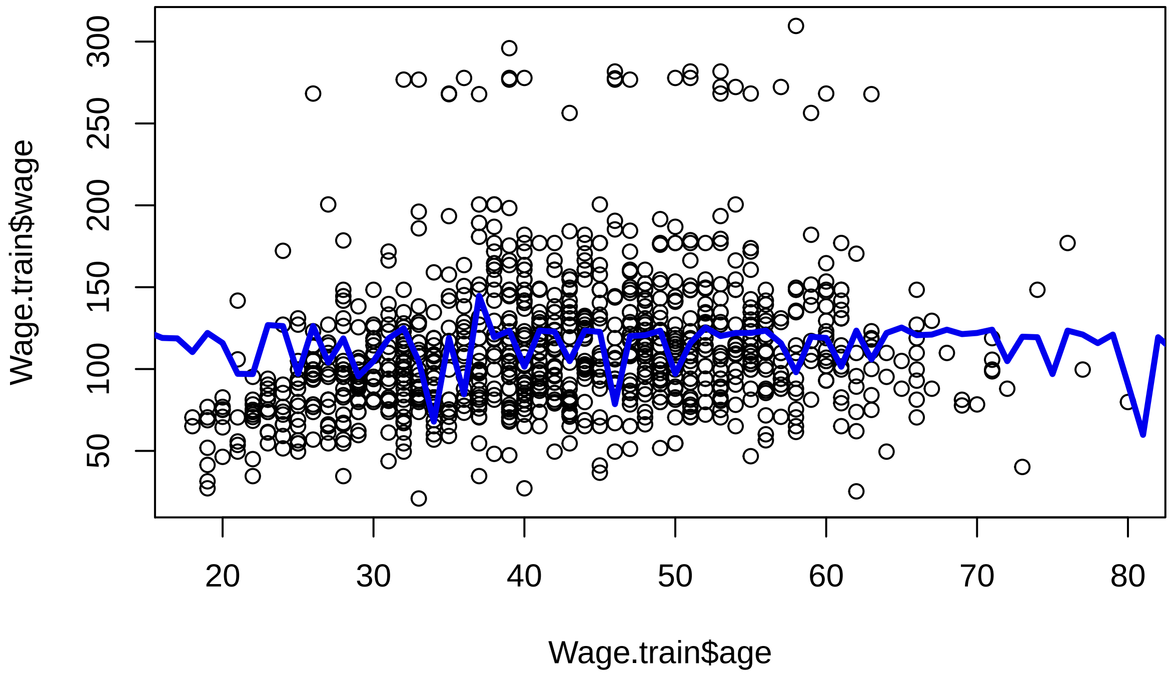
```
source('home1-part1-data.R')
```

Produce a scatterplot of `wage.train` vs `age.train` and add a kernel smooth for a `normal` kernel with `bandwidth = 3`. Observe the residual sum of squares.

```
smooth = ksmooth.train(Wage.train$age, Wage.train$wage, bandwidth = 3)
age.train = smooth[,1]
wage.train = smooth[,2]
RSS.train = sum((Wage.train$wage-wage.train)^2)
cat("RSS.train : ", RSS.train)
```

```
## RSS.train :  1625121
```

```
plot(Wage.train$age, Wage.train$wage)
lines(wage.train, col = 'blue2', lwd=3)
```

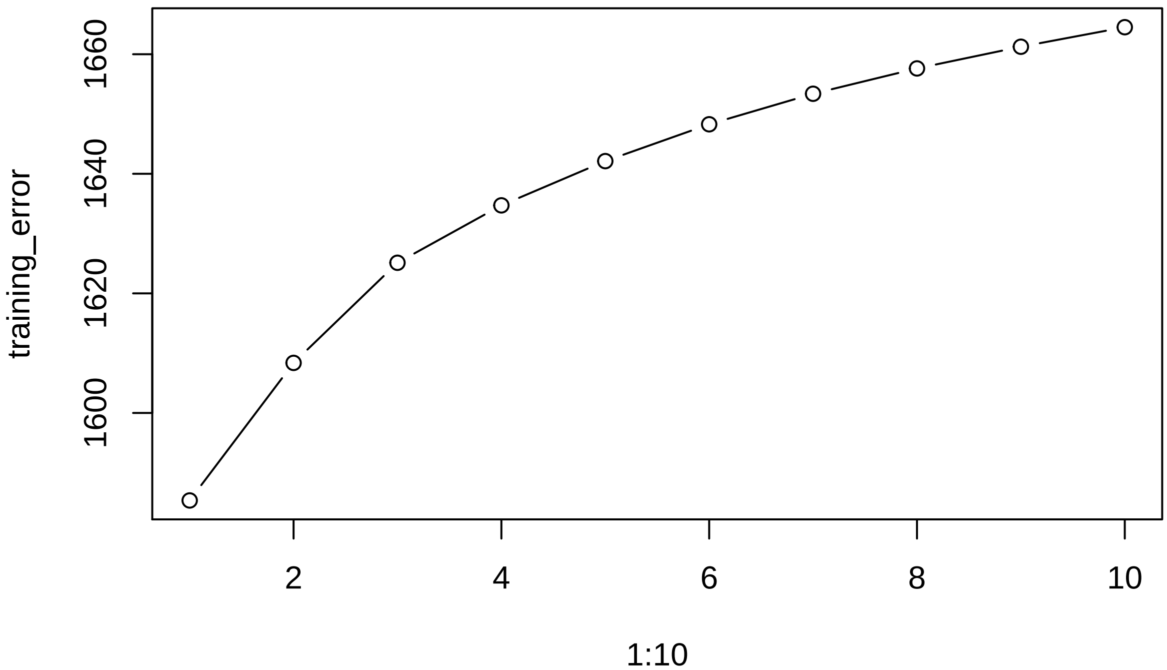


Plot the training error of the expected squared prediction error as a function of bandwidth for bandwidths = 1, 2, . . . , 10. Print the 10 values and explain the result briefly.

```
training_error = numeric(10)
for (i in 1:10) {
  trained = ksmooth.train(Wage.train$age, Wage.train$wage, bandwidth = i)
  training_error[i] = sum((Wage.train$wage-trained[,2])^2)/length(trained[,2])
}
print(training_error)
```

```
##  [1] 1585.364 1608.370 1625.121 1634.722 1642.120 1648.282 1653.387 1657.624
##  [9] 1661.252 1664.519
```

```
plot(1:10, training_error, type = 'b')
```



(Now, we will continue to experiment bias-variance tradeoff in optimal bandwidth problem)

Part 2. Optimal Bandwidth (refer to hw description in `README.md`)

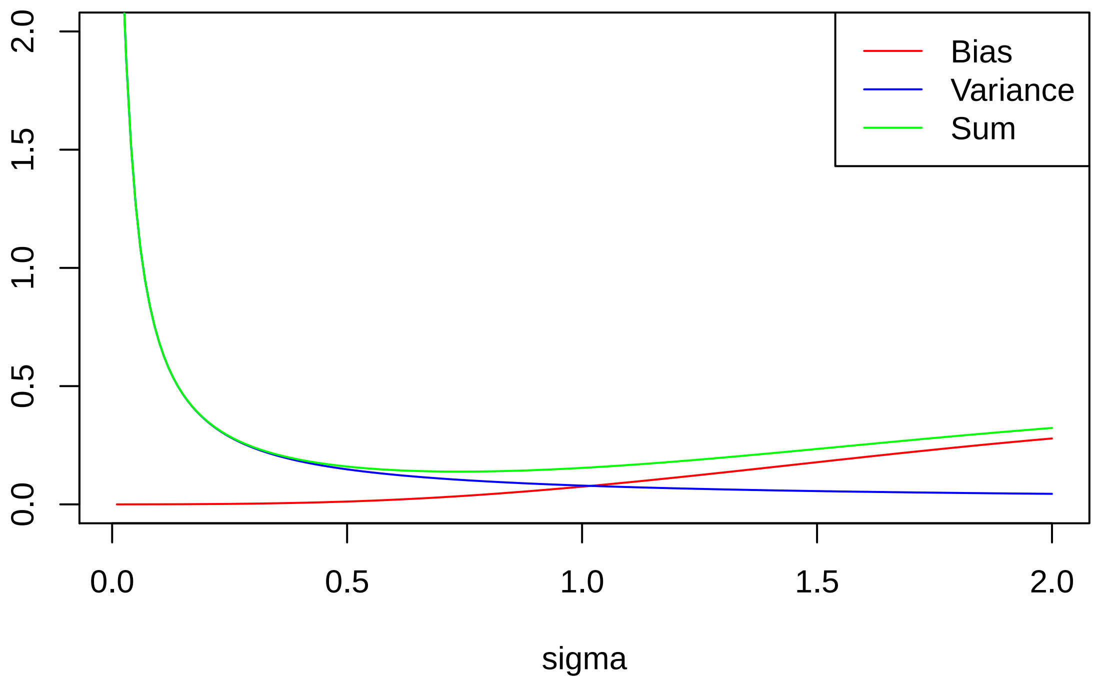
(a) Using a Gaussian kernel ϕ_σ, plot squared bias, variance, and their sum for $\sigma =$`seq(from = 0.01, to = 2, by = 0.01)`. Print the optimal choice for σ.

```
source('home1-part2-data.R')
```

(This process takes some time...!)

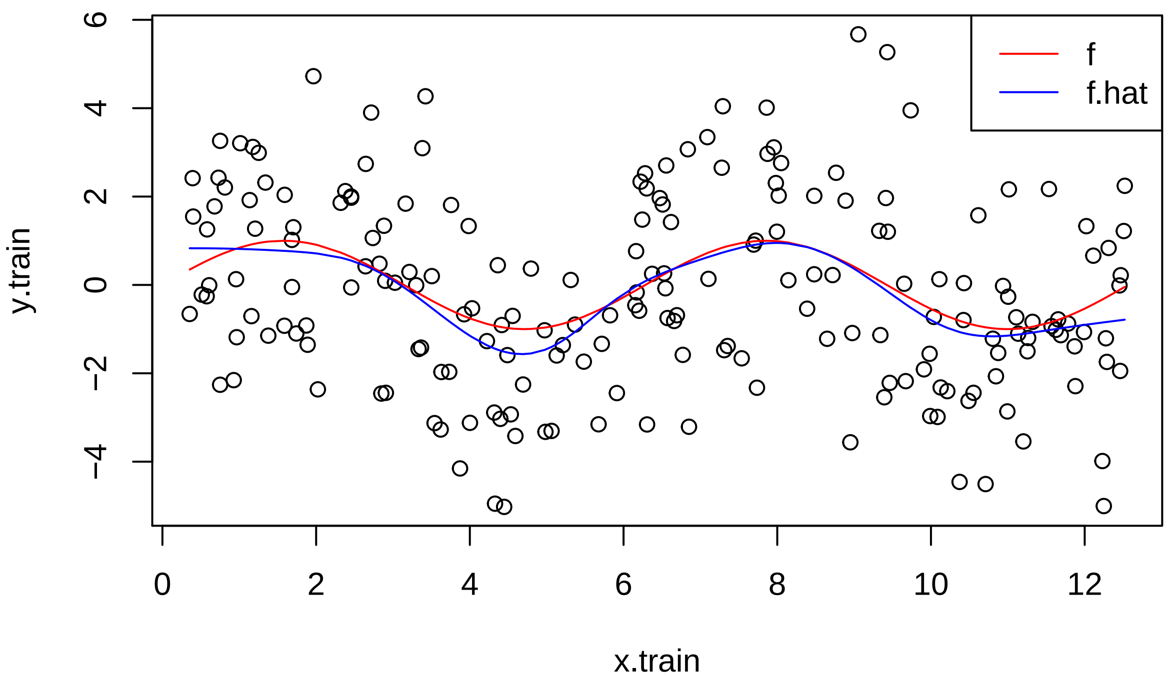
```
# Initialization
sigma = seq(from = 0.01, to = 2, by = 0.01)
n = length(sigma)
squared_norm <- function(x) sum(x^2)  # will be used for computing bias^2

# initialize empty list to store values
bias = numeric(n)          # stores bias^2
variance = numeric(n)
summation = numeric(n)

for (k in 1:n) {
  # W : weight matrix (kernel function value)
  # make sure to include normalizing part!
  W = matrix(nrow = n, ncol = n)
  for (i in 1:n) {         # move filter (kernel) through query point
    for (j in 1:n) {       # local neighborhood (all data due to Gauessian kernel)
      W[i,j] = dnorm(x.train[i]-x.train[j], 0, sigma[k])/sum(dnorm(x.train[i]-x.train, 0, sigma[k]))
    }
  }
  variance[k] = noise.var * sum(diag(t(W)%*%W)) / n
  bias[k] = squared_norm(W%*%f-f) / n
  summation[k] = variance[k] + bias[k]
}
```

```r
# Plotting
plot(sigma, bias, type = 'l', col = 'red', ylim = c(0.0, 2.0),
     xlab = 'sigma', ylab='')
lines(sigma, variance, col = 'blue')
lines(sigma, summation, col = 'green')
legend('topright', legend = c("Bias", "Variance", "Sum"),
       col = c("red", "blue", "green"), lty = 1)
```



(b) Plot the training sample, f, and $\hat{f}$ for the optimal choice of σ.

```r
opt = sigma[which.min(summation)]
W = matrix(nrow = n, ncol = n)
for (i in 1:n) {
  for (j in 1:n) {
    W[i,j] = dnorm(x.train[i]-x.train[j], 0, opt)/sum(dnorm(x.train[i]-x.train, 0, opt))
  }
}
plot(x.train, y.train)
lines(x.train, f, col = 'red')
lines(x.train, W%*%y.train, col = 'blue')
legend('topright', legend = c("f", "f.hat"), col = c("red", "blue"), lty = 1)
```



(c) Check the output for simulated data in Part 1.

```
Kreg1 = ksmooth(x=X,y=Y,kernel = "normal",bandwidth = 0.1)
Kreg2 = ksmooth(x=X,y=Y,kernel = "normal",bandwidth = 0.9)
Kreg3 = ksmooth(x=X,y=Y,kernel = "normal",bandwidth = 3.0)
plot(X,Y,pch=20)
lines(Kreg1, lwd=3, col="orange")
lines(Kreg2, lwd=3, col="purple")
lines(Kreg3, lwd=3, col="limegreen")
legend("topright", c("h=0.1","h=0.9","h=3.0"), lwd=6,
col=c("orange","purple","limegreen"))
```