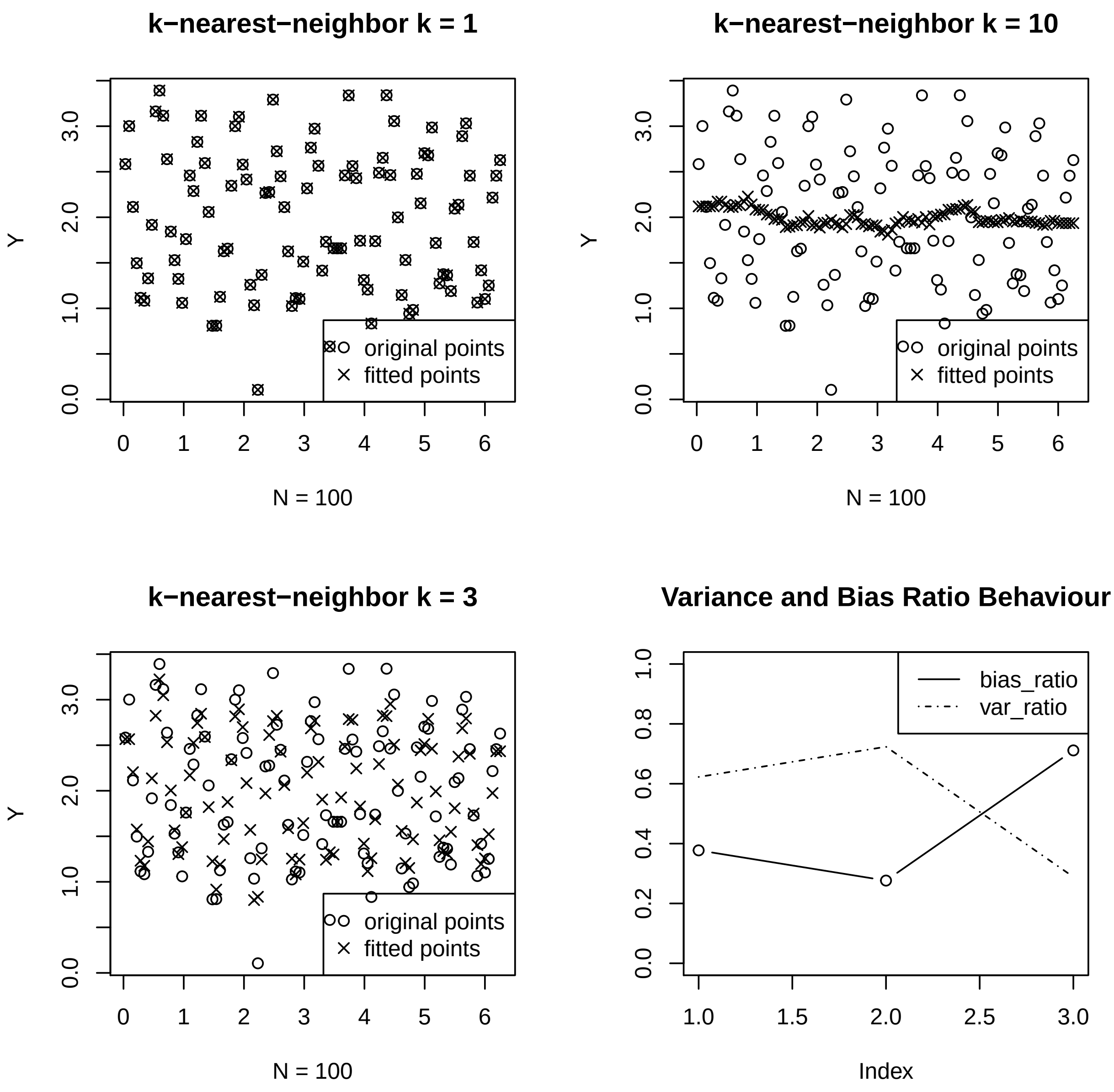
Problem Two 2

Basically, it is just a repetition of the problem before except that the training dataset has 100 points rather than 10. Furthermore, adding more points to fit the model does not really make the outcome better. It seems that when I tried to do knn, k = 3 gives a better prediction from the perspective of EPE. However, I did not find that N = 100 is helpful in getting better model.

```
set.seed(25041)
require(FNN)

## Loading required package:  FNN
## Warning:  package 'FNN' was built under R version 3.0.2

par(mfcol = c(2, 2))
require(fields)

## Loading required package:  fields
## Loading required package:  spam
## Loading required package:  grid
## Spam version 0.40-0 (2013-09-11) is loaded.
## Type 'help( Spam)' or 'demo( spam)' for a short introduction
## and overview of this package.
## Help for individual functions is also obtained by adding the
## suffix '.spam' to the function name, e.g.  'help( chol.spam)'.
##
## Attaching package:  'spam'
##
## äÿŃåĹŮåŕźèśąèćńåśŔèŤ¡äžĘfrom 'package:base':
##
##     backsolve, forwardsolve
##
## Loading required package:  maps

X <- (c(1:100) - 1/2)/100 * 2 * pi
Y <- cos(10 * X) + 2 + rnorm(100, sd = sqrt(0.1))
# 1) # This is the k-nn funtion that returns the estimated value of some point using k
# nearest neighbors with Euclidean metric
knn <- function(x, y, xseq, k) {
    if (k < length(x)) {
        dmat <- rdist(x, xseq)
        indices <- order(dmat)[2:(k + 1)]  # If you need to find less than 10 neighbors, it will not tak
        return(mean(y[indices]))
    } else {
        dmat <- rdist(x, xseq)
        indices <- order(dmat)[1:k]
        # If you need to find 10 neighbors, it will take the points itself as a neighbor
        return(mean(y[indices]))
    }
}
# Plot knn function for k = 1,3,10
knn_one <- sapply(X, knn, y = Y, xseq = X, k = 1)
plot(X, Y, main = "k-nearest-neighbor k = 1", xlab = "N = 100")
points(X, knn_one, pch = 4)
legend("bottomright", c("original points", "fitted points"), pch = c(1, 4))
knn_thr <- sapply(X, knn, y = Y, xseq = X, k = 3)
plot(X, Y, main = "k-nearest-neighbor k = 3", xlab = "N = 100")
```

```
points(X, knn_thr, pch = 4)
legend("bottomright", c("original points", "fitted points"), pch = c(1, 4))
knn_ten <- sapply(X, knn, y = Y, xseq = X, k = 10)
plot(X, Y, main = "k-nearest-neighbor k = 10", xlab = "N = 100")
points(X, knn_ten, pch = 4)
legend("bottomright", c("original points", "fitted points"), pch = c(1, 4))
# EPE(pi) and E(EPE(X)) Same idea as before except that I doubled the size of simulation
# for E(EPE(X))
set.seed(123123)
Eps <- matrix(rep(rnorm(100, sd = sqrt(0.1)), 1000), ncol = 1000)  # Firstly I generate random error
X_pre <- (c(1:1000) - 1/2)/1000 * 2 * pi  # The 500 randomly generated number from        UNIF(0, 2*pi)
Simu_Y <- t(matrix(1, nrow = 1000, ncol = 100) * (cos(10 * X_pre) + 2)) + Eps
# This is the model used to fit a prediction data(data_X) with original model constructed
# by Y and X, I use this model to fit predicted value
knn_model <- function(data_X, X, Y, k) {
    fit <- sapply(data_X, knn, y = Y, xseq = X, k = k)
    EPE <- matrix(NA, nrow = length(data_X), ncol = 3)
    for (i in 1:length(data_X)) {
        EPE[i, 1] <- mean((Simu_Y[, i] - fit[i])^2)
        EPE[i, 2] <- mean((Simu_Y[, i] - mean(Simu_Y[, i]))^2)
        EPE[i, 3] <- mean((fit[i] - mean(Simu_Y[, i]))^2)
    }
    # Since X's are draw from uniform distribution, so we can estimate the Expected EPE by
    # taking the average of 500 different EPE
    MeanEPE <- mean(EPE)/(2 * pi)
    var_ratio <- mean(EPE[, 2]/EPE[, 1])
    bias_ratio <- mean(EPE[, 3]/EPE[, 1])
    return(data.frame(Mean_EPE = MeanEPE, var_ratio = var_ratio, bias_ratio = bias_ratio))
}
knn_model(X_pre, X, Y, 1)

##   Mean_EPE var_ratio bias_ratio
## 1  0.02099    0.6226     0.3774

knn_model(X_pre, X, Y, 3)

##   Mean_EPE var_ratio bias_ratio
## 1  0.01511    0.7236     0.2764

knn_model(X_pre, X, Y, 10)

##   Mean_EPE var_ratio bias_ratio
## 1  0.06365    0.2887     0.7113

knn_model(X_pre, X, Y, 20)

##   Mean_EPE var_ratio bias_ratio
## 1  0.06322    0.2893     0.7107

knn_model(X_pre, X, Y, 50)

##   Mean_EPE var_ratio bias_ratio
## 1  0.06299    0.2879     0.7121

plot(rbind(knn_model(X_pre, X, Y, 1)$bias_ratio, knn_model(X_pre, X, Y, 3)$bias_ratio, knn_model(X_pre,
    X, Y, 10)$bias_ratio), type = "b", ylim = c(0, 1), ylab = "", main = "Variance and Bias Ratio Behavi
lines(rbind(knn_model(X_pre, X, Y, 1)$var_ratio, knn_model(X_pre, X, Y, 3)$var_ratio, knn_model(X_pre,
    X, Y, 10)$var_ratio), lty = 4)
legend("topright", c("bias_ratio", "var_ratio"), lty = c(1, 4))
```




```r
# Fit a constant function(The same as fitting a knn with k = 100 since we only have ten
# points in the trainning sample)
knn_model(X_pre, X, Y, 100)

##    Mean_EPE var_ratio bias_ratio
## 1   0.06274    0.2888     0.7112

# Comment: For constant fit, it is equivalent to fit a knn model with 10 points.
# Fit a linear model
fit_linear <- lm(Y ~ X)
predict_linear <- X_pre * fit_linear$coefficients[2] + fit_linear$coefficients[1]
EPE_linear <- matrix(NA, 1000)
for (i in 1:1000) {
    EPE_linear[i] <- mean((predict_linear[i] - Simu_Y[, i])^2)
```

```
}
mean(EPE_linear)/(2 * pi)  # This is the estimated E(EPE(X)) under linear model

## [1] 0.09428

Var_linear <- sum((mean(predict_linear) - predict_linear)^2)
Var_linear

## [1] 1.088

Bias_linear <- sum((colMeans(Simu_Y) - mean(predict_linear))^2)
sqrt(Bias_linear)

## [1] 22.37

# Fit a quadratic function
fit_quadra <- lm(Y ~ X + I(X^2))
predict_quadra <- X_pre^2 * fit_quadra$coefficients[3] + X_pre * fit_quadra$coefficients[2] +
    fit_quadra$coefficients[1]
EPE_quadra <- matrix(NA, 1000)
for (i in 1:1000) {
    EPE_quadra[i] <- mean((predict_quadra[i] - Simu_Y[, i])^2)
}
mean(EPE_quadra)/(2 * pi)  # This is the estimated E(EPE(X)) under quadratic model

## [1] 0.09438

Var_quadra <- sum((mean(predict_quadra) - predict_quadra)^2)
Var_quadra

## [1] 2.195

Bias_quadra <- sum((colMeans(Simu_Y) - mean(predict_quadra))^2)
sqrt(Bias_quadra)

## [1] 22.37
```