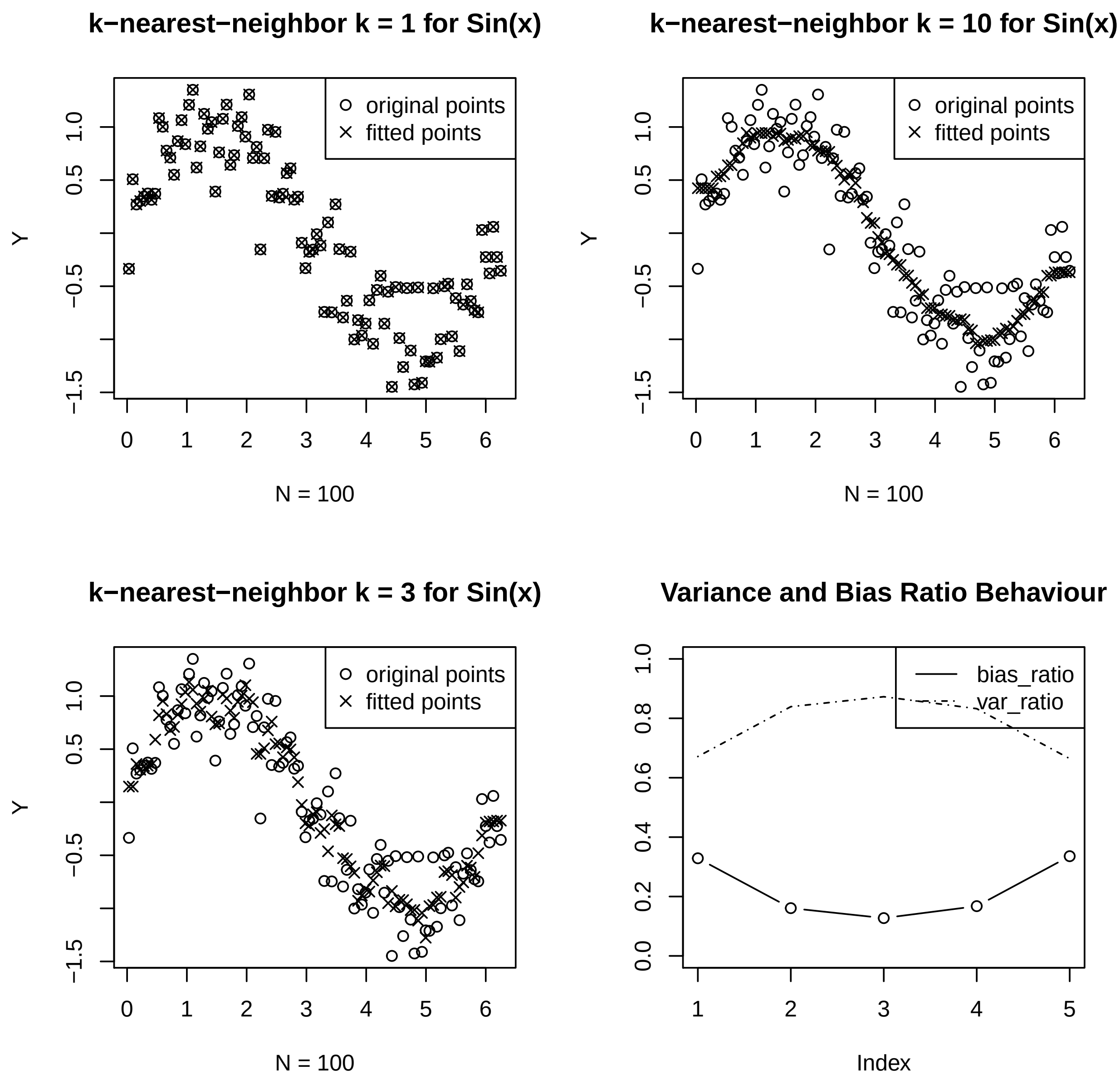
# Problem Two Repeat

Note: I only repeat the problem with N = 100, and it turns out that fitting this model using knn is much better than fitting the previous model using knn. It is reasonable since the core of previous model is cos(10*x) and the core of this model is sin(x); we reduced the osiciallting frequency by changing 10 to 1. Moreover, according to the graph, the bias achieves a local minimum when k = 10, which I think is a sign of good fit since variance here explains more EPE.

```
set.seed(25041)
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 <- sin(X) + 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 for Sin(x)", xlab = "N = 100")
points(X, knn_one, pch = 4)
legend("topright", 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 for Sin(x)", xlab = "N = 100")
points(X, knn_thr, pch = 4)
legend("topright", c("original points", "fitted points"), pch = c(1, 4))
knn_ten <- sapply(X, knn, y = Y, xseq = X, k = 10)
```

```r
plot(X, Y, main = "k-nearest-neighbor k = 10 for Sin(x)", xlab = "N = 100")
points(X, knn_ten, pch = 4)
legend("topright", 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(12345)
Eps <- matrix(rep(rnorm(100, sd = sqrt(0.1)), 1000), ncol = 1000)
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) * (sin(X_pre))) + Eps
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[, 1])/(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   0.0354    0.6712     0.3288

knn_model(X_pre, X, Y, 3)

##   Mean_EPE var_ratio bias_ratio
## 1  0.02473    0.8391     0.1609

knn_model(X_pre, X, Y, 10)

##   Mean_EPE var_ratio bias_ratio
## 1  0.02297    0.8727     0.1273

knn_model(X_pre, X, Y, 20)

##   Mean_EPE var_ratio bias_ratio
## 1  0.02594    0.8324     0.1676

knn_model(X_pre, X, Y, 50)

##   Mean_EPE var_ratio bias_ratio
## 1  0.03402    0.6642     0.3358

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)$bias_ratio, knn_model(X_pre, X, Y, 20)$bias_ratio, knn_model(X_pre, X, Y, 50)$bias_ratio),
    type = "b", ylim = c(0, 1), ylab = "", main = "Variance and Bias Ratio Behaviour")
lines(rbind(knn_model(X_pre, X, Y, 1)$var_ratio, knn_model(X_pre, X, Y, 3)$var_ratio, knn_model(X_pre,
    X, Y, 10)$var_ratio, knn_model(X_pre, X, Y, 20)$var_ratio, knn_model(X_pre, X, Y, 50)$var_ratio),
    lty = 4)
legend("topright", c("bias_ratio", "var_ratio"), lty = c(1, 4))
```




```
# 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   0.1005    0.3318     0.6682
```

```
# 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] 0.05232

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

## [1] 341.6

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

## [1] 22.55

# 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] 0.05243

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

## [1] 342.3

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

## [1] 22.55
```