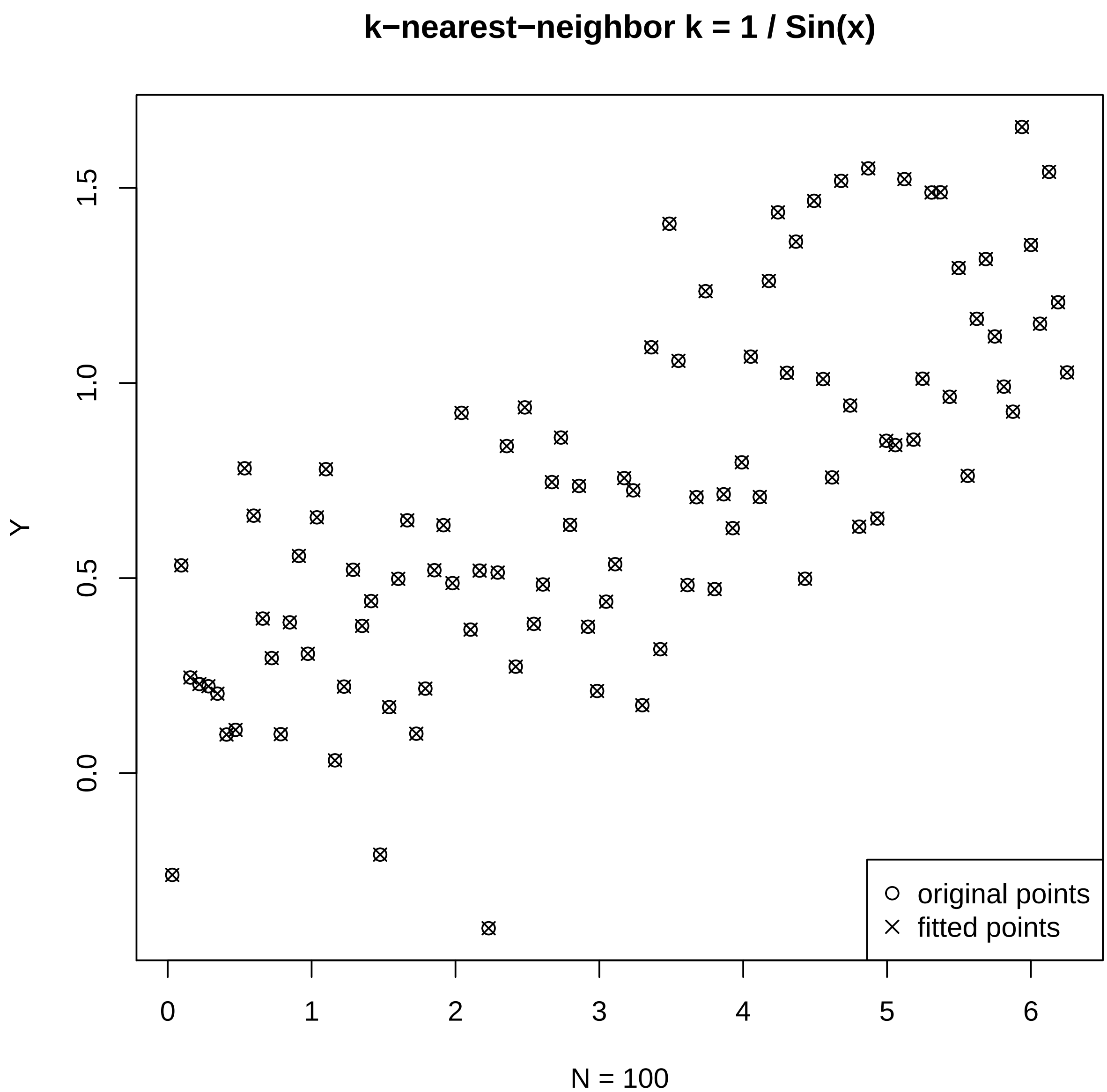
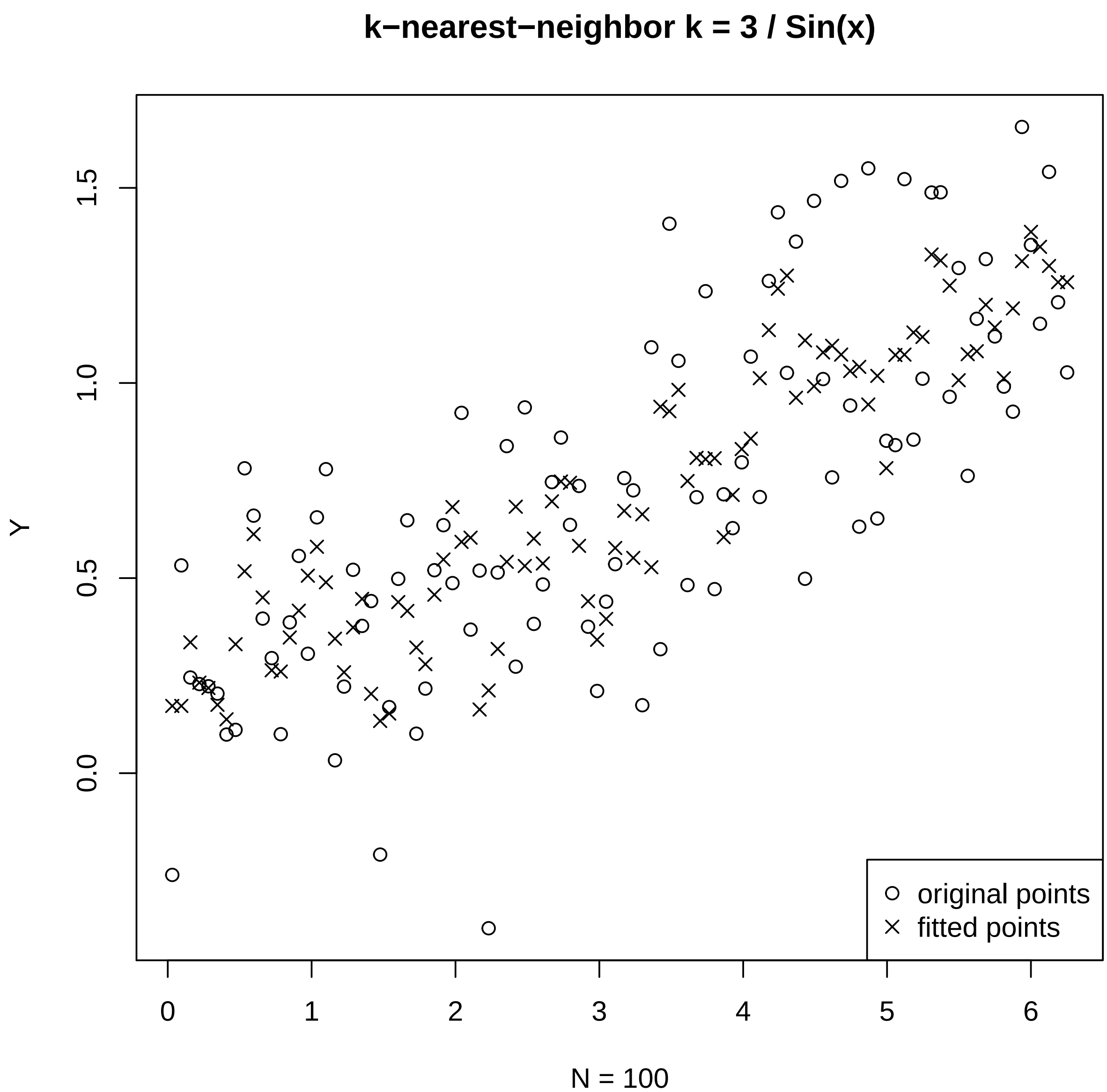
Problem Two Repeat Again

This model has a better fit using knn and linear model, which is what we had expected since we don't have oscillating function here. We can easily interpret a trend of data so that the fit would be more accurate. It is demonstrated by our code below that a large portion of EPE comes from variance, from some perspective, it reflects that we have pretty good fit for the data.

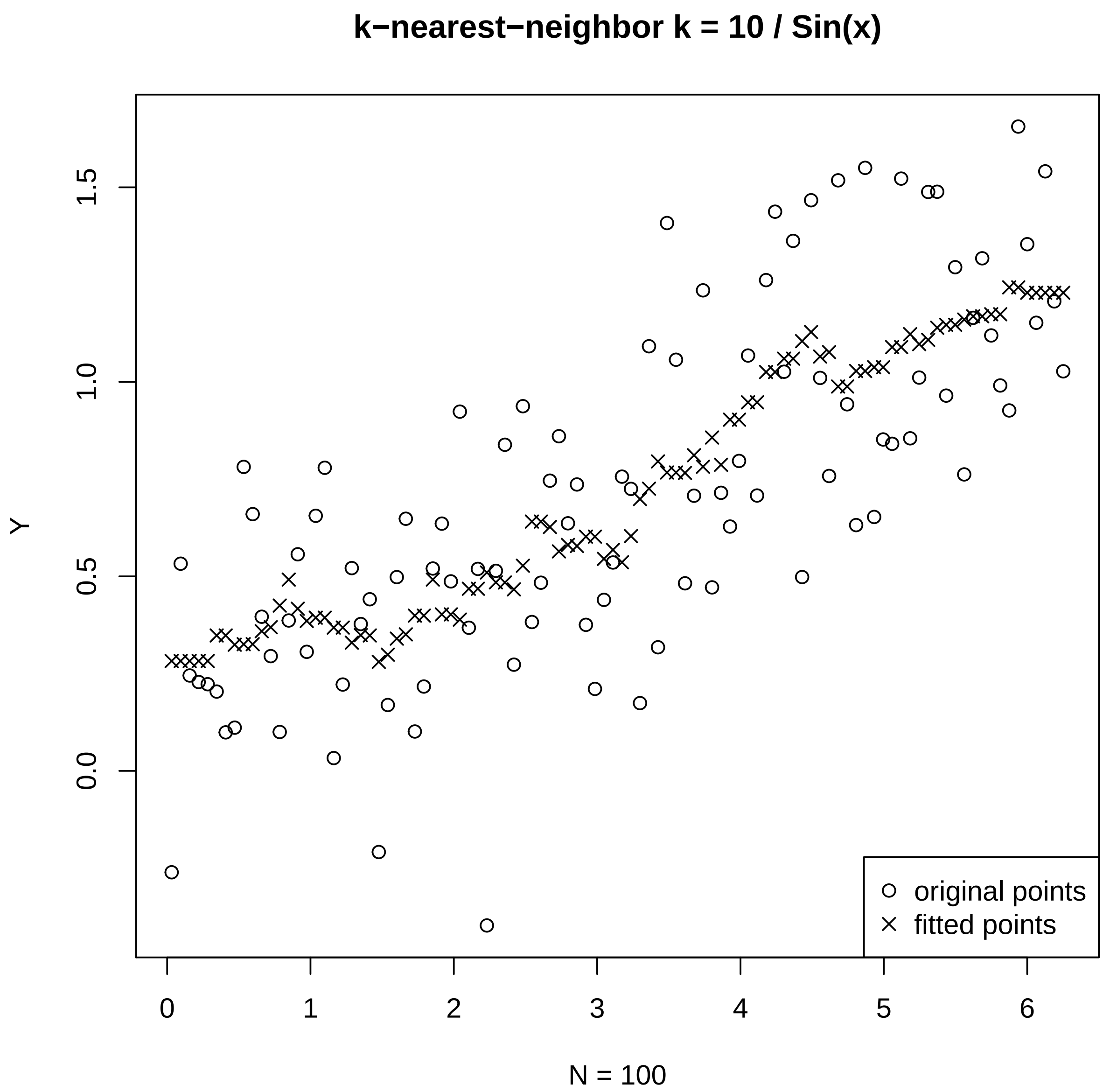
```
set.seed(25041)
require(FNN)

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

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'
##
## äÿÑåĹŮåŕźèśqèćńåśŔèŤ¡äžĘfrom 'package:base':
##
##     backsolve, forwardsolve
##
## Loading required package:  maps

X <- (c(1:100) - 1/2)/100 * 2 * pi
Y <- X * 0.2 + 0.1 + 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 / Sin(x)", 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 / Sin(x)", xlab = "N = 100")
points(X, knn_thr, pch = 4)
legend("bottomright", c("original points", "fitted points"), pch = c(1, 4))
```




```r
knn_ten <- sapply(X, knn, y = Y, xseq = X, k = 10)
plot(X, Y, main = "k-nearest-neighbor k = 10 / Sin(x)", xlab = "N = 100")
points(X, knn_ten, pch = 4)
legend("bottomright", c("original points", "fitted points"), pch = c(1, 4))
```




```r
# 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) * (0.2 * X_pre + 0.1)) + Eps
# The model I use is as follows
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)
```

```r
    }
    # 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  0.01931     0.639      0.361
```

```r
knn_model(X_pre, X, Y, 3)
```

```
##   Mean_EPE var_ratio bias_ratio
## 1  0.01223    0.8394     0.1606
```

```r
knn_model(X_pre, X, Y, 10)
```

```
##   Mean_EPE var_ratio bias_ratio
## 1  0.01055    0.9216    0.07842
```

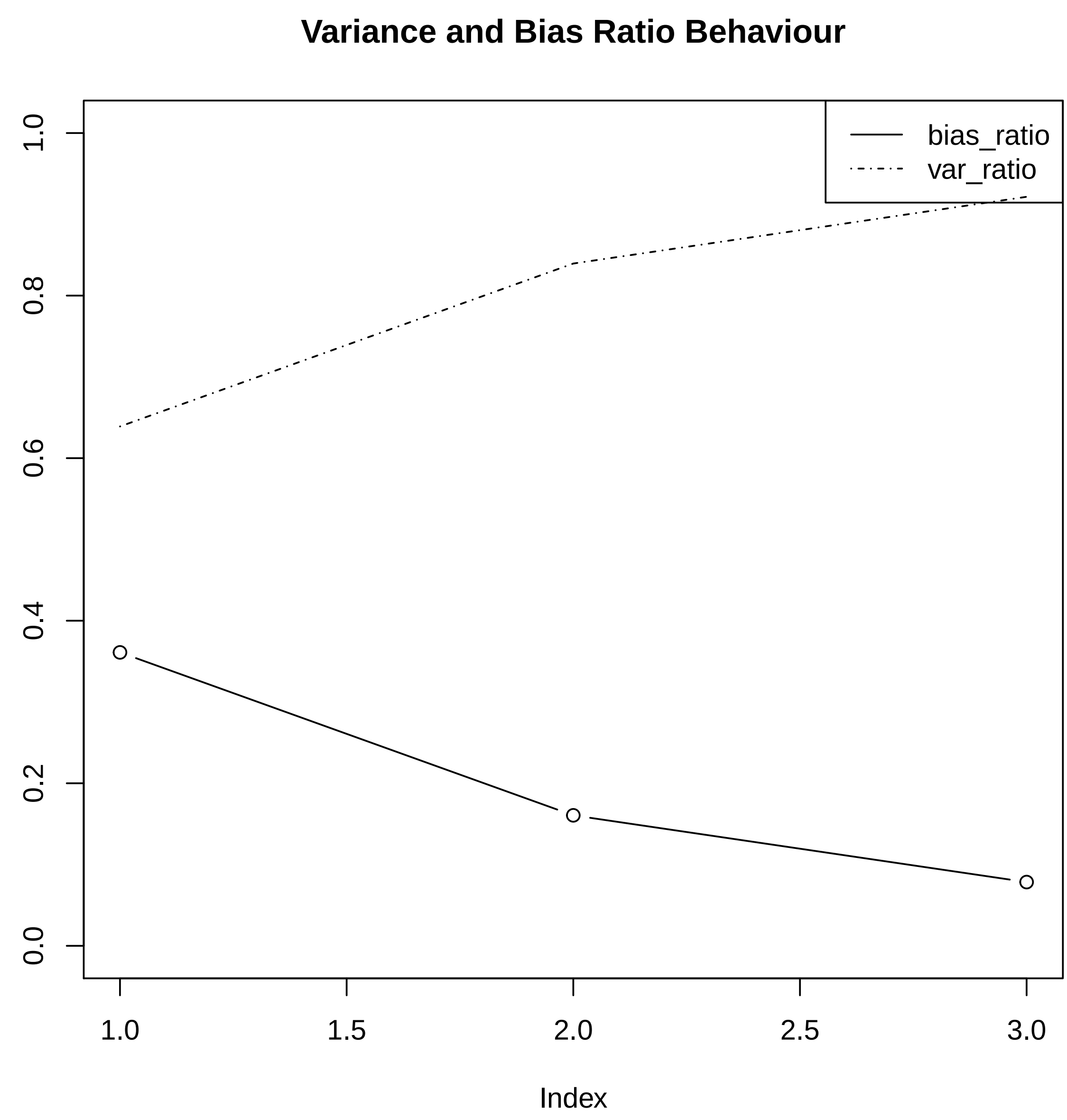
```r
knn_model(X_pre, X, Y, 20)
```

```
##   Mean_EPE var_ratio bias_ratio
## 1  0.01043    0.9324    0.06765
```

```r
knn_model(X_pre, X, Y, 50)
```

```
##   Mean_EPE var_ratio bias_ratio
## 1  0.01163    0.8714     0.1286
```

```r
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))
```




```
# 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  0.02365    0.5391     0.4609
```

```
# 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] 0.0147

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

## [1] 108.7

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

## [1] 11.49

# 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] 0.01481

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

## [1] 109.5

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

## [1] 11.49
```