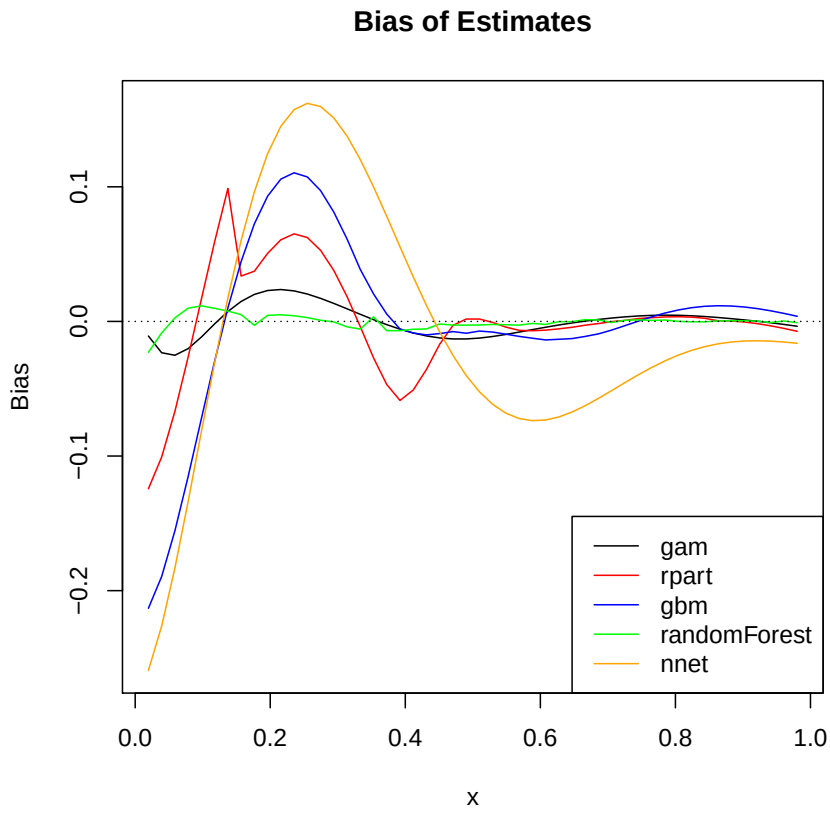
STAT7400 HW9, 2017

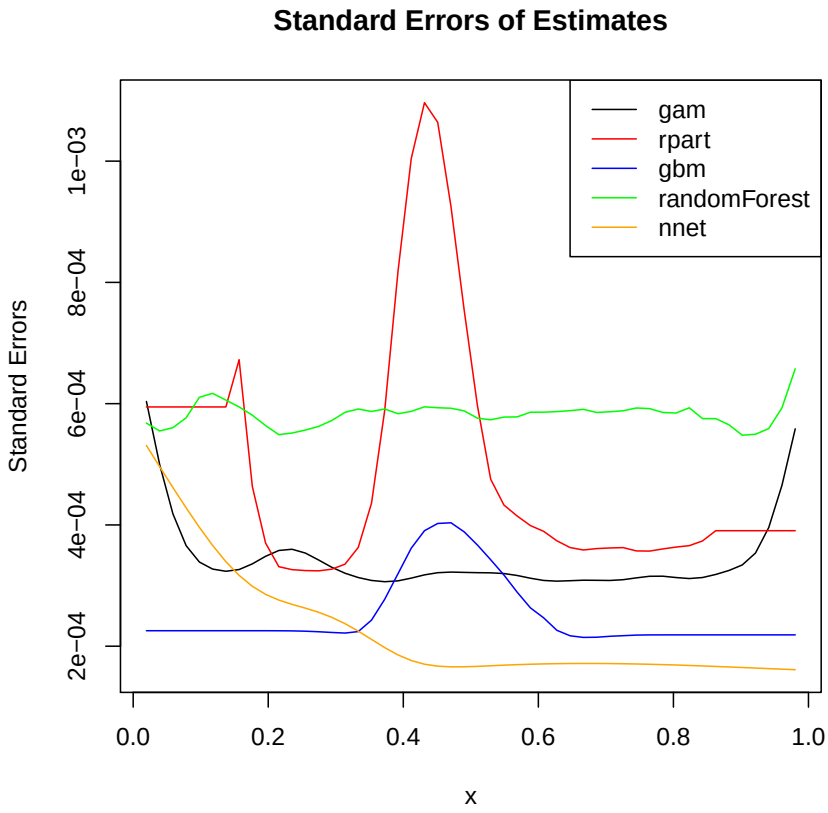
Yiheng Liu

Apr 9, 2017

Problem 1

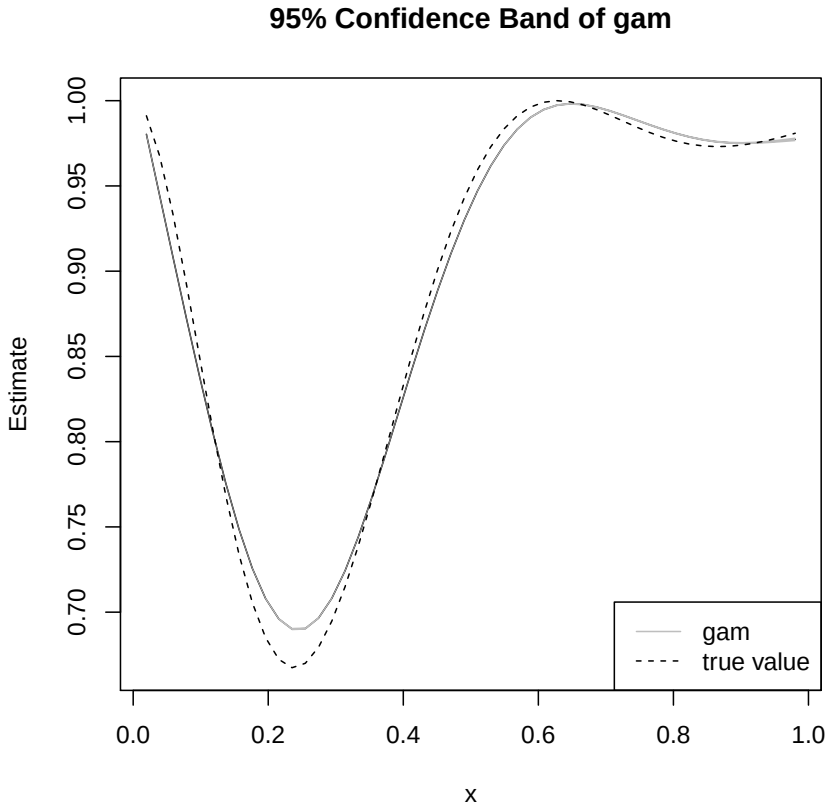
(a) By Monte Carlo simulation of 10000 times, the bias and standard error of the estimates at each x_i are given below.





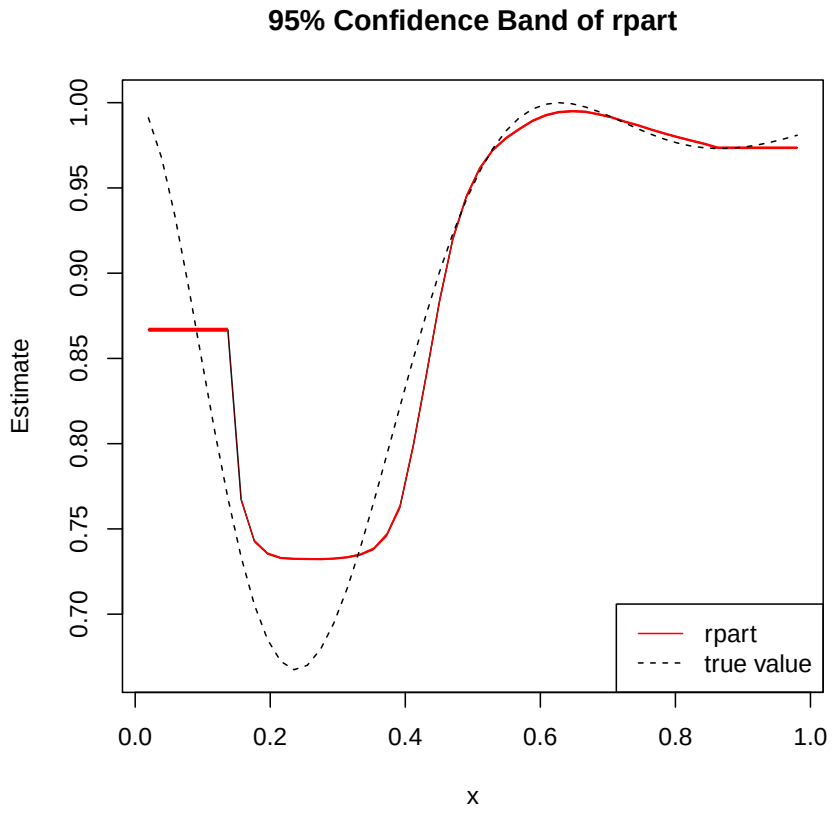
As we can see, there is a tradeoff between bias and standard error for different methods. For example, `nnet` has a large bias but relatively small standard error. On the contrary, `randomForest` has a large standard error but small bias.

To better visualize the performance of each method, a 95% confidence band of each method is given below.



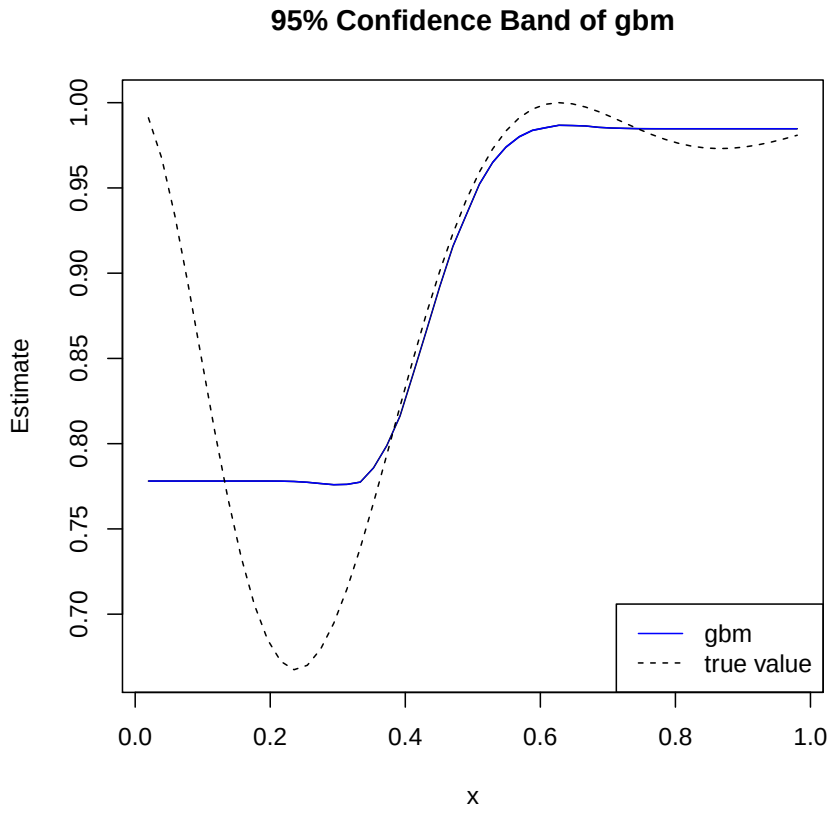

```
> fit.gam.se
```

```
 [1] 0.0006036186 0.0004995894 0.0004182980 0.0003658148 0.0003387883
 [6] 0.0003271902 0.0003235826 0.0003264654 0.0003358002 0.0003482775
[11] 0.0003578892 0.0003599011 0.0003536321 0.0003422366 0.0003301757
[16] 0.0003203522 0.0003132401 0.0003084399 0.0003063472 0.0003077871
[21] 0.0003123079 0.0003176384 0.0003212415 0.0003222395 0.0003217290
[26] 0.0003212578 0.0003210415 0.0003198442 0.0003166322 0.0003121166
[31] 0.0003084541 0.0003072598 0.0003080268 0.0003088574 0.0003086824
[36] 0.0003084122 0.0003097117 0.0003127012 0.0003153395 0.0003154523
[41] 0.0003132466 0.0003115732 0.0003132872 0.0003183770 0.0003249242
[46] 0.0003340209 0.0003536869 0.0003956624 0.0004655435 0.0005584702
```



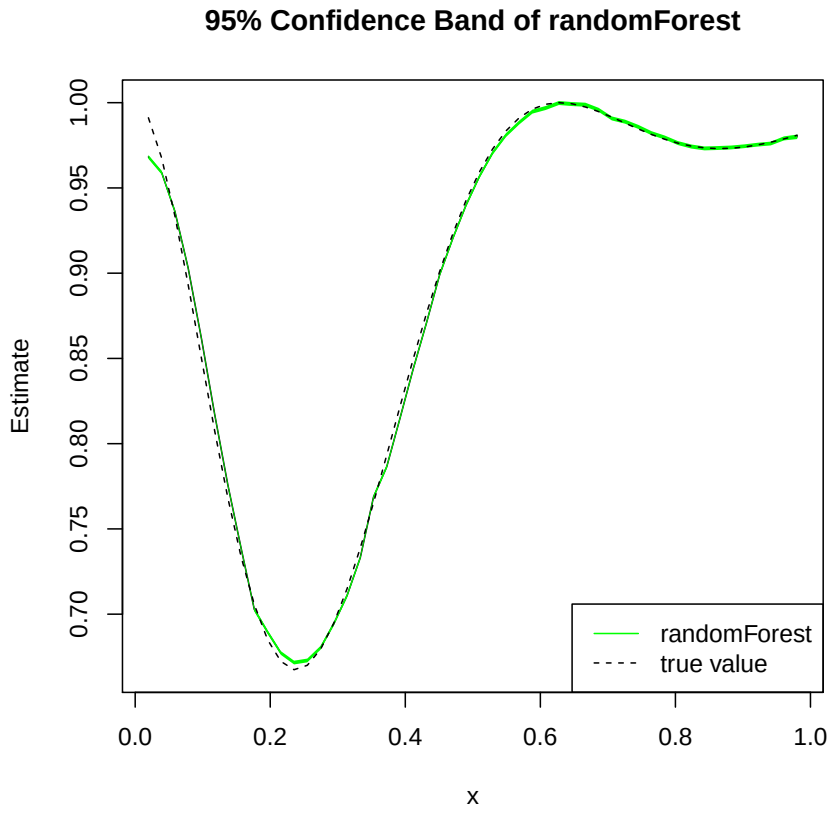

```
> fit.rp.se

 [1] 0.0005945781 0.0005945781 0.0005945781 0.0005945781 0.0005945781
 [6] 0.0005945781 0.0005945781 0.0006724089 0.0004633862 0.0003700611
[11] 0.0003311089 0.0003262605 0.0003247654 0.0003242603 0.0003269851
[16] 0.0003353656 0.0003629542 0.0004358950 0.0005894914 0.0008181651
[21] 0.0010042556 0.0010963772 0.0010639565 0.0009251416 0.0007539628
[26] 0.0005965494 0.0004754026 0.0004328447 0.0004147131 0.0003989502
[31] 0.0003894572 0.0003737980 0.0003626757 0.0003587016 0.0003609929
[36] 0.0003620229 0.0003626029 0.0003571695 0.0003568710 0.0003604016
[41] 0.0003634959 0.0003658254 0.0003736404 0.0003904854 0.0003904854
[46] 0.0003904854 0.0003904854 0.0003904854 0.0003904854 0.0003904854
```



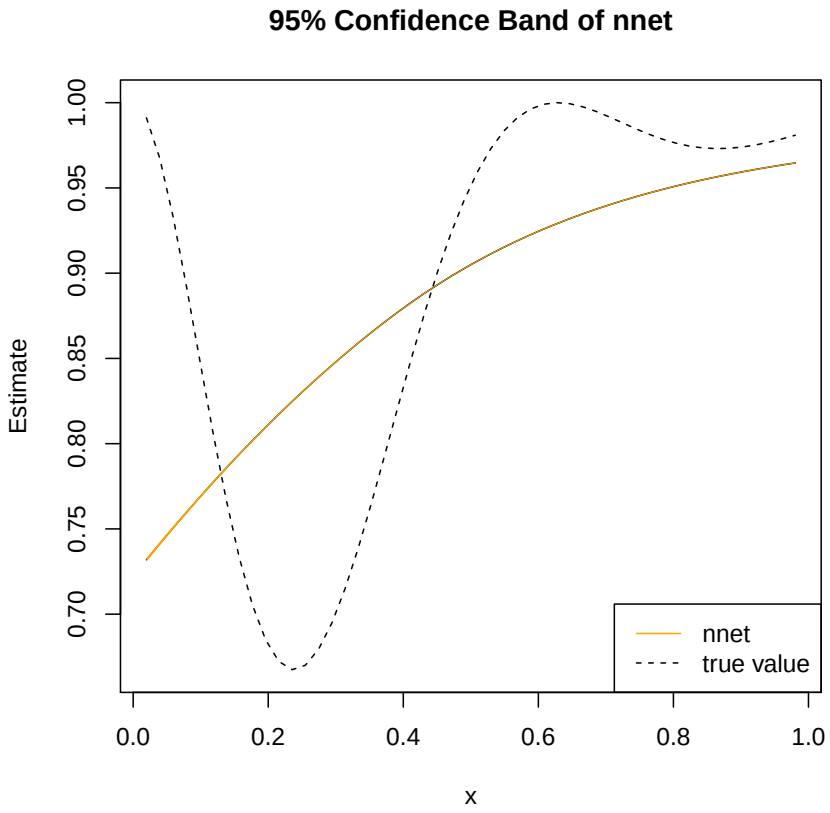

```
> fit.gbm.se

 [1] 0.0002255236 0.0002255236 0.0002255236 0.0002255236 0.0002255236
 [6] 0.0002255236 0.0002255236 0.0002255236 0.0002255236 0.0002255236
[11] 0.0002254810 0.0002252734 0.0002247495 0.0002238417 0.0002226695
[16] 0.0002216973 0.0002241116 0.0002430596 0.0002775364 0.0003195558
[21] 0.0003616528 0.0003904743 0.0004024284 0.0004035652 0.0003885649
[26] 0.0003668121 0.0003425993 0.0003174072 0.0002891010 0.0002632636
[31] 0.0002467704 0.0002261628 0.0002171830 0.0002145004 0.0002147986
[36] 0.0002163664 0.0002173341 0.0002183389 0.0002186100 0.0002186745
[41] 0.0002186811 0.0002186811 0.0002186811 0.0002186811 0.0002186811
[46] 0.0002186811 0.0002186811 0.0002186811 0.0002186811 0.0002186811
```




```
> fit.rf.se

 [1] 0.0005678695 0.0005549407 0.0005604254 0.0005767460 0.0006104868
 [6] 0.0006170185 0.0006059165 0.0005945881 0.0005809481 0.0005638427
[11] 0.0005488813 0.0005515523 0.0005564883 0.0005624065 0.0005722519
[16] 0.0005857188 0.0005912003 0.0005871103 0.0005911351 0.0005834272
[21] 0.0005872718 0.0005947821 0.0005932224 0.0005924741 0.0005880810
[26] 0.0005761243 0.0005734173 0.0005778203 0.0005781819 0.0005858717
[31] 0.0005859446 0.0005869097 0.0005884116 0.0005906476 0.0005855141
[36] 0.0005866750 0.0005882381 0.0005928336 0.0005918551 0.0005854727
[41] 0.0005845287 0.0005932117 0.0005753164 0.0005752913 0.0005650773
[46] 0.0005479484 0.0005495855 0.0005589171 0.0005931583 0.0006575088
```

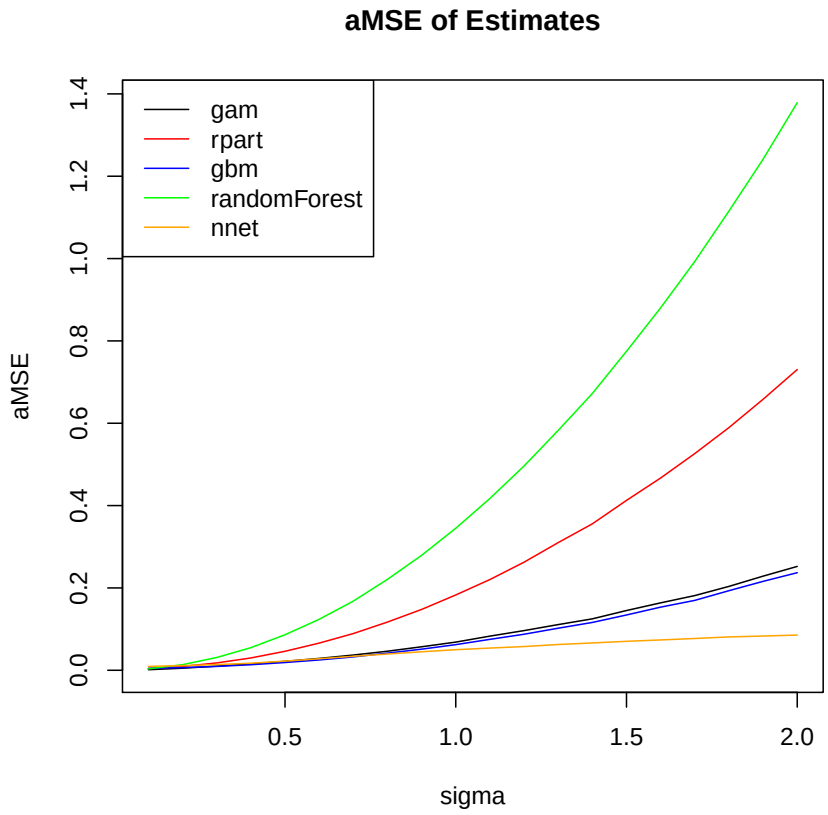



```
> fit.nnet.se

 [1] 0.0005310145 0.0004960510 0.0004616388 0.0004281329 0.0003960708
 [6] 0.0003662084 0.0003394792 0.0003168199 0.0002988495 0.0002855309
[11] 0.0002760470 0.0002690014 0.0002628012 0.0002560170 0.0002476313
[16] 0.0002371800 0.0002248018 0.0002112168 0.0001976556 0.0001856227
[21] 0.0001763590 0.0001703085 0.0001670990 0.0001659452 0.0001660453
[26] 0.0001667781 0.0001677397 0.0001687034 0.0001695603 0.0001702678
[31] 0.0001708155 0.0001712057 0.0001714455 0.0001715433 0.0001715077
[36] 0.0001713474 0.0001710708 0.0001706863 0.0001702027 0.0001696291
[41] 0.0001689748 0.0001682501 0.0001674652 0.0001666313 0.0001657596
[46] 0.0001648618 0.0001639495 0.0001630346 0.0001621288 0.0001612438
```

As we can see from the graphs, `randomForest` has the best performance when σ is small. Even though `nnet` has a relatively small standard error, which means the estimator is more stable, its performance is not good enough.

(b) Set the simulation sample size to be 10000 and let σ range from 0.1 to 2, the graph of integrated mean square error is given below.



As σ increases, the integrated mean square error will also increase. For this particular problem, `randomForest` is very sensitive to large σ (noise). On the contrary, though `nnet` is not as accurate as `randomForest` when σ is small, it is pretty stable when σ becomes large.

Appendix

```
# Problem 1 (a)
library(mgcv)
library(rpart)
library(gbm)
library(randomForest)
library(nnet)
n <- 50
sigma <- 0.1
N <- 10^4
i <- c(1:n)
x <- i/(n + 1)
```

```
m <- function(x) {
    1 - sin(5 * x)^2 * exp(-4 * x)
}

m.x <- m(x)

# Matrices to store the fitted values
fit.gam <- matrix(NA, N, n)
fit.rp <- matrix(NA, N, n)
fit.gbm <- matrix(NA, N, n)
fit.rf <- matrix(NA, N, n)
fit.nnet <- matrix(NA, N, n)


set.seed(100)
for (j in 1:N) {
    eps <- rnorm(n, 0, sigma)
    y <- m.x + eps
    data <- as.data.frame(cbind(y, x))
    fit.gam[j, ] <- gam(y ~ s(x))$fitted
    fit.rp[j, ] <- predict(rpart(y ~ x), data)
    fit.gbm[j, ] <- predict(gbm(y ~ x, n.trees = 5000), data, n.trees = 5000)
    fit.rf[j, ] <- predict(randomForest(y ~ x), data)
    fit.nnet[j, ] <- predict(nnet(y ~ x, size = 10, entropy = TRUE,
                                  decay = 0.001, maxit = 300), data)
}

# Calculate average of estimator
fit.gam.avg <- colMeans(fit.gam)
fit.rp.avg <- colMeans(fit.rp)
fit.gbm.avg <- colMeans(fit.gbm)
fit.rf.avg <- colMeans(fit.rf)
fit.nnet.avg <- colMeans(fit.nnet)


# Calculate bias
fit.gam.bias <- fit.gam.avg - m.x
fit.rp.bias <- fit.rp.avg - m.x
fit.gbm.bias <- fit.gbm.avg - m.x
fit.rf.bias <- fit.rf.avg - m.x
fit.nnet.bias <- fit.nnet.avg - m.x

# Plot bias
plot(x, fit.gam.bias, ylab = "Bias", main = "Bias of Estimates",
```

```
     ylim = range(c(fit.gam.bias, fit.rp.bias, fit.gbm.bias, fit.rf.bias,
                    fit.nnet.bias)), type = "l")
lines(x, fit.rp.bias, col = "red")
lines(x, fit.gbm.bias, col = "blue")
lines(x, fit.rf.bias, col = "green")
lines(x, fit.nnet.bias, col = "orange")
abline(0, 0, lty = 3)
legend("bottomright", legend = c("gam", "rpart", "gbm", "randomForest", "nnet"),
       lty = 1, col = c("black", "red", "blue", "green", "orange"))

# Calcualte standard error
fit.gam.se <- apply(fit.gam, 2, sd)/sqrt(N)
fit.rp.se <- apply(fit.rp, 2, sd)/sqrt(N)
fit.gbm.se <- apply(fit.gbm, 2, sd)/sqrt(N)
fit.rf.se <- apply(fit.rf, 2, sd)/sqrt(N)
fit.nnet.se <- apply(fit.nnet, 2, sd)/sqrt(N)

# Plot standard error
plot(x, fit.gam.se, ylab = "Standard Errors",
     main = "Standard Errors of Estimates",
     ylim = range(c(fit.gam.se, fit.rp.se, fit.gbm.se, fit.rf.se, fit.nnet.se)),
     type = "l")
lines(x, fit.rp.se, col = "red")
lines(x, fit.gbm.se, col = "blue")
lines(x, fit.rf.se, col = "green")
lines(x, fit.nnet.se, col = "orange")
legend("topright", legend = c("gam", "rpart", "gbm", "randomForest", "nnet"),
       lty = 1, col = c("black", "red", "blue", "green", "orange"))

# Plot 95% confidence bands gam
plot(x, fit.gam.avg, ylab = "Estimate", main = "95% Confidence Band of gam",
     ylim = range(m.x), type = "l")
polygon(c(x, rev(x)), c(fit.gam.avg - 2 * fit.gam.se,
                        rev(fit.gam.avg + 2 * fit.gam.se)), col = "grey",
        border = NA)
lines(x, m.x, lty = 2)
legend("bottomright", legend = c("gam", "true value"), lty = c(1, 2),
       col = c("grey", "black"))

# rpart
plot(x, fit.rp.avg, ylab = "Estimate", main = "95% Confidence Band of rpart",
     ylim = range(m.x), type = "l")
polygon(c(x, rev(x)), c(fit.rp.avg - 2 * fit.rp.se,
                        rev(fit.rp.avg + 2 * fit.rp.se)), col = "red",
        border = NA)
```

```
lines(x, m.x, lty = 2)
legend("bottomright", legend = c("rpart", "true value"), lty = c(1, 2),
       col = c("red", "black"))

# gbm
plot(x, fit.gbm.avg, ylab = "Estimate", main = "95% Confidence Band of gbm",
     ylim = range(m.x), type = "l")
polygon(c(x, rev(x)), c(fit.gbm.avg - 2 * fit.gbm.se,
                        rev(fit.gbm.avg + 2 * fit.gbm.se)), col = "blue",
        border = NA)
lines(x, m.x, lty = 2)
legend("bottomright", legend = c("gbm", "true value"), lty = c(1, 2),
       col = c("blue", "black"))

# randomForest
plot(x, fit.rf.avg, ylab = "Estimate",
     main = "95% Confidence Band of randomForest", ylim = range(m.x),
     type = "l")
polygon(c(x, rev(x)), c(fit.rf.avg - 2 * fit.rf.se,
                        rev(fit.rf.avg + 2 * fit.rf.se)), col = "green",
        border = NA)
lines(x, m.x, lty = 2)
legend("bottomright", legend = c("randomForest", "true value"), lty = c(1, 2),
       col = c("green", "black"))

# nnet
plot(x, fit.nnet.avg, ylab = "Estimate", main = "95% Confidence Band of nnet",
     ylim = range(m.x), type = "l")
polygon(c(x, rev(x)), c(fit.nnet.avg - 2 * fit.nnet.se,
                        rev(fit.nnet.avg + 2 * fit.nnet.se)), col = "orange",
        border = NA)
lines(x, m.x, lty = 2)
legend("bottomright", legend = c("nnet", "true value"), lty = c(1, 2),
       col = c("orange", "black"))

# (b)
sigma <- seq(0.1, 2, 0.1)
N <- 10^4
fit.gam.amse <- rep(NA, length(sigma))
fit.rp.amse <- rep(NA, length(sigma))
fit.gbm.amse <- rep(NA, length(sigma))
fit.rf.amse <- rep(NA, length(sigma))
fit.nnet.amse <- rep(NA, length(sigma))

# Matrices to store the fitted values
```

```
fit.gam <- matrix(NA, N, n)
fit.rp <- matrix(NA, N, n)
fit.gbm <- matrix(NA, N, n)
fit.rf <- matrix(NA, N, n)
fit.nnet <- matrix(NA, N, n)
m.true <- matrix(m.x, N, n, byrow = TRUE)

# Calcualte aMSE
set.seed(100)
for (i in 1:length(sigma)) {
    for (j in 1:N) {
        eps <- rnorm(n, 0, sigma[i])
        y <- m.x + eps
        data <- as.data.frame(cbind(y, x))
        fit.gam[j, ] <- gam(y ~ s(x))$fitted
        fit.rp[j, ] <- predict(rpart(y ~ x), data)
        fit.gbm[j, ] <- predict(gbm(y ~ x, n.trees = 5000), data,
                                n.trees = 5000)
        fit.rf[j, ] <- predict(randomForest(y ~ x), data)
        fit.nnet[j, ] <- predict(nnet(y ~ x, size = 10, entropy = TRUE,
                                      decay = 0.001, maxit = 300), data)
    }
    fit.gam.amse[i] <- mean(colMeans((fit.gam - m.true)^2))
    fit.rp.amse[i] <- mean(colMeans((fit.rp - m.true)^2))
    fit.gbm.amse[i] <- mean(colMeans((fit.gbm - m.true)^2))
    fit.rf.amse[i] <- mean(colMeans((fit.rf - m.true)^2))
    fit.nnet.amse[i] <- mean(colMeans((fit.nnet - m.true)^2))
}

# Plot aMSE
plot(sigma, fit.gam.amse, ylab = "aMSE", main = "aMSE of Estimates",
     ylim = range(c(fit.gam.amse, fit.rp.amse, fit.gbm.amse, fit.rf.amse,
                    fit.nnet.amse)), type = "l")
lines(sigma, fit.rp.amse, col = "red")
lines(sigma, fit.gbm.amse, col = "blue")
lines(sigma, fit.rf.amse, col = "green")
lines(sigma, fit.nnet.amse, col = "orange")
legend("topleft", legend = c("gam", "rpart", "gbm", "randomForest", "nnet"),
       lty = 1, col = c("black", "red", "blue", "green", "orange"))
```