midterm2_2023

Yiran Jia

2023-03-19

Problem 1 Incremental Stagewise Regression

Part A

```r
incrementalStagewise <- function(x, y, eps = 0.01, plot = TRUE) {

    # Get the sample size and number of features
    n <- nrow(x)
    p <- ncol(x)

    # standardize predictors
    x <- scale(x, center = TRUE, scale = TRUE)

    # Initialize the residual
    r <- y

    # set the maximum iteration
    max.ite <- 800

    # Store the coefficient values at each iteration For example, fix the
    # column number to be 1, this stores the value of the initialized
    # coefficient values fix the column number to be 2, this stores the value
    # of the beta after one iteration
    coeffs <- matrix(0, nrow = p, ncol = max.ite + 1)

    # Start iteration
    for (i in 1:max.ite) {

        # Find the predictor most correlated with the residual
        corr <- cor(x, r)
        j <- which.max(abs(corr))

        # Update the coefficient We need to be very careful. By the way we
        # construct the matrix coeffs, at the iteration i, need to not only
        # update the (i+1)th column, we need to update all column after that as
        # well, because we need to keep this record.
        delta.j <- as.numeric(eps * sign(x[, j] %*% r))
        coeffs[j, (i + 1):ncol(coeffs)] <- coeffs[j, i] + delta.j

        # Update the residual
        r <- r - delta.j * x[, j]

        # Break if all the residuals are uncorrelated with all the predictors
        if (all(abs(cor(x, r)) < 1e-10)) {
            break
        }

    }

    # Cut the coefficient matrix to just the number of iteration we run over
    coeffs <- coeffs[, 1:(i + 1)]

    # Draw plot if asked
    if (plot == TRUE) {

        plot(0:i, coeffs[1, ], type = "l", xlab = "Iteration", ylab = "Coefficient Value",
            main = "Forward Stagewise Regression", xlim = c(1, 1300))

        for (j in 2:p) {
            lines(0:i, coeffs[j, ], col = j)
        }

        legend("topright", legend = paste0("X", 1:p), col = 1:p, lty = 1)

    }

}
```

In the textbook, the author set the epsilon to be 0.01. This value should be chosen carefully to make sure we have a balance on the convergence rate and numerical stability. We will show in part B that if the value of epsilon is 0.05, then the convergence is faster; however it is too large, for example, 0.5, the algorithm is very hard to converge and the result is meaningless.

Part B

First we apply our function to synthetic data.

We set epsilon as the defaul value 0.01, and the algorithm run for maximum 800 iterations. For the synthetic data, we will set `n = 200` and `p = 20`, and generating them in the same way as the one in the lecture. And we compare it with the above with lasso we studied in lecture 9.

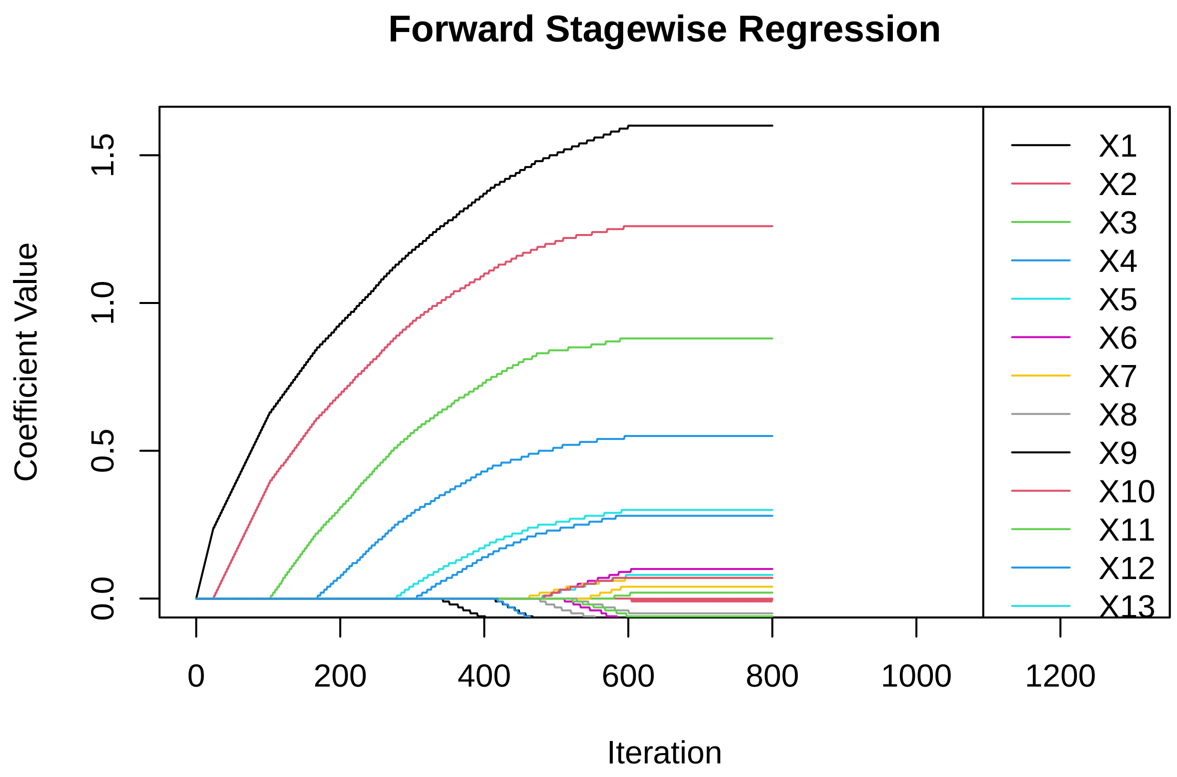
In this configuration, the result from algorithm 3.4 closely aligns with the one from lasso. In the first plot, matching the legend color, we see that the first five are the main features, and the y-axis indicates their coefficient values. In the second plot, we also see that the first five are the the main features, with roughly the same coefficient values as the ones shown in the first plot.

Knowing the truth from the simulation we made, we see that both algorithm 3.4 and lasso performs well and their estimations on the coefficients are reasonable. Both algorithms correctly show the relationship between the features. For example, in both plots, the value of the first coefficient is larger than the second one. And in the true the first feature does have larger value coefficient than the second feature.

Notice the true coefficient of the first feature is 5, but the estimation from both algorithms 3.4 and lasso are about 1.6. This does not mean the algorithms are off, but because we centered and scaled the predictor before applying each of the algorithm. From Math282A lecture 10 centering and scaling, the new estimator equals to the product of the original estimator and the standard deviation of the corresponding predictor.

For example, the true coefficient of the first feature is 5, multiplying 5 by `sd(x[,1])`, we get 1.37, which is the estimator return by both algorithms 3.4 and lasso.

```
# Synthetic data Since we are comparing the algorithm 3.4 with lasso, we need
# to make sure they are using the same data.
set.seed(123)
n = 200
p = 20
x = matrix(runif(n * p), n, p)
y = x %*% c(5, 4, 3, 2, 1, rep(0, p - 5)) + 2 * rnorm(n)
dat = data.frame(x, y)

# Plot and compare with lasso result

# our function already standardize the predictors
incrementalStagewise(x, y, eps = 0.01)
```

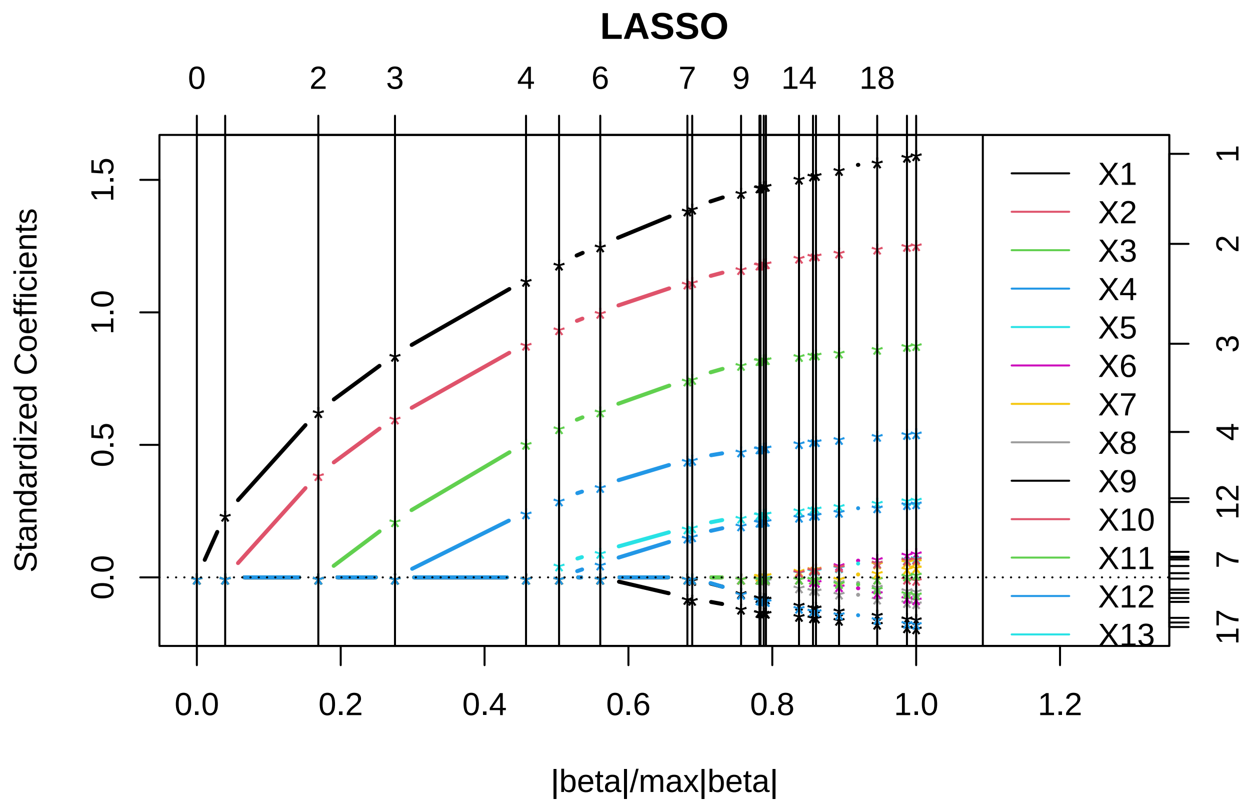


```
require(lars)
```

```
## Loading required package: lars
```

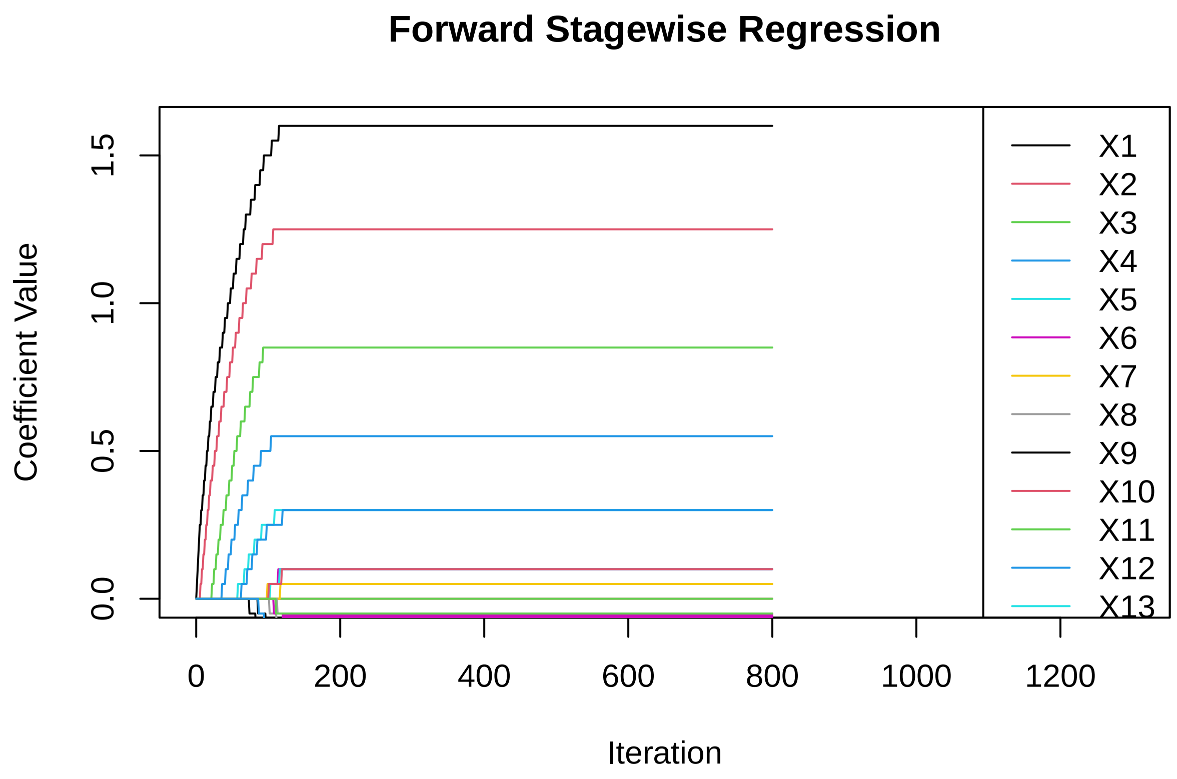
```
## Loaded lars 1.3
```

```
# standardize predictors before fitting lasso
x <- scale(x, center = TRUE, scale = TRUE)
# lars() by default will return standardized coefficients for better comparison
# among the coefficients. However, since we are not standardizing in algorithm
# 3.4, we will set normalize = FALSE in lars().
fit.lasso.1 <- lars(x, y, normalize = FALSE)
plot(fit.lasso.1, col = 1:p, lwd = 2, xlim = c(0, 1.3), lty = 1)
legend("topright", legend = paste0("X", 1:p), col = 1:p, lty = 1)
```

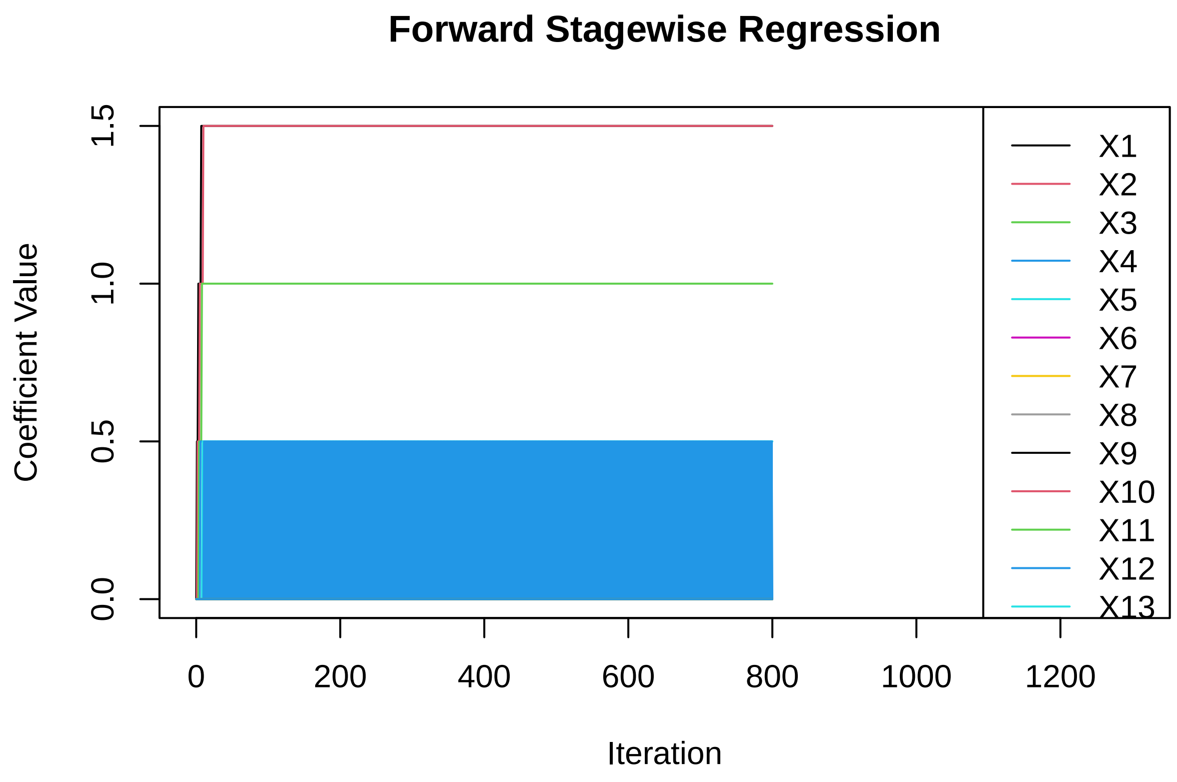


Before we go to the next set up, let's look at what happens if we set epsilon to be 0.5 and 0.1 in algorithm 3.4. We indeed see if the value of epsilon is 0.05, then the convergence is faster; if it is too large, the result is hard to interpret.

```
# Same synthetic data as above Since we are comparing the algorithm 3.4 with
# lasso, we need to make sure they are using the same data.
set.seed(123)
n = 200
p = 20
x = matrix(runif(n * p), n, p)
y = x %*% c(5, 4, 3, 2, 1, rep(0, p - 5)) + 2 * rnorm(n)
dat = data.frame(x, y)

incrementalStagewise(x, y, eps = 0.05)
```

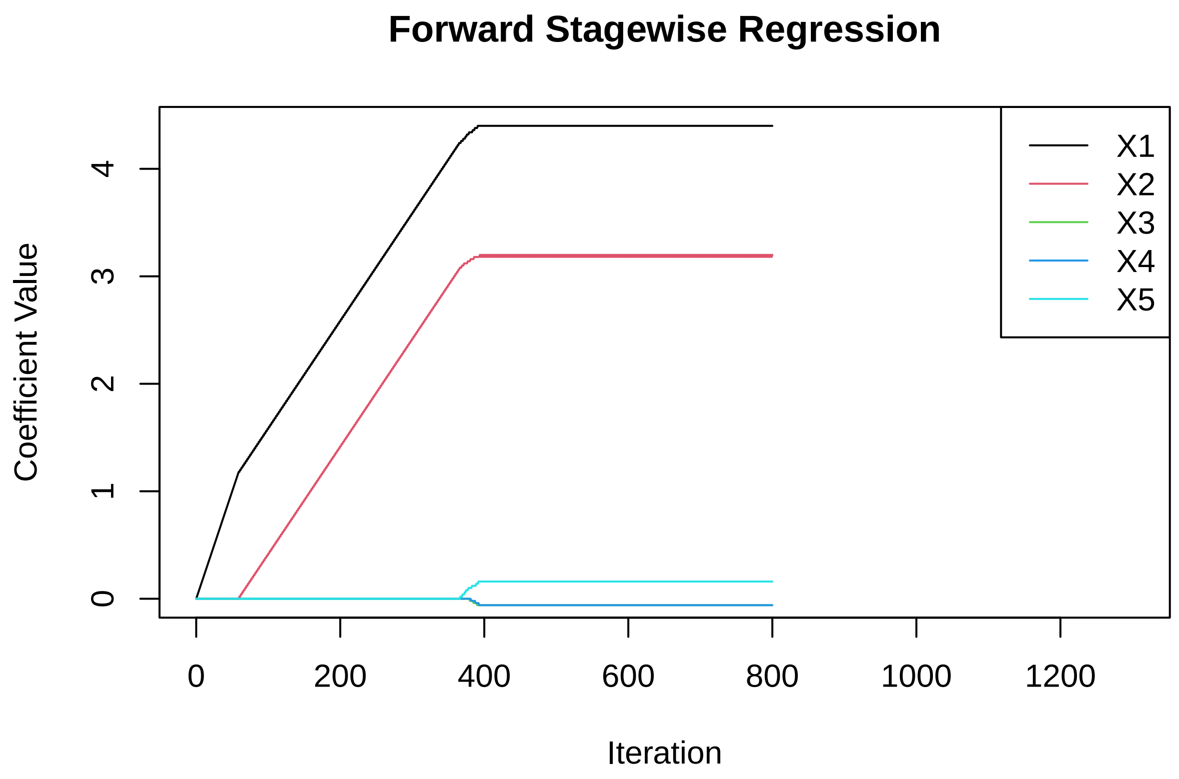


```
incrementalStagewise(x, y, eps = 0.5)
```

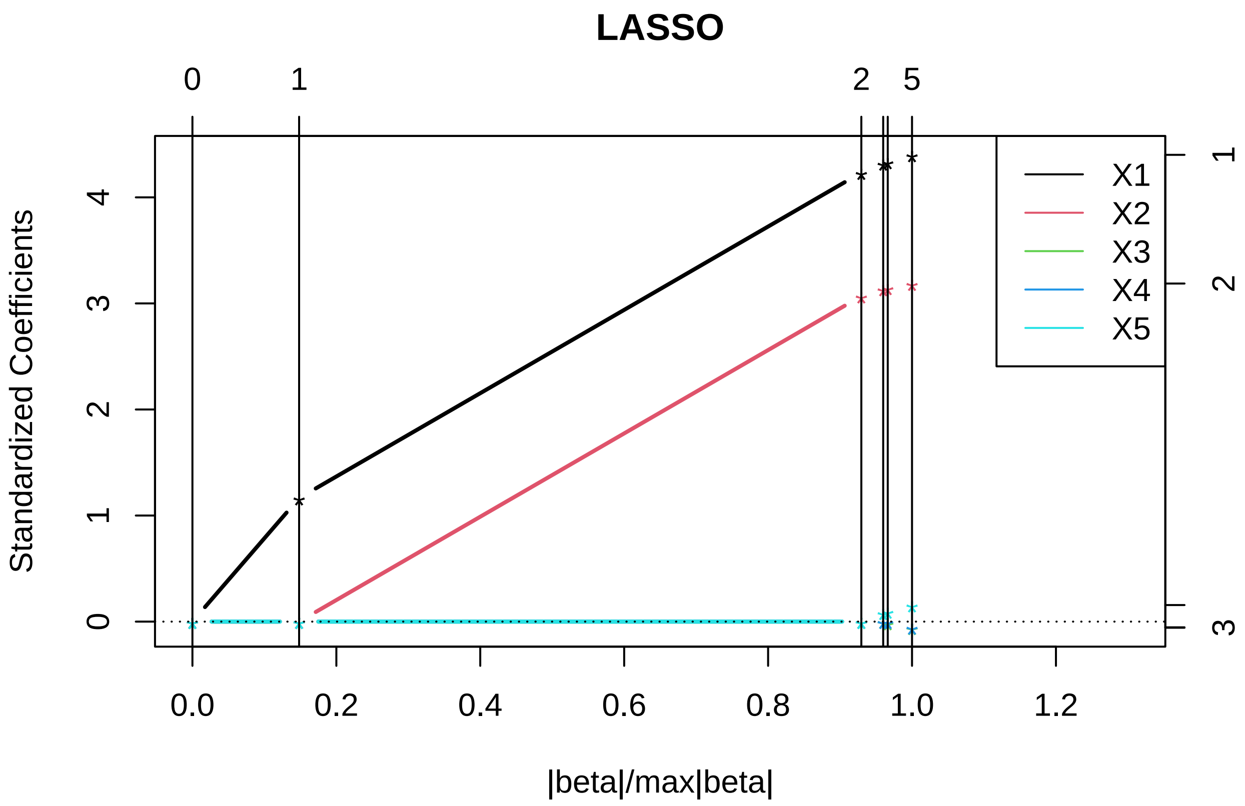


Now let's work with another configuration. Notice the synthetic data is different. Our observation and analysis is similar as the above one. Since it does not hurt too much, we increase the step size to from 0.01 to 0.02 to speed up the convergence. This time, both algorithms' estimation on the coefficient is close to the true values. In addition, lasso does better than algorithm 3.4 since its estimation on the last three coefficietns are zer0, while the ones obtained from algorithm 3.4 has small nonzero value.

```
# Synthetic data Since we are comparing the algorithm 3.4 with lasso, we need
# to make sure they are using the same data.
set.seed(124)
n = 200
p = 5
x = matrix(rnorm(n * p), n, p)
y = x %*% c(5, 3, 0, 0, 0) + rnorm(n)
dat = data.frame(x, y)

# Plot and compare with lasso result our function already standardize the
# predictors
incrementalStagewise(x, y, eps = 0.02)
```

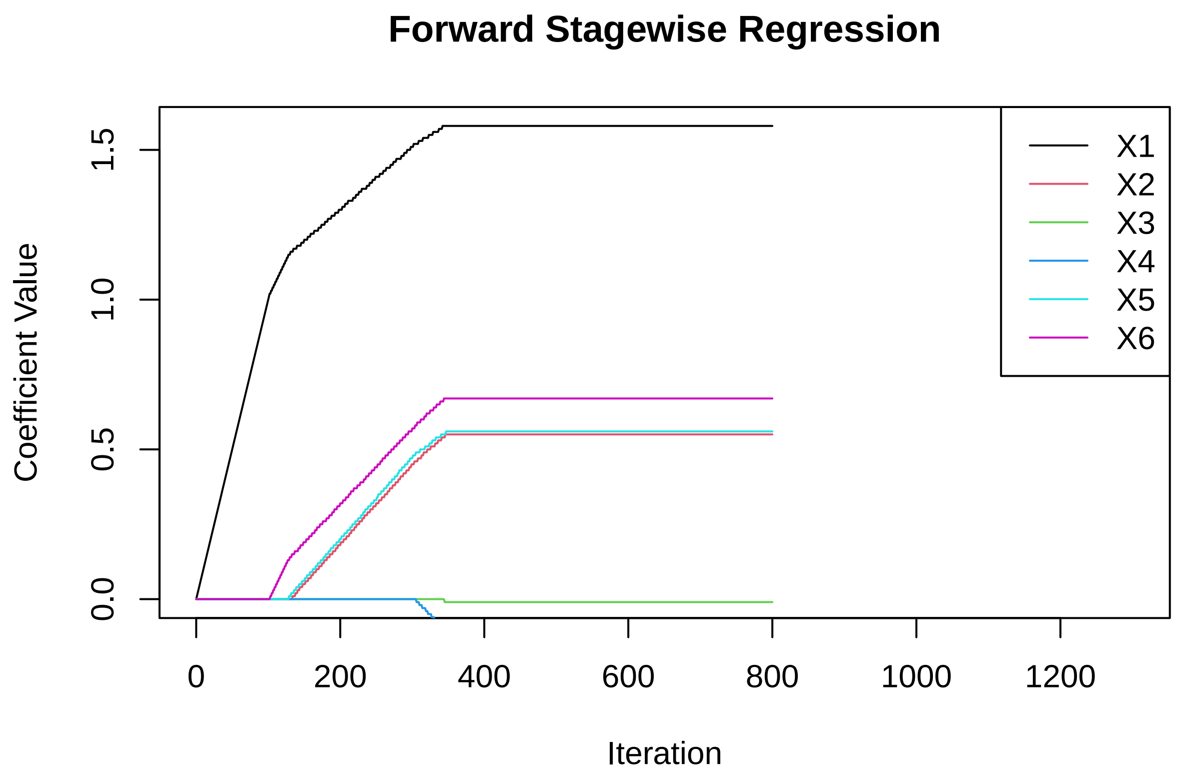


```r
require(lars)
# standardize predictors before fitting lasso
x <- scale(x, center = TRUE, scale = TRUE)
# lars() by default will return standardized coefficients for better comparison
# among the coefficients. However, since we are not standardizing in algorithm
# 3.4, we will set normalize = FALSE in lars().
fit.lasso.1 <- lars(x, y, normalize = FALSE)
plot(fit.lasso.1, col = 1:p, lwd = 2, xlim = c(0, 1.3), lty = 1)
legend("topright", legend = paste0("X", 1:p), col = 1:p, lty = 1)
```

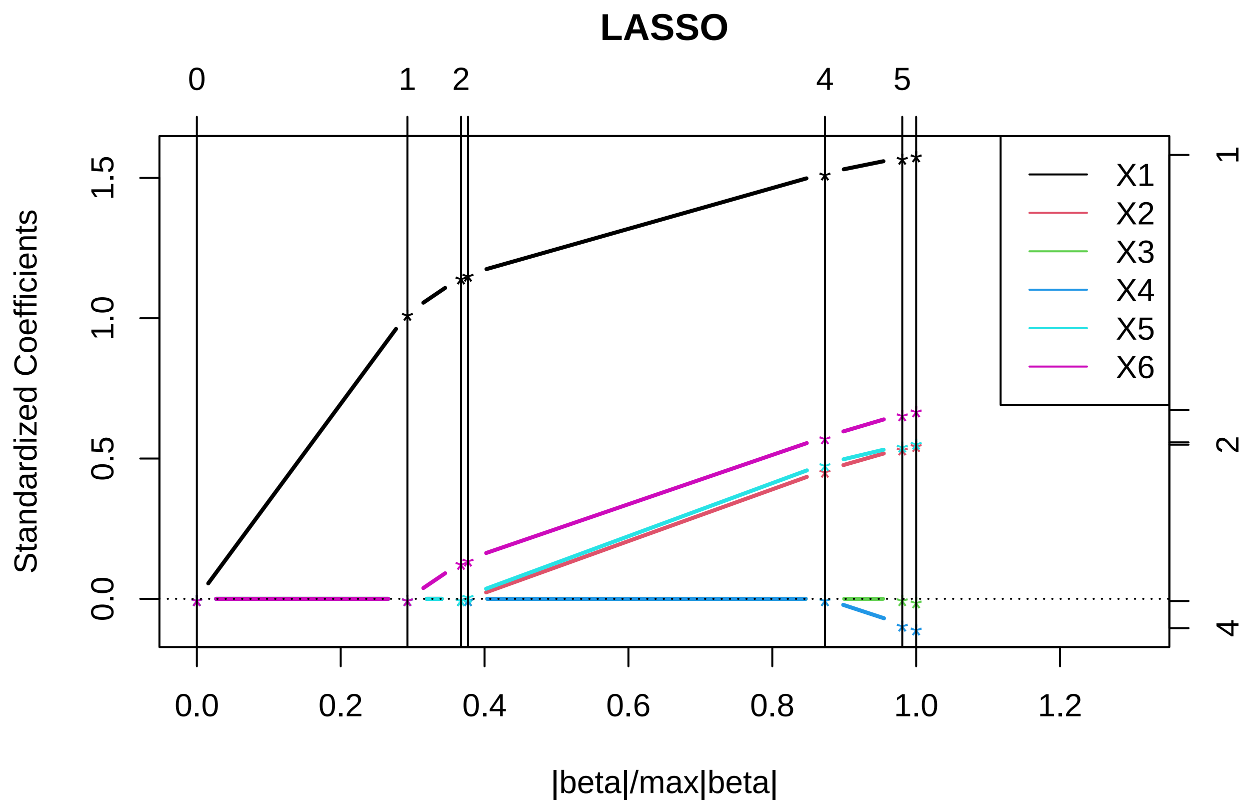


Now let's work with another configuration. Similar observation as above.

```
# Synthetic data Since we are comparing the algorithm 3.4 with lasso, we need
# to make sure they are using the same data.
set.seed(124)
n = 200
p = 6
x = matrix(runif(n * p), n, p)
y = x %*% c(5, 2, 0, 0, 2, 3) + 2 * rnorm(n)
dat = data.frame(x, y)

# Plot and compare with lasso result our function already standardize the
# predictors
incrementalStagewise(x, y, eps = 0.01)
```



```
require(lars)
# standardize predictors before fitting lasso
x <- scale(x, center = TRUE, scale = TRUE)
# lars() by default will return standardized coefficients for better comparison
# among the coefficients. However, since we are not standardizing in algorithm
# 3.4, we will set normalize = FALSE in lars().
fit.lasso.1 <- lars(x, y, normalize = FALSE)
plot(fit.lasso.1, col = 1:p, lwd = 2, xlim = c(0, 1.3), lty = 1)
legend("topright", legend = paste0("X", 1:p), col = 1:p, lty = 1)
```



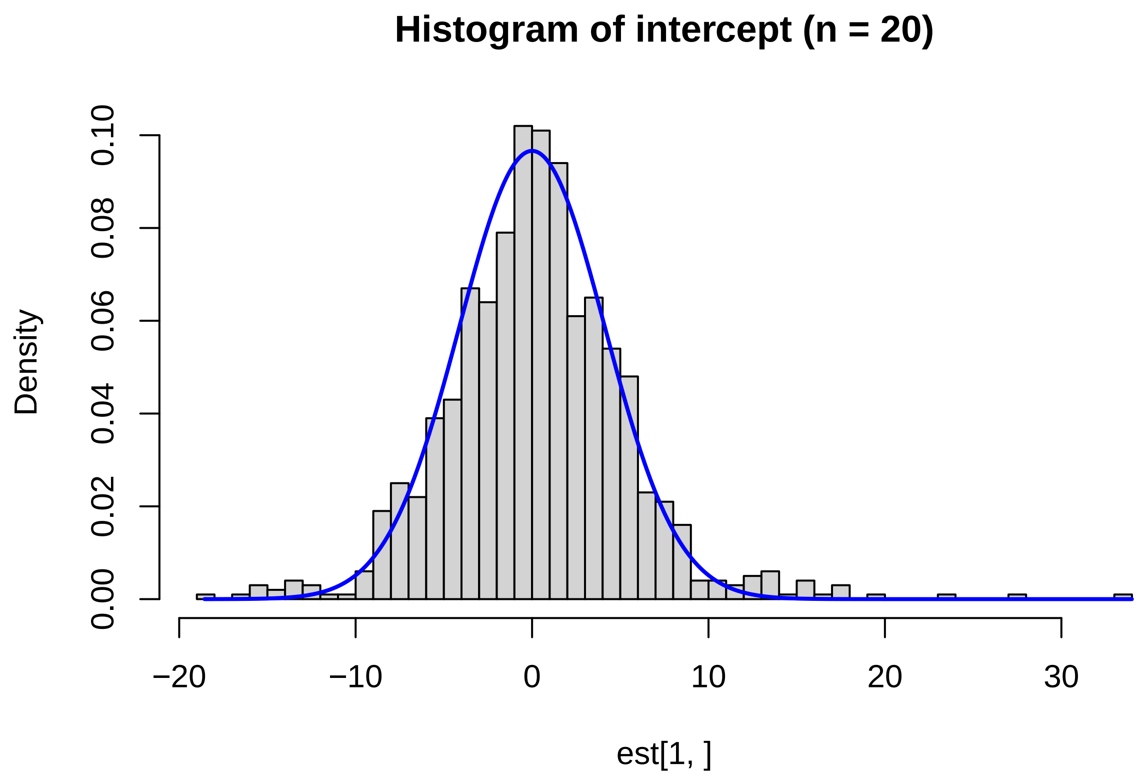
Problem 2

```
sample.size <- c(20, 50, 100, 200, 500, 1000)
true.beta.0 <- 0.3
true.beta.1 <- 0.7

# logistic function
f <- function(s, beta.0 = true.beta.0, beta.1 = true.beta.1) {
    return(1/(1 + exp(-beta.0 - beta.1 * s)))
}

# function prepared for computing matrix J later
g1 <- function(s, beta.0 = true.beta.0, beta.1 = true.beta.1) {
    return((exp(beta.0 + beta.1 * s)/(1 + exp(beta.0 + beta.1 * s))^2) * (1))
}
g2 <- function(s, beta.0 = true.beta.0, beta.1 = true.beta.1) {
    return((exp(beta.0 + beta.1 * s)/(1 + exp(beta.0 + beta.1 * s))^2) * (s))
}
g3 <- function(s, beta.0 = true.beta.0, beta.1 = true.beta.1) {
    return((exp(beta.0 + beta.1 * s)/(1 + exp(beta.0 + beta.1 * s))^2) * (s^2))
}
```

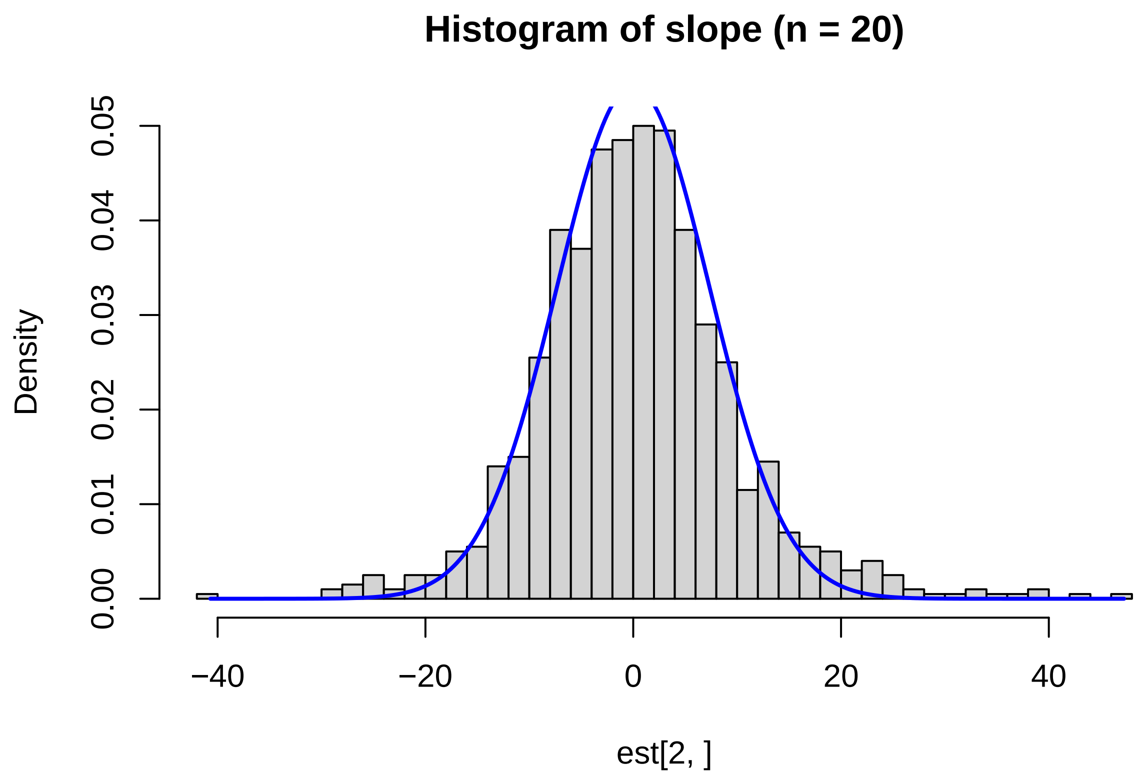
```r
mle <- function(n = sample.size[1], R = 1000) {
    # Store the estimation at each iteration
    store.inter <- numeric(R)
    store.slope <- numeric(R)

    set.seed(126)
    for (i in 1:R) {
        # Generate the data
        x <- runif(n, 0, 1)
        y <- rbinom(n, 1, f(x))

        # Fit the logistic model
        fit <- glm(y ~ x, family = binomial(link = "logit"))

        # Record the coefficients
        coef0 <- fit$coefficients[1]
        coef1 <- fit$coefficients[2]

        # Since we want to discuss asymptotic distribution (with rate sqrt(n),
        # instead of looking at \hat beta.0 or \hat beta.1, we look at
        # \sqrt(n)*(\hat beta.0 - true.beta.0) and \sqrt(n)*(\hat beta.1 -
        # true.beta.1)
        store.inter[i] <- sqrt(n) * (coef0 - true.beta.0)
        store.slope[i] <- sqrt(n) * (coef1 - true.beta.1)

    }

    return(rbind(store.inter, store.slope))
}

# Collect all six configurations (six different sample sizes) in a single
# matrix, which has dimension 12x1000
est <- rbind(mle(n = sample.size[1]), mle(n = sample.size[2]), mle(n = sample.size[3]),
    mle(n = sample.size[4]), mle(n = sample.size[5]), mle(n = sample.size[6]))

draw.density <- function(which.para = "beta.0", min.hist, max.hist) {

    mu <- 0

    # Follow slide 20 in lecture 6, we want to compute matrix J first. We know
    # X as written in the problem statement is unif[0,1]. Meanwhile the X
    # written in the slide is the vector (1,X), where X is unif[0,1]. Being
    # aware of this abuse of notation, we can compute J from pure probability
    # perspective, since the constant 1 is independent from X.

    # The dimension of matrix J is 2x2, because the parameter of interest is
    # the intercept and the slope. We compute each component of matrix
    # individually first.

    # By the information of mu and J, we can obtain the joint distribution of
    # (\sqrt(n)*(\hat beta.0 - true.beta.0), \sqrt(n)*(\hat beta.1 -
    # true.beta.1))

    J <- matrix(NA, 2, 2)
    J[1, 1] <- integrate(g1, lower = 0, upper = 1)[[1]]
    J[1, 2] <- integrate(g2, lower = 0, upper = 1)[[1]]
    J[2, 1] <- J[1, 2]  # by symmetry
    J[2, 2] <- integrate(g3, lower = 0, upper = 1)[[1]]

    J.inv <- solve(J)

    # Since we are only interested in comparing the asymptotic distribution of
    # \hat beta.0 and \hat beta.1 individually, let's use the above information
    # to derive density function for \sqrt(n)*(\hat beta.0 - true.beta.0) and
    # sqrt(n)*(\hat beta.1 - true.beta.1). B oth mean are 0, we just need to
    # derive their asymptotic variance.

    sigma.hat.beta.0 <- J.inv[1, 1]
    sigma.hat.beta.1 <- J.inv[2, 2]

    if (which.para == "beta.0") {
        den <- dnorm(seq(min.hist, max.hist, length.out = 1000), mean = mu, sd = sqrt(sigma.hat.beta.0)
    } else {
        den <- dnorm(seq(min.hist, max.hist, length.out = 1000), mean = mu, sd = sqrt(sigma.hat.beta.1)
    }

    return(den)
}
```

Finally let's generate 12 histograms with asymptotic distribution. We observe each of the plot and see that in general our histogram aligns with the asymptotic distribution, as n increase, the alignment improves. Our simulation helps us understand the theory better and valid our theory from the perspective of practice.

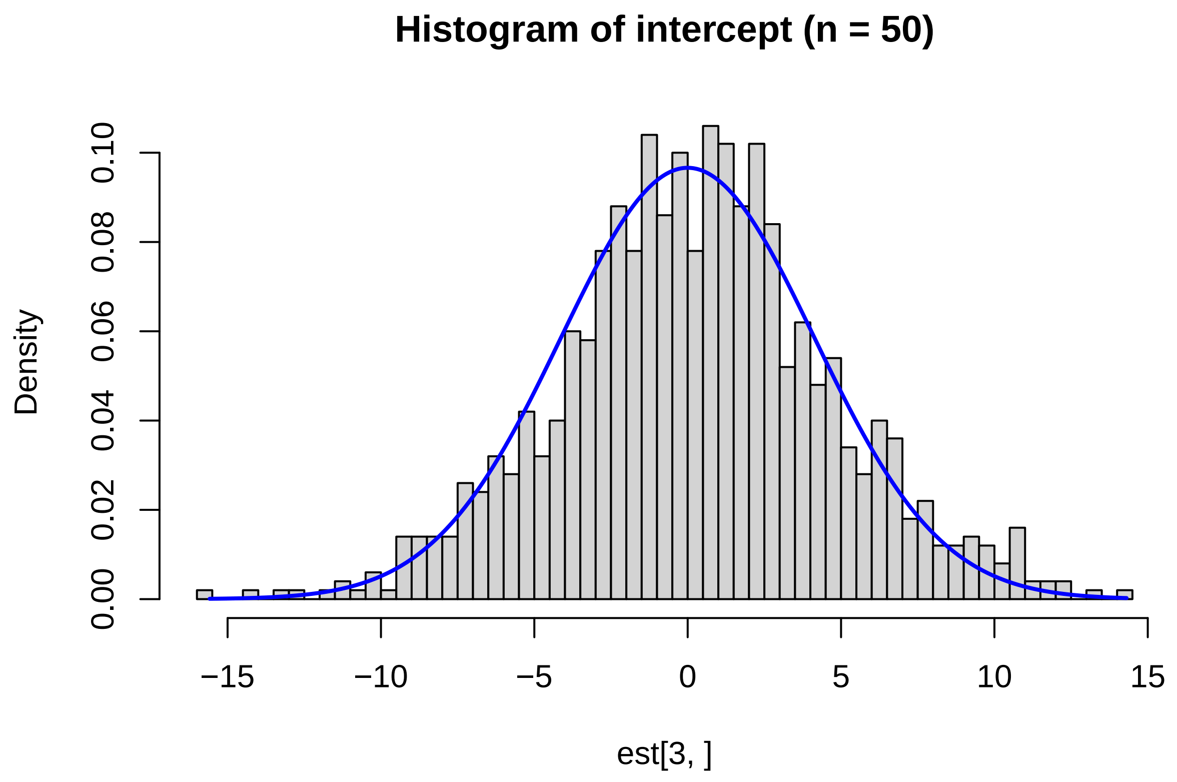
```
# PLOT 1: n = 20, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[1, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 20)"))
turn.den.1 <- draw.density(which.para = "beta.0", min.hist = min(est[1, ]), max.hist = max(est[1,
    ]))
lines(seq(min(est[1, ]), max(est[1, ]), length.out = 1000), turn.den.1, col = "blue",
    lwd = 2)
```



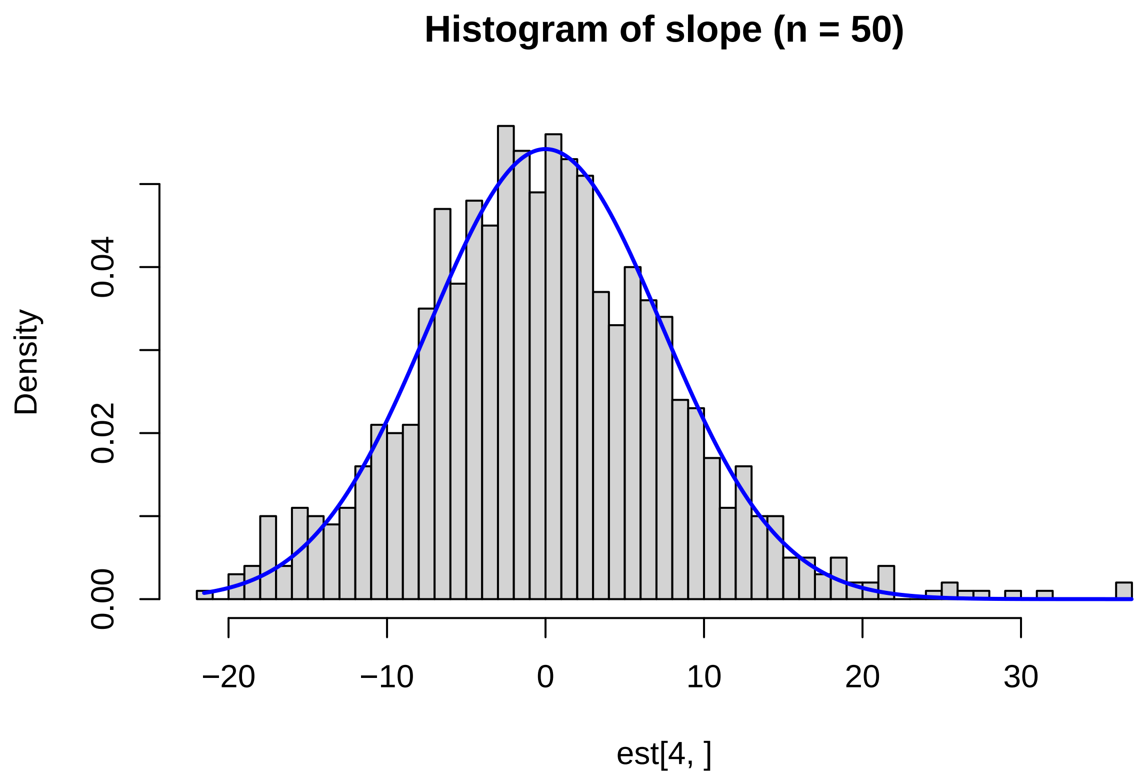
```
# PLOT 2: n = 20, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[2, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 20)"))
turn.den.2 <- draw.density(which.para = "beta.1", min.hist = min(est[2, ]), max.hist = max(est[2,
    ]))
lines(seq(min(est[2, ]), max(est[2, ]), length.out = 1000), turn.den.2, col = "blue",
    lwd = 2)
```



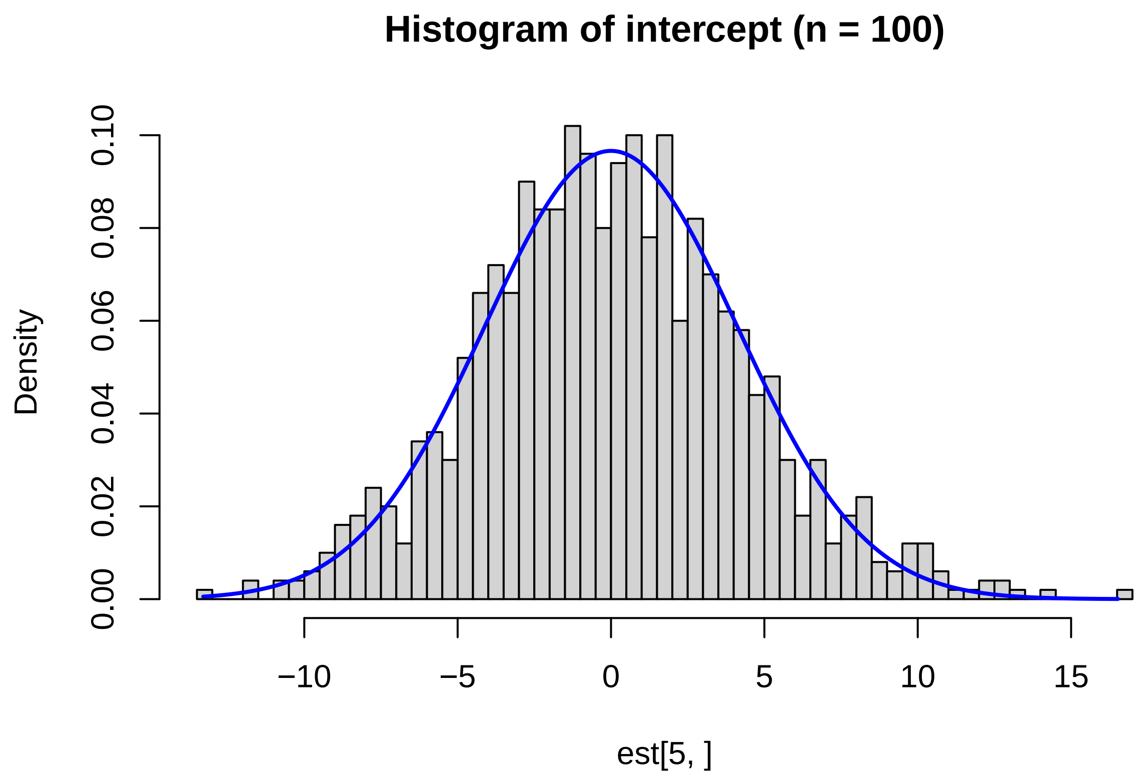
```
# PLOT 3: n = 50, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[3, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 50)"))
turn.den.3 <- draw.density(which.para = "beta.0", min.hist = min(est[3, ]), max.hist = max(est[3,
    ]))
lines(seq(min(est[3, ]), max(est[3, ]), length.out = 1000), turn.den.3, col = "blue",
    lwd = 2)
```



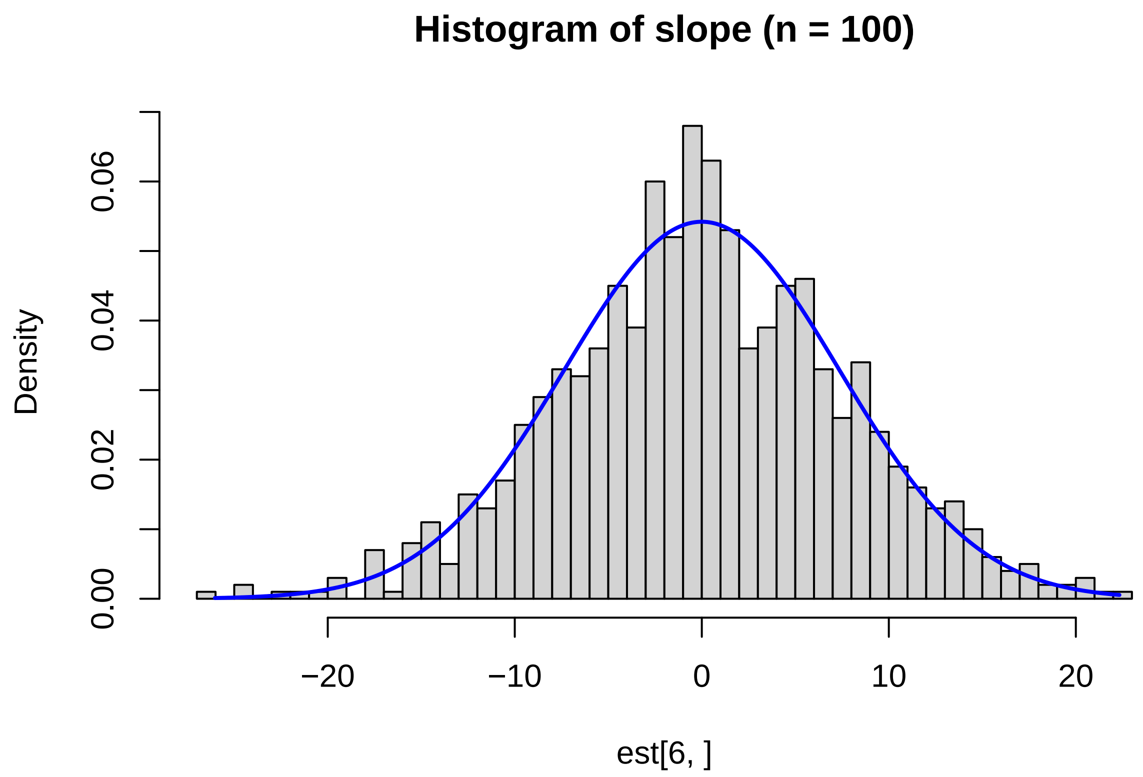
```r
# PLOT 4: n = 50, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[4, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 50)"))
turn.den.4 <- draw.density(which.para = "beta.1", min.hist = min(est[4, ]), max.hist = max(est[4,
    ]))
lines(seq(min(est[4, ]), max(est[4, ]), length.out = 1000), turn.den.4, col = "blue",
    lwd = 2)
```



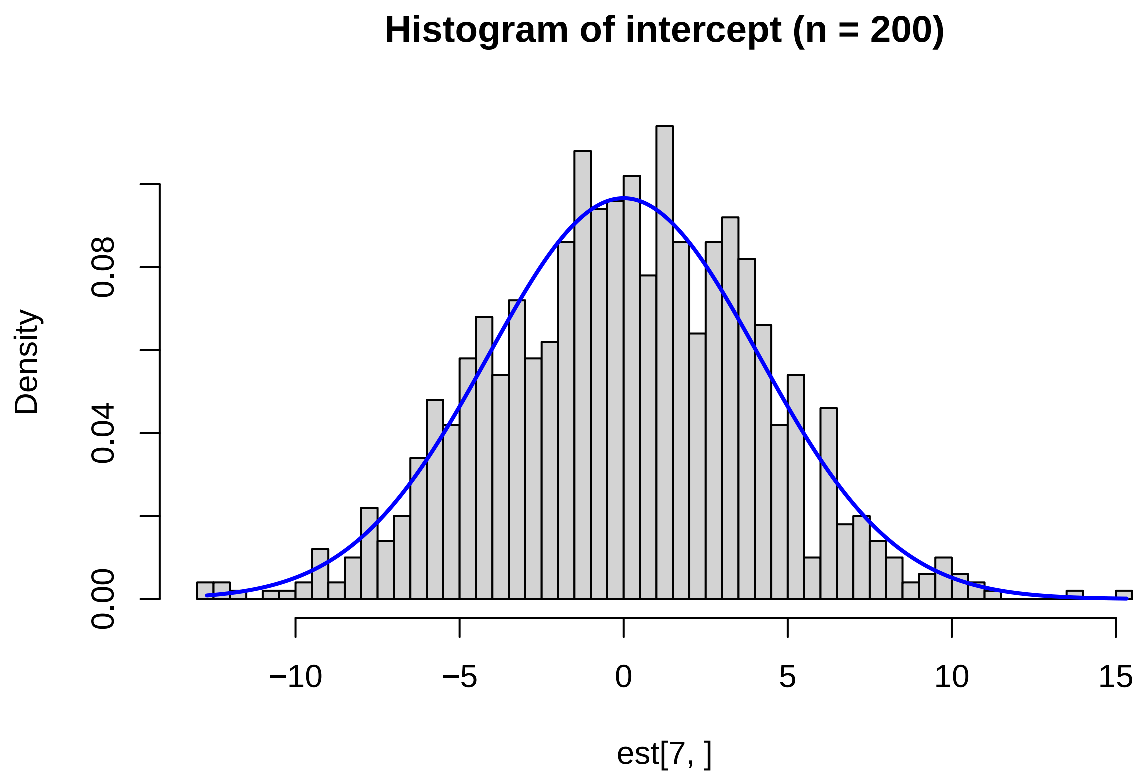
```
# PLOT 5: n = 100, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[5, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 100)"))
turn.den.5 <- draw.density(which.para = "beta.0", min.hist = min(est[5, ]), max.hist = max(est[5,
    ]))
lines(seq(min(est[5, ]), max(est[5, ]), length.out = 1000), turn.den.5, col = "blue",
    lwd = 2)
```



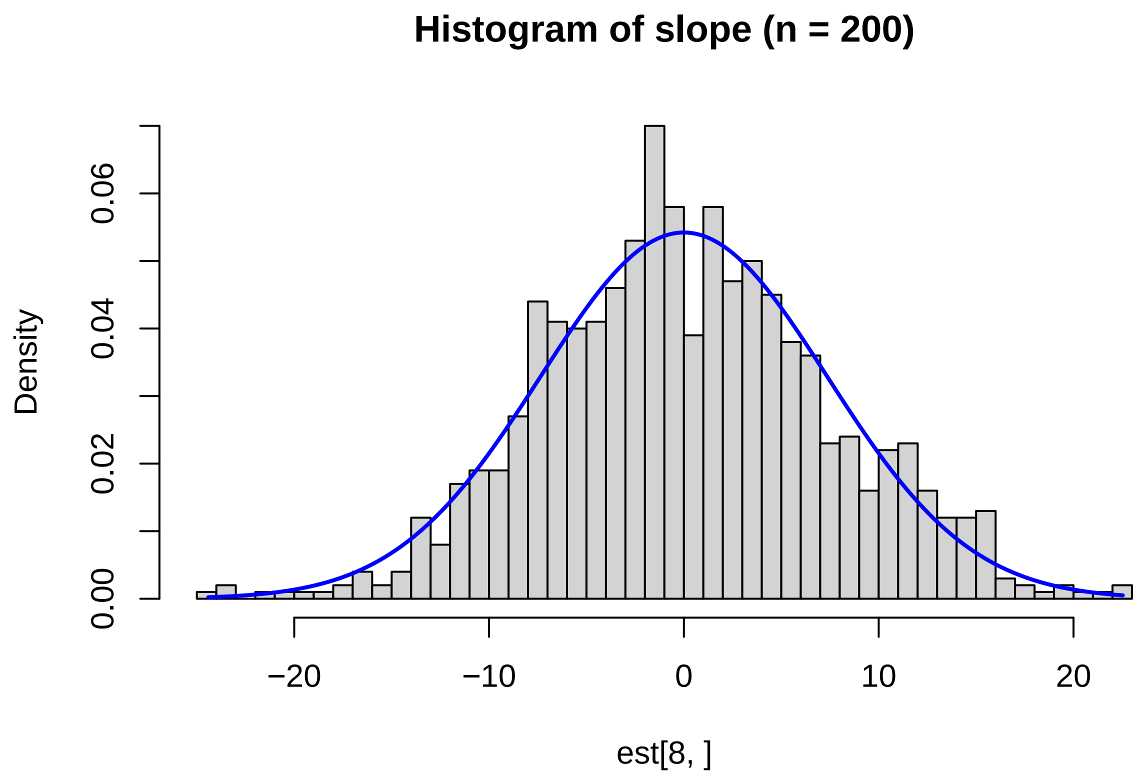
```
# PLOT 6: n = 100, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[6, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 100)"))
turn.den.6 <- draw.density(which.para = "beta.1", min.hist = min(est[6, ]), max.hist = max(est[6,
    ]))
lines(seq(min(est[6, ]), max(est[6, ]), length.out = 1000), turn.den.6, col = "blue",
    lwd = 2)
```



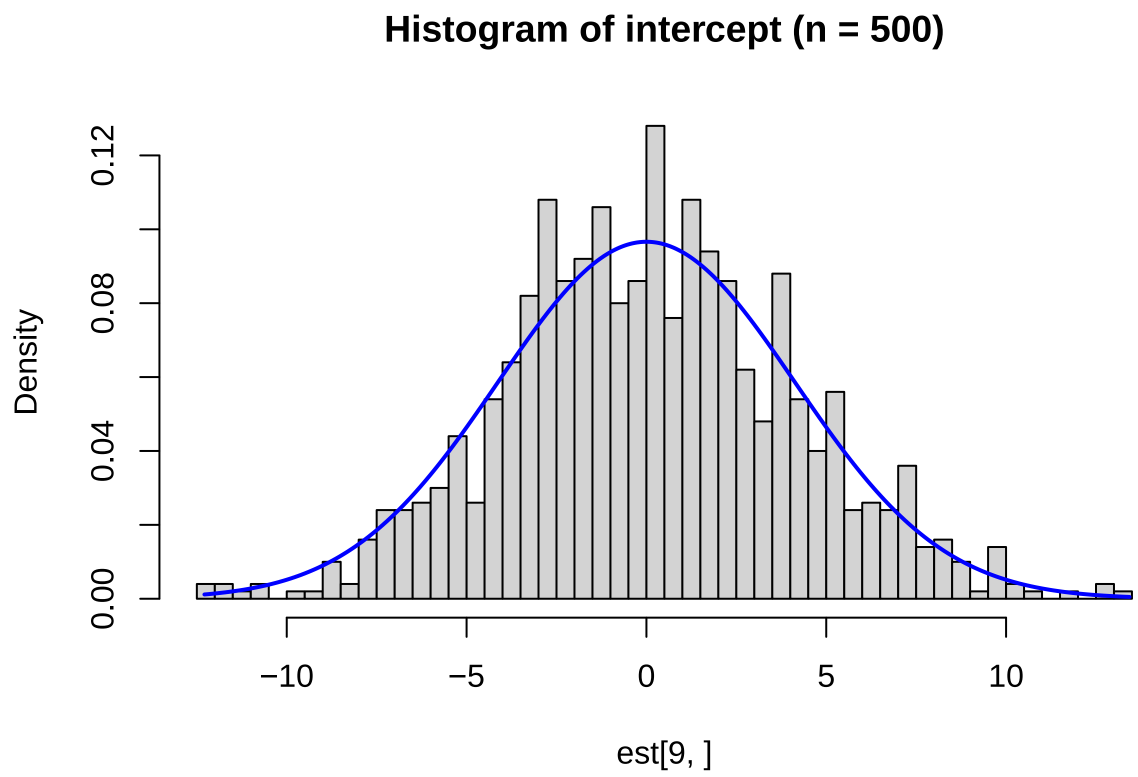
```r
# PLOT 7: n = 200, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[7, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 200)"))
turn.den.7 <- draw.density(which.para = "beta.0", min.hist = min(est[7, ]), max.hist = max(est[7,
    ]))
lines(seq(min(est[7, ]), max(est[7, ]), length.out = 1000), turn.den.7, col = "blue",
    lwd = 2)
```



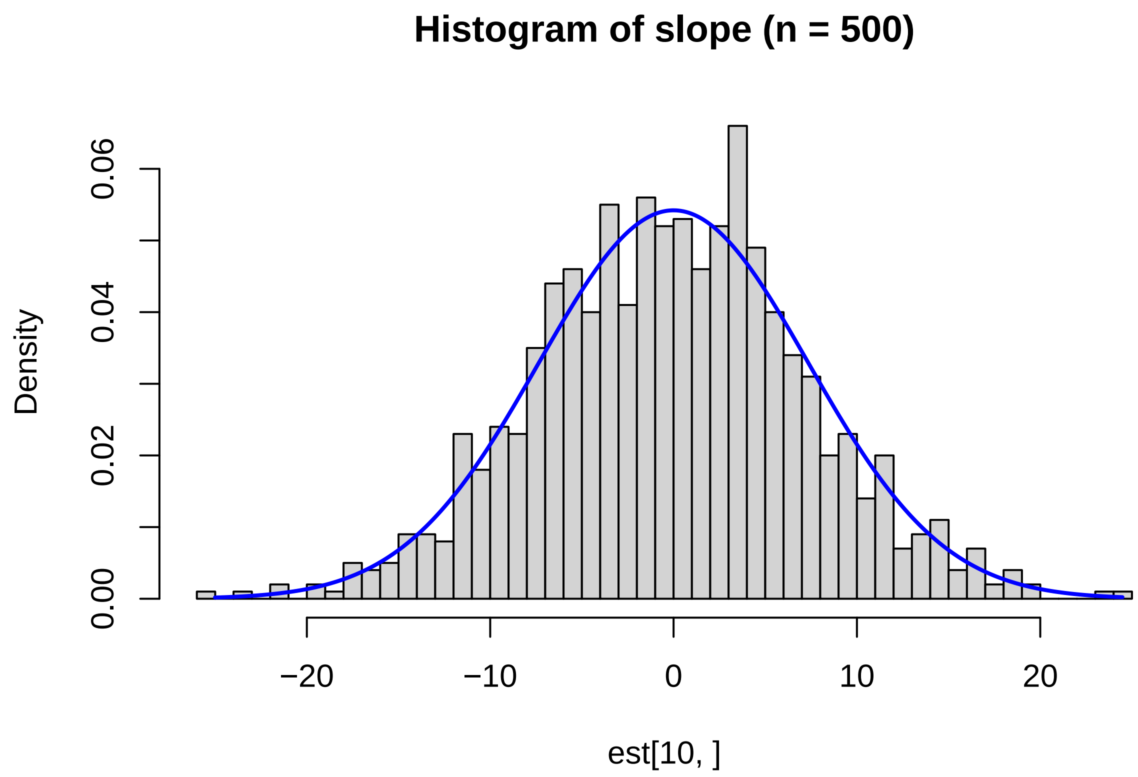
```r
# PLOT 8: n = 200, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[8, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 200)"))
turn.den.8 <- draw.density(which.para = "beta.1", min.hist = min(est[8, ]), max.hist = max(est[8,
    ]))
lines(seq(min(est[8, ]), max(est[8, ]), length.out = 1000), turn.den.8, col = "blue",
    lwd = 2)
```



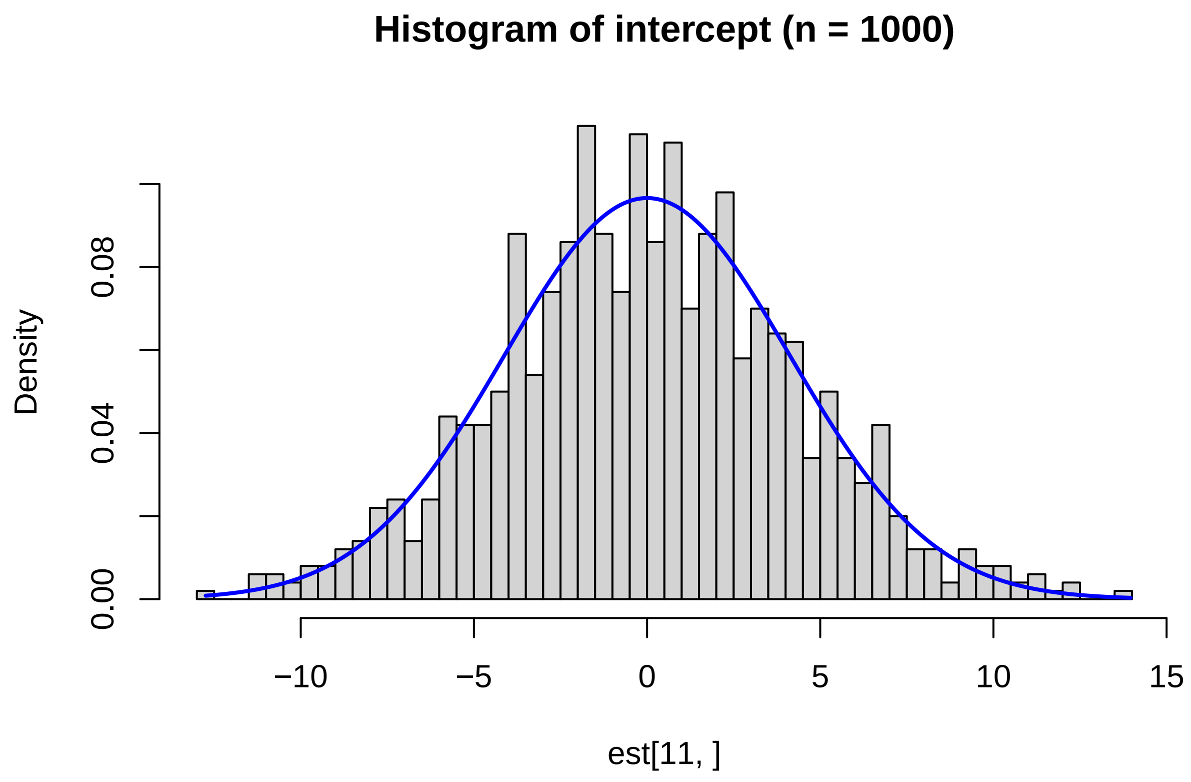
```
# PLOT 9: n = 500, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[9, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 500)"))
turn.den.9 <- draw.density(which.para = "beta.0", min.hist = min(est[9, ]), max.hist = max(est[9,
    ]))
lines(seq(min(est[9, ]), max(est[9, ]), length.out = 1000), turn.den.9, col = "blue",
    lwd = 2)
```



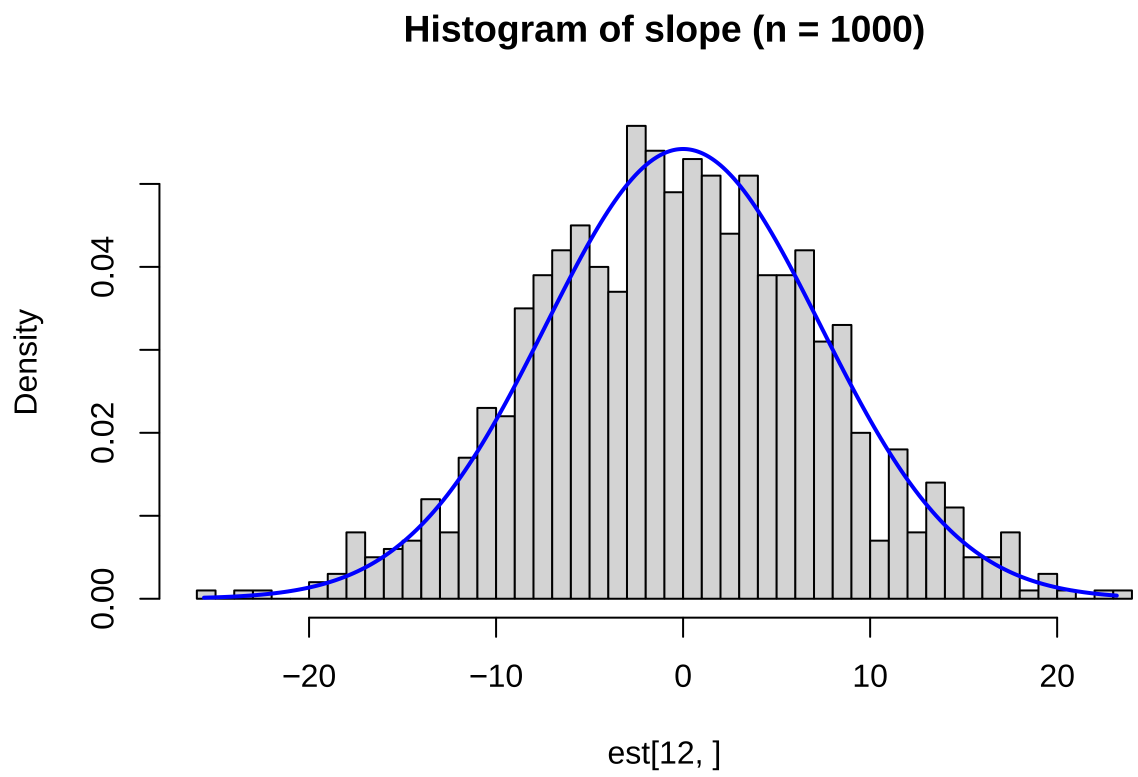
```
# PLOT 10: n = 500, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[10, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 500)"))
turn.den.10 <- draw.density(which.para = "beta.1", min.hist = min(est[10, ]), max.hist = max(est[10,
    ]))
lines(seq(min(est[10, ]), max(est[10, ]), length.out = 1000), turn.den.10, col = "blue",
    lwd = 2)
```



```
# PLOT 11: n = 1000, \sqrt(n)*(\hat beta.0 - true.beta.0)
hist(est[11, ], breaks = 50, freq = FALSE, main = paste0("Histogram of intercept (n = 1000)"))
turn.den.11 <- draw.density(which.para = "beta.0", min.hist = min(est[11, ]), max.hist = max(est[11,
    ]))
lines(seq(min(est[11, ]), max(est[11, ]), length.out = 1000), turn.den.11, col = "blue",
    lwd = 2)
```



```
# PLOT 12: n = 1000, \sqrt(n)*(\hat beta.1 - true.beta.1)
hist(est[12, ], breaks = 50, freq = FALSE, main = paste0("Histogram of slope (n = 1000)"))
turn.den.12 <- draw.density(which.para = "beta.1", min.hist = min(est[12, ]), max.hist = max(est[12,
    ]))
lines(seq(min(est[12, ]), max(est[12, ]), length.out = 1000), turn.den.12, col = "blue",
    lwd = 2)
```


Histogram of slope (n = 1000)
Density
0.00
0.02
0.04
−20
−10
0
10
20
est[12,]

“‘