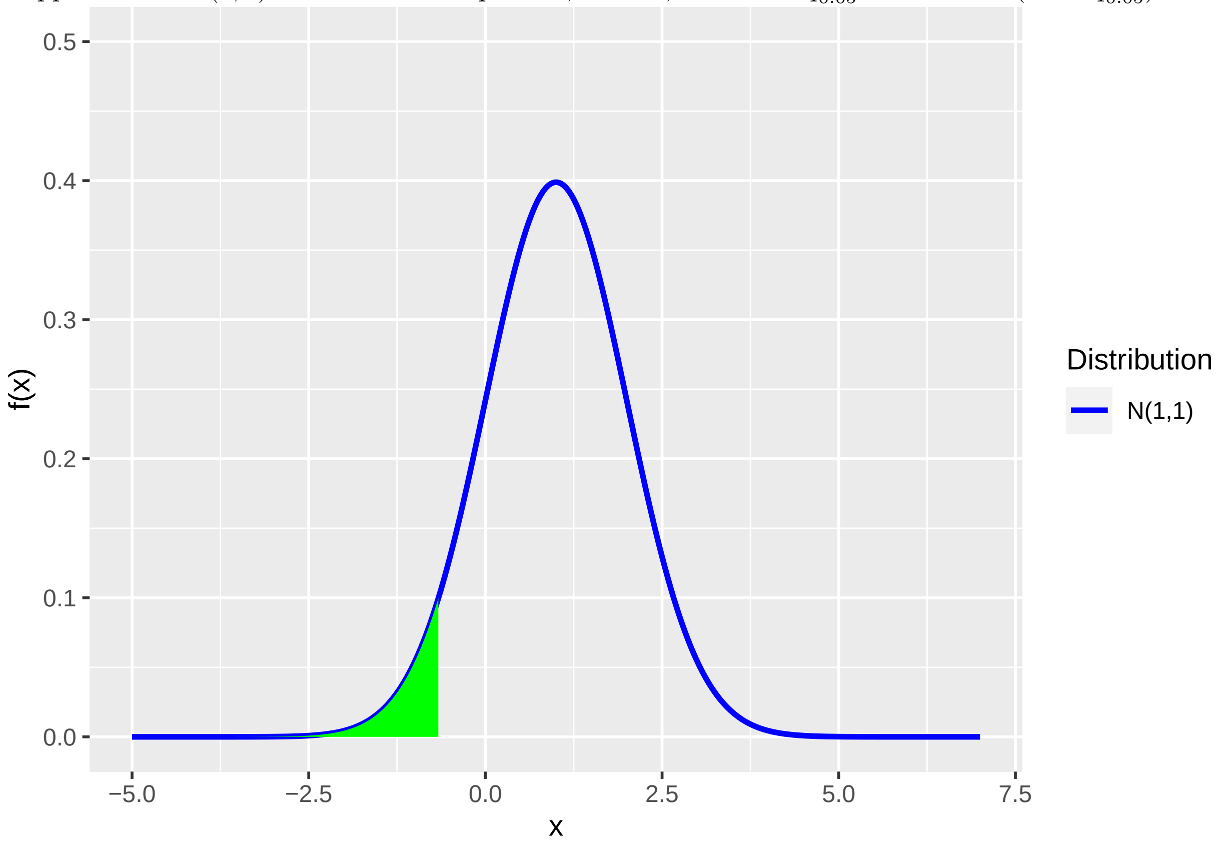
Value at Risk and Expected Shortfall

jsn-jin @ UCLA

Contents

Find Quantiles from Normal Distributions Using `qnorm`

Suppose $X \sim N(1,1)$. Find the 0.05 quantile, that is, the value $q^X_{0.05}$ such that $P(X \leq q^X_{0.05}) = 0.05$.



```r
alpha <- 0.05
mu.x <- 1
sigma.x <- 1

# Syntax for qnorm function:
# qnorm(p, mean, sd, lower.tail).
q_alpha = qnorm(alpha, mu.x, sigma.x)

sprintf("The %.2f quantile is %.3f", alpha, q_alpha)
```

```
## [1] "The 0.05 quantile is -0.645"
```

Let's check the probability that X is at most -0.645 is 0.05 using monte carlo simulation.

```r
set.seed(2020)
runs <- 5000
X <- rnorm(runs, mu.x, sigma.x)
sum(X <= q_alpha) / runs
```

```
## [1] 0.0532
```

The result is approximately equal to 0.05 if it is rounded to 2 decimal places.

Determine Value-at-Risk (VaR) and Expected Shortfall (ES)

Let r be the continuously compounded monthly return on a stock and W_0 be the initial wealth to be invested over the month. Assume that $r \sim N(0.01, 0.1^2)$ and $W_0 = \$10,000$.

Determine 5% VaR

Let W_1 be the end of month wealth and $L_1 \equiv W_1 - W_0$ be the profit of the investment, which can be negative. The monthly VaR on the W_0 investment with probability α is denoted by VaR_α such that

$$P(L_1 \le VaR_\alpha) = \alpha.$$

Note that

$$L_1 = W_1 - W_0 = W_0 \cdot (e^r - 1).$$

Thus, we have

$$P\Bigg(W_0 \cdot (e^r - 1) \le VaR_\alpha\Bigg) = \alpha.$$

It implies that

$$P\left(r \le x\right) = \alpha.$$

if $VaR_\alpha = W_0 \cdot (e^x - 1)$. From the definition of quantile function (inverse CDF), we know $x = q^r_\alpha$.

Therefore, to compute 5% VaR, we compute

1. $q^r_{5\%}$
2. $VaR_{5\%} = \$10,000 \cdot (e^{q^r_{5\%}} - 1)$

```
library(scales)
W0 <- 10000
alpha <- 0.05
mu.r = 0.01
sigma.r = 0.1

q_r <- qnorm(alpha, mu.r, sigma.r)
VaR_r <- W0 * (exp(q_r) - 1)
sprintf("The %s VaR is %.3f", percent(alpha), VaR_r)
```

```
## [1] "The 5% VaR is -1431.440"
```

Determine 5% ES

Given the profit is not larger than VaR_α, the expected shortfall (ES) at α measures the conditional expected profit, i.e.,

$$ES_\alpha = E[L_1 | L_1 \leq VaR_\alpha] = \alpha^{-1} \int_{-\infty}^{VaR_\alpha} x f_{L_1}(x) dx = \alpha^{-1} \int_0^\alpha VaR_u du.$$

Next, we introduce several ways to compute ES_α, where $\alpha = 5\%$.

Numerical Integration

```
# Define the integrand
Integrand <- function(u) {W0 * (exp(qnorm(u, mu.r, sigma.r)) - 1) / alpha}
res <- integrate(Integrand, lower = 0, upper = alpha)
res
```

```
## -1776.533 with absolute error < 0.00033
```

```
sprintf("The %s ES is %#.3f", percent(alpha), res[1])
```

```
## [1] "The 5% ES is -1776.533"
```

Monte Carlo Simulation

```
set.seed(123)
runs <- 5000

r <- rnorm(runs, mu.r, sigma.r) # used to generate random numbers
L1 <- W0 * (exp(r) - 1)
indicator <- as.numeric(L1 <= VaR_r)
ES <- mean(L1 * indicator) / mean(indicator)
sprintf("The simulated %s ES is %#.3f", percent(alpha), ES)
```

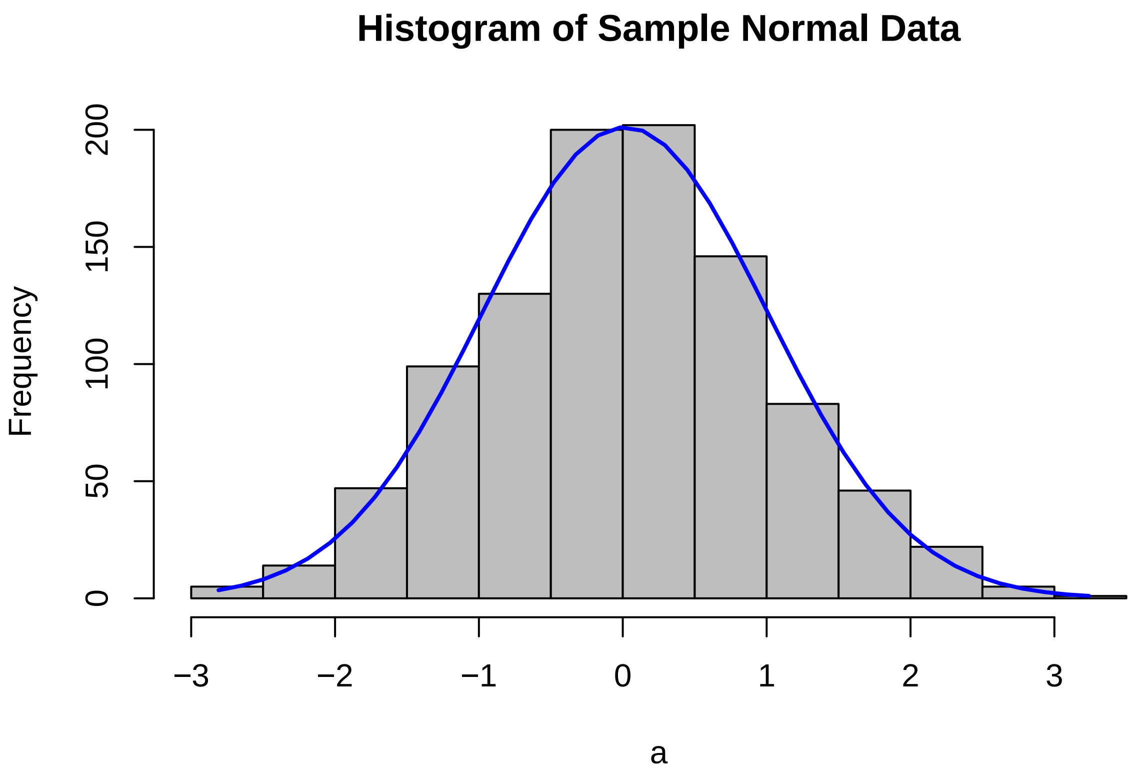
```
## [1] "The simulated 5% ES is -1763.887"
```

As we can see, the simulated -1763.887 is very close to the conditional expectation -1776.533 (theoretic value).

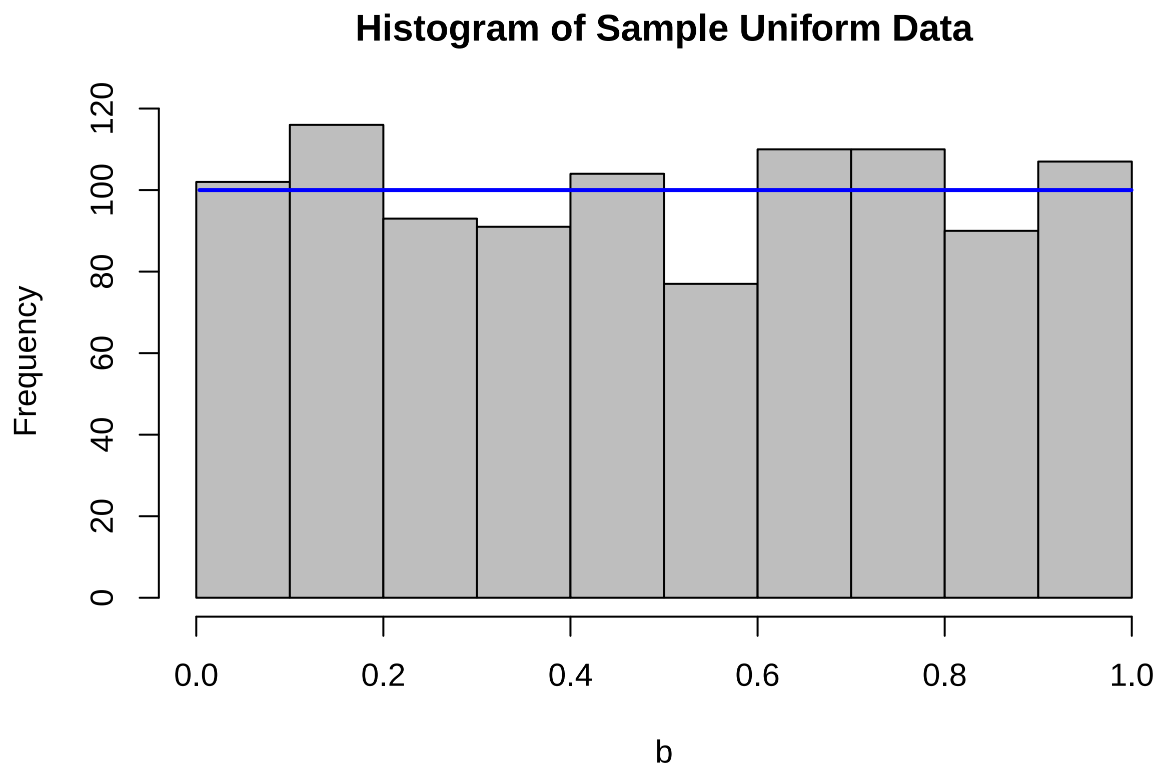
Generating Random Numbers from Commonly Seen Distributions

In many cases, we need to generate random numbers from certain distributions, i.e., normal distribution, Possion distribution, binomial distribution. Here are some commonly used functions for this purposes.

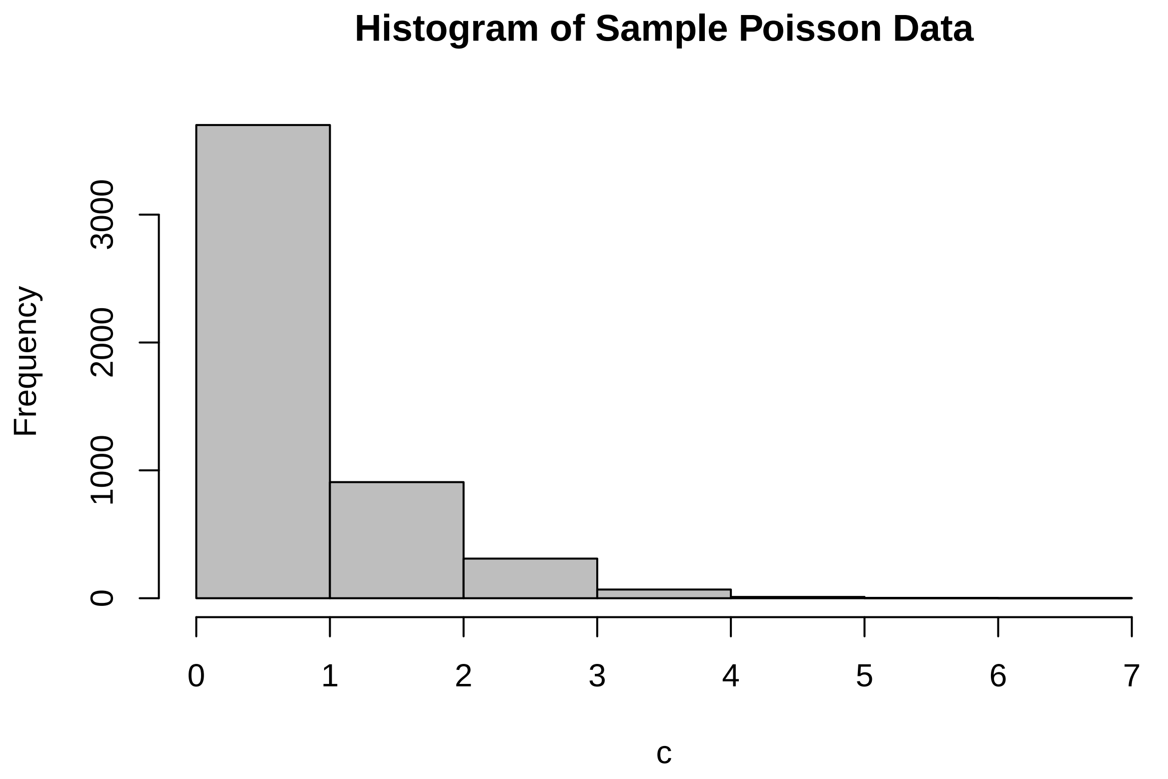
```r
set.seed(123)
# Normal Distribution
a <- rnorm(1000, mean = 0, sd = 1)
h <- hist(a, breaks = 10, col = "grey", main = "Histogram of Sample Normal Data")
afit <- seq(min(a), max(a), length = 40)
yfit <- dnorm(afit, mean = mean(a), sd = sd(a))
yfit <- yfit * diff(h$mids[1:2]) * length(a)
lines(afit, yfit, col = "blue", lwd = 2)
```



```r
# Unif Distribution
b <- runif(1000, min = 0, max = 1)
h1 <- hist(b, breaks = 10, col = "grey", main = "Histogram of Sample Uniform Data")
bfit <- seq(min(b), max(b), length = 40)
yfit <- dunif(bfit, min = 0,  max = 1)
yfit <- yfit * diff(h1$mids[1:2]) * length(b)
lines(bfit, yfit, col = "blue", lwd = 2)
```



```r
# Possion Distribution
c <- rpois(5000, lambda = 1)
h2 <- hist(c, breaks = 7, col = "grey", main = "Histogram of Sample Poisson Data")
```



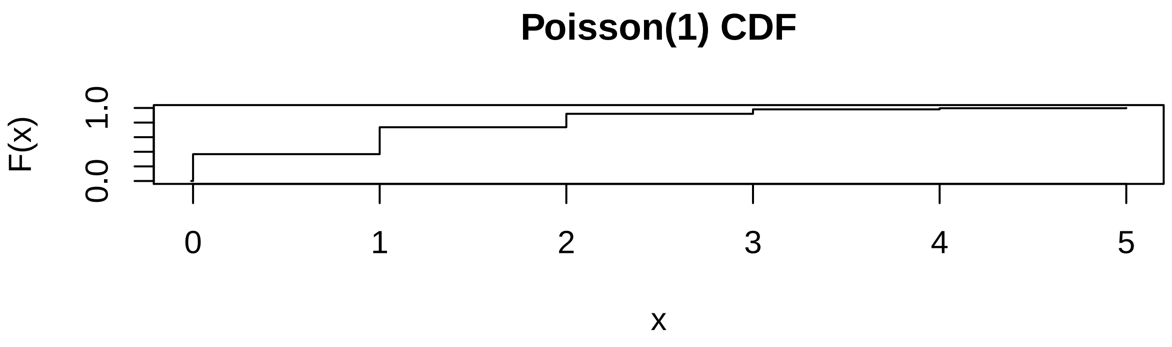
```r
summary(c) # What is the theoretical mean of c?
```

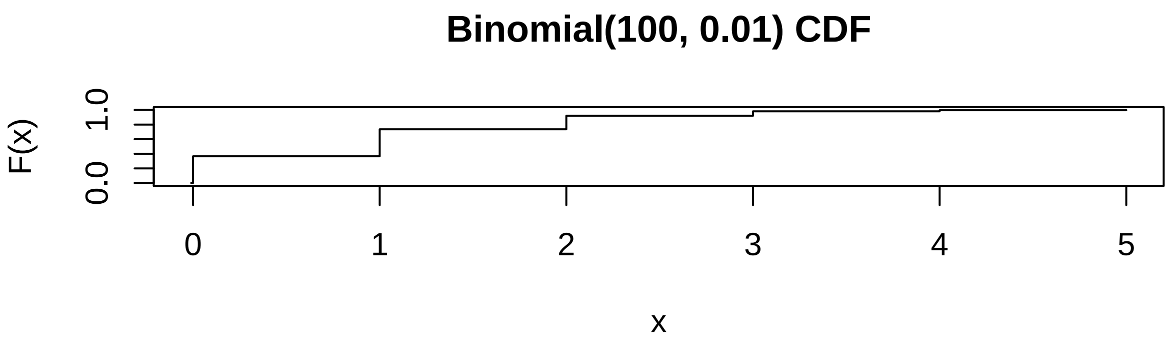
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  1.0000  0.9948  2.0000  7.0000
```

```r
var(c) # What is the theoretical variance of c?
```

```
## [1] 0.9641658
```

```r
# Check the relationship between Possion Distribution and Binomial Distribution
par(mfrow = c(2, 1))
x <- seq(-0.01, 5, 0.01)
plot(x, ppois(x, 1), type = "s", ylab = "F(x)", main = "Poisson(1) CDF")
plot(x, pbinom(x, 100, 0.01), type = "s", ylab = "F(x)", main = "Binomial(100, 0.01) CDF")
```





```r
par(mfrow = c(1, 1))
```

Monte Carlo Simulation: The Finite Sample Performance of the Least Squares Estimator

Next, let's use Monte Carlo simulation to study the asympototic distribution of the linear least squares estimator. We consider the following model:

$$y_i = x_i\beta + \epsilon_i,$$

where the true value of $beta_1$ is 2 and the true value of β_0 is 2. Also suppose that the sample size $N = 1000$, x_i is independently generated from a standard normal distribution, i.e., $x_i \sim N(0, 1)$. The error term ϵ_i also follows a standard normal distribution and is independent of x_i.

We are interested in the asympototic distribution of the ordinary least squares estimator of β, which is denoted as $\hat{\beta}_{LS}$. Recall what you have learned in the ECON 103 class. The OLS estimator $\hat{\beta}_{LS}$ can be calculated using the following formula:

$$\hat{\beta}_{LS} = \frac{\sum\limits_{i=1}^{N} x_i y_i}{\sum\limits_{i=1}^{N} x_i^2}.$$

Utilizing both Central Limit Theorem (CLT) and Law of Large Numbers (LLN) gives us the asympototic distribution of the above least squares estimator,

$$\sqrt{N}(\hat{\beta}_{LS} - \beta) \xrightarrow{d} N(0, \sigma_\epsilon^2 \cdot \mathbf{E}[x_i^2]^{-1}).$$

We will use Monte Carlo simulations to study the finite sample performance of the least squares estimator. The simulation steps are:

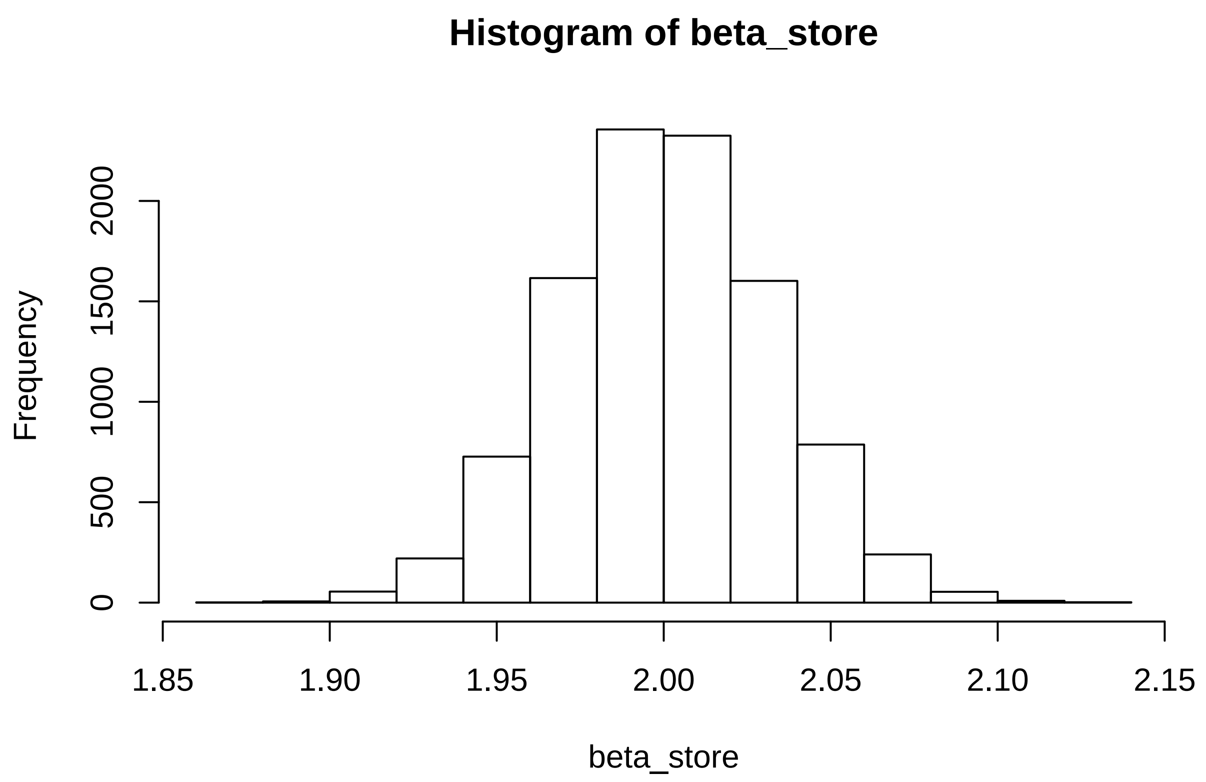
(1) For each simulation (indexed by k), generate $(x_i, y_i)_{i=1}^N$ from the corresponding distribution of x_i and the model.
(2) Calculate the least sqaures estimator $\hat{\beta}_k$ and store its value in a vector.
(3) Repeat (1) and (2) for K times (here K will be the number of simulations). We then get $\{\hat{\beta}_k\}_{k=1}^K$, which enables us to approximate the distribution of $\sqrt{N}(\hat{\beta}_{LS} - \beta)$.

Here we set the number of simulations to be 10000, i.e., $K = 10000$.

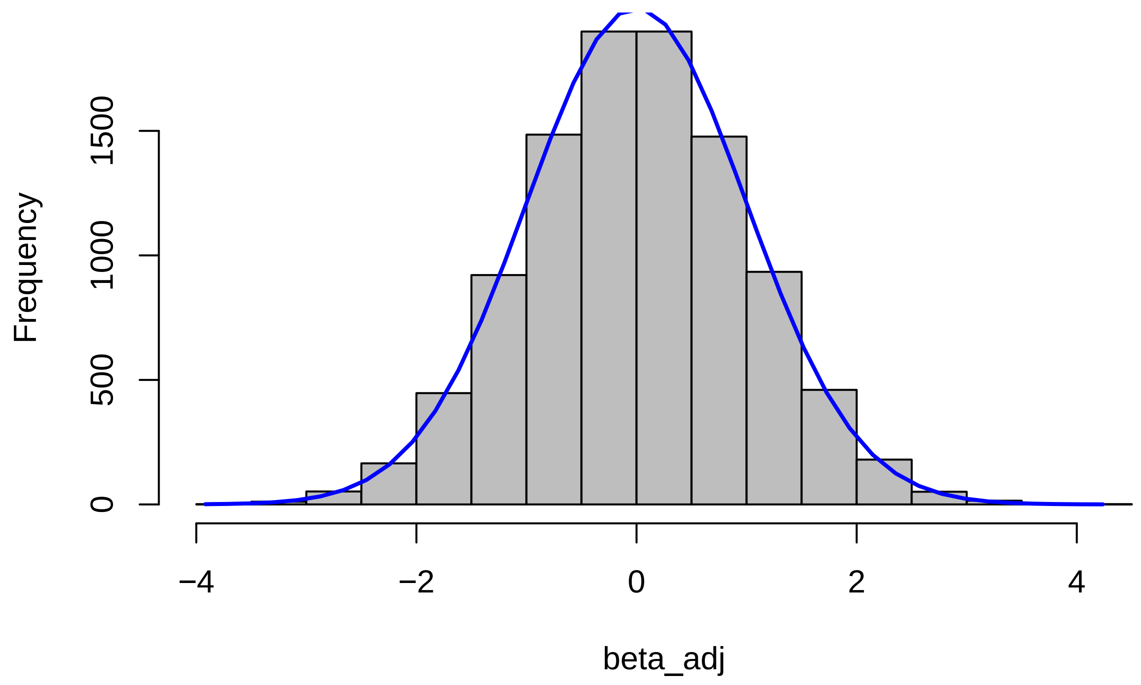
```r
K <- 10000
beta_0 <- 2 # define the value of the true beta
N <- 1000

beta_store <- matrix(rep(0, K), ncol = 1)

for (k in 1:K){
  X <- matrix(rnorm(1000, mean = 0, sd = 1), ncol = 1)
  error <- matrix(rnorm(1000, mean = 0, sd = 1), ncol = 1)
  Y <- beta_0 * X + error

  beta_LS <- (t(X) %*% Y) / (t(X) %*% X)
  beta_store[k,] <- beta_LS
}
```

Now, we have finished the Monte Carlo part, we will go to analyze the finite sample performance of the least squares estimator based on the data we generated.

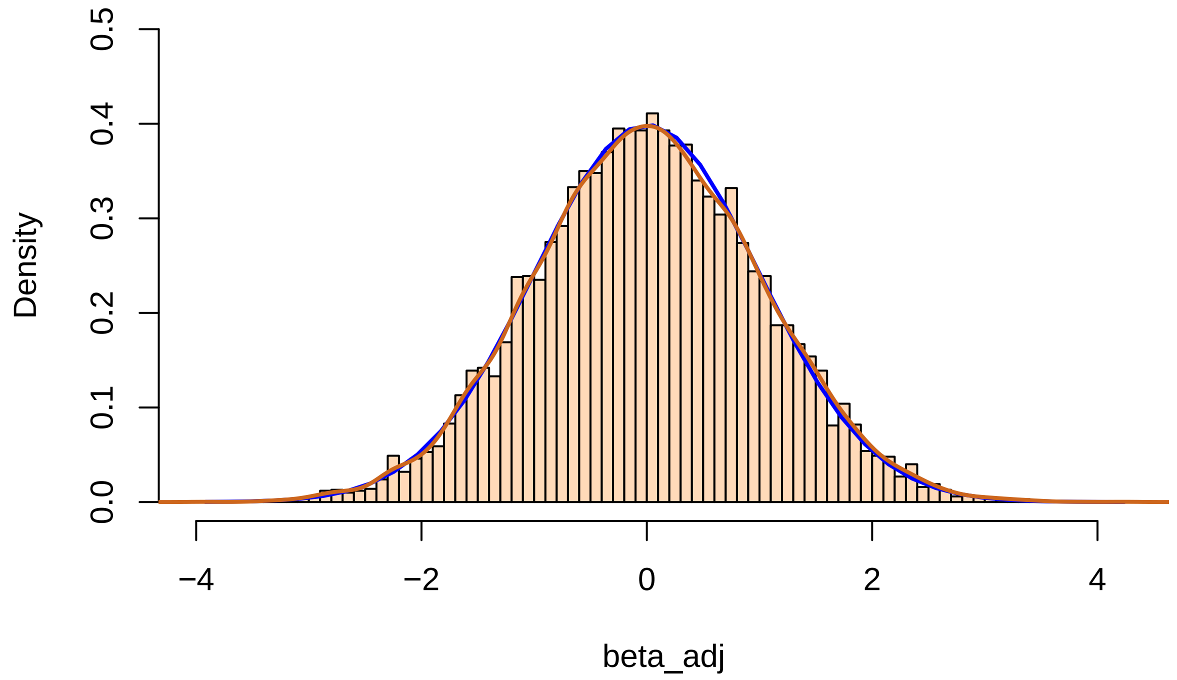
```r
hist(beta_store)
```



```r
# Looks like a normal distribution, but however, it's not standard normal.

beta_adj <- sqrt(N) * (beta_store - 2)
h1 <- hist(beta_adj, breaks = 20, col = "grey", main = "")

betafit <- seq(min(beta_adj), max(beta_adj), length = 40)
yfit <- dnorm(betafit, mean = 0, sd = 1)
yfit <- yfit * diff(h1$mids[1:2]) * length(beta_adj)
lines(betafit, yfit, col = "blue", lwd = 2)
```



```r
hist(beta_adj,
 col = "peachpuff",
 border = "black",
 breaks = 100,
 prob = TRUE,
 main = "" ,
 ylim = c(0,0.5))

betafit <- seq(min(beta_adj), max(beta_adj), length = 40)
yfit <- dnorm(betafit, mean = 0, sd = 1)
# yfit <- yfit * diff(h1$mids[1:2]) * length(beta_adj)

lines(betafit, yfit, col = "blue", lwd = 2)

lines(density(beta_adj),
      lwd = 2,
      col = "chocolate3")
```



An Important Function: Sample()

The function "*sample*" takes a sample of the specified size from the elements of x using either with or without replacement.

```r
# 100 Bernoulli trials
sample(c(0,1), 100, replace = TRUE)
```

```
##   [1] 0 0 0 1 0 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0
##  [38] 0 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 1
##  [75] 0 0 0 0 1 1 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1
```

```r
x <- 1:12
# Random permutation
sample(x)
```

```
##  [1]  6  2 11  1  7  8  3  4  5 12  9 10
```

```r
# Bootstrap resampling -- only if length(x) > 1 !
sample(x, replace = TRUE)
```

```
##  [1] 11  5 10  2  5  2 10  4  9  3  9  9
```