Simulation

Zhentao Shi

April 1, 2020

Simulation

若夫乘天地之正，而御六气之辩，以游无穷者，彼且恶乎待哉

Probability theory has an infamous inception for its association with gambling. Monte Carlo, the European casino capital, is another unfortunate presence. However, naming it Macau simulation or Hong Kong Jockey Club simulation does not make me feel any better. I decide to simply call it "simulation".

Simulation has been widely used for (i) checking finite-sample performance of asymptotic theory; (ii) bootstrap, an automated data-driven inference procedure; (iii) generating non-standard distributions; (iv) approximating integrals with no analytic expressions. In this lecture, we will focus on (i) and (ii), whereas (iii) and (iv) will be deferred to the next lecture on integration.

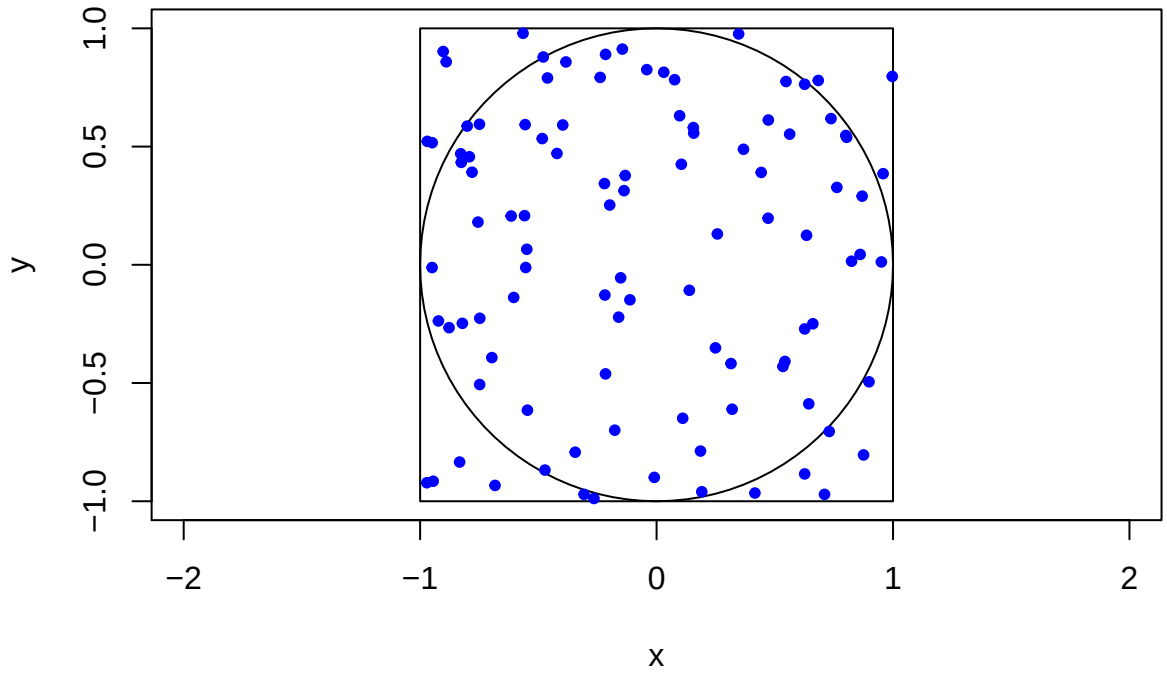
From now on, we will start to write script. A script is a piece of code for a particular purpose. A script of thousands of lines is not written from the beginning to the end; we develop it recursively. We cut a big job into small manageable tasks. Write a small piece, test it, and encapsulate it into a user-defined function whenever necessary. Small pieces are integrated by the super structure. This is just like building an Airbus 380. The engines and wings are made in UK, the fuselage is made in Germany and so on. All pieces are assembled in Toulouse, France, and then the giant steel bird can fly. Finally, add comments to the script to facilitate readability. Without comments you will forget what you did when you open the script again one month later.

Example

Zu Chongzhi (429–500 AD), an ancient Chinese mathematician, calculated π being between 3.1415926 and 3.1415927, which for 900 years held the world record of the most accurate π. He used a deterministic approximation algorithm. Now imagine that we present to Zu Chongzhi, with full respect and admiration, a modern computer. How can he achieve a better approximation?

```
require(plotrix)
require(grid)

plot(c(-1, 1), c(-1, 1), type = "n", asp = 1, xlab = "x", ylab = "y")
rect(-1, -1, 1, 1)
draw.circle(0, 0, 1)
points(x = runif(100) * 2 - 1, y = runif(100) * 2 - 1, pch = 20, col = "blue")
```



Standing on the shoulder of laws of large numbers, π can be approximated by a stochastic algorithm. Since

$$\frac{\text{area of a circle}}{\text{area of the smallest encompassing square}} = \frac{\pi r^2}{(2r)^2} = E\left[\mathbf{1}\left\{x^2 + y^2 \leq 1\right\}\right],$$

it implies $\pi = 4 \times E[1\{x^2 + y^2 \leq 1\}]$. The mathematical expectation is unknown, and it can be approximated by simulation.

```
n <- 10000000
Z <- 2 * matrix(runif(n), ncol = 2) - 1 # uniform distribution in [-1, 1]

inside <- mean(sqrt(rowSums(Z^2)) <= 1) # the center of the cirle is (0, 0)
cat("The estimated pi = ", inside * 4, "\n")
```

```
## The estimated pi =  3.142578
```

The sample size can be made as large as the computer's memory permits, and we can iterate it with average of averages and so on, for higher accuracy.

Finite Sample Evaluation

In the real world, a sample is finite. The distribution of a statistic in finite sample depends on the sample size n, which has only in rare cases a simple mathematical expression. Fortunately,

the expression can often be simplified when we imagine the sample size being arbitrarily large. Asymptotic theory is such mathematical apparatus to approximate finite sample distributions. It is so far the most useful analytical tool that helps us understand the behavior of estimators and tests, either in econometrics or in statistics in general. Simulation is one way to evaluate the accuracy of approximation.

Even though real-data empirical example can also be used to illustrate a statistical procedure, artificial data are convenient and informative. The prevalent paradigm in econometrics is to assume that the data are generated from a model. We, as researchers, check how close the estimate is to the model characterized by a set of unknown parameters. In simulations we have full control of the data generation process, including the true parameter. In a real example, however, we have no knowledge about the true model, so we cannot directly evaluate the quality of parameter estimation.

(It would be a different story if we are mostly interested in prediction, as we often encounter in machine learning. In such cases, we can split the data into two parts: one part for modeling and estimation, and the other for verification.)

Example

In OLS theory, the classical views X as fixed regressions and only cares about the randomness of the error term. Modern econometrics textbook emphasizes that a random X is more appropriate for econometrics applications. In rigorous textbooks, the moment of X is explicitly stated as $E[X_i X_i'] < \infty$. *Is asymptotic inferential theory for the OLS estimator—consistency and asymptotic normality—valid when X follows a Pareto distribution with shape coefficient 1.5?* A Pareto distribution with shape coefficient between 1 and 2 has finite population mean but infinite variance. Therefore this case violates the assumptions for OLS stated in most of econometric textbooks.

We write a script to investigate this problem. The following steps develop the code.

1. given a sample size, get the OLS `b_hat` and its associated `t_value`.
2. wrap `t_value` as a user-defined function so that we can reuse it for many times.
3. given a sample size, report the size under two distributions.
4. wrap step 3 again as a user-defined function, ready for different sample sizes.
5. develop the super structure to connect the workhorse functions.
6. add comments and documentation.

```
# the workhorse functions
simulation <- function(n, type = "Normal", df = df) {
  # a function gives the t-value under the null
  if (type == "Normal") {
    e <- rnorm(n)
  } else if (type == "T") {
    e <- rt(n, df)
  }

  X <- cbind(1, VGAM::rpareto(n, shape = 1.5))
  Y <- X %*% b0 + e
  rm(e)

  bhat <- solve(t(X) %*% X, t(X) %*% Y)
  bhat2 <- bhat[2] # parameter we want to test

  e_hat <- Y - X %*% bhat
```

```
  sigma_hat_square <- sum(e_hat^2) / (n - 2)
  sig_B <- solve(t(X) %*% X) * sigma_hat_square
  t_value_2 <- (bhat2 - b0[2]) / sqrt(sig_B[2, 2])

  return(t_value_2)
}

# report the empirical test size
report <- function(n) {
  # collect the test size from the two distributions
  # this function contains some repetitive code, but is OK for such a simple one
  TEST_SIZE <- rep(0, 3)

  # e ~ normal distribution, under which the t-dist is exact
  Res <- plyr::ldply(.data = 1:Rep, .fun = function(i) simulation(n, "Normal"))
  TEST_SIZE[1] <- mean(abs(Res) > qt(.975, n - 2))
  TEST_SIZE[2] <- mean(abs(Res) > qnorm(.975))

  # e ~ t-distribution, under which the exact distribution is complicated.
  # we rely on asymptotic normal distribution for inference instead
  Res <- plyr::ldply(.data = 1:Rep, .fun = function(i) simulation(n, "T", df))
  TEST_SIZE[3] <- mean(abs(Res) > qnorm(.975))

  return(TEST_SIZE)
}

## the super structure
# set the parameters
Rep <- 1000
b0 <- matrix(1, nrow = 2)
df <- 1 # t dist. with df = 1 is Cauchy

# run the calculation of the empirical sizes for different sample sizes
NN <- c(5, 10, 200, 5000)
RES <- plyr::ldply(.data = NN, .fun = report)
names(RES) <- c("exact", "normal.asym", "cauchy.asym") # to make the results readable
RES$n <- NN
RES <- RES[, c(4, 1:3)] # beautify the print
print(RES)
```

```
##      n exact normal.asym cauchy.asym
## 1    5 0.058       0.179       0.167
## 2   10 0.056       0.092       0.096
## 3  200 0.047       0.049       0.044
## 4 5000 0.049       0.049       0.025
```

The first column is the when the error is normal, and we use the exactly distribution theory to find the critical value (according to the t distribution.) The second column still uses the normal

distribution in the error term, but change the critical value to be from the normal distribution, which is based on asymptotic approximation. When sample size is small, obvious size distortion is observed; but the deviation is mitigated when the sample size increases. When the error distribution is Cauchy, the so called "exact distribution" is no longer exact— it is exact only if the error is normal and independent from x. If we attempt to use the asymptotic normal approximation, we find that the asymptotic approximation breaks down. The test size does not converge to the nominal 5% as the sample size increases.

In this simulation design, X is always Pareto while we vary the distribution of the error term. When we look at the table, we witness that the distribution of X indeed does not matter. Why? Since

$$\sqrt{n}(\hat{\beta} - \beta_0)|X = (X'X/n)^{-1}(X'e/\sqrt{n}),$$

the k-th element of the vector coefficient conditional on X is

$$\widehat{\beta}_k|X = \eta_k'\widehat{\beta}|X \sim N\left(\beta_k, \sigma^2 \left(X'X\right)_{kk}^{-1}\right).$$

The t-statistic

$$T_k = \frac{\widehat{\beta}_k - \beta_k}{\sqrt{s^2\left[(X'X)^{-1}\right]_{kk}}} = \frac{\widehat{\beta}_k - \beta_k}{\sqrt{\sigma^2\left[(X'X)^{-1}\right]_{kk}}} \cdot \frac{\sqrt{\sigma^2}}{\sqrt{s^2}} = \frac{\left(\widehat{\beta}_k - \beta_k\right)/\sqrt{\sigma^2\left[(X'X)^{-1}\right]_{kk}}}{\sqrt{\frac{e'}{\sigma}M_X\frac{e}{\sigma}/\left(n-K\right)}}.$$

Even though $X'X/n$ does not converge to a stable probabilistic limit, the self-normalized t statistic does not break down. Regardless the distribution of X, when the error term is normal, the numerator of the above expression follows a standard normal distribution and the demonimator follows a χ^2, and moreover these two components are independent. The resulting statistic follows a t-distribution.

Bootstrap

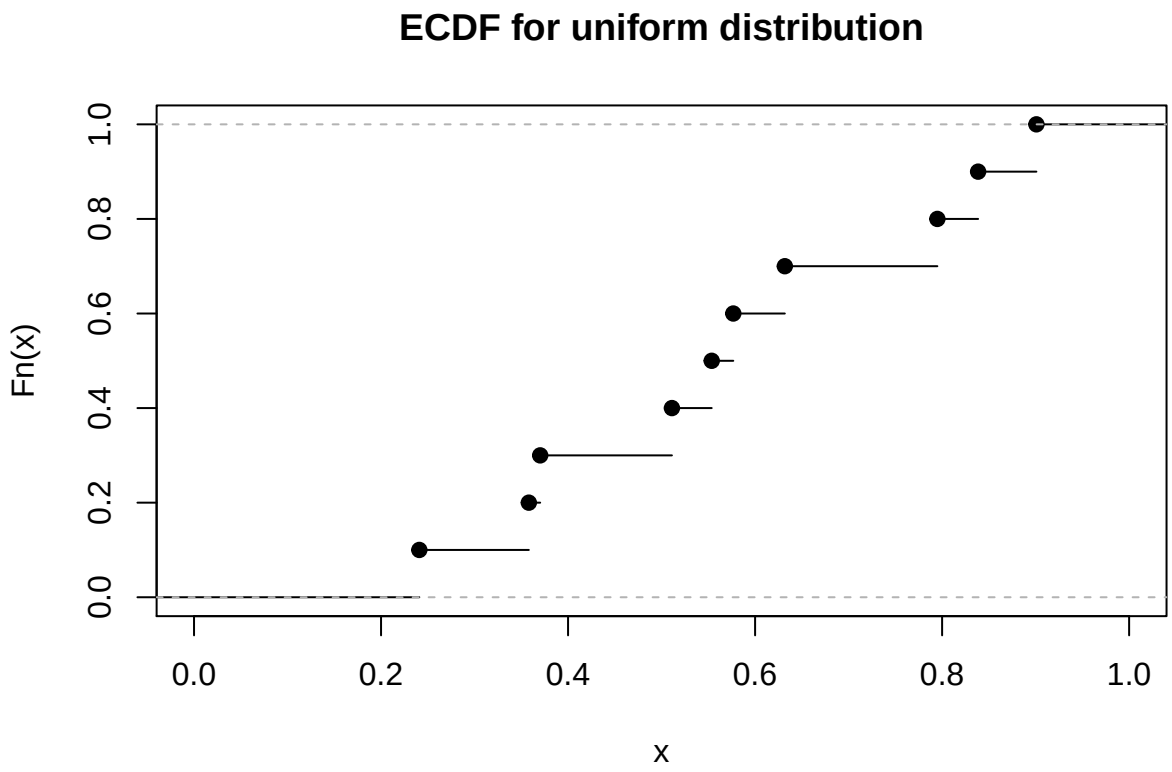
Bootstrap, originated from Efron (1979), is an extremely powerful and influential idea for estimation and inference. Here we give a brief introductoin. Textbook exposition can be found in Cameron and Trivedi (2005) (Chapter 11).

Let $X_1, X_2, \ldots, X_n \sim F$ be an i.i.d. sample of n observations following a distribution F. The finite sample distribution of a statistic $T_n(\theta) \sim G_n(\cdot, F)$ usually depends on the sample size n, as well as the known true distribution F. Asymptotic theory approximates $G_n(\cdot, F)$ by its limit

$$G(\cdot, F) := \lim_{n\to\infty} G_n(\cdot, F).$$

In particular, if $T_n(\theta)$ is *asymptotically pivotal*, then $G_n(\cdot, F)$ is independent of F and it becomes $G(\cdot)$.

```
runif(10) %>%
  ecdf() %>%
  plot(, xlim = c(0, 1), main = "ECDF for uniform distribution")
```



Instead of referring to the limiting distribution, Bootstrap replaces the unknown distribution F in $G_n(\cdot, F)$ by a consistent estimator F_n of the true distribution, for example, the empirical distribution function $\hat{F}_n(\cdot) = n^{-1} \sum_{i=1}^{n} 1\{\cdot \leq X_i\}$. Bootstrap inference is drawn from the bootstrap distribution

$$G_n^*(\cdot) := G_n(\cdot, \hat{F}_n)$$

Implementation of bootstrap is a simulation exercise. In an i.i.d. environment we sample over each observation with equal weight, which is called *nonparametric bootstrap*. However, the species of bootstrap creatures has many varieties adapted to their living environment. In a dependent dataset such as time series (Chang 2004), clustering data or networks, we must adjust the sampling schedule to preserve the dependence structure. In regression context if we hold the indepdendent variables fixed and only bootstrap the residual, we call it *parametric bootstrap*. If the error term is heteroskedascity, the relationship between X and $\hat{e}$ can be preserved by *wild bootstrap* (Davidson and MacKinnon 2010).

```
n <- 9 # sample size
boot_Rep <- 3 # bootstrap 3 times

real_sample <- rnorm(n) # the real sample
d0 <- tibble(no = 1:n, x = real_sample)
print(d0)
```

```
## # A tibble: 9 x 2
##      no      x
```

```
##   <int>  <dbl>
## 1     1 -0.855
## 2     2 -1.20
## 3     3 -0.335
## 4     4  0.366
## 5     5 -1.97
## 6     6  0.629
## 7     7 -1.18
## 8     8  1.96
## 9     9 -0.801
```

```
d_boot <- list() # save the bootstrap sample
for (b in 1:boot_Rep) {
  boot_index <- sample(1:n, n, replace = TRUE)
  d_boot[[b]] <- tibble(no = boot_index, x = real_sample[boot_index])
}

d_boot %>% as_tibble(, .name_repair = "minimal") %>% print()
```

```
## # A tibble: 9 x 3
##   ``$no     $x ``$no     $x ``$no     $x
##   <int>  <dbl> <int>  <dbl> <int>  <dbl>
## 1     6  0.629     6  0.629     8  1.96
## 2     8  1.96      2 -1.20      6  0.629
## 3     8  1.96      2 -1.20      9 -0.801
## 4     5 -1.97      5 -1.97      8  1.96
## 5     9 -0.801     1 -0.855     1 -0.855
## 6     3 -0.335     3 -0.335     7 -1.18
## 7     9 -0.801     2 -1.20      8  1.96
## 8     1 -0.855     4  0.366     3 -0.335
## 9     7 -1.18      9 -0.801     4  0.366
```

In many regular cases, it is possible to show in theory the *consistency* of bootstrap: the statistic of interest and its bootstrap version converge to the same asymptotic distribution, or $G_n^*(a) \xrightarrow{p} G(a)$ for a such that $G(a)$ is continuous. However, bootstrap consistency can fail when the distribution of the statistic is discontinuous in the limit. Bootstrap is invalid in such cases. For instance, naive bootstrap fails to replicate the asymptotic distribution of the two-stage least squares estimator under weak instruments. More sophisticated alternatives are need to fix the inconsistency of bootstrap, which we don't mention in this lecture.

Bootstrap Estimation

Bootstrap is simple enough to be done by a `ply`-family function for repeated simulations. Alternatively, R package boot provides a general function `boot()`.

Bootstrap is useful when the analytic formula of the variance of an econometric estimator is too complex to derive or code up.

Example

One of the most popular estimators for a sample selection model is Heckman (1977)'s two-step method. Let the outcome equation be

$$y_i = x_i\beta + u_i$$

and the selection equation be

$$D_i = z_i\gamma + v_i$$

To obtain a point estimator, we simply run a Probit in the selection model, predict the probability of participation, and then run an OLS of y_i on x_i and $\lambda(\hat{D}_i)$ in the outcome model, where $\lambda(\cdot)$ is the inverse Mill's ratio. However, as we can see from Heckman (1979)'s original paper, the asymptotic variance expression of the two-step estimator is very complicated. Instead of following the analytic formula, we can simply bootstrap the variance.

```
# the dataset comes from Greene( 2003 ): example 22.8, page 786
library(sampleSelection)
data(Mroz87)
# equations
selection_eq <- lfp ~ -1 + age + faminc + exper + educ
outcome_eq <- wage ~ exper + educ

# Heckman two-step estimation
heck <- sampleSelection::heckit(selection_eq, outcome_eq, data = Mroz87)
print(lmtest::coeftest(heck))
```

```
##
## z test of coefficients:
##
##                   Estimate  Std. Error z value  Pr(>|z|)
## age           -3.8037e-02  4.7335e-03 -8.0357 9.306e-16 ***
## faminc         1.0433e-05  4.3179e-06  2.4163   0.01568 *
## exper          7.4747e-02  7.1535e-03 10.4491 < 2.2e-16 ***
## educ           6.3923e-02  1.6283e-02  3.9257 8.646e-05 ***
## (Intercept)   -1.9611e+00  1.7375e+00 -1.1287   0.25904
## exper          1.6135e-02  3.3126e-02  0.4871   0.62619
## educ           4.8222e-01  8.0096e-02  6.0205 1.739e-09 ***
## invMillsRatio -3.0237e-01  9.6202e-01 -0.3143   0.75328
## sigma          3.1080e+00           NA      NA        NA
## rho           -9.7290e-02           NA      NA        NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Below is the function for a single bootstrap. For convenience, we save the point estimates of `heckit` but ignore the estimated variance. (This is a lazy option though, it must produce the same result if we take the effor to manually program the Heckit point estimation.) Implementation is just a repeated evaluation.

```r
n <- nrow(Mroz87)
boot_heck <- function() {
    indices <- sample(1:n, n, replace = T)  # resample the index set
    Mroz87_b <- Mroz87[indices, ]  # generate the bootstrap sample
    heck_b <- sampleSelection::heckit(selection_eq, outcome_eq, data = Mroz87_b)
    return(coef(heck_b))
}
# repeat the bootstrap
boot_Rep <- 199
Heck_B <- plyr::ldply(.data = 1:boot_Rep, .fun = function(i) boot_heck())

# collect the bootstrap outcomes
Heck_b_sd <- apply(Heck_B, 2, sd)[1:7]
print(Heck_b_sd)
```

```
##          age       faminc        exper         educ  (Intercept)
## 4.931608e-03 4.973624e-06 7.794018e-03 1.739699e-02 2.453931e+00
##        exper         educ
## 4.512413e-02 9.909452e-02
```

The standard errors from the analytical expression and those from bootstrap are comparable. Both are asymptotically consistent. The bootstrap estimates can also be used to directly compute the confidence intervals.

Bootstrap Test

Bootstrap is particularly helpful in inference. Indeed, we can rigorously prove that bootstrap enjoys high-order improvement relative to analytic asymptotic approximation if the test statistic is asymptotically pivotal (Hall and Horowitz 1996). Loosely speaking, if the test statistic is asymptotically pivotal, a bootstrap hypothesis testing can be more accurate than its analytic asymptotic counterpart.

Example

We use bootstrap to test a hypothesis about the population mean. The test is carried out by a t-statistic. The distribution of the sample is either *normal* or *zero-centered chi-square*. It will show that the bootstrap test size is more precise than that of the asymptotic approximation.

We first prepare the workhorse functions.

```r
# the t-statistic for a null hypothesis mu
T_stat <- function(Y, mu) sqrt(n) * (mean(Y) - mu)/sd(Y)

# the bootstrap function
boot_test <- function(Y, boot_Rep) {
    # INPUT Y: the sample boot_Rep: number of bootstrap replications

    n <- length(Y)
    boot_T <- rep(0, boot_Rep)

    # bootstrap in action
```

```
    for (r in 1:boot_Rep) {
        indices <- sample.int(n, n, replace = T)  # resampling the index
        resampled_Y <- Y[indices]  # construct a bootstrap artificial sample
        boot_T[r] <- abs(T_stat(resampled_Y, mean(Y)))
        # the bootstrapped t-statistic mu is replaced by 'mean(Y)' to mimic the
        # situation under the null
    }

    # bootstrap critical value
    boot_critical_value <- quantile(boot_T, 1 - alpha)
    # bootstrap test decision
    return(abs(T_stat(Y, mu)) > boot_critical_value)
}
```

A key point for bootstrap test is that the null hypothesis must be imposed no matter the hypothesized parameter is true value or not. Therefore the bootstrap t-statistic is

$$T_n^* = \frac{\bar{X}^* - \bar{X}}{s^*/\sqrt{n}}.$$

That is, the bootstrap t-statistic is centered at $\bar{X}$, the sample mean of F_n, rather than θ, the population mean of F. This is because in the bootstrap world the "true" distribution is F_n. If we wrongly center the bootstrap t-statistic at θ, then the test will have no power when the null hypothesis is false.

The following chuck of code report the rejection probability from three decision rules.

```
compare <- function() {
    # this function generates a sample of n observations and it returns the
    # testing results from three decision rules

    if (distribution == "normal") {
        X <- rnorm(n)
    } else if (distribution == "chisq") {
        X <- rchisq(n, df = 3) - 3
    }

    t_value_X <- T_stat(X, mu)  # T-statistic

    # compare it to the 97.5% of t-distribution
    exact <- abs(t_value_X) > qt(0.975, df = n - 1)
    # compare it to the 97.5% of normal distribution
    asym <- abs(t_value_X) > 1.96
    # decision from bootstrap
    boot_rule <- boot_test(X, boot_Rep)

    return(c(exact, asym, boot_rule))
}
```

```r
# set the parameters
n <- 20
distribution <- "normal"
boot_Rep <- 199
MC_rep <- 2000
alpha <- 0.05
mu <- 0

# Monte Carlo simulation and report the rejection probability
res <- plyr::ldply(.data = 1:MC_rep, .fun = function(i) compare())
colnames(res) <- c("exact", "asym", "bootstrap")
print(colMeans(res))
```

```
##     exact      asym bootstrap
##    0.0425    0.0560    0.0435
```

Here the nominal size of the test is 5%. The program reports the empirical size —the ratio between the number of rejections to the total number of replications. The closer is the empirical size to the nominal size, the more accurate is the test. We find here the bootstrap test is more accurate than the asymptotic test.

When the underlying distribution is a χ^2, the exact distribution is difficult to derive analytically. However, we can still compare the asymptotic size with the bootstrap size.

```r
distribution <- "chisq"

res <- plyr::ldply(.data = 1:MC_rep, .fun = function(i) compare())
colnames(res) <- c("exact?", "asym", "bootstrap")
print(colMeans(res))
```

```
##    exact?      asym bootstrap
##    0.0635    0.0790    0.0550
```

Again, here the "exact test" is no longer exact. The asymptotic test works fairly reasonable, while the bootstrap is closer to the nominal size 5%.

Reading

Efron and Hastie: Ch 10 and 11

Reference

Cameron, A Colin, and Pravin K Trivedi. 2005. *Microeconometrics: Methods and Applications*. Cambridge University Press.

Chang, Yoosoon. 2004. "Bootstrap Unit Root Tests in Panels with Cross-Sectional Dependency." *Journal of Econometrics* 120 (2): 263–93.

Davidson, Russell, and James G MacKinnon. 2010. "Wild Bootstrap Tests for Iv Regression." *Journal of Business & Economic Statistics* 28 (1): 128–44.

Efron, B. 1979. "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics* 7 (1): 1–26.

Hall, Peter, and Joel L Horowitz. 1996. "Bootstrap Critical Values for Tests Based on Generalized-Method-of-Moments Estimators." *Econometrica*, 891–916.

Heckman, James J. 1977. "Sample Selection Bias as a Specification Error (with an Application to the Estimation of Labor Supply Functions)." National Bureau of Economic Research Cambridge, Mass., USA.