Simulation

Zhentao Shi

Feb 7, 2018

Simulation

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 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. We do not write a script of thousands of lines 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 perhaps encapsulate it into a user-defined function. 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 PC. How can he achieve a better approximation? Of course, we suppose that he would not google it.

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

```
n = 10000
Z =  matrix(   runif(n)  , ncol = 2 )
# uniform distribution ranging (0,1)

inside = mean(  sqrt( rowSums( (Z-.5)^2 ) )  <=  .5 )
# the center of the cirle is (0.5,0.5)
pi_hat = 4 * inside # the area of a circle = pi * r^2
print(pi_hat)
```

```
## [1] 3.184
```

Finite Sample Evaluation

Asymptotic theory is apparatus to approximate finite sample distributions. In the real world, nothing is infinite, so apparently all asymptotic results are fake. However, asymptotic theory is so far the best possible instrument 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 is convenient and boast advantages. The prevalent paradigm in statistics is to assume that the data are generated from a model. We, as researchers, check how close is the estimate to the model, which is often characterized by a set of unknown parameters. In simulation we have full control of the data generation process, so we also know 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 approach is to view 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. 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 mean but infinite variance.)

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

```
rm(list = ls( ) )
library(plyr)
library(VGAM) # for the Pareto distribution

set.seed(888)
# set the parameters
Rep = 1000
b0 = matrix(1, nrow = 2 )
df = 1 # t dist. with df = 1 is Cauchy

# 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, 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] )

  out = c(bhat2, t_value_2)
  names(out) = c("bhat2", "t_value")
  return( out )
}

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

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

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

  return(TEST_SIZE)
}


pts0 = Sys.time()
# run the calculation of the empirical sizes for different sample sizes
NN = c(5, 10, 200, 5000)
RES = ldply(.data = NN, .fun = report )
names(RES) = c("exact", "normal.asym", "cauchy.asym") # to make the results readable
```

```r
RES$n = NN
RES = RES[, c(4,1:3)] # beautify the results
print(RES)
```

```
##      n exact normal.asym cauchy.asym
## 1    5 0.049       0.143       0.150
## 2   10 0.066       0.107       0.086
## 3  200 0.045       0.049       0.035
## 4 5000 0.049       0.050       0.020
```

```r
print( Sys.time() - pts0 )
```

```
## Time difference of 5.198113 secs
```

When we look at the table, we witness that the distribution of X indeed does not matter. Why?

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

Even though $X'X/n$ does not converge, the self-normalized t statistic is immune of the problem. What matters is the distribution of the error term. 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 not 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.

Bootstrap

Bootstrap, originated from Efron (1979), is an extremely powerful and influential idea for estimation and inference.

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 depend on the sample size n, as well as the known true distribution F. Asymptotic theory approximate $G_n(\cdot, F)$ by its limit

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

if $T_n(\theta)$ if *asymptotically pivotal* then $G_n(\cdot, F)$ is independent of F.

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. Bootstrap inference is drawn from the bootstrap distribution

$$G_n^*(\cdot) := G_n(\cdot, F_n)$$

Implementation of bootstrap is almost always a Monte Carlo simulation. In i.i.d. environment we sample over each observation with equal weight, while in dependent dataset such as time series,

clustering data or networks, we must adjust the sampling schedule to preserve the dependence structure.

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 $G_n^*(a) \to G(a)$ 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.

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(1979)'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 bootstrap the variance.

```
library(plyr)
library(AER)
library(sampleSelection)
# the dataset comes from Greene( 2003 ): example 22.8, page 786

data(Mroz87)
# equations
selection_eq = lfp ~ age + faminc + exper + educ
outcome_eq = wage ~ exper + educ

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

```
##
## z test of coefficients:
##
```

```
##                   Estimate  Std. Error z value  Pr(>|z|)
## (Intercept)   -1.1385e-01  4.0442e-01 -0.2815  0.778319
## age           -3.6696e-02  6.7110e-03 -5.4680 4.552e-08 ***
## faminc         1.0319e-05  4.3393e-06  2.3780  0.017409 *
## exper          7.4511e-02  7.2030e-03 10.3444 < 2.2e-16 ***
## educ           6.8872e-02  2.3978e-02  2.8723  0.004075 **
## (Intercept)   -1.9500e+00  1.8023e+00 -1.0819  0.279277
## exper          1.6062e-02  3.3892e-02  0.4739  0.635554
## educ           4.8147e-01  8.2271e-02  5.8523 4.848e-09 ***
## invMillsRatio -3.0324e-01  9.8862e-01 -0.3067  0.759051
## sigma          3.1080e+00           NA      NA        NA
## rho           -9.7565e-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, I keep using `heckit` but only save the point estimates.

```
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 = heckit(selection_eq, outcome_eq, data = Mroz87_b)
    return(coef(heck_b))
}
```

Implementation is just a repeated evaluation.

```
# repeat the bootstrap
boot_Rep = 199
Heck_B = ldply(.data = 1:boot_Rep, .fun = function(i) boot_heck())

# collect the bootstrap outcomes
Heck_b_sd = apply(Heck_B, 2, sd)
print(Heck_b_sd)
```

```
##   (Intercept)           age        faminc         exper          educ
##  3.736464e-01  6.347649e-03  4.835453e-06  7.969160e-03  2.415344e-02
##   (Intercept)         exper          educ invMillsRatio         sigma
##  2.245913e+00  4.147212e-02  8.976462e-02  1.534939e+00  4.357649e-01
##           rho
##  4.327475e-01
```

Bootstrap Test

Bootstrap is particularly helpful in inference. Indeed, we can rigorously prove that bootstrap enjoys high-order improvement relative to asymptotic approximation if the test statistic is asymptotically pivotal. 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 via 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.

```
library(plyr)

# 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.

```
library(plyr)

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 9.75% of t-distribution
    exact = abs(t_value_X) > qt(0.975, df = n - 1)
    # compare it to the 9.75% of normal distribution
    asym = abs(t_value_X) > 1.96
    # decision from bootstrap
    boot_decision = boot_test(X, boot_Rep)

    return(c(exact, asym, boot_decision))
}

# set the parameters
n = 20
distribution = "normal"
boot_Rep = 199
MC_rep = 2000
alpha = 0.05
mu = 0

set.seed(111)

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

```
##     exact      asym bootstrap
##    0.0475    0.0660    0.0450
```

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 better than the asymptotic test.

When the underlying distribution is a χ^2, there is no explicit exact distribution. However, we can still compare the asymptotic size with the bootstrap size.

```
distribution = "chisq"
```

```r
set.seed(666)
# Simulate and report the rejection probability
res = ldply( .data = 1:MC_rep, .fun = function(i) compare())
colnames(res) = c("exact","asym", "bootstrap")
print( colMeans(res))
```

```
##     exact      asym bootstrap
##    0.0735    0.0875    0.0600
```

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