Lecture 3: Monte Carlo Simulation

Zhentao Shi

February 11, 2017

Monte Carlo simulation has been widely used for

- check the finite-sample performance of asymptotic theory
- generate non-standard distribution (MCMC)
- approximate integrals with no analytic expression (SMM)
- bootstrap

A Teaser

example: estimate π

```
n = 2 * 100000000
Z =  matrix( 2 * ( runif(n)-0.5 ) , ncol = 2 ) # uniform distribution centered at zero, ranging (-1,1)

inside = mean(  sqrt( rowSums( Z^2 ) )  <=  1 )
pi_hat = 4 * inside # the total area of the square base is 4.
print(pi_hat)
```

```
## [1] 3.141594
```

Finite Sample Evaluation

Example: the test size of the OLS estimator.

Question: Is the asymptotic theory valid when X follows a Cauchy distribution? This example also intends to illustrate the recursive process of code development.

1. given sample size, get OLS `b_hat` and its `t_value`
2. wrap the `t_value` so that we can replicate for many many times
3. give sample size, report the size under two distributions
4. wrap it over different sample sizes.
5. develop the super structure
6. add comments and documentation

```
rm(list = ls( ) )
library(plyr)

# set the parameters
Rep = 1000
b0 = matrix(1, nrow = 2 )
df = 1

# the workhorse functions
MonteCarlo = 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, rcauchy(n) )
  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( c(bhat2, t_value_2) )
}

# 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) MonteCarlo(n, "Normal")  )
  names(Res) = c("bhat2", "t_value")
  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) MonteCarlo(n, "T", df)  )
  names(Res) = c("bhat2", "t_value")
  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, 2000)
RES = ldply(.data = NN, .fun = report )
names(RES) = c("exact", "normal.asym", "t.asym")
RES$n = NN
RES = RES[, c(4,1:3)] # beautify the results
print(RES)
```

```
##      n exact normal.asym t.asym
## 1    5 0.059       0.153  0.156
## 2   10 0.047       0.087  0.100
## 3  200 0.056       0.056  0.042
## 4 2000 0.053       0.053  0.023
```

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

```
## Time difference of 2.788362 secs
```

Bootstrap

Bootstrap, originated from Efron (1979), is an extremely powerful and influential idea for statistical 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, bootstrap fails to replicate the asymptotic distribution of the two-stage least squares estimator under weak instruments.

Execution in R

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 Estimation

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
##  3.888299e-01  6.960423e-03  4.868586e-06  7.717776e-03  2.392319e-02
```

```
##   (Intercept)         exper          educ invMillsRatio         sigma
##  2.447683e+00  4.161005e-02  1.016779e-01  1.537508e+00  4.439006e-01
##           rho
##  4.361096e-01
```

Bootstrap Test

Bootstrap is particularly helpful in statistical inference. Indeed, it is possible to show in theory the higher-order improvement of bootstrap. Loosely speaking, if the test statistic is asymptotically pivotal, a bootstrap hypothesis testing can be more accurate than its analytic asymptotic counterpart.

Example: a bootstrap test for the population mean. The test is carried out via a t-statistic. The distribution of the sample is either *normal* or *zero-centered chi-square*. It shows 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) (mean(Y) - mu)/sqrt(var(Y)/n)

# 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.

```r
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 = 500
alpha = 0.05
mu = 0

# 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
##     0.038     0.050     0.040
```

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.

```r
distribution = "chisq"

# Monte Carlo simulation and report the rejection probability
res = ldply( .data = 1:MC_rep, .fun = function(i) compare())[2:3]
colnames(res) = c("asym", "bootstrap")
print( colMeans(res))
```

```
##      asym bootstrap
##     0.102     0.082
```