

Lab 4

A Quick Introduction to Simulation Psychology 310

Instructions. Submit your file as an R Markdown Rmd file.

Preamble. Today’s assignment involves looking at some elementary approaches to statistical simulation in R. Several problems are given embedded in the text, and colored red. Please solve these and submit both your answers and the R code that produced them. Use a random number seed at the start of each of your answers, so we can replicate your findings if we have any questions. Use 100,000 replications if you have a fast computer, or some smaller number (10,000) if you have an older, slower computer.

1 Introduction

In many cases in statistics (and in other areas of science), we have probability models, but we cannot find “closed form” analytic expressions for the answers to questions that are of scientific value.

Sometimes the answers exist, but we haven’t found them. In other cases, the answers do not yet exist, but will be found in the future. In still others, there will never be a closed form expression for the quantity we are interested in.

In such cases, we have a strategy open to us that can give us answers that are accurate enough to be useful, and get us out of the bind created by the unavailability of an analytic answer.

This strategy involves using the computer to simulate observations generated by our probability model, over, and over, and over. We collect these simulated observations, and, because of the law of large numbers, we then have an approximation to the probability that we are searching for.

In the following sections, we examine some fundamental strategies for implementing this *Monte Carlo* strategy.

2 The Tallest Boy

A fundamental situation in statistics is the process of sampling followed by ordering and selection. For example, we give 100 people a test and then select the person scoring highest on the test to receive a scholarship.

When sampling is followed by selection of an observation with a particular rank, the result is called an *order statistic*. So, the n th order statistic is the largest observation in a sample of size n .

Order statistic theory is complicated. Whole books are written about it. It

gets a whole lot more complicated if the original observations are not independent.

Suppose, for the time being, the observations are independent and normal. Here is a typical problem.

A high school has 400 boys in the 12th grade. The heights of boys that age have a normal distribution with a mean of 70 and a standard deviation of 3. What is the expected value of the height of the tallest boy in the class?

This is a problem in order statistic theory. The tallest boy in the class is the n th order statistic.

We can “build up” a calculation to allow us to simulate the distribution of the tallest boy.

First, let’s simulate a single class of size 400.

```
> set.seed(12345)
> one.class <- rnorm(400, 70, 3)
> head(one.class)

[1] 71.76 72.13 69.67 68.64 71.82 64.55
```

Now let’s select the tallest boy.

```
> set.seed(12345)
> max(rnorm(400, 70, 3))

[1] 78.24
```

He’s about 6 feet 6 and a quarter inches. Now, we could simulate a second class of size 400.

```
> max(rnorm(400, 70, 3))

[1] 79.99
```

He’s almost exactly 6 feet 8, quite a bit taller than the tallest boy from the preceding year. Clearly, the tallest boy, over repeated years, is a random variable. What does its distribution look like?

This is where the `replicate` function in R comes in.

Suppose I replicate 10 years.

```
> set.seed(12345)
> replicate(10, max(rnorm(400, 70, 3)))

[1] 78.24 79.99 77.37 79.87 78.28 77.79 78.27 78.11 77.78 77.29
```

I can use R to simulate 100,000 years of high school classes in just a few seconds. I can store the result in a variable called `tallest.boy`. Here we go

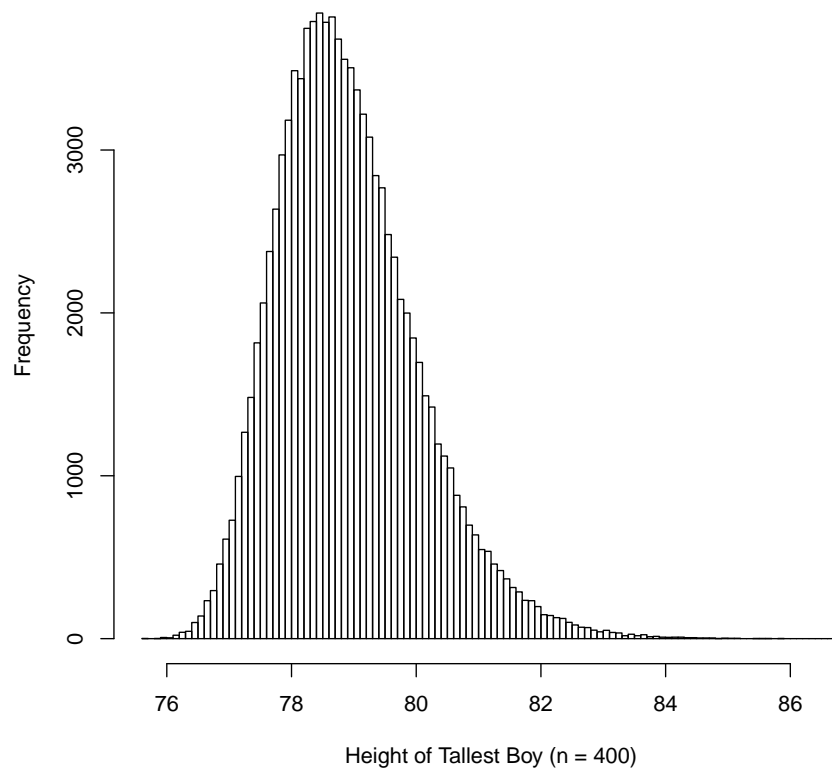
```
> set.seed(12345)
> tallest.boy <- replicate(1e+05, max(rnorm(400, 70, 3)))
```

(This took 4 seconds on my computer.)

Now we can examine the mean, standard deviation, and distributional shape easily.

```
> mean(tallest.boy)
[1] 78.91
> sd(tallest.boy)
[1] 1.133
> hist(tallest.boy, breaks = 100, xlab = "Height of Tallest Boy (n = 400)")
```

Histogram of tallest.boy



We can see that the distribution is skewed, with a mean of a little less than 6 feet 7 inches.

We could create a whole table of values of the average height of the tallest boy as a function of class size. We'd find that larger classes have tallest boys who tend to be taller than the tallest boy from smaller classes.

Let's explore that notion with a second question.

High school A has 400 boys, while high school B has only 100. What percentage of the time will the tallest boy from high school A be taller than the tallest boy from high school B if the two high schools draw independently and randomly from the same population?

With this problem we introduce a technique for extending the usefulness of a simulation. Again, let's break the problem down. Let's generate one class of size 400 and another of size 100 and select the tallest boy from each.

```
> set.seed(12345)
> max(rnorm(400, 70, 3))

[1] 78.24

> max(rnorm(100, 70, 3))

[1] 76.72
```

we can see that the larger class wins this time. There are several ways we can proceed to now examine 100,000 pairs of classes. One way is to store all the data from both classes, then do a side-by-side comparison.

```
> set.seed(12345)
> A <- replicate(1e+05, max(rnorm(400, 70, 3)))
> B <- replicate(1e+05, max(rnorm(100, 70, 3)))
> A.wins <- A > B # comparison via logical expression
> head(A.wins, 15) # first 15 years

[1] TRUE TRUE TRUE TRUE TRUE TRUE FALSE TRUE TRUE FALSE TRUE
[12] TRUE FALSE FALSE TRUE
```

The statement `A.wins <- A > B` examines the expression `A > B`. This evaluates to `TRUE` if A is greater than B, otherwise it evaluates to `FALSE`.

The neat part is that `TRUE` and `FALSE` are coded internally as 1 and 0, so taking the mean of the `A.wins` variable gives you the proportion of the time the larger high school had the tallest boy.

```
> mean(A.wins)

[1] 0.8025
```

Since this is a binomial proportion, we can set confidence limits as well, using the technique discussed in the Cases handout. (Actually, as we discover later in this exercise, there is a better way...)

An additional point is that, if you just need an answer in a hurry, you can write the whole problem in one line!

```

> set.seed(12345)
> mean(replicate(1e+05, max(rnorm(400, 70, 3))) > replicate(1e+05, max(rnorm(100,
+ 70, 3))))
[1] 0.8025

```

Question 1. Suppose that intelligence is one-dimensional, measured by IQ, and you are very smart. You have an IQ of 150, and the distribution of IQ scores in the general population is normal with a mean of 100 and a standard deviation of 15. If you walk into a room consisting of 25 people who were sampled independently and at random from the general population, what is the probability that you are the smartest person in the room?

Question 2. So how big would the number of people in the room have to be before the probability that you are the smartest person in the room got down to about 0.50?

Question 3. You throw two fair dice. What is the probability that you throw doubles? Write a simulation program to estimate this probability, then compare it to the true probability.

3 Statistical Assumptions, Robustness, and Monte Carlo Assessment of Error Rates

In the preceding section, we learned a neat trick for calculating the probability of an event happening. Code the event as a TRUE-FALSE variable, replicate it many times, and compute the mean of the outcomes.

This approach is widely utilized in Monte Carlo investigations of the *empirical rejection rates* for statistical tests, and *empirical coverage rates* for confidence intervals. We speak of the *nominal alpha* of a test when we refer to the planned significance level. Since many tests are based on asymptotic distribution theory, the actual alpha may be quite different from the nominal alpha.

In a similar vein, the actual proportion of times a confidence interval “covers” a parameter may be either higher or lower than the nominal confidence level.

Let’s look at an example of this. In class, we learned a basic technique for constructing a confidence interval on a single proportion. This technique works pretty well if N is large and p is not too far from 0.50, or, more generally, if Np and $N(1 - p)$ are both greater than 5.

This confidence interval, discussed in the Cases handout, is constructed as

$$\hat{p} \pm Z_{1-\alpha/2} \times \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

So if we observe a sample proportion of $\hat{p} = 0.65$ based on $n = 100$, a 95% confidence interval would be constructed as

$$0.65 \pm 1.96 \times \sqrt{\frac{0.65(0.35)}{100}}$$

An R function to compute this interval takes only a couple of lines:

```
> simple.interval <- function(phat, n, conf) {
+   z <- qnorm(1 - (1 - conf)/2)
+   dist <- z * sqrt(phat * (1 - phat)/n)
+   lower = phat - dist
+   upper = phat + dist
+   return(list(lower = lower, upper = upper))
+ }
> simple.interval(0.65, 100, 0.95)

$lower
[1] 0.5565

$upper
[1] 0.7435
```

The approach in the preceding example does not directly compensate for the fact that the standard error is estimated from the same data used to estimate the sample proportion. E.B. Wilson's approach asks, which values of p are barely far enough away from \hat{p} so that \hat{p} would reject them. These points are the endpoints of the Wilson confidence interval.

The Wilson approach requires us to solve the equations.

$$z = \frac{\hat{p} - p}{\sqrt{p(1-p)/n}} \quad (1)$$

and

$$-z = \frac{\hat{p} - p}{\sqrt{p(1-p)/n}} \quad (2)$$

Be careful to note that the denominator has p , *not* \hat{p} . If we square both of the above equations, and simplify by defining $\theta = z^2/n$, we arrive at

$$(\hat{p} - p)^2 = \theta p(1 - p) \quad (3)$$

This can be rearranged into a quadratic equation in p , which we learned how to solve in high school algebra with a simple if messy formula. The solution can be expressed as

$$p = \frac{1}{1 + \theta} \left(\hat{p} + \theta/2 \pm \sqrt{\hat{p}(1 - \hat{p})\theta + \theta^2/4} \right) \quad (4)$$

We can easily write an R function to implement this result.

```
> wilson.interval <- function(phat, n, conf) {
+   z <- qnorm(1 - (1 - conf)/2)
+   theta <- z^2/n
+   mult <- 1/(1 + theta)
+   dist <- sqrt(phat * (1 - phat) * theta + theta^2/4)
+   upper = mult * (phat + theta/2 + dist)
+   lower = mult * (phat + theta/2 - dist)
+   return(list(lower = lower, upper = upper))
+ }
> wilson.interval(0.65, 100, 0.95)
```

```

$lower
[1] 0.5525

$upper
[1] 0.7364

```

We have identified two different methods for constructing a confidence interval, and the natural question to ask is which method is better. There are a number of ways of characterizing the performance of confidence intervals. For example, we can examine how close the actual coverage probability is to the nominal value. In this case, we can, rather easily, compute the exact coverage probabilities for each interval, because R allows us to compute exact probabilities from the binomial distribution, and N is small. Therefore, we can

1. Compute every possible value of \hat{p}
2. Determine the confidence interval for that value
3. See whether the confidence interval contains the true value of p
4. Add up the probabilities for intervals that do cover p

```

> actual.coverage.probability <- function(n, p, conf) {
+   x <- 0:n
+   phat <- x/n
+   probs <- dbinom(x, n, p)
+   wilson <- wilson.interval(phat, n, conf)
+   simple <- simple.interval(phat, n, conf)
+   s <- 0
+   w <- 0
+   for (i in 1:n + 1) if ((simple$lower[i] < p) & (simple$upper[i] > p))
+     s <- s + probs[i]
+   for (i in 1:n + 1) if ((wilson$lower[i] < p) & (wilson$upper[i] > p))
+     w <- w + probs[i]
+   return(list(simple.coverage = s, wilson.coverage = w))
+ }
> actual.coverage.probability(30, 0.95, 0.95)

$simple.coverage
[1] 0.7821

$wilson.coverage
[1] 0.9392

```

Note that the Wilson interval is close to the nominal coverage level, while the traditional interval performs rather poorly.

Suppose that we had not realized that the exact probabilities were available to us. We could still get an excellent approximation of the exact probabilities by Monte Carlo simulation.

Monte Carlo simulation works as follows:

1. Choose your parameters

2. Choose a number of replications
3. For each replication:
 - (a) Generate data according to the model and parameters
 - (b) Calculate the test statistic or confidence interval
 - (c) Keep track of performance, e.g., whether the test statistic rejects, or whether the confidence interval includes the true parameter
4. Display the results

This “loop and accumulate” strategy is central to most complex Monte Carlo simulations. Study the code below carefully to see how the accumulator variables are zeroed out at the beginning, then added to as the `for-next` loop goes through all the replications.

In the function below, we simulate 10,000 Monte Carlo replications

```

> estimate.coverage.probability <- function(n, p, conf, reps, seed.value = 12345) {
+   ## Set seed, create empty matrices to hold results
+   set.seed(seed.value)
+   ## These are counting variables
+   coverage.wilson <- 0
+   coverage.simple <- 0
+   ## Loop through the Monte Carlo replications
+   for (i in 1:reps) {
+     ## create the simulated proportion
+     phat <- rbinom(1, n, p)/n
+     ## calculate the intervals
+     wilson <- wilson.interval(phat, n, conf)
+     simple <- simple.interval(phat, n, conf)
+     ## test for coverage, and update the count if successful
+     if ((simple$lower < p) & (simple$upper > p))
+       coverage.simple <- coverage.simple + 1
+     if ((wilson$lower < p) & (wilson$upper > p))
+       coverage.wilson <- coverage.wilson + 1
+   }
+   ## convert results from count to probability
+   s <- coverage.simple/reps
+   w <- coverage.wilson/reps
+   ## return as a named list
+   return(list(simple.coverage = s, wilson.coverage = w))
+ }
>
>
> ## Try it out
> estimate.coverage.probability(30, 0.95, 0.95, 10000)

$simple.coverage
[1] 0.7864

$wilson.coverage
[1] 0.9397

```

To get a better idea of the overall performance of the two interval estimation methods when $N = 30$, we might examine coverage rates as a function of p . With

our functions written, we are all set to go. We simply set up a vector of p values, and store the results as we go.

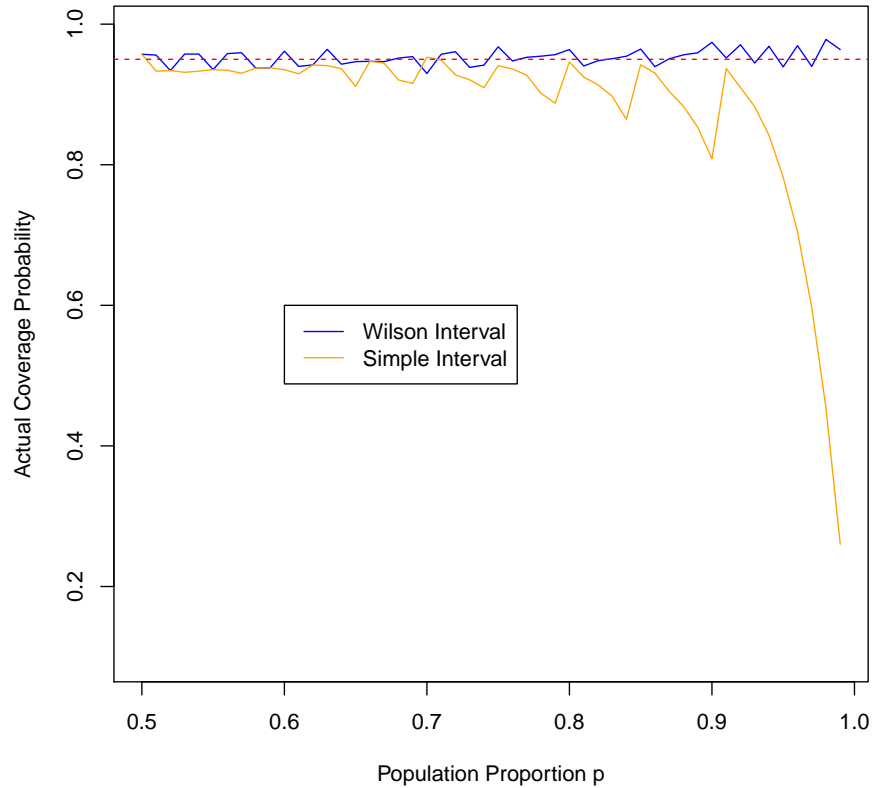
Here is some code:

```
> ## set up empty vectors to hold 50 cases
> p <- matrix(NA, 50, 1)
> wilson <- matrix(NA, 50, 1)
> simple <- matrix(NA, 50, 1)
> ## step from .50 to .99, saving results as we go
> for (i in 1:50) {
+   p[i] <- (49 + i)/100
+   res <- actual.coverage.probability(30, p[i], 0.95)
+   wilson[i] <- res$wilson.coverage
+   simple[i] <- res$simple.coverage
+ }
```

Below, we graph the results, presenting coverage probability as a function of p . The performance advantage of the Wilson interval when p gets close to 1 is obvious.

```
> plot(p, wilson, type = "l", col = "blue", ylim = c(0.1, 0.99), xlab = "Population Proportion p",
+   ylab = "Actual Coverage Probability", main = "Confidence Interval Performance (N = 30)")
> lines(p, simple, col = "orange")
>
> abline(0.95, 0, lty = 2, col = "red")
> legend(0.6, 0.6, c("Wilson Interval", "Simple Interval"), col = c("blue",
+   "orange"), lty = c(1, 1))
```

Confidence Interval Performance (N = 30)



Question 4. The two-sample t test assumes that population variances are equal. It is not robust to violations of this assumption if the sample sizes are unequal. To demonstrate this, write a Monte Carlo program to estimate, on the basis of 10,000 replications, the proportion of times a two-sample t will reject at the 0.05 significance level, two-tailed, when $n_1 = 100$, $n_2 = 20$, $\sigma_1 = 5$, $\sigma_2 = 15$, and both population means are 0. This is the empirical α .

Question 5. The coins in the box. Use Monte Carlo methods to answer the following question. There is a box with three drawers. Each drawer has two sides. One drawer has a gold coin in each side. One drawer has a silver coin in each side. The third drawer has a gold coin in one side and a silver coin in the other. You are blindfolded. A drawer is selected at random. You then choose a side at random and the coin in that side is revealed to be gold. What is the probability that given such a situation, the coin in the other side of the drawer is gold? Choose a clever representation of this game, and play it at least 10,000 times, keeping track of what happens. Note that you don't have to have a

physical representation of the box. You can just code it as a list of numbers and solve this problem in a few lines of code if you are careful!! Note: Legend has it that duels were actually fought over this problem. Some people believed that the probability is $1/2$, while others thought it was $2/3$.