For Loops

Contents

Simulating the Law of Large Numbers

The Law of Large Numbers says that as sample sizes increase, the mean of the sample will approach the true mean of the distribution. We are going to simulate this phenomenon!

We'll start by making a vector of sample sizes from 1 to 50, to represent increasing sample sizes.

Create a vector called `sample_sizes` that is made up of the numbers 1 through 50. You can use `seq()` or `:` notation.

```
set.seed(60637)
sample_sizes <- 1:50
```

We'll make an empty tibble to store the results of the for loop:

```
estimates <- tibble(n = integer(), sample_mean = double())
```

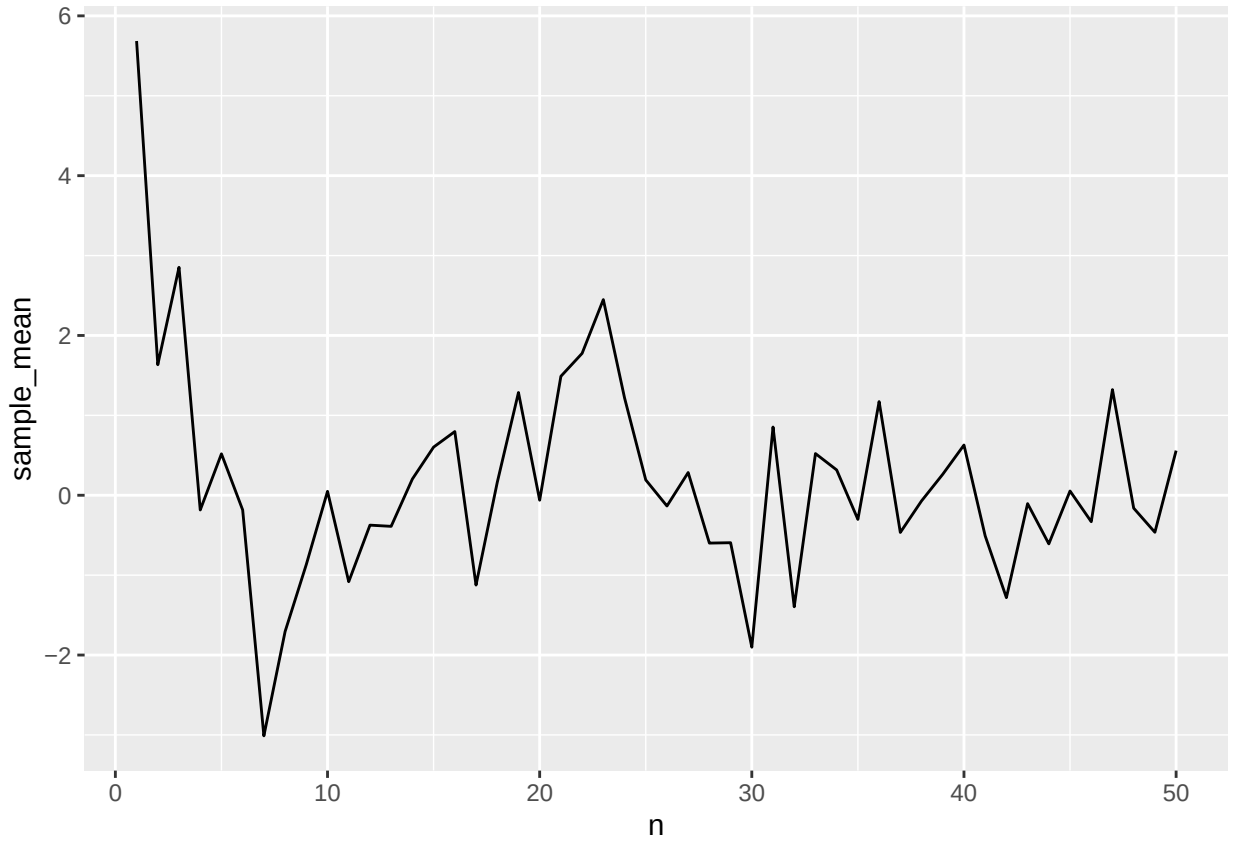
Write a loop over the `sample_sizes` you specified above. In the loop, for each sample size you will:

1. Calculate the mean of a sample from the random normal distribution with mean = 0 and sd = 5.
2. Make a named vector and the new rows to your tibble using bind_rows().

```
set.seed(60637)
for (n in sample_sizes) {
  sample_mean <- mean(rnorm(n, mean = 0, sd = 5))
  estimates <- bind_rows(estimates, c(n = n, sample_mean = sample_mean))
}
```

We can use ggplot2 to view the results. Fill in the correct information for the data and x and y variables, so that the `n` column of the `estimates` tibble is plotted on the x-axis, while the `sample_mean` column of the `estimates` tibble is plotted on the y-axis.

```
estimates %>%
  ggplot(aes(x = n, y = sample_mean)) +
  geom_line()
```



As the sample size (n) increases, the sample mean becomes closer to 0, or farther away from 0?

Answer: Closer.

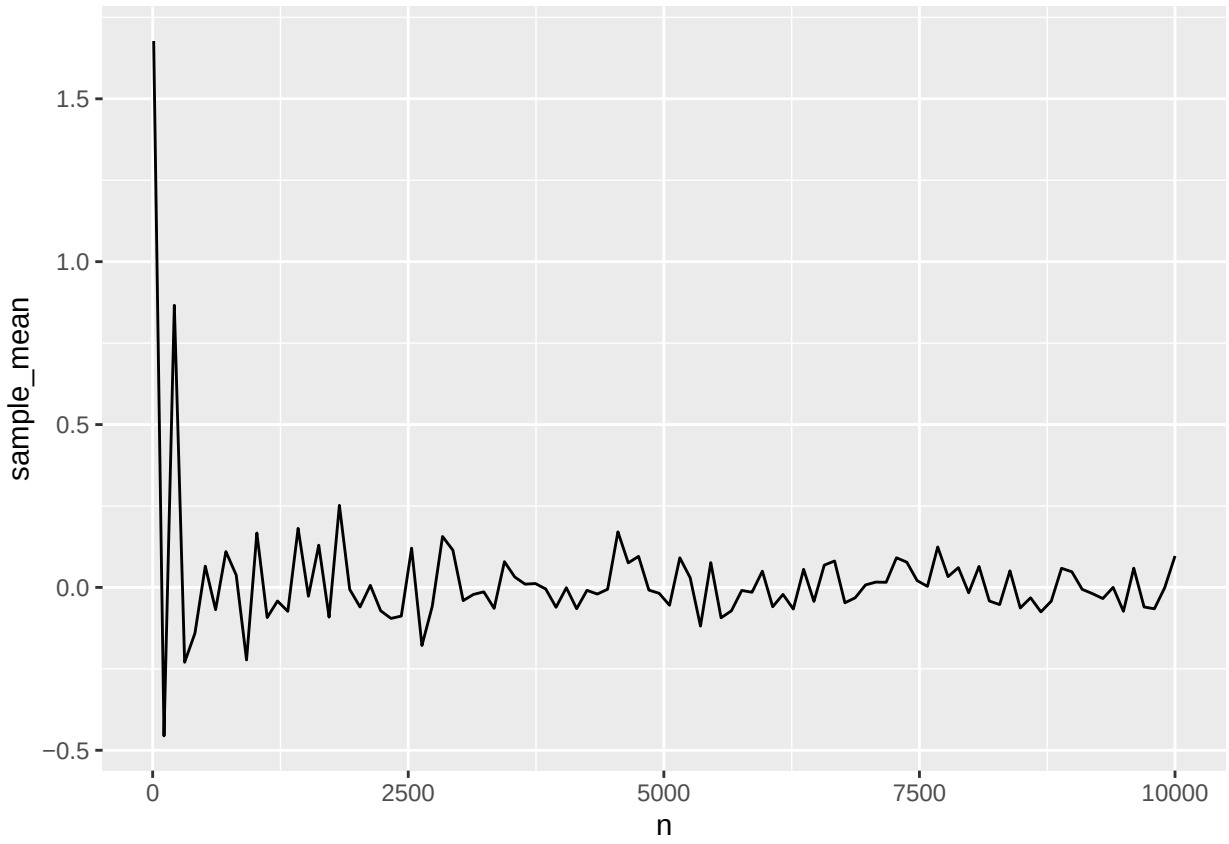
Rerun the above code with a wider range of sample sizes. Try several different sample size combinations. What happens when you increase the sample size to 100? 500? 1000? Feel free to use the `seq()` function to generate a sensibly spaced sequence. Play around with it!

Note: From our Monte Carlo lab, remember that no matter the size of the draw, you're still only taking one single draw for each sample size, so you might get an odd draw just by chance.

```
set.seed(60637)
sample_sizes <- seq(10, 10000, length.out = 100)
estimates <- tibble(n = integer(), sample_mean = double())

for (n in sample_sizes) {
  sample_mean <- mean(rnorm(n, mean = 0, sd = 5))
  estimates <- bind_rows(estimates, c(n = n, sample_mean = sample_mean))
}

estimates %>%
  ggplot(aes(x = n, y = sample_mean)) +
  geom_line()
```



How does this compare to before?

Answer: Just like before, we can see that for small samples, the mean fluctuates quite a bit. Past a certain point, larger samples seem to add little.

Extending Our Simulation

Looking at your results, you might think a small sample size is sufficient for estimating a mean, but your data had a relatively small standard deviation compared to the mean. Let's run the same simulation as before with different standard deviations.

Do the following:

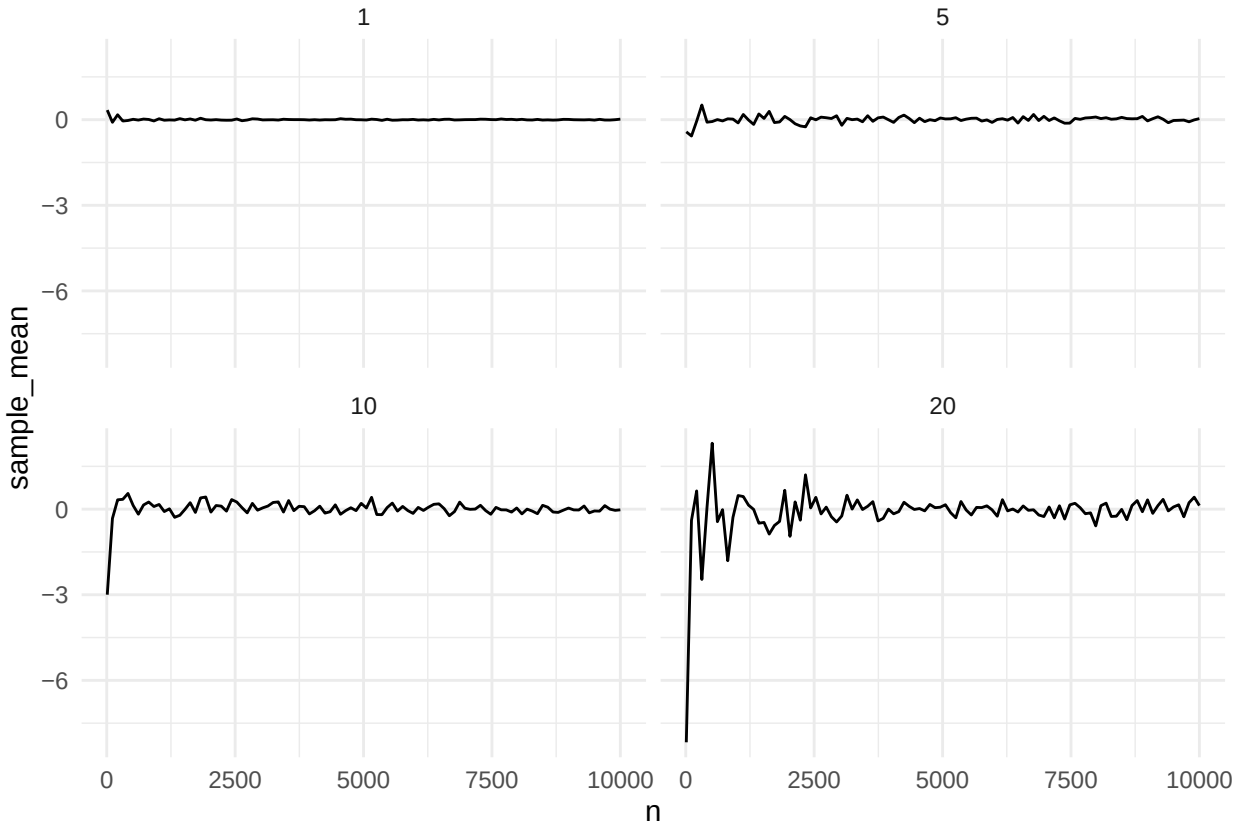
1. Create a vector called `population_sd` of length 4 with values 1, 5, 10, and 20 (you're welcome to add larger numbers if you wish).
2. Make an empty tibble to store the output. Compared to before, this has an extra column for the changing population standard deviations.
3. Write a loop inside a loop over `population_sd` and then `sample_sizes`.
4. Then, make a ggplot graph where the x and y axes are the same, but we facet (aka we create small multiples of individual graphs) on `population_sd`.

```
set.seed(60637)
population_sd <- c(1, 5, 10, 20)
estimates <- tibble(n = integer(), sample_mean = double(), population_sd = integer())

for (sd in population_sd){
  for (n in sample_sizes) {
```

```
    sample_mean <- mean(rnorm(n, mean = 0, sd = sd))
    estimates <- bind_rows(estimates, c(n = n, sample_mean = sample_mean, population_sd = sd))
  }
}

estimates %>%
  ggplot(aes(x = n, y = sample_mean)) +
  geom_line() +
  facet_wrap(~ population_sd) +
  theme_minimal()
```



How do these estimates differ as you increase the standard deviation?

Answer: We can see that as the standard deviation increases, the spread of our mean calculations also increases—the wider the distribution, the larger the draw we require for the sample mean to converge towards the true mean. This makes sense, because as the population standard deviation of the distribution we draw from increases, the likelier it is we draw extreme numbers.

Making the code more efficient.

At this point in your coding career, efficiency is minimally important. However, we discussed that loops in R are much slower if we build our output item by item (or row by row in this case.) We do better if we preallocate space.

```r
build_up <- function(sample_sizes = 1:30){
set.seed(60637)
estimates <- tibble(n = integer(), sample_mean = double())

for (n in sample_sizes) {
  sample_mean <- mean(rnorm(n, mean = 0, sd = 5))
  estimates <- bind_rows(estimates, c(n = n, sample_mean = sample_mean))
}
estimates
}

prealloc <- function(sample_sizes = 1:30){
  estimates <- vector("list", length(sample_sizes))
  set.seed(60637)
  for (i in seq_along(sample_sizes)) {
    sample_mean <- mean(rnorm(sample_sizes[i], mean = 0, sd = 5))
    # if you make this a tibble it's much slower!
    estimates[[i]] <- c(n = sample_sizes[i], sample_mean = sample_mean)
  }

  bind_rows(estimates)
}


bench::mark(prealloc(),
            build_up()) %>%
  select(expression, min, median, mem_alloc)
```

```
## # A tibble: 2 x 4
##   expression      min   median mem_alloc
##   <bch:expr> <bch:tm> <bch:tm> <bch:byt>
## 1 prealloc() 658.56us 808.44us     371KB
## 2 build_up()   3.25ms   3.95ms     120KB
```