The Perils of Micronumerosity: A simulation

Alex Stephenson

Purpose

This simulation aims to show the challenges of sample sizes, or what Goldberger referred to as "micronumerosity." Our simulation example replicates a simulation in Aronow and Miller (2019). They describe their simulation on page 133:

> "To investigate the small sample properties of the sample mean and associated measures of uncertainty, we will use computer simulations. We assume that we observe n i.i.d. draws of $X \sim U(0,1)$. To perform a simulation, we simulate n = 10, n = 100, or n = 1,000 draws from $U(0,1)$, and compute the sample mean, estimated standard error, and 90%/95%/99% normal approximation–based confidence intervals. We perform this simulation many, many times (here, we use 1 million simulations), and collect the estimates. Due to the law of large numbers, the empirical joint distribution of these quantities from our simulations approximates the true joint distribution of the estimators arbitrarily well. Our simulations are essentially plug-in estimates of the features of a random variable (an estimator) using a large number of simulated i.i.d. draws from its distribution" (Aronow and Miller 2019).

Aronow and Miller report these results.

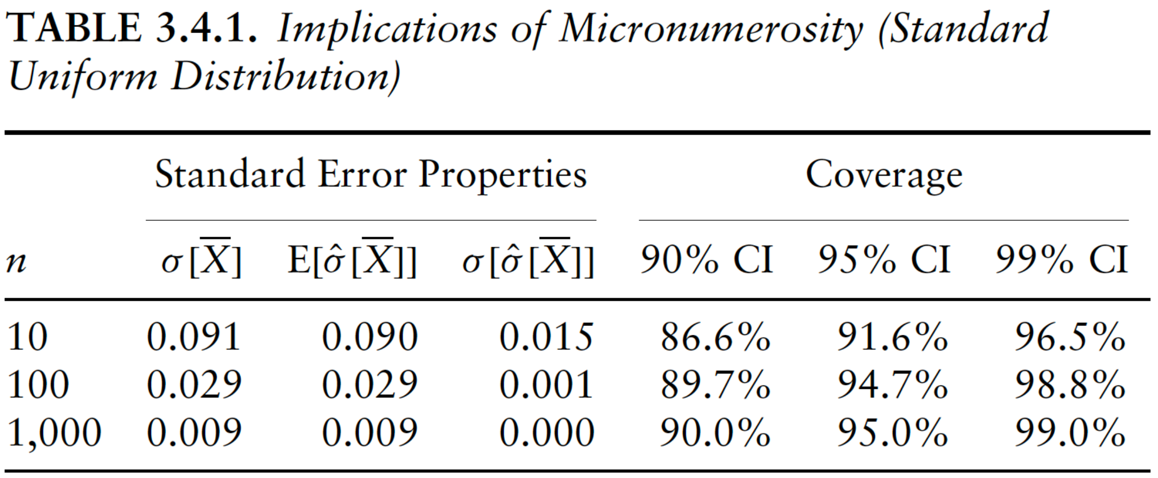

TABLE 3.4.1. *Implications of Micronumerosity (Standard Uniform Distribution)*

	Standard Error Properties			Coverage		
n	$\sigma[\overline{X}]$	$\mathrm{E}[\hat{\sigma}[\overline{X}]]$	$\sigma[\hat{\sigma}[\overline{X}]]$	90% CI	95% CI	99% CI
10	0.091	0.090	0.015	86.6%	91.6%	96.5%
100	0.029	0.029	0.001	89.7%	94.7%	98.8%
1,000	0.009	0.009	0.000	90.0%	95.0%	99.0%

Figure 1: Simulation Results: Aronow and Miller

The goal of this note is to replicate their results as closely as possible.

Functions

To simulate, we first need to build the architecture. Here are the functions I came up with to run a simulation.

```r
get_draws <- function(N){
  # A function to get sample draws from specified distributions
  # N = size of draw
  CI_90 <- 1.64
  CI_95 <- 1.96
  CI_99 <- 2.58
  s <- runif(N, 0, 1)
  avg <- mean(s)
  std_err <- sd(s)/sqrt(N)
  return(list(mean = avg, se = std_err,
                l_90 = avg - CI_90*std_err,
                u_90 = avg + CI_90*std_err,
                l_95 = avg-CI_95*std_err,
                u_95 = avg + CI_95*std_err,
                l_99 = avg-CI_99*std_err,
                u_99 = avg+CI_99*std_err))
}

in_interval <- function(val, lwr, upp){
  # A function that determines whether a given
  # value is inside an interval
  # val = a numeric value
  # lwr = numeric lower bound of interval
  # upp = numeric upper bound of interval
  if(lwr < val & val < upp){
    return(1)
  }
  return(0)
}

sim <- function(N, value){
  # Wrapper function to run a simulation with desired sample size N
  # N = an integer for size of sample
  # value = True mean value
  out <- get_draws(N)
  coverage90 <- in_interval(value, out$l_90, out$u_90)
  coverage95 <- in_interval(value, out$l_95, out$u_95)
  coverage99 <- in_interval(value, out$l_99, out$u_99)
  return(list(avg = out$mean,
              se = out$se,
              coverage90 = coverage90,
              coverage95 = coverage95,
              coverage99 = coverage99))
}

run_sim <- function(N,size, value){
  # Wrapper function to run the simulation for a given sample size
  # N = number of runs
  # size = sample size
  # value = true value of Expectation
  sim_sizes <- vector("list", N)
  for(i in 1:N){
    sim_sizes[[i]] <- sim(N=size, value = value)
  }
  sim_d <- map_dfr(sim_sizes,
                   `[`,
                   c("avg",
                     "se",
                     "in_int_90",
                     "in_int_95",
                     "in_int_99"))%>%
    mutate(e_se = sd(se),
         mean_se = mean(se))%>%
    summarise(across(everything(),
                   ~mean(.x, na.rm = TRUE)
                   ),
              sd_se = sd(se)
              )%>%
  select(se, mean_se, e_se,
         coverage90,
         coverage95,
         coverage99)
  return(sim_d)
}
```

A lot is going on at the end of the function. The important basic point is that we simulate many draws. The `map_dfr()` function comes from the `purrr` package. It is a way of pulling out each element of the list and turning it into a data frame. The next two columns add the expected mean of the standard error and the standard deviation of the standard error. Finally, we select the columns of interest and rename our coverage columns.

Running the simulation

We set a seed to ensure reproducibility. Setting a seed ensures that we get the same answer every time we run this code from scratch (or someone else runs it from scratch).

```
set.seed(42)
```

We run a million simulations. The simulation will take a long time because the for loop is slow. For future simulations of this nature, we should come up with a vectorized solution. Aronow and Miller claim they ran 1,000,000 simulations, so we do the same. That said, we can make this simulation point with an order of magnitude less.

```
df <- map_dfr(list(10,100,1000),
              ~run_sim(N=1000000,
                       size = .x,
                       value = 0.5))
```

Presentation

Finally, we will try for a reasonable replication of the results in the book. We take advantage of the `knitr::kable()` function to convert our data frame into a nice Latex table.

```
df %>%
  mutate(across(everything(), ~round(.x, 3)),
         across(starts_with("cover"),~100*.x))%>%
  tibble::add_column(n = c(10,100, 1000))%>%
  select(n, everything())%>%
  knitr::kable(
    caption = "Implications of Micronumerosity (Standard Uniform Distribution)",
    col.names =  c("n", "$\\hat{\\sigma}[\\bar{X}]$",
                   "$E[\\hat{\\sigma}[\\bar{X}]]$",
                   "$\\sigma[\\hat{\\sigma}[\\bar{X}]$",
                   "90\\% CI", "95\\% CI",
                   "99\\% CI"),
    escape = FALSE,
    booktabs = TRUE)%>%
  kable_styling(latex_options = "HOLD_position")%>%
  add_header_above(c(" " = 1,"Standard Error Properties" = 3, "Coverage" = 3))
```

Table 1: Implications of Micronumerosity (Standard Uniform Distribution)

	Standard Error Properties			Coverage		
n	$\hat{\sigma}[\bar{X}]$	$E[\hat{\sigma}[\bar{X}]]$	$\sigma[\hat{\sigma}[\bar{X}]$	90% CI	95% CI	99% CI
10	0.090	0.090	0.015	86.6	91.6	96.6
100	0.029	0.029	0.001	89.6	94.7	98.8
1000	0.009	0.009	0.000	89.8	95.0	99.0