# SURV-740 Homework 4: Randomization Inference

Namit Shrivastava

2025-10-30

### Question 1

```
set.seed(41279)
N <- 100000
mu_super <- 15
var_super <- 3
population <- rnorm(N, mean = mu_super, sd = sqrt(var_super))

pop_mean <- mean(population)
pop_var <- var(population) * (N - 1) / N
```

#### 1a

```
set.seed(41279)
N <- 100000
mu_super <- 15
var_super <- 3
population <- rnorm(N, mean = mu_super, sd = sqrt(var_super))

# Part (a)
pop_mean <- mean(population)
pop_var <- var(population) * (N - 1) / N

results_a <- tibble(
  pop_mean = pop_mean,
```

```
  pop_var = pop_var
)

results_a %>% mutate(across(everything(), fmt_num, digits = 5)) %>% kable()
```

| pop_mean | pop_var |
|---|---|
| 15.00332 | 2.99917 |

I simulated a finite population of size 100,000 from a Normal(15, 3) superpopulation. The simulated finite population mean is 15.00332 and the finite population variance is 2.99917. Now, because the draws come from a Normal distribution with mean 15 and variance 3, that Normal distribution here serves as the superpopulation model that justifies the finite population that I analyzed here.

**1b**

```
set.seed(41279)
n_sample <- 1000
sample_indices <- sample.int(N, size = n_sample, replace = FALSE)
sample_one <- population[sample_indices]

sample_mean <- mean(sample_one)
sample_var <- var(sample_one)
fpc <- sqrt(1 - n_sample / N)
se_mean <- sqrt(sample_var / n_sample) * fpc
ci_b <- sample_mean + qnorm(c(0.025, 0.975)) * se_mean

results_b <- tibble(
  estimate = sample_mean,
  se = se_mean,
  ci_lower = ci_b[1],
  ci_upper = ci_b[2]
)

results_b %>% mutate(across(everything(), fmt_num, digits = 5)) %>% kable()
```

| estimate | se | ci_lower | ci_upper |
|---|---|---|---|
| 15.02522 | 0.05424 | 14.91892 | 15.13153 |

In this single simple random sample without replacement, the estimated mean is 15.02522. Now after accounting for the finite population correction, the standard error is 0.05424, which in turn yields a 95% confidence interval of (14.91892, 15.13153).

**1c**

```
set.seed(41279)
B <- 2000

sim_results <- replicate(B, {
    idx <- sample.int(N, size = n_sample, replace = FALSE)
    samp <- population[idx]
    c(mean = mean(samp), variance = var(samp))
}) %>%
    t() %>%
    as_tibble()

quantile_means <- quantile(sim_results$mean, probs = c(0.025, 0.975))

quantile_tbl <- tibble(
    quantile = c("2.5%", "97.5%"),
    value = fmt_num(quantile_means, digits = 5)
)

quantile_tbl %>% kable(col.names = c("Quantile", "Estimated mean"))
```

| Quantile | Estimated mean |
|---|---|
| 2.5% | 14.89640 |
| 97.5% | 15.11060 |

```
plot_data <- sim_results %>%
    pivot_longer(cols = everything(), names_to = "statistic", values_to = "value")

ggplot(plot_data, aes(x = value)) +
```

```
    geom_histogram(bins = 40, color = "white", fill = "steelblue", alpha = 0.8) +
    facet_wrap(~ statistic, scales = "free", ncol = 1) +
    labs(x = "Value", y = "Frequency") +
    theme_minimal()
```

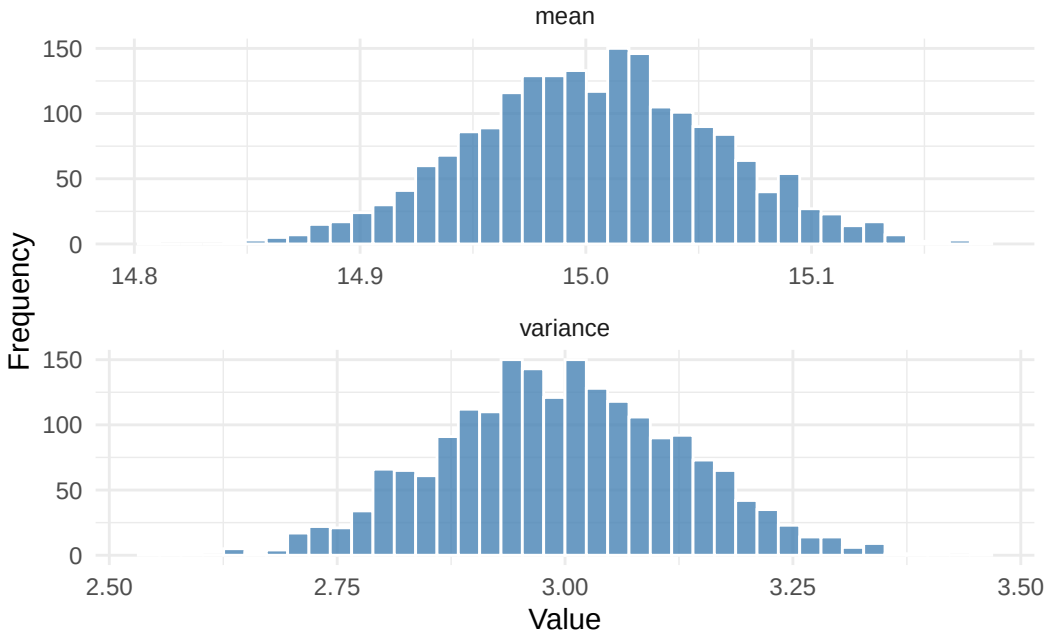


Figure 1: Sampling distribution approximations from 2,000 SRSWOR samples.

The approximate sampling distribution of the estimated mean is tightly centered around 15.00372, while the sampling distribution of the estimated variance is more skewed. The empirical 2.5th and 97.5th percentiles of the estimated mean are 14.89640 and 15.11060.

### 1d

The 95% confidence interval from the single sample in part 1b is (14.91892, 15.13153), whereas the central 95% of the sampling distribution of the estimated mean spans (14.89640, 15.11060). So, the percentile interval here summarizes how the estimator varies across repeated samples, while the confidence interval uses the estimated standard error from one sample to approximate the uncertainty about the true finite population mean. Now they target related but distinct quantities i.e., the former is an empirical description of estimator variability, while the latter is an inferential statement about the parameter.

Now to be more precise, the confidence interval (part b) makes an inferential statement about the unknown finite population mean, it estimates where likely falls based on one observed sample and uses the sample's standard error. Meanwhile, the percentile interval (part c) is a descriptive summary of the sampling distribution that shows where sample means actually fall across 2,000 repeated samples.

Now, the confidence interval's validity rests on the assumption that the sampling distribution is approximately normal, which actually the percentile interval empirically validates. So, both intervals are similar because the normal approximation works well for large samples (n=1,000), demonstrating that the theoretical standard error from part (b) accurately characterizes the variability observed in the simulated sampling distribution.

### 1e

```
set.seed(41279)
z_alpha <- qnorm(0.975)

ci_bounds <- sim_results %>%
    mutate(
        se = sqrt(variance / n_sample) * fpc,
        lower = mean - z_alpha * se,
        upper = mean + z_alpha * se,
        covers = (lower <= pop_mean) & (upper >= pop_mean)
    )

coverage_rate <- mean(ci_bounds$covers)

tibble(
    Coverage = fmt_num(coverage_rate, digits = 4)
) %>% kable()
```

| Coverage |
|---|
| 0.9480 |

Out of the 2,000 simulated samples, 1896.0000 of the confidence intervals covered the true finite population mean, giving an empirical coverage rate of 0.9480. This is extremely close to the nominal 95% level, differing by only 0.20 percentage points and validates that the normal approximation and finite population correction work well in this setting with large sample size (n=1,000) and also, demonstrates the procedure has the stated level of coverage.

## Question 2

```
blocks <- c(3, 5, 5, 52, 21, 34, 3, 0, 0, 0, 17, 14, 0, 0, 2, 54, 11, 11, 0, 23)
a <- length(blocks)
A <- 270
M <- 60

sample_mean_blocks <- mean(blocks)
sample_var_blocks <- var(blocks)
fpc_blocks <- sqrt(1 - a / A)
se_blocks <- sqrt(sample_var_blocks / a) * fpc_blocks

p_hat <- sample_mean_blocks / M
se_p <- se_blocks / M

logit <- function(p) log(p / (1 - p))
inv_logit <- function(x) 1 / (1 + exp(-x))

phi_hat <- logit(p_hat)
se_phi <- se_p / (p_hat * (1 - p_hat))
ci_phi <- phi_hat + qnorm(c(0.025, 0.975)) * se_phi
ci_p <- inv_logit(ci_phi)

q2_summary <- tibble(
    quantity = c("Sample mean rentals", "Standard error", "Rental proportion", "SE proportio
    value = c(sample_mean_blocks, se_blocks, p_hat, se_p, phi_hat, se_phi)
)

q2_summary %>% mutate(value = fmt_num(value, digits = 5)) %>% kable()
```

| quantity | value |
|---|---|
| Sample mean rentals | 12.75000 |
| Standard error | 3.59152 |
| Rental proportion | 0.21250 |
| SE proportion | 0.05986 |
| Logit estimate | -1.30992 |
| SE logit | 0.35770 |

Using the block-level data from Module 8, the sample mean number of rental units is 12.7500, which corresponds to a rental proportion of 0.2125 once I divide by the 60 dwellings in each

block. Applying the delta method to the logit transformation yields the following 95% confidence interval for logit(Q(Y)):

```
tibble(
    bound = c("Lower", "Upper"),
    logit_QY = ci_phi,
    QY = ci_p
) %>%
    mutate(across(-bound, fmt_num, digits = 5)) %>%
    kable()
```

| bound | logit_QY | QY |
|---|---|---|
| Lower | -2.01100 | 0.11805 |
| Upper | -0.60884 | 0.35232 |

So the estimated logit is -1.30992 with a 95% confidence interval from -2.01100 to -0.60884. Transforming back to the rental proportion gives a 95% confidence interval of (0.11805, 0.35232), or equivalently about (7.083, 21.139) rental units per block.

## Question 3

```
p_i <- blocks / M
total_units <- a * M
total_pop_units <- A * M

p_hat_clusters <- mean(p_i)
f_cluster <- a / A
s_p2 <- var(p_i)
var_cluster <- (1 - f_cluster) * s_p2 / a
se_cluster <- sqrt(var_cluster)
ci_cluster <- p_hat_clusters + qnorm(c(0.025, 0.975)) * se_cluster

f_unit <- total_units / total_pop_units
var_srs <- (1 - f_unit) * p_hat_clusters * (1 - p_hat_clusters) / total_units
se_srs <- sqrt(var_srs)

design_effect <- var_cluster / var_srs
effective_n <- total_units / design_effect
```

```r
q3_table <- tibble(
    quantity = c(
        "Estimated rental proportion", "SE (two-stage cluster)",
        "95% CI (two-stage cluster) lower", "95% CI (two-stage cluster) upper",
        "Variance (two-stage cluster)", "Variance (SRS of individuals)",
        "Design effect", "Effective sample size"
    ),
    value = c(
        p_hat_clusters, se_cluster, ci_cluster[1], ci_cluster[2],
        var_cluster, var_srs, design_effect, effective_n
    )
)

q3_table %>% mutate(value = fmt_num(value, digits = 5)) %>% kable()
```

| quantity | value |
|---|---|
| Estimated rental proportion | 0.21250 |
| SE (two-stage cluster) | 0.05986 |
| 95% CI (two-stage cluster) lower | 0.09518 |
| 95% CI (two-stage cluster) upper | 0.32982 |
| Variance (two-stage cluster) | 0.00358 |
| Variance (SRS of individuals) | 0.00013 |
| Design effect | 27.74911 |
| Effective sample size | 43.24463 |

### Answers

- **3a.** My estimate of the proportion of rental housing units is 0.21250.
- **3b.** The estimated variance under the two-stage cluster design is 0.003583, giving a 95% confidence interval of (0.09518, 0.32982).
- **3c.** Under an SRS of 1,200 individual housing units, the estimated variance would be only 0.000129. The design effect is 27.7491, so the effective sample size of the cluster design is about 43.24 respondents. In simple plain language, I would say that it means the 1,200 interviews collected via cluster sampling carry the same amount of information about the rental proportion as roughly 43 interviews from a simple random sample.
- **3d.** Now, because the design effect is so large, the two-stage cluster plan produces much higher sampling variance than an SRS of the same size. I would therefore prefer the SRS design if it were operationally feasible. If I naively analyzed the clustered data as

though they came from a simple random sample of 1,200 units, I would understate the true variance by about a factor of 27.75, leading to confidence intervals that are far too narrow and that would undercover the true population proportion.

## Question 4

### Original population Y = {5, 10, 35, 100}

```
population1 <- c(5, 10, 35, 100)
samples1 <- combn(population1, 2)

estimates1 <- tibble(
    sample = apply(samples1, 2, paste, collapse = ","),
    arithmetic = colMeans(samples1),
    harmonic = 2 / colSums(1 / samples1),
    geometric = sqrt(samples1[1, ] * samples1[2, ])
)

pop1_mean <- mean(population1)

moment_summary <- function(vec, true_mean) {
    tibble(
        bias = mean(vec) - true_mean,
        variance = mean((vec - mean(vec))^2),
        mse = mean((vec - true_mean)^2)
    )
}

moments1 <- bind_rows(
    moment_summary(estimates1$arithmetic, pop1_mean) %>% mutate(estimator = "Arithmetic"),
    moment_summary(estimates1$harmonic, pop1_mean) %>% mutate(estimator = "Harmonic"),
    moment_summary(estimates1$geometric, pop1_mean) %>% mutate(estimator = "Geometric")
) %>%
    relocate(estimator)

estimates1 %>% kable()
```

| sample | arithmetic | harmonic | geometric |
|---|---|---|---|
| 5,10 | 7.5 | 6.666667 | 7.071068 |
| 5,35 | 20.0 | 8.750000 | 13.228757 |

| sample | arithmetic | harmonic | geometric |
|---|---|---|---|
| 5,100 | 52.5 | 9.523810 | 22.360680 |
| 10,35 | 22.5 | 15.555556 | 18.708287 |
| 10,100 | 55.0 | 18.181818 | 31.622777 |
| 35,100 | 67.5 | 51.851852 | 59.160798 |

```
moments1 %>% mutate(across(-estimator, fmt_num, digits = 5)) %>% kable()
```

| estimator | bias | variance | mse |
|---|---|---|---|
| Arithmetic | 0.00000 | 477.08333 | 477.08333 |
| Harmonic | -19.07838 | 239.45707 | 603.44177 |
| Geometric | -12.14127 | 286.10160 | 433.51210 |

For this skewed population, only the arithmetic mean is unbiased. The harmonic and geometric means exhibit negative bias and much larger mean squared errors because the extreme value of 100 exerts a strong pull on the arithmetic scale but cannot be captured well by reciprocal or multiplicative averaging.

### Alternative population Y = {5, 10, 16, 18}

```
population2 <- c(5, 10, 16, 18)
samples2 <- combn(population2, 2)

estimates2 <- tibble(
    sample = apply(samples2, 2, paste, collapse = ","),
    arithmetic = colMeans(samples2),
    harmonic = 2 / colSums(1 / samples2),
    geometric = sqrt(samples2[1, ] * samples2[2, ])
)

pop2_mean <- mean(population2)

moments2 <- bind_rows(
    moment_summary(estimates2$arithmetic, pop2_mean) %>% mutate(estimator = "Arithmetic"),
    moment_summary(estimates2$harmonic, pop2_mean) %>% mutate(estimator = "Harmonic"),
    moment_summary(estimates2$geometric, pop2_mean) %>% mutate(estimator = "Geometric")
) %>%
```

```
    relocate(estimator)

estimates2 %>% kable()
```

| sample | arithmetic | harmonic | geometric |
|---|---|---|---|
| 5,10 | 7.5 | 6.666667 | 7.071068 |
| 5,16 | 10.5 | 7.619048 | 8.944272 |
| 5,18 | 11.5 | 7.826087 | 9.486833 |
| 10,16 | 13.0 | 12.307692 | 12.649111 |
| 10,18 | 14.0 | 12.857143 | 13.416408 |
| 16,18 | 17.0 | 16.941177 | 16.970563 |

```
moments2 %>% mutate(across(-estimator, fmt_num, digits = 5)) %>% kable()
```

| estimator | bias | variance | mse |
|---|---|---|---|
| Arithmetic | 0.00000 | 8.72917 | 8.72917 |
| Harmonic | -1.54703 | 13.36827 | 15.76157 |
| Geometric | -0.82696 | 10.84744 | 11.53130 |

With the milder population, all three estimators behave more similarly. The arithmetic mean remains unbiased with the smallest variance, while the harmonic and geometric means still have mild negative bias. Their mean squared errors shrink dramatically compared to the first population, reflecting the lower skewness.

### Overall comparison

For finite population inference in survey settings, the arithmetic mean is the most defensible choice because it is unbiased regardless of population shape. While the geometric mean can have lower MSE in highly skewed populations as is seen in Population 1, this advantage relies on knowing the population's skewness in advance, like something typically unavailable in practice. The harmonic mean should generally be avoided due to severe downward bias on skewed populations. Hence, the arithmetic mean's unbiasedness property, combined with its design-based justification in simple random sampling, makes it the preferred estimator for most applications.

## Question 5

- **Frequentist pro:** Frequentist procedures often come with guaranteed long-run properties (such as unbiasedness or nominal coverage), which makes it easy for me to communicate design-based error rates to stakeholders.
- **Frequentist con:** Those guarantees can rely on idealized sampling assumptions; when those assumptions fail like I was thinking for example, with clustering as in Question 3, the nominal properties can disappear unless I make careful adjustments.
- **Bayesian pro:** A Bayesian analysis lets me incorporate prior knowledge coherently and delivers full posterior distributions, so I can make probability statements about parameters directly.
- **Bayesian con:** Posterior results can be sensitive to the prior specification, and defending that prior choice to skeptical collaborators can sometimes be harder than justifying a design-based frequentist analysis.