HW 1

10.6)

Percentile confidence interval is

$C^{pc} = [q^*_{\alpha/2}, q^*_{1-\alpha/2}$

b/c this is CI uses quantiles, it respects monotonic transformations.

Let $m(q^*) = \hat{\theta} + s(\hat{\theta}) q^*$

$$\Rightarrow C = [m(q^*_{\alpha/2}), m(q^*_{1-\alpha/2})]$$
$$= [\hat{\theta} + s(\hat{\theta}) q^*_{\alpha/2}, \hat{\theta} + s(\hat{\theta}) q^*_{1-\alpha/2}]$$

Since C^{pc} respects monotonic transformations,

$$P(T^* \le q^*_{\alpha/2}) = P(\hat{\theta}^* \le \hat{\theta} + s(\hat{\theta}) q^*_{\alpha/2}) = \alpha/2$$

and likewise

$$P(T^* \ge q^*_{1-\alpha/2}) = P(\hat{\theta}^* \ge \hat{\theta} + s(\hat{\theta}) q^*_{1-\alpha/2}) = \alpha/2$$

10.10) $\mu = E(Y) > 0$ $\quad \theta = \mu^{-1}$

$\hat{\mu} = \bar{Y}_n$ $\quad \hat{\theta} = \hat{\mu}^{-1}$

a) For $\hat{\theta}$ to be unbiased for θ, $E(\hat{\theta}) = \theta$

$$E[\hat{\theta}] = E[\hat{\mu}^{-1}]$$
$$= E\left[\frac{1}{\hat{\mu}}\right]$$
$$\neq \frac{1}{E[\hat{\mu}]}$$

in general by Jensen inequality

So $\hat{\theta}$ is biased for θ

b) $E[\hat{\theta} - \theta] = E\left[\frac{1}{\hat{\mu}} - \frac{1}{\mu}\right]$

$$= E[1/\hat{\mu}] - E[1/\mu]$$

Since $\mu > 0$, and $1/x$ is strictly convex for positive x, by Jensens inequality

$$E[1/\hat{\mu}] > 1/E[\hat{\mu}] = \theta$$

So $E[\hat{\theta} - \theta]$ is upward biased

c) The percentile CI is appropriate when the sample parameter is symmetric around the true parameter which is not the case here. So it is not appropriate for this context

10.11)

b) (x^*, e^*)

$e^* = y^* - x^{*\prime}\hat{\beta}$

so $y^* = x^{*\prime}\hat{\beta} + e^*$

Given this, we get that $\hat{\beta}$ for the bootstrap estimator is

$\hat{\beta}^* = (X^{*\prime}X^*)^{-1}X^{*\prime}y^*$

Rearranging gives us

$\hat{\beta}^* = \hat{\beta} + (X^{*\prime}X^*)^{-1}X^{*\prime}e^*$

The parametric bootstrap gives us the estimator

$$\beta^* = (X^{*\prime} X^*)^{-1} X^{*\prime} y^*$$
$$= \hat\beta + (X^{*\prime} X^*)^{-1} X^{*\prime} \eta$$

We note that $\eta = e^*$, so these are identical.

10.23)

a)

$$C^{asy} = [\hat\theta - z_{1-\alpha/2}\, s(\hat\theta),\ \hat\theta + z_{1-\alpha/2}\, s(\hat\theta)]$$

$$\hat\theta = \hat\beta_1 \hat\beta_2$$

$$s(\hat\theta) = \sqrt{\frac{1}{n} \hat R' \hat V_\beta \hat R}$$

$$\hat R = \begin{pmatrix} \hat\beta_2 \\ \hat\beta_1 \end{pmatrix}$$

$$\hat V_\beta = \left(\frac{1}{n} \sum x_i x_i'\right)^{-1} \left(\frac{1}{n} \sum x_i x_i' \hat e_i^2\right) \left(\frac{1}{n} \sum x_i x_i'\right)$$

b) $C^{percentile} = [q^*_{\alpha/2}, q^*_{1-\alpha/2}]$

We obtain $q^*_{\alpha/2}, q^*_{1-\alpha/2}$ by running the bootstrap algorithm and obtaining a distribution of $\hat{\theta}^* = \hat{\beta}_1^* \hat{\beta}_2^*$.

Then $q^*_{\alpha/2}, q^*_{1-\alpha/2}$ are the $\alpha/2$ and $1-\alpha/2$ percentiles of that distribution

10.25) No. w/ heteroskedastic errors, Up is biased and inconsistent w/ regular asymptotic theory. However, the bootstrapping procedure's iid sampling loses the heteroskedasticity, so we don't have to worry about it.

HW1

2023-03-30

```r
# Load packages
pacman::p_load(tidyverse, boot, car)
```

Question 1 Monte Carlo Simulation

```r
# Set true values of parameters
y_true = c(1, 1)
sigma = matrix(diag(c(0.25^2,1)), ncol = 2)
theta = y_true[1]/y_true[2]

# Simulating parameters
n = 50
B = 10000

# Set seed
set.seed(123)

# Initialize simulation vectors
sim.theta = rep(0, B)
sim.mu1 = rep(0,B)
sim.mu2 = rep(0, B)
sim.sigma = matrix(0, B, 3)

# Monte Carlo loop
for (sim in 1:B) {
  # Simulate data
  sim.y = tibble(y1 = rnorm(n, mean = y_true[1], sd = sigma[1,1]),
                 y2 = rnorm(n, mean = y_true[2], sd = sigma[2,2]))
  mu_hat = c(mean(sim.y$y1), mean(sim.y$y2))
  sigma_hat = with(sim.y, c(sd(y1), cov(y1, y2), sd(y2)))/n

  # Parameter of interest
  theta_hat = mu_hat[1]/mu_hat[2]

  # Store the simulated values
  sim.theta[sim] = theta_hat
  sim.mu1[sim] = mu_hat[1]
  sim.mu2[sim] = mu_hat[2]
  sim.sigma[sim,] = sigma_hat
}

# Calculate the mean and standard error of theta_hat
mean_theta_hat = mean(sim.theta); mean_theta_hat
```

```
## [1] 1.020602
```

```
se_theta_hat = sd(sim.theta); se_theta_hat
```

```
## [1] 0.1531086
```

```
# Confidence interval of theta_hat
ci_mc = mean_theta_hat + c(-se_theta_hat, se_theta_hat)*1.96

# Var cov matrix
sigma_hat = c(mean(sim.sigma[,1]), rep(mean(sim.sigma[,2]), 2), mean(sim.sigma[,3])) |> matrix(nrow = 2
```

Question 2

Theta is defined as $\theta = \frac{\mu_1}{\mu_2} = f(\mu)$. To use the delta method, we take the derivative of θ in therms of μ_1 and μ_2.

$$g(\mu) = \frac{\partial f(\mu)}{\partial \mu} = \begin{pmatrix} \frac{\partial f(\mu)}{\partial \mu_1} \\ \frac{\partial f(\mu)}{\partial \mu_2} \end{pmatrix} = \begin{pmatrix} \frac{1}{\mu_2} \\ \frac{-\mu_1}{\mu_2^2} \end{pmatrix}$$

Using this, we can calculate the variance as

$$\widehat{\text{var}}(\hat{\theta}) = g(\hat{\theta})' \hat{V}_{\hat{\theta}} g(\hat{\theta})$$

So the standard error is

$$\widehat{\text{se}}(\hat{\mu}) = \sqrt{\widehat{\text{var}}(\hat{\theta})}$$

We already calculated the $\hat{V}_{\hat{\theta}}$ in part 1, so we just need to enumerate $g(\theta)$

```
g_mu_hat = c(1/y_true[1], -y_true[1]/y_true[2]) |> matrix()
varhat_theta_hat = t(g_mu_hat)%*%sigma_hat%*%g_mu_hat
sehat_theta_hat_delta = sqrt(varhat_theta_hat) |> as.numeric()

# Confidence interval
ci_delta = 1 + c(-sehat_theta_hat_delta, sehat_theta_hat_delta)*1.96
```

Now check with the delta_method function from the car package.

```
# Set names of g_mu_hat
names(g_mu_hat) = c("mu1", "mu2")
rownames(g_mu_hat) = names(g_mu_hat)

# String for the expression of interest
f_string = "mu1 / mu2"

# Delta method
dm = car::deltaMethod(g_mu_hat, f_string, sigma_hat)
dm$SE # Exact match for the manual computation
```

```
## [1] 0.145384
```

Question 3 Jackknife Method

```r
# Simulated data
y_sim = tibble(y1 = rnorm(n, mean = y_true[1], sd = sigma[1,1]),
               y2 = rnorm(n, mean = y_true[2], sd = sigma[2,2]))

# Set parameters
n = nrow(y_sim)

# Initialize values
n = nrow(y_sim)
jack_theta = rep(0, n)

# Set seed
set.seed(123)

# Jackknife loop
for (i in 1:n){
  # Selects all the data except the one observation we're leaving out
  temp_data = y_sim[-i,]

  # Calculate the parameters
  mu_hat = c(mean(temp_data$y1), mean(temp_data$y2))

  # Parameter of interest
  theta_hat = mu_hat[1]/mu_hat[2]

  # Store the parameter
  jack_theta[i] = theta_hat
}

# Jackknife standard error
jack_mean = mean(jack_theta)
jack_se = ((n-1)/n)*sum((jack_theta - jack_mean)^2) |> sqrt();jack_se
```

```
## [1] 0.1617257
```

```r
# Confidence interval
ci_jack = jack_mean + c(-jack_se, jack_se)*1.96
```

Question 4 NonParametric Bootstrap

```r
# Bootstrap parameters
n = 50
B = 10000

# Initialize necessary vectors
boot_theta = rep(0, B)
list_boot_theta = vector('list', 5) # empty list to put results from different seeds into
boot_test = vector('list', 5)
```

```r
# Bootstrap loop
# Outside loop is to loop over the different seeds
for (seed in 123:127){
  set.seed(seed)
  y_sim = tibble(y1 = rnorm(n, mean = y_true[1], sd = sigma[1,1]),
                 y2 = rnorm(n, mean = y_true[2], sd = sigma[2,2]))

  # Inside loop is bootstrapping
  for (i in 1:B){
    # Get bootstrap sample
    idx = sample(n, replace = TRUE) # idk stands for index
    boot_samp = y_sim[idx,]

    # Get parameters
    mu_hat = c(mean(boot_samp$y1), mean(boot_samp$y2))

    # Parameter of interest
    boot_theta[i] = mu_hat[1]/mu_hat[2]
    boot_test[[i]] = boot_samp
  }
  list_boot_theta[[seed-122]] = boot_theta # Subtract 122 so the list indexes from 1
}

# Parameter of interest for each seed
boot_theta_hat = sapply(1:5, function(i) {mean(list_boot_theta[[i]])})
boot_se_theta_hat = sapply(1:5, function(i) {sd(list_boot_theta[[i]])});boot_se_theta_hat
```

```
## [1] 0.10379276 0.12699370 0.15338724 0.09707892 0.10356049
```

```r
# Normal CI
ci_boot = sapply(1:5, function(i){boot_theta_hat[[i]] + c(-boot_se_theta_hat[[i]], boot_se_theta_hat[[i
```

To calculate the other CIs, we will use the boot package

```r
# This is a function to get the parameter given a bootstrapped sample
fun_theta = function(x, i){
  temp_data = x[i,]
  boot_theta = mean(temp_data$y1)/mean(temp_data$y2)
  return(boot_theta)
}

# Initialize empty list for the result from the five seeds
list_boot_theta_ci = vector('list', 5)

# Loop over the seeds again
for (seed in 123:127) {
  set.seed(seed)
  y_sim = tibble(y1 = rnorm(n, mean = y_true[1], sd = sigma[1,1]),
                 y2 = rnorm(n, mean = y_true[2], sd = sigma[2,2]))
  theta.boot = boot(y_sim, statistic = fun_theta, R = 10000)

  list_boot_theta_ci[[seed-122]] = theta.boot
}
```

```r
# All the confidence intervals
# I'm not doing this in a loop because it takes away the nice formatting of the CIs
boot.ci(list_boot_theta_ci[[1]], conf=0.95, type=c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = list_boot_theta_ci[[1]], conf = 0.95, type = c("norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   ( 0.6586,  1.0661 )   ( 0.7173,  1.1215 )   ( 0.7171,  1.1205 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(list_boot_theta_ci[[2]], conf=0.95, type=c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = list_boot_theta_ci[[2]], conf = 0.95, type = c("norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   ( 0.6720,  1.1742 )   ( 0.7468,  1.2499 )   ( 0.7536,  1.2634 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(list_boot_theta_ci[[3]], conf=0.95, type=c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = list_boot_theta_ci[[3]], conf = 0.95, type = c("norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   ( 0.6324,  1.2260 )   ( 0.7438,  1.3291 )   ( 0.7542,  1.3851 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(list_boot_theta_ci[[4]], conf=0.95, type=c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
```

```
## boot.ci(boot.out = list_boot_theta_ci[[4]], conf = 0.95, type = c("norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   ( 0.6365,  1.0130 )   ( 0.6906,  1.0618 )   ( 0.6949,  1.0731 )
## Calculations and Intervals on Original Scale
```

```r
boot.ci(list_boot_theta_ci[[5]], conf=0.95, type=c("norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = list_boot_theta_ci[[5]], conf = 0.95, type = c("norm",
##     "perc", "bca"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   ( 0.6304,  1.0419 )   ( 0.6883,  1.0938 )   ( 0.6852,  1.0841 )
## Calculations and Intervals on Original Scale
```

```r
# The normal CI is not quite the same as my manual one, but close enough
```

Question 5 Parametric Bootstrap

```r
# Set parameters
B = 10000
n = 50

# Initialize empty vectors
boot_theta = rep(0, B)
list_boot_theta = vector('list', 5) # empty list to put results from different seeds into


for (seed in 123:127) {
  set.seed(seed)
  y_sim = tibble(y1 = rnorm(n, mean = y_true[1], sd = sigma[1,1]),
                 y2 = rnorm(n, mean = y_true[2], sd = sigma[2,2]))

  # Get bootstrap se's
  se_y1 = sd(y_sim$y1)
  se_y2 = sd(y_sim$y2)

  # Get bootstrap means
  mu1 = mean(y_sim$y1)
  mu2 = mean(y_sim$y2)

  # Run bootstrap
  for (i in 1:B) {
    y1_b = rnorm(n, mu1, se_y1)
```

```
    y2_b = rnorm(n, mu2, se_y2)

    # Parameter of interest
    theta_hat = mean(y1_b) / mean(y2_b)

    # Store parameters
    boot_theta[i] = theta_hat

  }
  list_boot_theta[[seed-122]] = boot_theta # Subtract 122 so the list indexes from 1
}

# Parameter of interest for each seed
boot_theta_hat_param = sapply(1:5, function(i) {mean(list_boot_theta[[i]])})
boot_se_theta_hat_param = sapply(1:5, function(i) {sd(list_boot_theta[[i]])});boot_se_theta_hat
```

```
## [1] 0.10379276 0.12699370 0.15338724 0.09707892 0.10356049
```

```
# Normal CI
ci_boot_param = sapply(1:5, function(i){boot_theta_hat[[i]] + c(-boot_se_theta_hat[[i]], boot_se_theta_h
```

Question 6 Table Comparing Results

```
# Data for table
results_df = tibble(Type = c("Monte Carlo", "Delta Method", "Jack Knife",
                             paste("NonParametric Bootstrap: Seed = ", 123:127, sep =""),
                             paste("Parametric Bootstrap: Seed = ", 123:127, sep ="")),
                    Theta = c(mean_theta_hat, 1, jack_mean, boot_theta_hat,
                              boot_theta_hat_param),
                    Standard_Error = c(se_theta_hat, as.numeric(sehat_theta_hat_delta), jack_se,
                              boot_se_theta_hat, boot_se_theta_hat_param),
                    "CI Lower" = c(ci_mc[1], ci_delta[1], ci_jack[1], ci_boot[1,],
                    ci_boot_param[1,]),
                    "CI_upper" = c(ci_mc[2], ci_delta[2], ci_jack[2], ci_boot[2,],
                    ci_boot_param[2,]))

results_df
```

```
## # A tibble: 13 x 5
##    Type                                Theta Standard_Error `CI Lower` CI_upper
##    <chr>                               <dbl>          <dbl>      <dbl>    <dbl>
##  1 Monte Carlo                         1.02          0.153       0.721     1.32
##  2 Delta Method                        1             0.145       0.715     1.28
##  3 Jack Knife                          1.04          0.162       0.725     1.36
##  4 NonParametric Bootstrap: Seed = 123 0.886         0.104       0.683     1.09
##  5 NonParametric Bootstrap: Seed = 124 0.950         0.127       0.701     1.20
##  6 NonParametric Bootstrap: Seed = 125 0.970         0.153       0.670     1.27
##  7 NonParametric Bootstrap: Seed = 126 0.846         0.0971      0.655     1.04
##  8 NonParametric Bootstrap: Seed = 127 0.861         0.104       0.658     1.06
##  9 Parametric Bootstrap: Seed = 123    0.885         0.103       0.683     1.09
```

```
## 10 Parametric Bootstrap: Seed = 124    0.953          0.130        0.701      1.20
## 11 Parametric Bootstrap: Seed = 125    0.973          0.154        0.670      1.27
## 12 Parametric Bootstrap: Seed = 126    0.845          0.0971       0.655      1.04
## 13 Parametric Bootstrap: Seed = 127    0.863          0.107        0.658      1.06
```

We can see from the table that all the results are very similar. The bootstrap methods seem to have their means biased downward somewhat, but by a small enough amount that the true value is still well within each confidence interval. All the standard errors are very close to that same true value as well no matter the method or seed.