ST502Project

2024-09-29

libraries

```r
library(ggplot2)
```

Global variables

```r
# Set parameters and define variables for global use
set.seed(42)
N <- 1000  # samples
p_vals <- seq(0.01, 0.99, length.out = 15)  # Range of true probabilities
alpha <- 0.05  # Significance level
B <- 100  # Number of bootstrap samples
z <- qnorm(1 - alpha / 2)  # z value
```

Wald

```r
waldCI <- function(y, n, alpha = 0.05) {
  if (y == 0) return(c(0, 0))
  if (y == n) return(c(1, 1))

  p_hat <- y / n
  error <- z * sqrt((p_hat * (1 - p_hat)) / n)
  return(c(p_hat - error, p_hat + error))
}

# Function to check if the true probability is within the confidence interval
check_coverage <- function(ci, p) {
  return(p >= ci[1] & p <= ci[2])
}

# Main function for simulation and coverage calculation
simulate_confidence_intervals <- function(N, n, alpha) {
  coverage_results <- list()

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)
    ci_wald <- t(sapply(y_samples, waldCI, n = n, alpha = alpha))
    wald_coverage <- mean(apply(ci_wald, 1, check_coverage, p = p))

    coverage_results[[as.character(p)]] <- data.frame(
      p = p,
      coverage = wald_coverage,
      method = "Wald"
    )
  }

  # Combine all results into a single data frame
```

```r
  coverage_results_df <- do.call(rbind, coverage_results)

  return(coverage_results_df)
}

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and create separate plots
for (n in n_values) {
  # Run simulation with desired parameters
  wald_coverage_results <- simulate_confidence_intervals(N, n)

  # Plot the coverage probabilities for the Wald interval
  library(ggplot2)

p_plot <-
  ggplot(wald_coverage_results, aes(x = p, y = coverage)) +
    geom_line(color = "blue", size = 1.2) +
    labs(title = paste("Wald Interval Coverage Probability for n =", n),
         x = "True Probability (p)",
         y = "Coverage Probability") +
    ylim(0.7, 1.0) +
    geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
    theme_minimal() +
    theme(legend.position = "none",
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot as a PNG file
  ggsave(filename = paste("wald_coverage_n_", n, ".png", sep = ""), plot = p_plot, width = 8, height = 
}
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## Warning: Removed 3 rows containing missing values (`geom_line()`).

## Warning: Removed 2 rows containing missing values (`geom_line()`).
## Removed 2 rows containing missing values (`geom_line()`).
```

Adjusted Wald

```r
AwaldCI <- function(y, n, alpha) {
  p <- (y + 2) / (n + 4)
  ci <- c(p - z * sqrt(p * (1 - p) / (n + 4)),
          p + z * sqrt(p * (1 - p) / (n + 4)))
  return(ci)
}

# Function to check if the true probability is within the confidence interval
check_coverage <- function(ci, p) {
  return(p >= ci[1] & p <= ci[2])
```

```r
}

# Main function for simulation and coverage calculation
simulate_Awald_intervals <- function(N, n, alpha) {
  coverage_results <- list()

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)
    ci_Awald <- t(sapply(y_samples, AwaldCI, n = n, alpha = alpha))
    Awald_coverage <- mean(apply(ci_Awald, 1, check_coverage, p = p))

    coverage_results[[as.character(p)]] <- data.frame(
      p = p,
      coverage = Awald_coverage,
      method = "Adjusted Wald"
    )
  }

  # Combine all results into a single data frame
  coverage_results_df <- do.call(rbind, coverage_results)

  return(coverage_results_df)
}

# Set seed for reproducibility
set.seed(123)

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and create separate plots
for (n in n_values) {
  # Run simulation with desired parameters
  Awald_coverage_results <- simulate_Awald_intervals(N = 1000, n = n)

  # Plot the coverage probabilities for the Adjusted Wald interval
  library(ggplot2)

p_plot <- ggplot(Awald_coverage_results, aes(x = p, y = coverage)) +
    geom_line(color = "blue", size = 1.2) +
    labs(title = paste("Adjusted Wald Interval Coverage Probability for n =", n),
         x = "True Probability (p)",
         y = "Coverage Probability") +
    ylim(0.7, 1.0) +
    geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
    theme_minimal() +
    theme(legend.position = "none",,
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot as a PNG file
  ggsave(filename = paste("Awald_coverage_n_", n, ".png", sep = ""), plot = p_plot, width = 8, height =
}
```

Score

```r
scoreCI <- function(y, n, alpha) {
  p <- y / n
  ci <- c(
    (p + z^2 / (2 * n) - z * sqrt((p * (1 - p) + z^2 / (4 * n)) / n)) / (1 + z^2 / n),
    (p + z^2 / (2 * n) + z * sqrt((p * (1 - p) + z^2 / (4 * n)) / n)) / (1 + z^2 / n)
  )
  return(ci)
}

# Function to check if the true probability is within the confidence interval
check_coverage <- function(ci, p) {
  return(p >= ci[1] & p <= ci[2])
}

# Main function for simulation and coverage calculation
simulate_score_intervals <- function(N, n, alpha ) {
  coverage_results <- list()

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)
    ci_score <- t(sapply(y_samples, scoreCI, n, alpha))
    score_coverage <- mean(apply(ci_score, 1, check_coverage, p = p))

    coverage_results[[as.character(p)]] <- data.frame(
      p = p,
      coverage = score_coverage,
      method = "Score"
    )
  }

  # Combine all results into a single data frame
  coverage_results_df <- do.call(rbind, coverage_results)

  return(coverage_results_df)
}

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and create separate plots
for (n in n_values) {
  # Run simulation with desired parameters
  score_coverage_results <- simulate_score_intervals(N , n )

  # Plot the coverage probabilities for the Score interval
  library(ggplot2)

  p_plot <- ggplot(score_coverage_results, aes(x = p, y = coverage)) +
    geom_line(color = "green", size = 1.2) +
    labs(title = paste("Score Interval Coverage Probability for n =", n),
         x = "True Probability (p)",
         y = "Coverage Probability") +
```

```
    ylim(0.7, 1.0) +
    geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
    theme_minimal() +
    theme(legend.position = "none",,
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot as a PNG file
  ggsave(filename = paste("score_coverage_n_", n, ".png", sep = ""), plot = p_plot, width = 8, height =
}
```

Clopper Pearson (Exact)

```
exactCI <- function(y, n, alpha) {
  if (y == 0) return(c(0, 1 - (alpha / 2)^(1/n)))
  if (y == n) return(c((alpha / 2)^(1/n), 1))

  lower <- 1 / (1 + (n - y + 1) / (y * qf(1 - alpha / 2, 2 * y, 2 * (n - y + 1), lower.tail = FALSE)))
  upper <- 1 / (1 + (n - y) / ((y + 1) * qf(alpha / 2, 2 * (y + 1), 2 * (n - y), lower.tail = FALSE)))

  return(c(lower, upper))
}

# Function to check if the true probability is within the confidence interval
check_coverage <- function(ci, p) {
  return(p >= ci[1] & p <= ci[2])
}

# Main function for simulation and coverage calculation
simulate_exact_intervals <- function(N, n, alpha) {
  p_vals <- seq(0.01, 0.99, length.out = 100)
  coverage_results <- list()

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)
    ci_exact <- t(sapply(y_samples, exactCI, n, alpha))
    exact_coverage <- mean(apply(ci_exact, 1, check_coverage, p = p))

    coverage_results[[as.character(p)]] <- data.frame(
      p = p,
      coverage = exact_coverage,
      method = "Exact"
    )
  }

  # Combine all results into a single data frame
  coverage_results_df <- do.call(rbind, coverage_results)

  return(coverage_results_df)
}

# Sample sizes to iterate over
n_values <- c(15, 30, 100)
```

```r
# Loop through each sample size and create separate plots
for (n in n_values) {
  # Run simulation with desired parameters
  exact_coverage_results <- simulate_exact_intervals(N, n, alpha)

  # Plot the coverage probabilities for the Exact (Clopper-Pearson) interval
  library(ggplot2)

  p_plot <- ggplot(exact_coverage_results, aes(x = p, y = coverage)) +
    geom_line(color = "purple", size = 1.2) +
    labs(title = paste("Exact Interval (Clopper-Pearson) Coverage Probability for n =", n),
         x = "True Probability (p)",
         y = "Coverage Probability") +
    ylim(0.7, 1.0) +
    geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
    theme_minimal() +
    theme(legend.position = "none",
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot as a PNG file
  ggsave(filename = paste("exact_coverage_n_", n, ".png", sep = ""), plot = p_plot, width = 8, height =
}
```

Bootstrap based Intervals for raw

```r
bootstrap_percentileCI <- function(y, n, B , alpha) {
  # Early exit for edge cases
  if (y == 0) return(list(percentile = c(0, 0)))
  if (y == n) return(list(percentile = c(1, 1)))

  p_hat <- y / n
  # Generate bootstrap samples for the mean
  boot_samples <- replicate(B, mean(rbinom(n, size = 1, prob = p_hat)))

  # Percentile-based confidence interval
  ci_percentile <- quantile(boot_samples, probs = c(alpha / 2, 1 - alpha / 2), na.rm = TRUE)

  # Return the confidence interval as a named list
  return(list(percentile = ci_percentile))
}

# Function to check coverage
check_coverage <- function(ci, p) {
  return(p >= ci[1] & p <= ci[2])
}

# Main function for Bootstrap Percentile interval
simulate_bootstrap_percentile_intervals <- function(N , n, alpha , B ) {
  p_vals <- seq(0.01, 0.99, length.out = 15)
  coverage_results <- list()

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)
```

```r
    ci_percentile_list <- vector("list", length(y_samples))

    for (i in seq_along(y_samples)) {
      ci_percentile_list[[i]] <- bootstrap_percentileCI(y_samples[i], n, B = B, alpha = alpha)$percentil
    }

    ci_percentile_matrix <- do.call(rbind, ci_percentile_list)
    percentile_coverage <- mean(apply(ci_percentile_matrix, 1, check_coverage, p = p))

    coverage_results[[as.character(p)]] <- data.frame(
      p = p,
      coverage = percentile_coverage,
      method = "Bootstrap Percentile"
    )
  }

  # Combine all results into a single data frame
  coverage_results_df <- do.call(rbind, coverage_results)

  return(coverage_results_df)
}

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and create separate plots
for (n in n_values) {
  # Run simulation with desired parameters
  bootstrap_percentile_results <- simulate_bootstrap_percentile_intervals(N , n, alpha, B )

  # Plot the coverage probabilities for the Bootstrap Percentile interval
  library(ggplot2)

  p_plot <- ggplot(bootstrap_percentile_results, aes(x = p, y = coverage)) +
    geom_line(color = "orange", size = 1.2) +
    labs(title = paste("Bootstrap Percentile Interval Coverage Probability for n =", n),
         x = "True Probability (p)",
         y = "Coverage Probability") +
    ylim(0.7, 1.0) +
    geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
    theme_minimal() +
    theme(legend.position = "none",
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot as a PNG file
  ggsave(filename = paste("bootstrap_percentile_coverage_n_", n, ".png", sep = ""), plot = p_plot, widtl
}
```

```
## Warning: Removed 3 rows containing missing values (`geom_line()`).

## Warning: Removed 2 rows containing missing values (`geom_line()`).
## Removed 2 rows containing missing values (`geom_line()`).
```

Bootstrap t Interval

```r
bootstrap_tCI <- function(y, n, B , alpha ) {
  if (y == 0) return(c(0, 0))
  if (y == n) return(c(1, 1))

  p_hat <- y / n
  # Generate bootstrap samples
  boot_samples <- replicate(B, mean(rbinom(n, size = 1, prob = p_hat)))

  # Compute t-statistics
  t_stat <- (boot_samples - p_hat) / sqrt((p_hat * (1 - p_hat)) / n)

  # T-interval confidence interval
  ci_t <- p_hat + quantile(t_stat, probs = c(alpha / 2, 1 - alpha / 2), na.rm = TRUE) * sqrt((p_hat * (

  return(ci_t)
}

# Parameters
bootstrap_t_results <- data.frame()

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Bootstrap T-Interval Coverage Simulation
for (n in n_values) {
  for (p in p_vals) {
  y_samples <- rbinom(B, size = n, prob = p)

  ci_t_interval <- t(sapply(y_samples, bootstrap_tCI, n = n, B, alpha = alpha))
  t_interval_coverage <- mean(apply(ci_t_interval, 1, check_coverage, p = p))

  # Store results for bootstrap t-interval
  bootstrap_t_results <- rbind(bootstrap_t_results, data.frame(
    p = p,
    coverage = t_interval_coverage,
    method = "bootstrap_t_interval"
    ))
  }
}
# Plot for Bootstrap T-Interval
for(n in n_values) {
p_plot <- ggplot(bootstrap_t_results, aes(x = p, y = coverage, color = method, linetype = method)) +
  geom_line(size = 1.2) +
  labs(title = paste("Bootstrap T-Interval Coverage (n = ", n, ")"),
       x = "True Probability (p)",
       y = "Coverage Probability") +
  geom_hline(yintercept = 0.95, linetype = "dashed", color = "red") +  # 95% target line
  theme_minimal() + ylim(0.7, 1.0) +
  theme(legend.position = "bottom",
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))
```

```
bootstrap_t_results

ci_t_interval

# Save the plot as a PNG file
  ggsave(filename = paste("bootstrap_t_coverage_n_", n, ".png", sep = ""), plot = p_plot, width = 8, hei
}
```

```
## Warning: Removed 6 rows containing missing values (`geom_line()`).
## Removed 6 rows containing missing values (`geom_line()`).
## Removed 6 rows containing missing values (`geom_line()`).
```

Proportion that miss above or below for bootstrapped

```
# Load necessary library
library(ggplot2)

# Function to check coverage details
check_coverage_details <- function(ci, p) {
  if (p >= ci[1] & p <= ci[2]) {
    return(c(capture = 1, miss_above = 0, miss_below = 0))  # Captures the true value
  } else if (p < ci[1]) {
    return(c(capture = 0, miss_above = 1, miss_below = 0))  # Misses above
  } else {
    return(c(capture = 0, miss_above = 0, miss_below = 1))  # Misses below
  }
}


# Initialize a data frame to store results
bootstrap_coverage_results <- data.frame()

# Function for Bootstrap t Interval
bootstrap_tCI <- function(y, n, B = 100, alpha = 0.05) {
  if (y == 0) return(c(0, 0))
  if (y == n) return(c(1, 1))

  p_hat <- y / n
  # Generate bootstrap samples
  boot_samples <- replicate(B, mean(rbinom(n, size = 1, prob = p_hat)))

  # Compute t-statistics
  t_stat <- (boot_samples - p_hat) / sqrt((p_hat * (1 - p_hat)) / n)

  # T-interval confidence interval
  ci_t <- p_hat + quantile(t_stat, probs = c(alpha / 2, 1 - alpha / 2), na.rm = TRUE) * sqrt((p_hat * (

  return(ci_t)
}

# Generate samples and compute coverage probabilities for both bootstrap methods
for (p in p_vals) {
  y_samples <- rbinom(100, size = n, prob = p)  # Generate random samples

  # Calculate bootstrap CIs for percentile
```

```r
  ci_bootstrap_percentile <- lapply(y_samples, bootstrap_percentileCI, n = n, B = B, alpha = alpha)
  ci_percentile <- do.call(rbind, lapply(ci_bootstrap_percentile, function(x) x$percentile))

  # Calculate bootstrap CIs for t interval
  ci_bootstrap_t <- t(sapply(y_samples, bootstrap_tCI, n = n, B = B, alpha = alpha))

  # Coverage and miss probabilities for percentile intervals
  percentile_coverage_details <- t(apply(ci_percentile, 1, check_coverage_details, p = p))
  percentile_coverage <- colMeans(percentile_coverage_details)

  # Coverage and miss probabilities for t intervals
  t_coverage_details <- t(apply(ci_bootstrap_t, 1, check_coverage_details, p = p))
  t_coverage <- colMeans(t_coverage_details)

  # Store the results for both methods
  bootstrap_coverage_results <- rbind(bootstrap_coverage_results,
                                      data.frame(
                                        p = p,
                                        coverage = percentile_coverage[1],  # Capture rate
                                        miss_above = percentile_coverage[2],  # Miss above
                                        miss_below = percentile_coverage[3],  # Miss below
                                        method = "bootstrap_percentile"
                                      ),
                                      data.frame(
                                        p = p,
                                        coverage = t_coverage[1],  # Capture rate
                                        miss_above = t_coverage[2],  # Miss above
                                        miss_below = t_coverage[3],  # Miss below
                                        method = "bootstrap_t_interval"
                                      ))
}

# Display the results
bootstrap_coverage_results
```

```
##              p coverage miss_above miss_below               method
## capture   0.01     0.59       0.00       0.41 bootstrap_percentile
## capture1  0.01     0.58       0.01       0.41 bootstrap_t_interval
## capture2  0.08     0.93       0.02       0.05 bootstrap_percentile
## capture3  0.08     0.94       0.02       0.04 bootstrap_t_interval
## capture4  0.15     0.89       0.05       0.06 bootstrap_percentile
## capture5  0.15     0.91       0.04       0.05 bootstrap_t_interval
## capture6  0.22     0.96       0.02       0.02 bootstrap_percentile
## capture7  0.22     0.94       0.03       0.03 bootstrap_t_interval
## capture8  0.29     0.93       0.04       0.03 bootstrap_percentile
## capture9  0.29     0.91       0.05       0.04 bootstrap_t_interval
## capture10 0.36     0.93       0.03       0.04 bootstrap_percentile
## capture11 0.36     0.91       0.04       0.05 bootstrap_t_interval
## capture12 0.43     0.98       0.00       0.02 bootstrap_percentile
## capture13 0.43     0.97       0.01       0.02 bootstrap_t_interval
## capture14 0.50     0.90       0.04       0.06 bootstrap_percentile
## capture15 0.50     0.94       0.03       0.03 bootstrap_t_interval
## capture16 0.57     0.91       0.06       0.03 bootstrap_percentile
## capture17 0.57     0.93       0.04       0.03 bootstrap_t_interval
```

```
## capture18 0.64     0.96       0.01       0.03 bootstrap_percentile
## capture19 0.64     0.95       0.02       0.03 bootstrap_t_interval
## capture20 0.71     0.93       0.05       0.02 bootstrap_percentile
## capture21 0.71     0.95       0.03       0.02 bootstrap_t_interval
## capture22 0.78     0.96       0.03       0.01 bootstrap_percentile
## capture23 0.78     0.97       0.03       0.00 bootstrap_t_interval
## capture24 0.85     0.93       0.06       0.01 bootstrap_percentile
## capture25 0.85     0.93       0.05       0.02 bootstrap_t_interval
## capture26 0.92     0.89       0.08       0.03 bootstrap_percentile
## capture27 0.92     0.91       0.07       0.02 bootstrap_t_interval
## capture28 0.99     0.64       0.36       0.00 bootstrap_percentile
## capture29 0.99     0.63       0.36       0.01 bootstrap_t_interval
```

Proportion that miss above and below for Wald

```r
# Initialize a data frame to store the results for Wald intervals
wald_coverage_results <- data.frame()

# Loop over the true probability values to compute coverage and miss rates for the Wald interval
for (p in p_vals) {
  y_samples <- rbinom(1000, size = n, prob = p)  # Generate random samples
  ci_wald <- t(sapply(y_samples, waldCI, n = n, alpha = alpha))  # Wald CI for each sample

  # Check coverage and proportion of misses for Wald intervals
  wald_coverage_details <- t(apply(ci_wald, 1, check_coverage_details, p = p))
  wald_coverage <- colMeans(wald_coverage_details)

  # Store the results for Wald Interval (without average length)
  wald_coverage_results <- rbind(wald_coverage_results, data.frame(
    p = p,
    coverage = wald_coverage[1],  # Capture rate
    miss_above = wald_coverage[2],  # Miss above
    miss_below = wald_coverage[3],  # Miss below
    method = "Wald"
  ))
}

# View the Wald interval results
wald_coverage_results
```

```
##              p coverage miss_above miss_below method
## capture   0.01    0.620      0.001      0.379   Wald
## capture1  0.08    0.908      0.017      0.075   Wald
## capture2  0.15    0.928      0.017      0.055   Wald
## capture3  0.22    0.935      0.010      0.055   Wald
## capture4  0.29    0.928      0.017      0.055   Wald
## capture5  0.36    0.939      0.029      0.032   Wald
## capture6  0.43    0.951      0.023      0.026   Wald
## capture7  0.50    0.939      0.022      0.039   Wald
## capture8  0.57    0.959      0.020      0.021   Wald
## capture9  0.64    0.940      0.033      0.027   Wald
## capture10 0.71    0.930      0.048      0.022   Wald
## capture11 0.78    0.931      0.054      0.015   Wald
## capture12 0.85    0.938      0.051      0.011   Wald
## capture13 0.92    0.921      0.068      0.011   Wald
```

```
## capture14 0.99    0.642      0.358      0.000   Wald
```

Proportion that miss above and below for Adj Wald

```r
# Initialize a data frame to store the results for Adjusted Wald intervals
Awald_coverage_results <- data.frame()

# Loop over the true probability values to compute coverage and miss rates for the Adjusted Wald interv
for (p in p_vals) {
  y_samples <- rbinom(1000, size = n, prob = p)  # Generate random samples
  ci_Awald <- t(sapply(y_samples, AwaldCI, n = n, alpha = alpha))  # Adjusted Wald CI for each sample

  # Check coverage and proportion of misses for Adjusted Wald intervals
  Awald_coverage_details <- t(apply(ci_Awald, 1, check_coverage_details, p = p))
  Awald_coverage <- colMeans(Awald_coverage_details)

  # Store the results for Adjusted Wald Interval (without average length)
  Awald_coverage_results <- rbind(Awald_coverage_results, data.frame(
    p = p,
    coverage = Awald_coverage[1],  # Capture rate
    miss_above = Awald_coverage[2],  # Miss above
    miss_below = Awald_coverage[3],  # Miss below
    method = "AWald"
  ))
}

# View the Adjusted Wald interval results
Awald_coverage_results
```

```
##              p coverage miss_above miss_below method
## capture   0.01    0.979      0.021      0.000  AWald
## capture1  0.08    0.965      0.021      0.014  AWald
## capture2  0.15    0.959      0.029      0.012  AWald
## capture3  0.22    0.955      0.027      0.018  AWald
## capture4  0.29    0.956      0.029      0.015  AWald
## capture5  0.36    0.937      0.030      0.033  AWald
## capture6  0.43    0.945      0.027      0.028  AWald
## capture7  0.50    0.945      0.031      0.024  AWald
## capture8  0.57    0.936      0.039      0.025  AWald
## capture9  0.64    0.947      0.024      0.029  AWald
## capture10 0.71    0.962      0.013      0.025  AWald
## capture11 0.78    0.966      0.011      0.023  AWald
## capture12 0.85    0.963      0.012      0.025  AWald
## capture13 0.92    0.948      0.014      0.038  AWald
## capture14 0.99    0.990      0.000      0.010  AWald
```

Proportion that miss above and below for Score

```r
# Initialize a data frame to store the results for Score intervals
score_coverage_results <- data.frame()

# Loop over the true probability values to compute coverage and miss rates for the Score interval
for (p in p_vals) {
  y_samples <- rbinom(1000, size = n, prob = p)  # Generate random samples
  ci_score <- t(sapply(y_samples, scoreCI, n = n, alpha = alpha))  # Score CI for each sample
```

```r
  # Check coverage and proportion of misses for Score intervals
  score_coverage_details <- t(apply(ci_score, 1, check_coverage_details, p = p))
  score_coverage <- colMeans(score_coverage_details)

  # Store the results for Score Interval (without average length)
  score_coverage_results <- rbind(score_coverage_results, data.frame(
    p = p,
    coverage = score_coverage[1],  # Capture rate
    miss_above = score_coverage[2],  # Miss above
    miss_below = score_coverage[3],  # Miss below
    method = "Score"
  ))
}

# View the Score interval results
score_coverage_results
```

```
##              p coverage miss_above miss_below method
## capture   0.01    0.923      0.077      0.000  Score
## capture1  0.08    0.957      0.034      0.009  Score
## capture2  0.15    0.933      0.036      0.031  Score
## capture3  0.22    0.965      0.019      0.016  Score
## capture4  0.29    0.953      0.024      0.023  Score
## capture5  0.36    0.941      0.036      0.023  Score
## capture6  0.43    0.939      0.025      0.036  Score
## capture7  0.50    0.947      0.030      0.023  Score
## capture8  0.57    0.950      0.028      0.022  Score
## capture9  0.64    0.963      0.019      0.018  Score
## capture10 0.71    0.940      0.030      0.030  Score
## capture11 0.78    0.960      0.018      0.022  Score
## capture12 0.85    0.932      0.030      0.038  Score
## capture13 0.92    0.947      0.010      0.043  Score
## capture14 0.99    0.921      0.000      0.079  Score
```

Proportion that miss above and below for Clopper Pearson

```r
# Initialize a data frame to store the results for Exact intervals
exact_coverage_results <- data.frame()

# Loop over the true probability values to compute coverage and miss rates for the Exact interval
for (p in p_vals) {
  y_samples <- rbinom(1000, size = n, prob = p)  # Generate random samples
  ci_exact <- t(sapply(y_samples, exactCI, n = n, alpha = alpha))  # Exact CI for each sample

  # Check coverage and proportion of misses for Exact intervals
  exact_coverage_details <- t(apply(ci_exact, 1, check_coverage_details, p = p))
  exact_coverage <- colMeans(exact_coverage_details)

  # Store the results for Clopper-Pearson (Exact) Interval (without average length)
  exact_coverage_results <- rbind(exact_coverage_results, data.frame(
    p = p,
    coverage = exact_coverage[1],  # Capture rate
    miss_above = exact_coverage[2],  # Miss above
    miss_below = exact_coverage[3],  # Miss below
    method = "Exact"
```

```
  ))
}

# View the Exact interval results
exact_coverage_results
```

```
##              p coverage miss_above miss_below method
## capture   0.01    0.987      0.013      0.000  Exact
## capture1  0.08    0.976      0.016      0.008  Exact
## capture2  0.15    0.975      0.017      0.008  Exact
## capture3  0.22    0.973      0.017      0.010  Exact
## capture4  0.29    0.968      0.019      0.013  Exact
## capture5  0.36    0.952      0.013      0.035  Exact
## capture6  0.43    0.963      0.026      0.011  Exact
## capture7  0.50    0.956      0.021      0.023  Exact
## capture8  0.57    0.960      0.015      0.025  Exact
## capture9  0.64    0.967      0.019      0.014  Exact
## capture10 0.71    0.956      0.016      0.028  Exact
## capture11 0.78    0.963      0.011      0.026  Exact
## capture12 0.85    0.968      0.012      0.020  Exact
## capture13 0.92    0.976      0.013      0.011  Exact
## capture14 0.99    0.994      0.000      0.006  Exact
```

Average widths

```
library(ggplot2)

# Function to calculate average length of confidence intervals
calculate_average_length <- function(N, n, alpha) {
  lengths <- data.frame(p = numeric(), length = numeric(), method = character())

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)

    # Calculate lengths for each method
    ci_wald <- t(sapply(y_samples, waldCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_wald[, 2] - ci_wald[, 1], method = "Wald"))

    ci_Awald <- t(sapply(y_samples, AwaldCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_Awald[, 2] - ci_Awald[, 1], method = "Adjus

    ci_score <- t(sapply(y_samples, scoreCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_score[, 2] - ci_score[, 1], method = "Score

    ci_exact <- t(sapply(y_samples, exactCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_exact[, 2] - ci_exact[, 1], method = "Exact

    ci_percentile <- sapply(y_samples, function(y) bootstrap_percentileCI(y, n, B, alpha)$percentile)
    lengths <- rbind(lengths, data.frame(p = p, length = ci_percentile[2, ] - ci_percentile[1, ], metho

    ci_t <- sapply(y_samples, function(y) bootstrap_tCI(y, n, B, alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_t[2, ] - ci_t[1, ], method = "Bootstrap T")
  }

  # Calculate average lengths
```

```r
  average_lengths <- aggregate(length ~ p + method, data = lengths, FUN = mean)

  return(average_lengths)
}

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and create plots
for (n in n_values) {
  average_lengths_results <- calculate_average_length(N = 1000, n = n, alpha = 0.05)

  # Create the plot
  plot <- ggplot(average_lengths_results, aes(x = p, y = length, color = method)) +
    geom_smooth(se = FALSE, size = 1.2) +
    labs(title = paste("Average Length of Confidence Intervals for n =", n),
         x = "True Probability (p)",
         y = "Average Length of CIs") +
    ylim(0.0, 0.5) +
    scale_x_continuous(breaks = seq(0, 1, by = 0.25)) +
    theme_minimal() +
    theme(legend.position = "bottom",
    panel.background = element_rect(fill = "white", color = "white"),  # Set panel background to white
          plot.background = element_rect(fill = "white", color = "white"))

  # Save the plot
  ggsave(filename = paste("average_length_CI_n_", n, ".png", sep = ""), plot = plot, width = 8, height =
}
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'

## Warning: Removed 3 rows containing non-finite values (`stat_smooth()`).

## Warning: Removed 18 rows containing missing values (`geom_smooth()`).

## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

Calculate standard errors for average widths

```r
# Function to calculate average length and standard error of confidence intervals
calculate_average_length_with_se <- function(N, n, alpha) {
  lengths <- data.frame(p = numeric(), length = numeric(), method = character())

  for (p in p_vals) {
    y_samples <- rbinom(N, size = n, prob = p)

    # Calculate lengths for each method
    ci_wald <- t(sapply(y_samples, waldCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_wald[, 2] - ci_wald[, 1], method = "Wald"))

    ci_Awald <- t(sapply(y_samples, AwaldCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_Awald[, 2] - ci_Awald[, 1], method = "Adjus

    ci_score <- t(sapply(y_samples, scoreCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_score[, 2] - ci_score[, 1], method = "Score
```

```r
    ci_exact <- t(sapply(y_samples, exactCI, n = n, alpha = alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_exact[, 2] - ci_exact[, 1], method = "Exact

    ci_percentile <- sapply(y_samples, function(y) bootstrap_percentileCI(y, n, B, alpha)$percentile)
    lengths <- rbind(lengths, data.frame(p = p, length = ci_percentile[2, ] - ci_percentile[1, ], metho

    ci_t <- sapply(y_samples, function(y) bootstrap_tCI(y, n, B, alpha))
    lengths <- rbind(lengths, data.frame(p = p, length = ci_t[2, ] - ci_t[1, ], method = "Bootstrap T")
  }

  # Calculate average lengths and standard errors
  average_lengths <- aggregate(length ~ method, data = lengths, FUN = mean)
  standard_errors <- aggregate(length ~ method, data = lengths, FUN = function(x) sd(x) / sqrt(N))

  # Combine average lengths and standard errors
  combined_results <- merge(average_lengths, standard_errors, by = "method")
  colnames(combined_results) <- c("method", "avg_length", "se_length")

  return(combined_results)
}

# Initialize a table to store SE results with the correct structure
se_table <- data.frame(Method = character(6), n_15 = numeric(6), n_30 = numeric(6), n_100 = numeric(6))

# Sample sizes to iterate over
n_values <- c(15, 30, 100)

# Loop through each sample size and store the SE results
for (n in n_values) {
  results <- calculate_average_length_with_se(N = 1000, n = n, alpha = 0.05)

  # Assign the SE results to the appropriate column in se_table
  if (n == 15) {
    se_table$n_15 <- results$se_length
  } else if (n == 30) {
    se_table$n_30 <- results$se_length
  } else if (n == 100) {
    se_table$n_100 <- results$se_length
  }

  # Store the method names in the first column (this only needs to be done once)
  se_table$Method <- results$method
}

# Print the table
print(se_table)
```

```
##                 Method        n_15        n_30       n_100
## 1        Adjusted Wald 0.001939769 0.001869174 0.001364342
## 2 Bootstrap Percentile 0.005400166 0.003499173 0.001740917
## 3          Bootstrap T 0.005404468 0.003489766 0.001740458
## 4                Exact 0.003397251 0.002676765 0.001590292
## 5                Score 0.002755010 0.002318041 0.001491608
```

```
## 6                 Wald 0.005532912 0.003528280 0.001744987
```