Report on Lab 2

Author: Kevin Deshayes

Email: kede23@student.bth.se

Course Name: Mathematical Statistics

Course Code: MS1403

Contents

0.1 R code

```r
# --- Corrected Comments --- #

library(moments)
library(ggplot2)
library(gridExtra)

Log_Rayleigh_likelihood= function(sd, mu) {
    n = length(sd) # Number of observations
    result = n*log(pi) + sum(log(sd)) - n*log(2) - 2*n*log(mu) - (pi* sum(sd^2))/(4*mu^2)
    return(result)
}

numericalDerivative = function(f, data){
    # Ensure data is numeric
    if (!is.numeric(data)) {
        stop("Error: Data is not numeric.")
    }

    result = optim(
    par = 5,
    fn = function(mu) f(data, mu),  # Optimize over mu, passing data via closure
    method = "L-BFGS-B", # BFGS with restrictions
    lower = 0.001,  # Set a lower bound to avoid non-positive mu values
    control = list(fnscale= -1))

    return(result)
}

analyticalDerivative = function(mu, y){
    n = length(y)
    (-2*n)/mu + (pi * sum(y^2))/(2*mu^3)
}

analyticalMaximizing = function(y) {
    result = optim(par = 5,
                   fn = function(mu) Log_Rayleigh_likelihood(y, mu), # Objective function
                   gr = function(mu) analyticalDerivative(mu, y),    # Gradient function
```

```
                   method = "L-BFGS-B", # BFGS with restrictions
                   lower = 0.001,  # Set a lower bound to avoid non-positive mu values
                   control = list(fnscale = -1))
    return(result)
}

rr = function(mu, n) {
    u = runif(n)
    r_samples = 2 * mu * sqrt(-log(1 - u) / pi)
    return(r_samples)
}

calc_bias = function(estimates, true_value){
    return(mean(estimates) - true_value)
}

calc_mse = function(estimates, true_value){
    return(mean((estimates) - true_value)^2)
}

calculate_metrics = function(estimates, true_value){
    mean_estimate = mean(estimates)
    bias = calc_bias(estimates, true_value)
    mse = calc_mse(estimates, true_value)
    skewness_val = skewness(estimates)
    kurtosis_val = kurtosis(estimates)

    return(list(mean = mean_estimate, bias = bias, mse = mse, skewness = skewness_val, kurtosis = kurtosis_val))
}

computation = function(n) {
    r_values = rr(mu = 5, n) # Generate random samples with n in size from n_values
    log_likelihood = Log_Rayleigh_likelihood(r_values, mu = 5) # mu is an initial guess

    numerical_result = numericalDerivative(Log_Rayleigh_likelihood, r_values)
    analytical_maximization = analyticalMaximizing(r_values)

    return(list(
```

```
        numerical_result = numerical_result,
        analytical_maximization = analytical_maximization
    ))
}

clean_results = function(result, true_mu, method_type) {
    converged_estimates = list()

    # Filter out only converged results
    res = result[[method_type]]

    # Check if 'res' is a list and contains 'convergence'
    if (is.list(res) && !is.null(res[["convergence"]]) && res[["convergence"]] == 0) {
        converged_estimates = res$par  # Store the estimated parameter
    }

    if (length(converged_estimates) > 0) {
        # Calculate performance metrics
        metrics = calculate_metrics(converged_estimates, true_mu)

        return(list(
            converged_estimates = converged_estimates,
            metrics = metrics
        ))
    } else {
        return ("No converged models to analyze")
    }
}

# --- Plot Functions --- #
plot_mean_estimates <- function(estimate_df_numerical, estimate_df_analytical, true_mu) {

    # Calculate the mean estimate for each sample size in numerical method
    mean_numerical <- aggregate(Estimate ~ Sample_Size, data = estimate_df_numerical, FUN = mean)
    mean_numerical$Method <- "Numerical"

    # Calculate the mean estimate for each sample size in analytical method
    mean_analytical <- aggregate(Estimate ~ Sample_Size, data = estimate_df_analytical, FUN = mean)
```

```r
    mean_analytical$Method <- "Analytical"

    # Combine both data frames for plotting
    combined_mean_df <- rbind(mean_numerical, mean_analytical)

    # Plot mean estimates with a horizontal line at true_mu
    ggplot(combined_mean_df, aes(x = Sample_Size, y = Estimate, color = Method, group = Method)) +
        geom_line(aes(linetype = Method), linewidth = 1.2) + # Use linewidth instead of size
        geom_point(size = 3) +
        geom_hline(yintercept = true_mu, linetype = "dashed", color = "black", linewidth = 1) +
        labs(title = "Mean Estimates of Numerical and Analytical Methods",
             x = "Sample Size", y = "Mean Estimate") +
        theme_minimal() +
        scale_color_manual(values = c("Numerical" = "blue", "Analytical" = "red")) +
        theme(legend.position = "top")
}

# Function to plot the bell curves
plot_bell_curve <- function(numerical_estimates, analytical_estimates, true_mu) {
    # Convert estimates to data frames for plotting
    df_numerical <- data.frame(Estimate = numerical_estimates, Method = "Numerical")
    df_analytical <- data.frame(Estimate = analytical_estimates, Method = "Analytical")

    # Combine the two data frames
    df_combined <- rbind(df_numerical, df_analytical)

    # Get standard deviation for the normal distribution based on estimates
    combined_sd <- sd(df_combined$Estimate)

    # Create the normal distribution curve using the true_mu and calculated std deviation
    normal_curve <- data.frame(
        x = seq(min(df_combined$Estimate) - 1, max(df_combined$Estimate) + 1, length.out = 100),
        y = dnorm(seq(min(df_combined$Estimate) - 1, max(df_combined$Estimate) + 1, length.out = 100),
```

```
                    mean = true_mu,
                    sd = combined_sd)
    )

    # Scale the normal curve's density to match the density of estimates
    scale_factor <- max(density(numerical_estimates)$y, density(analytical_estimates)$y) / max(normal_curve$y)
    normal_curve$y <- normal_curve$y * scale_factor

    # Plot the curves
    ggplot(df_combined, aes(x = Estimate, color = Method)) +
        geom_density(aes(linetype = Method), size = 1.2) + # Plot density for numerical and analytical estimates
        geom_line(data = normal_curve, aes(x = x, y = y), color = "black", linetype = "dashed", size = 1.2) + # Add the normal distribution curve
        labs(title = "Bell Curves for Numerical, Analytical, and Normal Distributions",
             x = "Estimate", y = "Density") +
        theme_minimal() +
        scale_color_manual(values = c("Numerical" = "blue", "Analytical" = "red")) +
        theme(legend.position = "top")
}

# --- Main Computation --- #
set.seed(123) # For reproducibility in different runs of the code
n_value = c(10, 20, 50, 90, 140)
true_mu = 5

# Separate lists for the results
all_results_numerical = list()
all_results_analytical = list()

# Run the Monte Carlo simulation and store results
bias_mse_df_numerical = data.frame(Sample_Size = integer(), Bias = double(), MSE = double())
bias_mse_df_analytical = data.frame(Sample_Size = integer(), Bias = double(), MSE = double())

estimate_df_numerical = data.frame(Sample_Size = integer(), Estimate = double(), Method = character())
```

```
estimate_df_analytical = data.frame(Sample_Size = integer(), Estimate = double(), Method = character())

# --- Sample Sizes Loop --- #
n_reps = 1000

# Iterate through all given sample sizes
for (n in n_value) {
    cat("\n--- Processing n =", n, "---\n")

    numerical_estimates = c()
    analytical_estimates = c()

    # Repeat the computation 1000 times for each sample size
    for (rep in 1:n_reps) {
        result <- computation(n)

        # Clean numerical and analytical results separately
        cleaned_numerical = clean_results(result, true_mu, "numerical_result")
        cleaned_analytical = clean_results(result, true_mu, "analytical_maximization")

        # Store numerical results if they exist
        if (is.list(cleaned_numerical)) {
            numerical_estimates = c(numerical_estimates, cleaned_numerical$converged_estimates)
        }

        # Store analytical results if they exist
        if (is.list(cleaned_analytical)) {
            analytical_estimates = c(analytical_estimates, cleaned_analytical$converged_estimates)
        }
    }

    # After 1000 repetitions
    if (length(numerical_estimates) > 0) {
        cat("\nNumerical Method Results for n =", n, " after", n_reps, "repetitions:\n")
        numerical_metrics = calculate_metrics(numerical_estimates, true_mu)
        print(numerical_metrics)

        if (is.list(cleaned_numerical)) {
```

```
            estimate_df_numerical = rbind(estimate_df_numerical, data.frame(
                Sample_Size = rep(n, length(cleaned_numerical$converged_estimates)),
                Estimate = numerical_estimates,
                Method = "Numerical"
            ))
        }
    }

    if (length(analytical_estimates) > 0) {
        cat("\nAnalytical Method Results for n =", n, " after", n_reps, "repetitions:\n")
        analytical_metrics = calculate_metrics(analytical_estimates, true_mu)
        print(analytical_metrics)

        if (is.list(cleaned_analytical)) {
            estimate_df_analytical = rbind(estimate_df_analytical, data.frame(
                Sample_Size = rep(n, length(cleaned_analytical$converged_estimates)),
                Estimate = analytical_estimates,
                Method = "Analytical"
            ))
        }
    }


        print(plot_histogram(cleaned_analytical$converged_estimates, n))
}

# After the simulations
# Call the function to plot the mean estimates
print(plot_mean_estimates(estimate_df_numerical, estimate_df_analytical, true_mu = 5))
print(plot_bell_curve(numerical_estimates, analytical_estimates, true_mu = 5))
```

0.2 Results

n	Method	Mean	Bias	MSE	Skewness	Kurtosis
10	Numerical	4.905042	-0.09495757	0.00901694	0.2408448	3.216145
10	Analytical	4.905042	-0.09495774	0.009016973	0.2408448	3.216145
20	Numerical	4.950229	-0.04977106	0.002477158	0.1248401	3.030776
20	Analytical	4.950229	-0.04977123	0.002477175	0.1248401	3.030776
50	Numerical	4.984634	-0.01536577	0.0002361068	0.06191512	3.024987
50	Analytical	4.984634	-0.01536594	0.000236112	0.06191511	3.024987
90	Numerical	5.007175	0.007175166	5.1483×10^{-5}	0.05860747	3.049959
90	Analytical	5.007175	0.007175	5.1481×10^{-5}	0.05860747	3.049959
140	Numerical	4.995738	-0.00426153	1.8161×10^{-5}	0.08503703	3.152488
140	Analytical	4.995738	-0.004261697	1.8162×10^{-5}	0.08503702	3.152488

Table 1: Comparison of Numerical and Analytical Methods for different n after 1000 repetitions

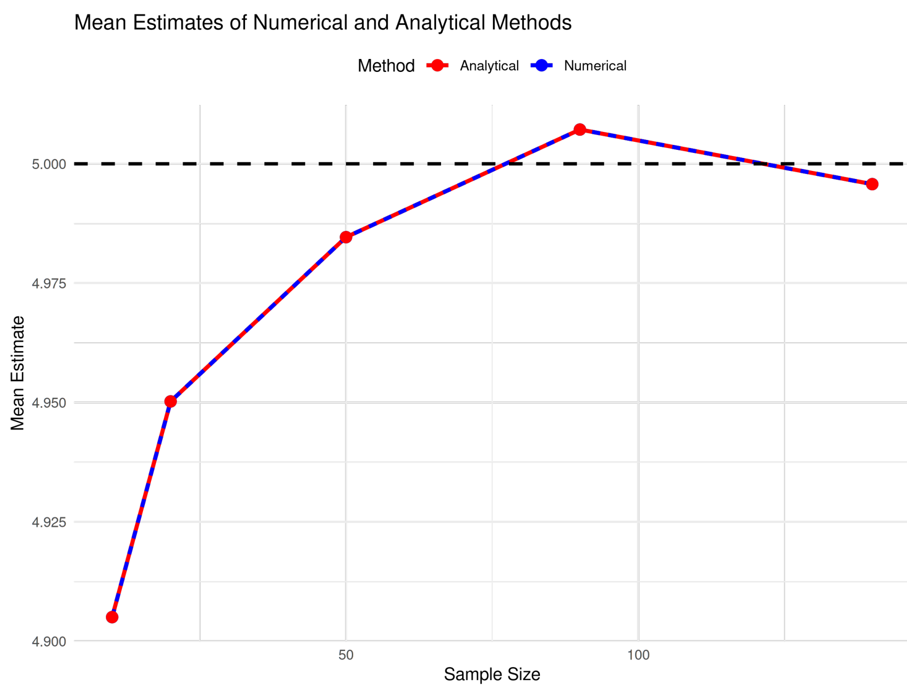

Figure 1: Comparison on estimations

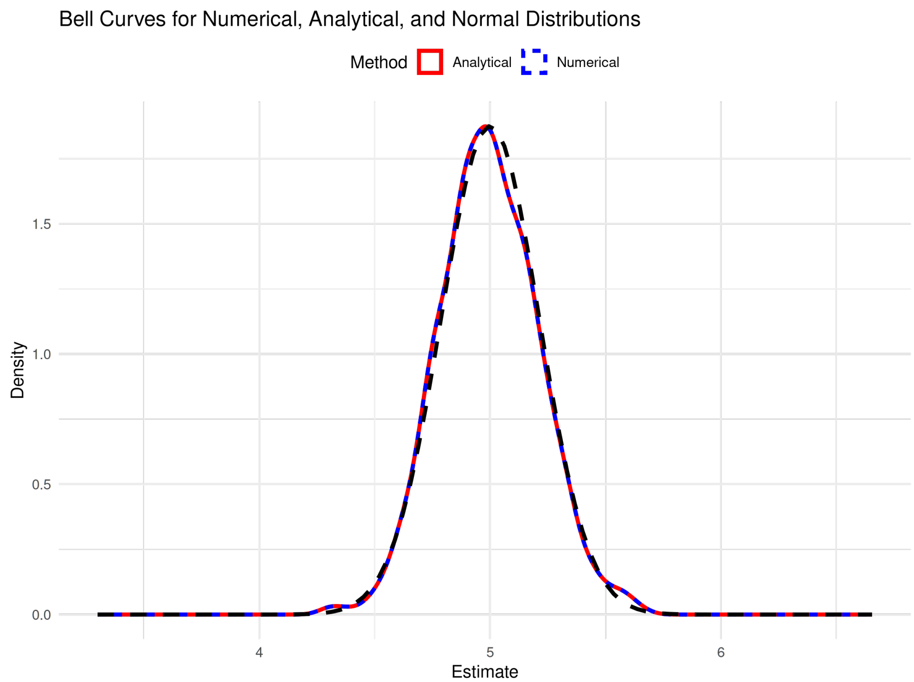

Figure 2: Bell curves for numerical and analytical estimators, compared to the normal distribution.

0.3 Analysis

0.3.1 Testing Environment Parameters

For each sample size, 1000 repetitions were conducted, with $\mu = 5$ being the true value for the simulations.

0.3.2 Observations

Both estimations generally follow the Law of Large Numbers, where the bias decreases as the sample size increases. There is a slight difference between the two estimators at $n = 50$ and $n = 140$. We also observe excess kurtosis close to 3 for all sample sizes, with values approaching 3 as the sample size increases.

0.3.3 Discussion

The observed skewness and kurtosis are as expected, with kurtosis near 3, which is typical for the Bernoulli distribution. As the sample size increases, the estimations become more similar to a normal distribution due to the Central Limit

Theorem. The higher the sample size, the closer the kurtosis gets to 3, indicating convergence towards normality.

Both estimators follow the Law of Large Numbers, showing a general decrease in bias. However, at $n = 140$, there is an unexpected increase in bias, which may be due to the randomness in the Monte Carlo simulation, where random numbers generated by the inverse function cause fluctuations. Despite this, the overall trend supports the Law of Large Numbers, where bias decreases asymptotically as N approaches infinity, though it does so asymmetrically, leading to spikes and fluctuations.

There is a minor discrepancy between the estimators at $n = 50$ and $n = 140$, but this difference, occurring at the eighth decimal place, is negligible in the context of this experiment. The small difference can be attributed to the nature of the methods used: the iterative algorithm behaves slightly differently from the algebraic estimation in the analytical method. However, the difference in results between the numerical and analytical approaches is minimal.

0.3.4 Conclusions

- Both estimators become more accurate as the sample size increases, following the Law of Large Numbers (LLN).
- The distribution assumes a normal shape as the sample size increases, consistent with the Central Limit Theorem (CLT).
- Although the two estimators use different methods, their results are very similar, and the differences are negligible.