Report on Lab 2

Author: Kevin Deshayes

Email: kede23@student.bth.se

Course Name: Mathematical Statistics

Course Code: MS1403

Date: October 14, 2024

Contents

0.1 R code

```r
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
```

```r
36                        method = "L-BFGS-B", # BFGS with
                          ↪  restrictions
37                        lower = 0.001,  # Set a lower bound to
                          ↪  avoid non-positive mu values
38                        control = list(fnscale = -1))
39        return(result)
40    }
41
42    rr = function(mu, n) {
43        u = runif(n)
44        r_samples = 2 * mu * sqrt(-log(1 - u) / pi)
45        return(r_samples)
46    }
47
48    calc_bias = function(estimates, true_value){
49        return(mean(estimates) - true_value)
50    }
51
52    calc_mse = function(estimates, true_value){
53        return(mean((estimates) - true_value)^2)
54    }
55
56    calculate_metrics = function(estimates, true_value){
57        mean_estimate = mean(estimates)
58        bias = calc_bias(estimates, true_value)
59        mse = calc_mse(estimates, true_value)
60        skewness_val = skewness(estimates)
61        kurtosis_val = kurtosis(estimates)
62
63        return(list(mean = mean_estimate, bias = bias, mse =
           ↪  mse, skewness = skewness_val, kurtosis =
           ↪  kurtosis_val))
64    }
65
66    computation = function(n) {
67        r_values = rr(mu = 5, n) # Generate random samples with
           ↪  n in size from n_values
68        log_likelihood = Log_Rayleigh_likelihood(r_values, mu =
           ↪  5) # mu is an initial guess
69
70        numerical_result =
           ↪  numericalDerivative(Log_Rayleigh_likelihood,
           ↪  r_values)
71        analytical_maximization =
           ↪  analyticalMaximizing(r_values)
```

3

```r
        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
```

4

```r
111          mean_analytical <- aggregate(Estimate ~ Sample_Size,
        ↪   data = estimate_df_analytical, FUN = mean)
112          mean_analytical$Method <- "Analytical"
113
114          # Combine both data frames for plotting
115          combined_mean_df <- rbind(mean_numerical,
        ↪   mean_analytical)
116
117          # Plot mean estimates with a horizontal line at
        ↪   true_mu
118          ggplot(combined_mean_df, aes(x = Sample_Size, y =
        ↪   Estimate, color = Method, group = Method)) +
119              geom_line(aes(linetype = Method), linewidth = 1.2)
                ↪   +   # Use linewidth instead of size
120              geom_point(size = 3) +
121              geom_hline(yintercept = true_mu, linetype =
                ↪   "dashed", color = "black", linewidth = 1) +
122              labs(title = "Mean Estimates of Numerical and
                ↪   Analytical Methods",
123                  x = "Sample Size", y = "Mean Estimate") +
124              theme_minimal() +
125              scale_color_manual(values = c("Numerical" = "blue",
                ↪   "Analytical" = "red")) +
126              theme(legend.position = "top")
127      }
128
129      # Function to plot the bell curves
130      plot_bell_curve <- function(numerical_estimates,
        ↪   analytical_estimates, true_mu) {
131          # Convert estimates to data frames for plotting
132          df_numerical <- data.frame(x = numerical_estimates,
        ↪   Method = "Numerical")
133          df_analytical <- data.frame(x = analytical_estimates,
        ↪   Method = "Analytical")
134
135          # Combine the two data frames
136          df_combined <- rbind(df_numerical, df_analytical)
137
138          # Get standard deviation for the normal distribution
        ↪   based on estimates
139          combined_sd <- sd(df_combined$x)
140
141          # Create the normal distribution curve using the
        ↪   true_mu and calculated std deviation
142          normal_curve <- data.frame(
```

```r
143              x = seq(min(df_combined$x) - 1, max(df_combined$x)
                 ↪  + 1, length.out = 100),
144              y = dnorm(seq(min(df_combined$x) - 1,
                 ↪  max(df_combined$x) + 1, length.out = 100),
145                    mean = true_mu,
146                    sd = combined_sd),
147            Method = "Normal"
148          )

149

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

153

154          # Plot the curves
155          ggplot() +
156              geom_density(data = df_combined, aes(x = x, color =
                 ↪  Method, linetype = Method), size = 1.5) +    #
                 ↪  Plot density for numerical and analytical
                 ↪  estimates
157              geom_line(data = normal_curve, aes(x = x, y = y,
                 ↪  color = Method, linetype = Method), size = 1.5)
                 ↪  +  # Add the normal distribution curve
158              labs(title = "Bell Curves for Numerical,
                 ↪  Analytical, and Normal Distributions",
159                  x = "Estimate", y = "Density") +
160              theme_minimal() +
161              scale_color_manual(values = c("Numerical" = "blue",
                 ↪  "Analytical" = "red", "Normal" = "black")) +
162              scale_linetype_manual(values = c("Numerical" =
                 ↪  "dashed", "Analytical" = "solid", "Normal" =
                 ↪  "dotted")) +
163              theme(legend.position = "top")
164      }

165

166

167

168

169

170      # --- Main Computation --- #
171      set.seed(123) # For reproducibility in different runs of
         ↪  the code
172      n_value = c(10, 20, 50, 90, 140)
```

```
173     true_mu = 5
174
175     # Separate lists for the results
176     all_results_numerical = list()
177     all_results_analytical = list()
178
179     # Run the Monte Carlo simulation and store results
180     bias_mse_df_numerical = data.frame(Sample_Size = integer(),
        ↪  Bias = double(), MSE = double())
181     bias_mse_df_analytical = data.frame(Sample_Size =
        ↪  integer(), Bias = double(), MSE = double())
182
183     estimate_df_numerical = data.frame(Sample_Size = integer(),
        ↪  Estimate = double(), Method = character())
184     estimate_df_analytical = data.frame(Sample_Size =
        ↪  integer(), Estimate = double(), Method = character())
185
186     # --- Sample Sizes Loop --- #
187     n_reps = 1000
188
189     # Iterate through all given sample sizes
190     for (n in n_value) {
191         cat("\n--- Processing n =", n, "---\n")
192
193         numerical_estimates = c()
194         analytical_estimates = c()
195
196         # Repeat the computation 1000 times for each sample
            ↪  size
197         for (rep in 1:n_reps) {
198             result <- computation(n)
199
200             # Clean numerical and analytical results
                ↪  separately
201             cleaned_numerical = clean_results(result, true_mu,
                ↪  "numerical_result")
202             cleaned_analytical = clean_results(result, true_mu,
                ↪  "analytical_maximization")
203
204             # Store numerical results if they exist
205             if (is.list(cleaned_numerical)) {
206                 numerical_estimates = c(numerical_estimates,
                    ↪  cleaned_numerical$converged_estimates)
207             }
208
```

```r
209            # Store analytical results if they exist
210            if (is.list(cleaned_analytical)) {
211                analytical_estimates = c(analytical_estimates,
                   ↪  cleaned_analytical$converged_estimates)
212            }
213         }
214
215         # After 1000 repetitions
216         if (length(numerical_estimates) > 0) {
217            cat("\nNumerical Method Results for n =", n, "
                  ↪  after", n_reps, "repetitions:\n")
218            numerical_metrics =
                  ↪  calculate_metrics(numerical_estimates, true_mu)
219            print(numerical_metrics)
220
221            if (is.list(cleaned_numerical)) {
222                estimate_df_numerical =
                      ↪  rbind(estimate_df_numerical, data.frame(
223                    Sample_Size = rep(n,
                          ↪  length(cleaned_numerical$converged_estimates)),
224                    Estimate = numerical_estimates,
225                    Method = "Numerical"
226                ))
227            }
228         }
229
230         if (length(analytical_estimates) > 0) {
231            cat("\nAnalytical Method Results for n =", n, "
                  ↪  after", n_reps, "repetitions:\n")
232            analytical_metrics =
                  ↪  calculate_metrics(analytical_estimates,
                  ↪  true_mu)
233            print(analytical_metrics)
234
235            if (is.list(cleaned_analytical)) {
236                estimate_df_analytical =
                      ↪  rbind(estimate_df_analytical, data.frame(
237                    Sample_Size = rep(n,
                          ↪  length(cleaned_analytical$converged_estimates)),
238                    Estimate = analytical_estimates,
239                    Method = "Analytical"
240                ))
241            }
242         }
243
```

```
244        }
245
246        # After the simulations
247        # Call the function to plot the mean estimates
248        print(plot_mean_estimates(estimate_df_numerical,
           ↪   estimate_df_analytical, true_mu = 5))
249        print(plot_bell_curve(numerical_estimates,
           ↪   analytical_estimates, true_mu = 5))
```

0.2   Results

n	Method	Mean	Bias	MSE	Skewness	Kurtosis
10	Numerical	4.905042000	-0.094957570	0.009016940	0.240844800	3.216145000
10	Analytical	4.905042000	-0.094957740	0.009016973	0.240844800	3.216145000
20	Numerical	4.950229000	-0.049771060	0.002477158	0.124840100	3.030776000
20	Analytical	4.950229000	-0.049771230	0.002477175	0.124840100	3.030776000
50	Numerical	4.984634000	-0.015365770	0.000236106	0.061915120	3.024987000
50	Analytical	4.984634000	-0.015365940	0.000236112	0.061915110	3.024987000
90	Numerical	5.007175000	0.007175166	5.1483×10^{-5}	0.058607470	3.049959000
90	Analytical	5.007175000	0.007175000	5.1481×10^{-5}	0.058607470	3.049959000
140	Numerical	4.995738000	-0.004261530	1.8161×10^{-5}	0.085037030	3.152488000
140	Analytical	4.995738000	-0.004261697	1.8162×10^{-5}	0.085037020	3.152488000

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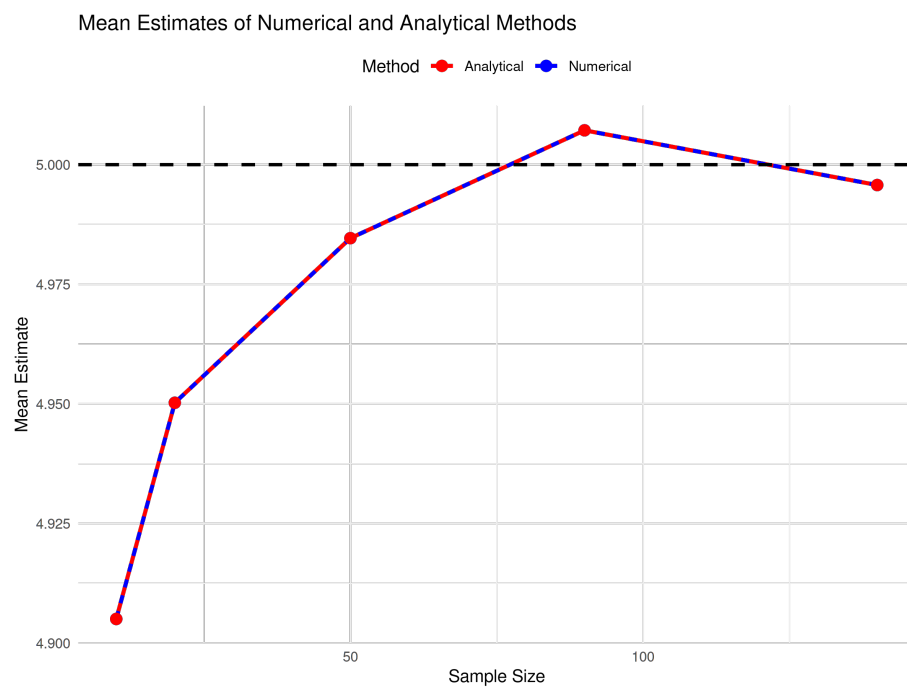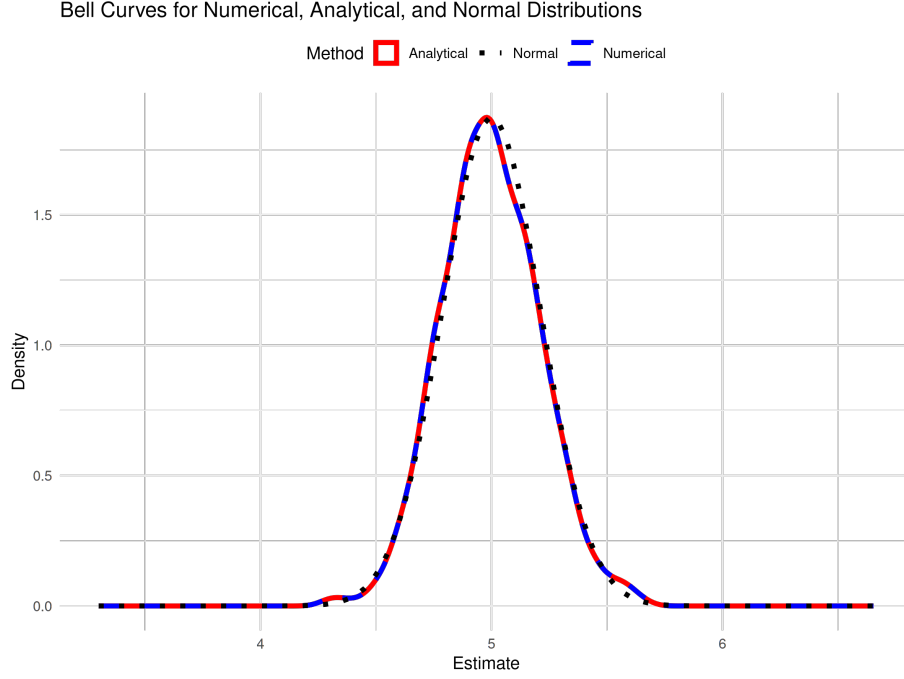
Figure 1: Comparison on estimations

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 distribution 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.