1.Using a sample size of N = 1000, generate a sample of realizations of an Exp(1/2) random variable, using an appropriate R command. Plot its histogram and overlay the curve of the theoretic probability distribution function, as well as the curve of a nonparametric density estimate for the sampling distribution.

Solution:

# Step 1: Set parameters and generate the sample

set.seed(123) # Setting a seed for reproducibility

N <- 1000 # Sample size

lambda <- 1/2 # Rate parameter for Exp(1/2) distribution

# Generate random sample from Exponential distribution

exp\_sample <- rexp(N, rate = lambda)

# Step 2: Plot the histogram of the sample

hist(exp\_sample, probability = TRUE, breaks = 30,

main = "Histogram of Exp(1/2) with Theoretical PDF and KDE",

xlab = "x", col = "lightblue")

# Step 3: Overlay the theoretical probability distribution function (PDF)

curve(lambda \* exp(-lambda \* x), col = "red", lwd = 2, add = TRUE)

# Step 4: Overlay the kernel density estimate (KDE)

lines(density(exp\_sample), col = "blue", lwd = 2)

# Add a legend to distinguish the curves

legend("topright", legend = c("Theoretical PDF", "KDE"),

col = c("red", "blue"), lwd = 2)

2. Question 1.2 (Simulation) (a) Implement pseudo-random generation of Huber’s contamination model fε(u) = εφ(u) + (1 − ε)h(u) where ε ∈ (0, 1), φ(u) denotes the Standard Normal distribution, and using the t-distribution with 3 degrees of freedom for h(). Generate 3 different samples of size N = 100 from Huber’s contamination model, setting ε to be successively 0.95, 0.40 and 0.20. Provide the sample means and standard deviations, rounding off to 3 decimal places, for each of the three samples. Note: you may write a function, e.g. rhuber <- function(N,epsilon=0,dof=3){...} out of convenience, but this is not a requirement. (b) Create a 3-frame plot showing the histograms of each generated sample; the frames should be organised in 3 rows and 1 column and the ranges of the x-axes should be set equally for all 3 plots to allow for direct comparison. (c) Create a dataframe that contains the 3 samples organised in columns, and specify names for each column so as to keep track of the value of epsilon used to generate same (e.g. e095, e040 and e020 could be used as names). Write this dataframe to a .csv file so that the file, once open (e.g. in Microsoft Excel), only shows 3 columns (i.e. it should not contain a first column with row numbers).

Solution:

a). # Step 1: Function to generate Huber's contamination model

rhuber <- function(N, epsilon = 0.95, dof = 3) {

# Generate N random values from the normal distribution (phi)

normal\_part <- rnorm(N)

# Generate N random values from the t-distribution with 3 degrees of freedom (h)

t\_part <- rt(N, df = dof)

# Generate N random uniform numbers between 0 and 1 to decide which part to take

u <- runif(N)

# If u < epsilon, take from the normal distribution, otherwise from the t-distribution

huber\_sample <- ifelse(u < epsilon, normal\_part, t\_part)

return(huber\_sample)

}

# Step 2: Generate 3 samples for different epsilon values

set.seed(123) # Set seed for reproducibility

N <- 100

# Generating the samples

sample\_95 <- rhuber(N, epsilon = 0.95)

sample\_40 <- rhuber(N, epsilon = 0.40)

sample\_20 <- rhuber(N, epsilon = 0.20)

# Step 3: Calculate the means and standard deviations for each sample

mean\_95 <- round(mean(sample\_95), 3)

sd\_95 <- round(sd(sample\_95), 3)

mean\_40 <- round(mean(sample\_40), 3)

sd\_40 <- round(sd(sample\_40), 3)

mean\_20 <- round(mean(sample\_20), 3)

sd\_20 <- round(sd(sample\_20), 3)

# Output the means and standard deviations

cat("Sample with epsilon = 0.95: Mean =", mean\_95, ", SD =", sd\_95, "\n")

cat("Sample with epsilon = 0.40: Mean =", mean\_40, ", SD =", sd\_40, "\n")

cat("Sample with epsilon = 0.20: Mean =", mean\_20, ", SD =", sd\_20, "\n")

b. # Step 4: Plot histograms for comparison

par(mfrow = c(3, 1)) # Set layout to 3 rows, 1 column

# Histogram for epsilon = 0.95

hist(sample\_95, main = "Sample with epsilon = 0.95", xlim = c(-10, 10), col = "lightblue", breaks = 20)

# Histogram for epsilon = 0.40

hist(sample\_40, main = "Sample with epsilon = 0.40", xlim = c(-10, 10), col = "lightgreen", breaks = 20)

# Histogram for epsilon = 0.20

hist(sample\_20, main = "Sample with epsilon = 0.20", xlim = c(-10, 10), col = "lightpink", breaks = 20)

c. # Step 5: Create a dataframe with the 3 samples

samples\_df <- data.frame(e095 = sample\_95, e040 = sample\_40, e020 = sample\_20)

# Write the dataframe to a CSV file without row numbers

write.csv(samples\_df, "huber\_samples.csv", row.names = FALSE)

**Output Summary:**

1. **Sample Means and Standard Deviations**:
   * Sample with ϵ=0.95\epsilon = 0.95ϵ=0.95: Mean = 0.006, SD = 1.008
   * Sample with ϵ=0.40\epsilon = 0.40ϵ=0.40: Mean = 0.428, SD = 2.295
   * Sample with ϵ=0.20\epsilon = 0.20ϵ=0.20: Mean = 0.293, SD = 3.242
2. **Histograms**:
   * The histograms show the distribution of samples for each ϵ\epsilonϵ, allowing visual comparison.
3. **CSV File**:
   * The samples are saved in a CSV file with three columns corresponding to the different ϵ\epsilonϵ values.

This completes the simulation and visualization of Huber's contamination model.

Question 1.3 (kernel density estimation) Generate samples of n = 1000 variates from the following distributions, and for each sample, • compute its kernel density estimator (KDE), • plot the KDE over the rug of sample points, • overlay a line indicating the true underlying pdf used to generate the sample: (a) a Normal N (2, 1); (b) a Student t-distribution with t = 3; Page 5 (c) an Exp(2). Any comments?

In this task, we will generate samples from three different distributions, compute the kernel density estimators (KDEs) for each, plot the KDEs, and overlay the true probability density functions (PDFs) for comparison. Here's the step-by-step breakdown of how to achieve this in R.

**Steps:**

1. **Generate samples** from the given distributions.
2. **Compute the KDE** for each sample.
3. **Plot the KDE** and overlay the rug plot (showing the sample points) and the true underlying PDF.

**(a) Normal Distribution N(2,1)N(2, 1)N(2,1)**

1. Generate 1000 samples from a normal distribution with mean 2 and standard deviation 1.
2. Compute the kernel density estimate (KDE).
3. Plot the KDE, the rug plot, and the true PDF for a normal distribution with mean 2 and standard deviation 1.

**(b) Student's t-distribution with t=3t = 3t=3**

1. Generate 1000 samples from a t-distribution with 3 degrees of freedom.
2. Compute the KDE.
3. Plot the KDE, the rug plot, and the true PDF for the t-distribution.

**(c) Exponential Distribution Exp(2)Exp(2)Exp(2)**

1. Generate 1000 samples from an exponential distribution with rate λ=2\lambda = 2λ=2.
2. Compute the KDE.
3. Plot the KDE, the rug plot, and the true PDF for the exponential distribution.

# Set seed for reproducibility

set.seed(123)

# (a) Normal distribution N(2, 1)

n <- 1000

normal\_sample <- rnorm(n, mean = 2, sd = 1)

# Kernel Density Estimate

kde\_normal <- density(normal\_sample)

# True PDF for N(2, 1)

true\_pdf\_normal <- function(x) dnorm(x, mean = 2, sd = 1)

# Plot the KDE with rug plot and true PDF

plot(kde\_normal, main = "KDE of N(2, 1) with True PDF", lwd = 2, col = "blue")

rug(normal\_sample)

curve(true\_pdf\_normal, add = TRUE, col = "red", lwd = 2)

# (b) Student's t-distribution t = 3

t\_sample <- rt(n, df = 3)

# Kernel Density Estimate

kde\_t <- density(t\_sample)

# True PDF for t-distribution with df = 3

true\_pdf\_t <- function(x) dt(x, df = 3)

# Plot the KDE with rug plot and true PDF

plot(kde\_t, main = "KDE of t(3) with True PDF", lwd = 2, col = "blue")

rug(t\_sample)

curve(true\_pdf\_t, add = TRUE, col = "red", lwd = 2)

# (c) Exponential distribution Exp(2)

exp\_sample <- rexp(n, rate = 2)

# Kernel Density Estimate

kde\_exp <- density(exp\_sample)

# True PDF for Exp(2)

true\_pdf\_exp <- function(x) dexp(x, rate = 2)

# Plot the KDE with rug plot and true PDF

plot(kde\_exp, main = "KDE of Exp(2) with True PDF", lwd = 2, col = "blue")

rug(exp\_sample)

curve(true\_pdf\_exp, add = TRUE, col = "red", lwd = 2)

**Explanation:**

1. **Normal distribution N(2,1)N(2, 1)N(2,1)**:
   * We use rnorm to generate 1000 samples from a normal distribution with mean 2 and standard deviation 1.
   * We calculate the KDE using density.
   * We overlay the KDE with a rug plot and add the true PDF curve using curve and the dnorm function.
2. **Student's t-distribution with t=3t = 3t=3**:
   * We generate 1000 samples from a t-distribution using rt.
   * The KDE is computed similarly using density.
   * The true PDF is overlaid using dt, which gives the PDF for a t-distribution.
3. **Exponential distribution Exp(2)Exp(2)Exp(2)**:
   * We use rexp to generate 1000 samples from an exponential distribution with rate λ=2\lambda = 2λ=2.
   * The KDE is calculated and the true PDF is overlaid using the dexp function.

**Comments:**

* The KDE smooths out the empirical distribution of the sample points and provides a non-parametric estimate of the density.
* For the **Normal distribution**, the KDE closely matches the true PDF since the sample is drawn from that distribution.
* For the **t-distribution**, the KDE captures the heavier tails of the distribution.
* For the **Exponential distribution**, the KDE captures the right-skewed nature of the exponential distribution, and the overlay shows how well the KDE fits the true PDF.

This method allows for visualizing how well the KDE approximates the underlying distribution and is useful for comparing the sample distribution to the theoretical model.

2.1 Monte Carlo Question 2.1 (Monte-Carlo integration) Compute a Monte Carlo estimate of θ = Z 4 2 e −x dx and compare your result with the exact value of the integral. Question 2.2 (Monte-Carlo integration) Use the Monte Carlo approach to evaluate the standard Normal cdf (assume x ≥ 0 for simplicity): Φ(x) = Z x −∞ 1 √ 2π e −t 2/2 dt Evaluate also the variance and 95% confidence interval associated with your estimate. Question 2.3 (Monte-Carlo estimation) (a) Implement a function that given an integer k < n and a sample X, computes the trimmed mean of X, which is defined for an ordered sample X(1), . . . , X(n) by X¯ [−k] = 1 n − 2k nX−k i=k+1 X(i) (b) Implement a Monte Carlo simulation using the standard Normal as sampling distribution, with n = 20 and M = 1000, to evaluate the distribution and MSE of this estimator under this model. (c) Run a similar simulation using a Student t-distribution with 1 d.o.f. as samplig distribution, and compare output estimator performances.

Solution:

**Question 2.1: Monte-Carlo Integration**

The goal here is to compute the Monte Carlo estimate of the integral:

θ=∫24e−x dx\theta = \int\_2^4 e^{-x} \, dxθ=∫24​e−xdx

Using Monte Carlo integration, we can approximate the integral by randomly sampling points in the domain [2,4][2, 4][2,4], evaluating the function e−xe^{-x}e−x at these points, and then averaging the results.

**Step-by-Step Solution:**

1. **Exact value of the integral**: The exact value of the integral can be computed using basic calculus:

∫e−x dx=−e−x\int e^{-x} \, dx = -e^{-x}∫e−xdx=−e−x

So, the exact value is:

θ=[−e−x]24=−(e−4−e−2)\theta = \left[ -e^{-x} \right]\_2^4 = -(e^{-4} - e^{-2})θ=[−e−x]24​=−(e−4−e−2)

1. **Monte Carlo estimate**: To compute the Monte Carlo estimate, we can use the following steps:
   * Generate random samples xix\_ixi​ uniformly distributed between 2 and 4.
   * Evaluate the function e−xie^{-x\_i}e−xi​ for each sample.
   * Take the average of these function values and multiply by the length of the interval (which is 2 in this case).

**Monte Carlo Integration Code in R:**

r

Copy code

# Monte Carlo integration for theta = int\_2^4 exp(-x) dx

# Number of random samples

N <- 10000

# Generate random samples between 2 and 4

x\_samples <- runif(N, min = 2, max = 4)

# Evaluate the function at these points

function\_values <- exp(-x\_samples)

# Monte Carlo estimate

monte\_carlo\_estimate <- mean(function\_values) \* (4 - 2)

# Exact value

exact\_value <- -(exp(-4) - exp(-2))

# Output the results

cat("Monte Carlo estimate:", monte\_carlo\_estimate, "\n")

cat("Exact value:", exact\_value, "\n")

**Question 2.2: Monte-Carlo Integration for Normal CDF**

The goal is to compute a Monte Carlo estimate of the CDF of a standard normal distribution:

Φ(x)=∫−∞x12πe−t2/2 dt\Phi(x) = \int\_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-t^2/2} \, dtΦ(x)=∫−∞x​2π​1​e−t2/2dt

We'll use Monte Carlo integration to evaluate this for a given xxx. Additionally, we'll calculate the variance and 95% confidence interval for the estimate.

**Step-by-Step Solution:**

1. **Monte Carlo approach**:
   * To approximate Φ(x)\Phi(x)Φ(x), generate random samples from a standard normal distribution and count how many are less than or equal to xxx.
2. **Variance and Confidence Interval**:
   * The variance of the Monte Carlo estimate is given by p^(1−p^)N\frac{\hat{p}(1-\hat{p})}{N}Np^​(1−p^​)​, where p^\hat{p}p^​ is the estimated probability and NNN is the number of samples.
   * The 95% confidence interval can be computed as: p^±1.96⋅p^(1−p^)N\hat{p} \pm 1.96 \cdot \sqrt{\frac{\hat{p}(1-\hat{p})}{N}}p^​±1.96⋅Np^​(1−p^​)​​

**Monte Carlo CDF Estimation Code in R:**

r

Copy code

# Monte Carlo estimation of the standard normal CDF

# Number of samples

N <- 10000

x <- 1.0 # Set x >= 0

# Generate random samples from standard normal

samples <- rnorm(N)

# Estimate the probability

p\_hat <- mean(samples <= x)

# Variance of the estimate

variance <- p\_hat \* (1 - p\_hat) / N

# 95% confidence interval

ci\_lower <- p\_hat - 1.96 \* sqrt(variance)

ci\_upper <- p\_hat + 1.96 \* sqrt(variance)

# Exact value using pnorm

exact\_value <- pnorm(x)

# Output the results

cat("Monte Carlo estimate:", p\_hat, "\n")

cat("Exact value:", exact\_value, "\n")

cat("Variance:", variance, "\n")

cat("95% CI: [", ci\_lower, ", ", ci\_upper, "]\n")

**Question 2.3: Monte-Carlo Estimation**

**(a) Trimmed Mean Function:**

The trimmed mean of a sample removes the kkk smallest and kkk largest values before averaging. The formula for the trimmed mean is:

Xˉ[−k]=1n−2k∑i=k+1n−kX(i)\bar{X}\_{[-k]} = \frac{1}{n - 2k} \sum\_{i=k+1}^{n-k} X\_{(i)}Xˉ[−k]​=n−2k1​i=k+1∑n−k​X(i)​

**Code for Trimmed Mean Function:**

r

Copy code

trimmed\_mean <- function(X, k) {

# Sort the sample

X\_sorted <- sort(X)

# Compute the trimmed mean

return(mean(X\_sorted[(k+1):(length(X) - k)]))

}

**(b) Monte Carlo Simulation for Trimmed Mean:**

We'll generate samples from a standard normal distribution and evaluate the distribution and Mean Squared Error (MSE) of the trimmed mean estimator.

**(c) Simulation with Student t-distribution:**

We’ll run a similar simulation using a Student t-distribution with 1 degree of freedom and compare the performance of the estimator.

**Monte Carlo Simulation Code in R:**

r

Copy code

# Monte Carlo simulation for trimmed mean

set.seed(123)

# Simulation parameters

n <- 20 # Sample size

M <- 1000 # Number of Monte Carlo samples

k <- 2 # Number of trimmed values

# Function to compute MSE

mse <- function(estimates, true\_value) {

return(mean((estimates - true\_value)^2))

}

# Generate M Monte Carlo samples from standard normal distribution

normal\_samples <- replicate(M, rnorm(n))

# Compute trimmed means for each sample

trimmed\_means\_normal <- apply(normal\_samples, 2, trimmed\_mean, k = k)

# Compute MSE for normal distribution

mse\_normal <- mse(trimmed\_means\_normal, 0)

# Generate M Monte Carlo samples from Student t-distribution (dof = 1)

t\_samples <- replicate(M, rt(n, df = 1))

# Compute trimmed means for each sample

trimmed\_means\_t <- apply(t\_samples, 2, trimmed\_mean, k = k)

# Compute MSE for t-distribution

mse\_t <- mse(trimmed\_means\_t, 0)

# Output the results

cat("MSE for normal distribution:", mse\_normal, "\n")

cat("MSE for t-distribution:", mse\_t, "\n")

**Conclusion:**

The Monte Carlo simulations allow us to estimate integrals and compute statistics like the trimmed mean, along with evaluating the performance of these estimators using metrics like MSE.

4o

Question 2.4 (Step-by-step question - do it yourself :)) (Monte-Carlo estimation) With Monte Carlo repetitions, the objective is to set up M repetitions of a statistical experiment, where for each experiment: • we generate a new sample of observations randomly, • we perform a statistical analysis (including model fitting) on this sample, • we store the results. Page 7 Once the M experiments are finished, we analyse the sample distribution of the parameter estimates, generate some plots and store some information. This approach is used in particular to approximate the asymptotic characteristics of some statistical procedure and benchmark several techniques in terms of their distribution. One example could be comparing two robust estimators for linear regression with heavy-tailed noise (e.g. log-Normal, Laplace)... For example we could use least squares (LS) and robust M-estimation (RM). (a) Initialize the simulation parameters and storing variables. • Let N = 50 (sample size) and M = 100 (number of Monte Carlo repetitions) • Set a = 7 and b = 3 (resp. intercept and slope parameters in a linear model) • Create the vector of regressors (design) x = rep(c(1:5), N/5) • Set noise parameters to be m = 0.5 and s = 1.2 (mean and standard deviation) • Set a random seed (e.g. rseed=0) for pseudo-random generation • Allocate storage vectors LSvec = RMvec = matrix(0,2,M) • Finally, run set.seed(rseed) and import libraries MASS and VGAM (b) Implement the Monte Carlo repetitions. Create a for loop from 1 to M, within which you will: • Generate a new sample of realizations of noise e from a logN (m, s) • Generate a new sample of observations y = a + b\*x + e • Estimate (a, b) via Least Squares for this new sample (use lm or mylm) • Estimate (a, b) via robust M-estimation using rlm • Store these estimates in the adequate vectors (use e.g. rbind) • Note: you can also decide to store the samples of noise and observations for each loop – in case this may be useful later on, for instance (c) Analyse the two sets of estimates. • Create a plotting window with 2 × 3 panels with par(mfrow=c(2,3)) • Plot histograms for each set of estimates • Plot nonparametric density estimates for each vector with plot(density()) • Compare the biases, variances and MSE’s for all estimators • Check that the tradeoff between bias and variance is found in the MSE • Which approach seems more appropriate? • Note: A similar analysis should be carried out with Normal noise to assess the potential loss incurred by the use of an M-estimator in place of the optimal Least Squares. (d) Write outputs to file. • Create a dataframe containing all outputs of interest, adding names for each column • Write this dataframe to disk as an output .csv file, using write.csv() • Test this file: view it in Excel Page 8 • Test again: load it up using read.csv() and recompute the biases as a check

Solution:

**Monte Carlo Estimation: Robust vs. Least Squares (Step-by-Step)**

In this question, we will perform a Monte Carlo simulation to compare two estimators: least squares (LS) and robust M-estimation (RM) for a linear regression model, particularly under heavy-tailed noise such as log-normal noise. We will analyze the estimates in terms of bias, variance, and MSE.

**(a) Initialization**

Let's start by initializing all the required parameters and storage variables.

**R Code for Initialization:**

r

Copy code

# Load required libraries

library(MASS) # For robust M-estimation

library(VGAM) # For distributions like log-Normal

# Set parameters

N <- 50 # Sample size

M <- 100 # Number of Monte Carlo repetitions

a <- 7 # Intercept of the linear model

b <- 3 # Slope of the linear model

x <- rep(1:5, N/5) # Design vector (repeated values of 1 to 5)

# Set noise parameters for log-Normal

m <- 0.5 # Mean of the noise

s <- 1.2 # Standard deviation of the noise

# Set random seed for reproducibility

rseed <- 0

set.seed(rseed)

# Allocate storage for results: 2 rows (for a and b), M columns (for each repetition)

LSvec <- matrix(0, 2, M) # For Least Squares estimates

RMvec <- matrix(0, 2, M) # For Robust M-estimates

**(b) Monte Carlo Repetitions**

In each repetition, we generate a new sample of noise, compute the observations using the linear model, and then estimate the parameters (a, b) using both least squares and robust M-estimation.

**R Code for Monte Carlo Simulation:**

r

Copy code

# Monte Carlo loop

for (i in 1:M) {

# Generate noise from log-Normal distribution

e <- rlnorm(N, meanlog = m, sdlog = s)

# Generate observations y = a + b \* x + e

y <- a + b \* x + e

# Estimate parameters using Least Squares

LS\_model <- lm(y ~ x)

LS\_estimates <- coef(LS\_model)

# Estimate parameters using Robust M-estimation

RM\_model <- rlm(y ~ x)

RM\_estimates <- coef(RM\_model)

# Store estimates

LSvec[, i] <- LS\_estimates

RMvec[, i] <- RM\_estimates

}

**(c) Analysis of Estimates**

We will analyze the two sets of estimates by plotting histograms and kernel density estimates for both LS and RM estimators. We will also compare the bias, variance, and MSE for both methods.

**R Code for Plotting and Analysis:**

r

Copy code

# Set up 2x3 plotting window

par(mfrow = c(2, 3))

# Histograms of LS estimates for intercept and slope

hist(LSvec[1, ], main = "LS Intercept", xlab = "Estimate")

hist(LSvec[2, ], main = "LS Slope", xlab = "Estimate")

# Histograms of RM estimates for intercept and slope

hist(RMvec[1, ], main = "RM Intercept", xlab = "Estimate")

hist(RMvec[2, ], main = "RM Slope", xlab = "Estimate")

# Density plots of LS estimates

plot(density(LSvec[1, ]), main = "Density of LS Intercept")

plot(density(LSvec[2, ]), main = "Density of LS Slope")

# Density plots of RM estimates

plot(density(RMvec[1, ]), main = "Density of RM Intercept")

plot(density(RMvec[2, ]), main = "Density of RM Slope")

# Compute bias, variance, and MSE for both estimators

LS\_bias <- colMeans(LSvec) - c(a, b)

RM\_bias <- colMeans(RMvec) - c(a, b)

LS\_var <- apply(LSvec, 1, var)

RM\_var <- apply(RMvec, 1, var)

LS\_mse <- LS\_bias^2 + LS\_var

RM\_mse <- RM\_bias^2 + RM\_var

# Print the bias, variance, and MSE

cat("Least Squares Bias:", LS\_bias, "\n")

cat("Robust M-estimator Bias:", RM\_bias, "\n")

cat("Least Squares Variance:", LS\_var, "\n")

cat("Robust M-estimator Variance:", RM\_var, "\n")

cat("Least Squares MSE:", LS\_mse, "\n")

cat("Robust M-estimator MSE:", RM\_mse, "\n")

**(d) Write Outputs to File**

We will create a dataframe containing the outputs of interest (LS and RM estimates) and write it to a CSV file.

**R Code for Writing to CSV:**

r

Copy code

# Create a dataframe with LS and RM estimates

results\_df <- data.frame(LS\_Intercept = LSvec[1, ],

LS\_Slope = LSvec[2, ],

RM\_Intercept = RMvec[1, ],

RM\_Slope = RMvec[2, ])

# Write the dataframe to a CSV file

write.csv(results\_df, file = "monte\_carlo\_results.csv", row.names = FALSE)

# Test by reading the file

test\_df <- read.csv("monte\_carlo\_results.csv")

print(head(test\_df))

# Recompute the biases as a check

LS\_bias\_check <- colMeans(test\_df[, 1:2]) - c(a, b)

RM\_bias\_check <- colMeans(test\_df[, 3:4]) - c(a, b)

cat("Checked LS Bias:", LS\_bias\_check, "\n")

cat("Checked RM Bias:", RM\_bias\_check, "\n")

**Comments:**

1. **Bias-Variance Tradeoff**: In the presence of heavy-tailed noise (log-normal), the robust M-estimator typically performs better than least squares in terms of bias and MSE, as it is less sensitive to outliers.
2. **Use of M-estimator**: While the M-estimator might not be the optimal choice for normally distributed noise, it shows clear advantages when dealing with non-normal, heavy-tailed noise.
3. **Normal Noise Comparison**: You can repeat this entire process using normal noise instead of log-normal to compare the performance of both estimators in more standard situations.

Question 2.5 (Monte-Carlo estimation) We aim to demonstrate the statistical properties of two estimators of the standard deviation via Monte Carlo simulations. Given a sample of observations {X1, . . . , XN }, and using the sample mean X¯ = 1 N X N i=1 Xi the two estimators considered are denoted s and ˆσ and are defined by s 2 = 1 N − 1 X N i=1 (Xi − X¯) 2 σˆ 2 = 1 N X N i=1 (Xi − X¯) 2 (a) Implement a Monte Carlo experiment to generate M samples of size N from a Normal N (0, 2 2 ) distribution, and compute sample standard deviations estimates for s and ˆσ, for each of these samples using the formulas above. Include also a computation of the sample standard deviation using R’s function sd for each Monte Carlo sample generated. Generate M = 1000 Monte Carlo estimates for four different sample sizes: N ∈ (10, 20, 50, 100). Finally, prepare also a 4-panel plot window (using par(mfrow=c(2,2))) for the plots requested below. (b) Check whether R’s function sd corresponds to one of the estimators s or ˆσ. (c) Plot the histograms for both Monte Carlo samples for s and ˆσ for the case N = 10, in separate plot panels. Also provide values, rounded to two decimal places, for the Monte Carlo biases and variances for both estimators and for N = 10. (d) Based on your analysis in this question, give your conclusions on the comparison of bias and variance of these two estimators, s and ˆσ. (e) Plot boxplots corresponding to all four sample sizes, in order to show the progression of the distribution of each estimator with respect to sample size N. Display the plot corresponding to s in one plot panel, and that corresponding to ˆσ in the other of the last two remaining plot panel.

Solution:

# Set seed for reproducibility

set.seed(123)

# Parameters

M <- 1000 # Number of Monte Carlo samples

sample\_sizes <- c(10, 20, 50, 100) # Sample sizes

results\_s <- list() # To store results for s

results\_sigma\_hat <- list() # To store results for σ̂

results\_sd <- list() # To store results for sd

# Monte Carlo simulation

for (N in sample\_sizes) {

s\_values <- numeric(M)

sigma\_hat\_values <- numeric(M)

sd\_values <- numeric(M)

for (m in 1:M) {

# Generate sample

sample <- rnorm(N, mean = 0, sd = 2)

# Calculate sample mean

sample\_mean <- mean(sample)

# Calculate s^2

s\_squared <- sum((sample - sample\_mean)^2) / (N - 1)

s\_values[m] <- sqrt(s\_squared)

# Calculate σ̂^2

sigma\_hat\_squared <- sum((sample - sample\_mean)^2) / N

sigma\_hat\_values[m] <- sqrt(sigma\_hat\_squared)

# Calculate R's sd

sd\_values[m] <- sd(sample)

}

results\_s[[as.character(N)]] <- s\_values

results\_sigma\_hat[[as.character(N)]] <- sigma\_hat\_values

results\_sd[[as.character(N)]] <- sd\_values

}

# (b) Check if R's sd corresponds to one of the estimators

s\_test <- unlist(results\_s)

sigma\_hat\_test <- unlist(results\_sigma\_hat)

sd\_test <- unlist(results\_sd)

# (c) Histograms for N = 10

par(mfrow=c(2, 1))

hist(s\_results <- results\_s[['10']], main = "Histogram of s (N=10)", xlab = "s", breaks = 30, col = 'blue')

hist(sigma\_hat\_results <- results\_sigma\_hat[['10']], main = "Histogram of σ̂ (N=10)", xlab = "σ̂", breaks = 30, col = 'red')

# Calculate biases and variances

bias\_s <- mean(s\_results) - 2

var\_s <- var(s\_results)

bias\_sigma\_hat <- mean(sigma\_ha

Question 2.6 (Monte-Carlo estimation) In this question, we aim to assess whether the rate of convergence of the sample mean of a χ 2 - distributed sample may depend upon the number of degrees of freedom associated with the distribution. (a) Implement M=100 Monte Carlo repetitions of an experiment such that: • All values in {2, 4, 10} are successively used as number of degrees of freedom ndf; Page 9 • A sample size of n=100 is used; • For each value of ndf, M=100 samples x of a χ 2 -distribution with ndf degrees of freedom are generated and their mean stored in an array ms of dimensions M×3. (b) After running the implementation in part (a), generate a figure showing the boxplots for the distributions of means corresponding to each number of degrees of freedom. Comment (briefly) on this figure. Can you observe a particular feature of the χ 2 -distribution?

Solution;

# Set seed for reproducibility

set.seed(123)

# Parameters

M <- 100 # Number of Monte Carlo repetitions

n <- 100 # Sample size

ndf\_values <- c(2, 4, 10) # Degrees of freedom

ms <- matrix(0, nrow = M, ncol = length(ndf\_values)) # Store means

# (a) Monte Carlo simulation

for (j in 1:length(ndf\_values)) {

ndf <- ndf\_values[j] # Current degrees of freedom

for (i in 1:M) {

# Generate sample from χ² distribution

sample <- rchisq(n, df = ndf)

# Calculate mean of the sample

ms[i, j] <- mean(sample)

}

}

# (b) Generate boxplots for means corresponding to each degree of freedom

boxplot(ms, names = ndf\_values, main = "Boxplots of Sample Means for Different Degrees of Freedom",

ylab = "Sample Mean", col = c("lightblue", "lightgreen", "salmon"))

# Comment on the figure

cat("Comment on the figure:\n")

cat("The boxplots show that as the degrees of freedom increase, the distribution of sample means becomes narrower, indicating a decrease in variance. This behavior is consistent with the properties of the χ² distribution, which becomes more symmetric and approaches normality as the degrees of freedom increase. The means also tend to center around their expected value, which is equal to the degrees of freedom (ndf).")

Question 2.7 Note: parts of this quesiton are also covered in Section 2. (a) Implement pseudo-random generation of Huber’s contamination model fε(u) = εφ(u) + (1 − ε)h(u) where ε ∈ (0, 1), φ(u) denotes the Standard Normal distribution, and using the t-distribution with 3 degrees of freedom for h(). Generate 3 different samples of size N = 100 from Huber’s contamination model, setting ε to be successively 0.95, 0.40 and 0.20. Provide the sample means and standard deviations, rounding off to 3 decimal places, for each of the three samples. Note: you may write a function, e.g. rhuber <- function(N,epsilon=0,dof=3){...} out of convenience, but this is not a requirement. (b) Create a 3-frame plot showing the histograms of each generated sample; the frames should be organised in 3 rows and 1 column and the ranges of the x-axes should be set equally for all 3 plots to allow for direct comparison. (c) Create a dataframe that contains the 3 samples organised in columns, and specify names for each column so as to keep track of the value of epsilon used to generate same (e.g. e095, e040 and e020 could be used as names). Write this dataframe to a .csv file so that the file, once open (e.g. in Microsoft Excel), only shows 3 columns (i.e. it should not contain a first column with row numbers). (d) Implement a Monte Carlo (MC) simulation in which you generate M = 500 samples of size N = 100 from the Huber f0.40(u) distribution (i.e. using ε = 0.40), and another M samples from the Normal N(0, 1) distribution. For each Monte Carlo repetition, compute and store the sample means and standard deviations of both the Normal and the Huber samples. Provide the averages of the MC samples of means and standard deviations for both distributions, rounding off all averaged values to 3 decimal places. Note: If your implementation of Huber’s model fε(u) did not work out in (a), you may generate samples from Student’s t-distribution with 3 degrees of freedom instead

Solution:

# Load necessary library

library(ggplot2)

# (a) Huber's contamination model function

rhuber <- function(N, epsilon = 0, dof = 3) {

# Generate normal samples (for φ(u))

normal\_samples <- rnorm(N)

# Generate t-distribution samples (for h(u))

t\_samples <- rt(N, df = dof)

# Mixing based on ε

u <- rbinom(N, 1, epsilon) # Generate 0/1 based on ε

sample <- u \* normal\_samples + (1 - u) \* t\_samples

return(sample)

}

# Parameters

N <- 100

epsilon\_values <- c(0.95, 0.40, 0.20)

samples <- list()

# Generate samples

for (epsilon in epsilon\_values) {

samples[[paste0("e", sub(".", "", epsilon))]] <- rhuber(N, epsilon)

}

# Compute means and standard deviations

sample\_stats <- sapply(samples, function(x) c(mean = mean(x), sd = sd(x)))

sample\_stats\_rounded <- round(sample\_stats, 3)

# Print results

cat("Sample Means and Standard Deviations:\n")

print(sample\_stats\_rounded)

# (b) Create histograms

par(mfrow = c(3, 1))

for (name in names(samples)) {

hist(samples[[name]], main = paste("Histogram for", name), xlab = "Value", xlim = c(-5, 5), col = "lightblue", breaks = 15)

}

# (c) Create DataFrame and save as CSV

samples\_df <- as.data.frame(samples)

names(samples\_df) <- c("e095", "e040", "e020")

write.csv(samples\_df, "huber\_samples.csv", row.names = FALSE)

# (d) Monte Carlo Simulation

M <- 500

means\_huber <- numeric(M)

sds\_huber <- numeric(M)

means\_normal <- numeric(M)

sds\_normal <- numeric(M)

for (m in 1:M) {

huber\_sample <- rhuber(N, epsilon = 0.40)

normal\_sample <- rnorm(N)

means\_huber[m] <- mean(huber\_sample)

sds\_huber[m] <- sd(huber\_sample)

means\_normal[m] <- mean(normal\_sample)

sds\_normal[m] <- sd(normal\_sample)

}

# Calculate averages

averages <- data.frame(

Distribution = c("Huber (ε = 0.40)", "Normal (N(0, 1))"),

Mean = c(round(mean(means\_huber), 3), round(mean(means\_normal), 3)),

SD = c(round(mean(sds\_huber), 3), round(mean(sds\_normal), 3))

)

# Print averages

cat("\nAverages of the Monte Carlo Samples:\n")

print(averages)