Question 1.1 (Simulation)

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.  
  
# Step 1: Set parameters and generate sample

set.seed(123) # for reproducibility

N <- 1000

lambda <- 1 / 2 # rate parameter

sample <- rexp(N, rate = lambda)

# Step 2: Plot the histogram

hist(sample, prob = TRUE, breaks = 30, col = "lightblue",

main = "Histogram of Exp(1/2) Sample with PDF and Density",

xlab = "x")

# Step 3: Overlay the theoretical PDF

curve(dexp(x, rate = lambda), add = TRUE, col = "red", lwd = 2)

# Step 4: Overlay the non-parametric density estimate

lines(density(sample), col = "blue", lwd = 2)

# Add a legend

legend("topright", legend = c("Theoretical PDF", "Kernel Density Estimate"),

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

# Load necessary library

library(ggplot2)

# Function to generate samples from Huber's contamination model

generate\_huber\_samples <- function(N, epsilon, dof = 3) {

# Decide which distribution to sample from

random\_choices <- runif(N)

# Generate samples

normal\_samples <- rnorm(N)

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

# Choose samples based on random choices

samples <- ifelse(random\_choices < epsilon, normal\_samples, t\_samples)

return(samples)

}

# Sample size

N <- 100

# Generate samples for each epsilon value

epsilon\_095 <- generate\_huber\_samples(N, 0.95)

epsilon\_040 <- generate\_huber\_samples(N, 0.40)

epsilon\_020 <- generate\_huber\_samples(N, 0.20)

# Calculate mean and standard deviation for each epsilon

mean\_095 <- round(mean(epsilon\_095), 3)

sd\_095 <- round(sd(epsilon\_095), 3)

mean\_040 <- round(mean(epsilon\_040), 3)

sd\_040 <- round(sd(epsilon\_040), 3)

mean\_020 <- round(mean(epsilon\_020), 3)

sd\_020 <- round(sd(epsilon\_020), 3)

# Print the results

cat("For epsilon = 0.95:\n")

cat("Mean:", mean\_095, "\n")

cat("Standard Deviation:", sd\_095, "\n\n")

cat("For epsilon = 0.40:\n")

cat("Mean:", mean\_040, "\n")

cat("Standard Deviation:", sd\_040, "\n\n")

cat("For epsilon = 0.20:\n")

cat("Mean:", mean\_020, "\n")

cat("Standard Deviation:", sd\_020, "\n\n")

# Part b) Plot histograms in a 3-frame layout

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

# Get common x-axis range for all plots

x\_limits <- range(c(epsilon\_095, epsilon\_040, epsilon\_020))

# Plot histograms

hist(epsilon\_095, main = "Histogram for epsilon = 0.95", xlim = x\_limits,

col = "lightblue", xlab = "Values", ylab = "Frequency", breaks = 10)

hist(epsilon\_040, main = "Histogram for epsilon = 0.40", xlim = x\_limits,

col = "lightblue", xlab = "Values", ylab = "Frequency", breaks = 10)

hist(epsilon\_020, main = "Histogram for epsilon = 0.20", xlim = x\_limits,

col = "lightblue", xlab = "Values", ylab = "Frequency", breaks = 10)

# Reset the plot layout

par(mfrow = c(1, 1))

# Part c) Create dataframe and save to CSV

df <- data.frame(e095 = epsilon\_095, e040 = epsilon\_040, e020 = epsilon\_020)

# Write dataframe to CSV without row numbers

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

**1.2 Nonparametric density estimation**

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

c) an Exp(2).

# Set sample size

n <- 1000

# Set up plotting space for 3 plots

par(mfrow = c(3, 1))

# (a) Normal N(2, 1)

# Generate sample

set.seed(123) # For reproducibility

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

# Compute Kernel Density Estimate

normal\_kde <- density(normal\_sample)

# Plot KDE with a rug plot of the sample points

plot(normal\_kde, main = "KDE for N(2, 1)", xlab = "x", ylab = "Density", col = "blue")

rug(normal\_sample, col = "red") # Rug plot

# Overlay the true PDF for N(2, 1)

curve(dnorm(x, mean = 2, sd = 1), col = "black", lwd = 2, add = TRUE)

# (b) Student t-distribution with t = 3

# Generate sample

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

# Compute Kernel Density Estimate

t\_kde <- density(t\_sample)

# Plot KDE with a rug plot of the sample points

plot(t\_kde, main = "KDE for t-distribution (df = 3)", xlab = "x", ylab = "Density", col = "blue")

rug(t\_sample, col = "red")

# Overlay the true PDF for t-distribution with df = 3

curve(dt(x, df = 3), col = "black", lwd = 2, add = TRUE)

# (c) Exponential Exp(2)

# Generate sample

exp\_sample <- rexp(n, rate = 1/2)

# Compute Kernel Density Estimate

exp\_kde <- density(exp\_sample)

# Plot KDE with a rug plot of the sample points

plot(exp\_kde, main = "KDE for Exp(2)", xlab = "x", ylab = "Density", col = "blue")

rug(exp\_sample, col = "red")

# Overlay the true PDF for Exp(2)

curve(dexp(x, rate = 1/2), col = "black", lwd = 2, add = TRUE)

Question 1.4 (Multivariate density estimation)

(a) Compute and plot a 2D KDE for the Old Faithful dataset in R, using the appropriate function(s) from R’s KernSmooth package.

(b) Compute and plot a nonparametric regression function estimate for the Old Faithful data using local polynomials from the same package.

# Load KernSmooth package

library(KernSmooth)

# Load the Old Faithful dataset

data(faithful)

# Perform 2D Kernel Density Estimation using bkde2D

kde2d\_result <- bkde2D(faithful, bandwidth = c(0.5, 5)) # Adjust bandwidth for smoothness

# Plot the 2D KDE using a filled contour plot

filled.contour(kde2d\_result$x1, kde2d\_result$x2, kde2d\_result$fhat,

color.palette = terrain.colors,

plot.title = title(main = "2D KDE of Old Faithful Data",

xlab = "Eruption Duration (mins)",

ylab = "Waiting Time (mins)"),

key.title = title("Density"))

# Perform local polynomial regression using locpoly

reg\_result <- locpoly(faithful$waiting, faithful$eruptions, bandwidth = 5) # Adjust bandwidth for smoothness

# Plot the original data points (scatter plot)

plot(faithful$waiting, faithful$eruptions,

main = "Nonparametric Regression: Eruption Duration vs Waiting Time",

xlab = "Waiting Time (mins)",

ylab = "Eruption Duration (mins)",

pch = 19, col = "blue")

# Overlay the nonparametric regression estimate (local polynomial fit)

lines(reg\_result$x, reg\_result$y, col = "red", lwd = 2)

Question 1.5 (Multivariate density estimation)

(a) Generate a random sample of size n = 1000 from the N2(μ, Σ) distribution.

(b) Fit a 2D (bivariate Normal) kernel density estimator to the sample (use R’s MASS::kde2d function).

(c) Fit a 2D kernel density estimator to the sample using a product of univariate kernels, i.e. implement the KDE defined by

A mathematical equation with numbers and symbols

Description automatically generated  
with d = 2, h1 = h2 = 0.5, and u ranging over the (x, y) grid computed by kde2d in the

previous question.

(d) For each KDE,

• plot the cloud of points and add a contour of the associated KDE;

• generate a perspective plot with angle θ = 30\*;

• generate a perspective plot with angle θ = −60\*.  
  
# Load necessary library

library(MASS)

# Set seed for reproducibility

set.seed(123)

# Parameters for the bivariate normal distribution

mu <- c(0, 0) # Mean vector

Sigma <- matrix(c(1, 0.5, 0.5, 1), nrow = 2) # Covariance matrix

# Generate random sample

n <- 1000 # Sample size

sample\_data <- mvrnorm(n, mu, Sigma)

# Compute the 2D KDE

kde\_result <- kde2d(sample\_data[, 1], sample\_data[, 2], n = 50)

# Plot the cloud of points and add a contour of the associated KDE

plot(sample\_data, pch = 19, col = rgb(0, 0, 1, 0.5),

main = "Bivariate Normal Sample with KDE Contour",

xlab = "X1", ylab = "X2")

contour(kde\_result$x, kde\_result$y, kde\_result$z, add = TRUE)

# Define the univariate kernel function (Gaussian)

K <- function(u) {

return(dnorm(u))

}

# Bandwidth for each dimension

h1 <- 0.5

h2 <- 0.5

# Function to compute the product of univariate kernels

kde\_product <- function(u1, u2, data, h1, h2) {

n <- nrow(data)

fhat <- 1/n \* sum(sapply(1:n, function(i) {

K((u1 - data[i, 1]) / h1) \* K((u2 - data[i, 2]) / h2)

}))

return(fhat)

}

# Create a grid for the KDE using kde2d results

grid\_points <- expand.grid(x = kde\_result$x, y = kde\_result$y)

kde\_values <- apply(grid\_points, 1, function(point) {

kde\_product(point[1], point[2], sample\_data, h1, h2)

})

# Reshape kde\_values into a matrix for contour plotting

kde\_matrix <- matrix(kde\_values, nrow = length(kde\_result$x), ncol = length(kde\_result$y))

# Plot the cloud of points and add the contour for product KDE

plot(sample\_data, pch = 19, col = rgb(0, 0, 1, 0.5),

main = "Bivariate Normal Sample with Product Kernel KDE Contour",

xlab = "X1", ylab = "X2")

contour(kde\_result$x, kde\_result$y, kde\_matrix, add = TRUE)

# Perspective plot for MASS::kde2d KDE

persp(kde\_result$x, kde\_result$y, kde\_result$z,

theta = 30, phi = 30, expand = 0.5,

main = "Perspective Plot of MASS::kde2d KDE",

xlab = "X1", ylab = "X2", zlab = "Density")

# Perspective plot for product kernel KDE

persp(kde\_result$x, kde\_result$y, kde\_matrix,

theta = -60, phi = 30, expand = 0.5,

main = "Perspective Plot of Product Kernel KDE",

xlab = "X1", ylab = "X2", zlab = "Density")

Question 2.1 (Monte-Carlo integration)

Compute a Monte Carlo estimate of  
  
A black and white math equation

Description automatically generated with medium confidence  
and compare your result with the exact value of the integral.  
  
# Set seed for reproducibility

set.seed(6040)

# Parameters

N <- 10000 # Number of random samples

a <- 2 # Lower bound of the integral

b <- 4 # Upper bound of the integral

# Generate N random points uniformly between 2 and 4

x <- runif(N, a, b)

# Monte Carlo estimate of the integral

monte\_carlo\_estimate <- (b - a) \* mean(exp(-x))

cat("Monte Carlo estimate:", round(monte\_carlo\_estimate, 6), "\n")

# Exact value of the integral

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

cat("Exact value:", round(exact\_value, 6), "\n")

Question 2.2 (Monte-Carlo integration)

Use the Monte Carlo approach to evaluate the standard Normal cdf (assume x ≥ 0 for simplicity):  
A math equations and symbols

Description automatically generated with medium confidence  
Evaluate also the variance and 95% confidence interval associated with your estimate.

# Set seed for reproducibility

set.seed(6040)

# Parameters

N <- 10000 # Number of random samples

x <- 1.96 # Point at which to evaluate the CDF (example: x = 1.96)

# Monte Carlo approximation of the normal CDF

samples <- rnorm(N) # Generate random samples from N(0, 1)

cdf\_estimate <- mean(samples <= x) # Proportion of samples less than or equal to x

# Variance of the estimate

variance <- cdf\_estimate \* (1 - cdf\_estimate) / N

# 95% confidence interval

stderr <- sqrt(variance)

ci\_lower <- cdf\_estimate - 1.96 \* stderr

ci\_upper <- cdf\_estimate + 1.96 \* stderr

# Output results

cat("Monte Carlo estimate of Φ(", x, ") =", round(cdf\_estimate, 6), "\n")

cat("Variance:", round(variance, 6), "\n")

cat("95% Confidence Interval: [", round(ci\_lower, 6), ",", round(ci\_upper, 6), "]\n")

Question 2.3 (Monte-Carlo estimation)

1. 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  
   A math equations with numbers

   Description automatically generated with medium confidence

# Function to compute the trimmed mean

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

n <- length(X)

X\_sorted <- sort(X) # Sort the sample

# Trim the k smallest and k largest values

trimmed\_X <- X\_sorted[(k+1):(n-k)]

# Compute the trimmed mean

return(mean(trimmed\_X))

}

# Example usage

X <- rnorm(20) # Sample from standard normal distribution

k <- 2 # Number of elements to trim from both ends

cat("Trimmed mean:", trimmed\_mean(X, k), "\n")

1. 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.

# Set seed for reproducibility

set.seed(6040)

# Monte Carlo Simulation function

monte\_carlo\_trimmed\_mean <- function(n, k, M) {

mse <- 0

trimmed\_means <- numeric(M) # Store trimmed means

for (i in 1:M) {

# Generate a sample from the standard normal distribution

X <- rnorm(n)

# Compute the trimmed mean

trimmed\_means[i] <- trimmed\_mean(X, k)

# Calculate MSE: compare trimmed mean to actual mean of normal (0 for N(0,1))

mse <- mse + (trimmed\_means[i] - 0)^2

}

# Calculate the MSE

mse <- mse / M

# Return the results

return(list(trimmed\_means = trimmed\_means, mse = mse))

}

# Run the simulation

n <- 20

k <- 2

M <- 1000

results <- monte\_carlo\_trimmed\_mean(n, k, M)

cat("MSE for normal distribution:", round(results$mse, 4), "\n")

1. Run a similar simulation using a Student t-distribution with 1 d.o.f. as samplig distribution, and compare output estimator performances.

# Monte Carlo Simulation for Student t-distribution with 1 d.o.f.

monte\_carlo\_trimmed\_mean\_tdist <- function(n, k, M) {

mse <- 0

trimmed\_means <- numeric(M)

for (i in 1:M) {

# Generate a sample from the t-distribution with 1 d.o.f.

X <- rt(n, df = 1)

# Compute the trimmed mean

trimmed\_means[i] <- trimmed\_mean(X, k)

# Calculate MSE: compare to actual mean of t(1) (which is 0)

mse <- mse + (trimmed\_means[i] - 0)^2

}

# Calculate the MSE

mse <- mse / M

# Return the results

return(list(trimmed\_means = trimmed\_means, mse = mse))

}

# Run the simulation

results\_tdist <- monte\_carlo\_trimmed\_mean\_tdist(n, k, M)

cat("MSE for Student t-distribution:", round(results\_tdist$mse, 4), "\n")

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.

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

# Initialize simulation parameters

N <- 50 # Sample size

M <- 100 # Number of Monte Carlo repetitions

a <- 7 # Intercept parameter

b <- 3 # Slope parameter

# Regressor (design) vector x

x <- rep(1:5, N / 5)

# Noise parameters for log-normal distribution

m <- 0.5 # Mean of the log-normal distribution

s <- 1.2 # Standard deviation of the log-normal distribution

# Set seed for reproducibility

rseed <- 0

set.seed(rseed)

# Allocate storage vectors for LS and RM estimation results

LSvec <- matrix(0, 2, M) # Matrix for least squares estimates (intercept and slope)

RMvec <- matrix(0, 2, M) # Matrix for robust M-estimation estimates (intercept and slope)

# Load required libraries

library(MASS) # For robust M-estimation (rlm function)

library(VGAM) # For log-normal distribution

(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

# Monte Carlo repetitions

for (i in 1:M) {

# Generate noise from log-normal distribution

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

# Generate new sample of observations

y <- a + b \* x + e

# Estimate (a, b) via Least Squares (LS)

ls\_model <- lm(y ~ x)

LSvec[, i] <- coef(ls\_model)

# Estimate (a, b) via Robust M-estimation (RM)

rm\_model <- rlm(y ~ x)

RMvec[, i] <- coef(rm\_model)

}

(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.

# Plotting window with 2x3 panels

par(mfrow = c(2, 3))

# Plot histograms for LS estimates

hist(LSvec[1, ], main = "LS Intercept Estimates", col = "lightblue", breaks = 15)

hist(LSvec[2, ], main = "LS Slope Estimates", col = "lightblue", breaks = 15)

# Plot histograms for RM estimates

hist(RMvec[1, ], main = "RM Intercept Estimates", col = "lightgreen", breaks = 15)

hist(RMvec[2, ], main = "RM Slope Estimates", col = "lightgreen", breaks = 15)

# Density plots

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

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

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

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

Bias, Variance, and MSE Calculations:

# Calculate bias, variance, and MSE

true\_intercept <- a

true\_slope <- b

# LS bias, variance, and MSE

ls\_bias\_intercept <- mean(LSvec[1, ]) - true\_intercept

ls\_var\_intercept <- var(LSvec[1, ])

ls\_mse\_intercept <- ls\_bias\_intercept^2 + ls\_var\_intercept

ls\_bias\_slope <- mean(LSvec[2, ]) - true\_slope

ls\_var\_slope <- var(LSvec[2, ])

ls\_mse\_slope <- ls\_bias\_slope^2 + ls\_var\_slope

# RM bias, variance, and MSE

rm\_bias\_intercept <- mean(RMvec[1, ]) - true\_intercept

rm\_var\_intercept <- var(RMvec[1, ])

rm\_mse\_intercept <- rm\_bias\_intercept^2 + rm\_var\_intercept

rm\_bias\_slope <- mean(RMvec[2, ]) - true\_slope

rm\_var\_slope <- var(RMvec[2, ])

rm\_mse\_slope <- rm\_bias\_slope^2 + rm\_var\_slope

cat("LS Intercept Bias:", ls\_bias\_intercept, "MSE:", ls\_mse\_intercept, "\n")

cat("LS Slope Bias:", ls\_bias\_slope, "MSE:", ls\_mse\_slope, "\n")

cat("RM Intercept Bias:", rm\_bias\_intercept, "MSE:", rm\_mse\_intercept, "\n")

cat("RM Slope Bias:", rm\_bias\_slope, "MSE:", rm\_mse\_slope, "\n")

(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

• Test again: load it up using read.csv() and recompute the biases as a check

# Create a dataframe with outputs of interest

results\_df <- data.frame(

LS\_Intercept = LSvec[1, ],

LS\_Slope = LSvec[2, ],

RM\_Intercept = RMvec[1, ],

RM\_Slope = RMvec[2, ]

)

# Write dataframe to CSV

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

# Load the file again to check

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

head(check\_df)

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

A math equations on a white background

Description automatically generated

(a) Implement a Monte Carlo experiment to generate M samples of size N from a Normal N (0, 22)

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.

Question 2.6 (Monte-Carlo estimation)

In this question, we aim to assess whether the rate of convergence of the sample mean of a χ2-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;

• A sample size of n=100 is used;

• For each value of ndf, M=100 samples x of a χ2-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 χ2-distribution?

# Set parameters

M <- 100 # Number of Monte Carlo repetitions

n <- 100 # Sample size

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

ms <- matrix(0, M, length(ndf\_values)) # Matrix to store sample means

# Monte Carlo simulation

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

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

for (i in 1:M) {

# Generate a sample from the chi-squared distribution

x <- rchisq(n, df = ndf)

# Store the mean of the sample

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

}

}

# Boxplot of the sample means for each degree of freedom

boxplot(ms,

main = "Boxplots of Sample Means for χ²-distribution",

names = ndf\_values,

xlab = "Degrees of Freedom",

ylab = "Sample Mean",

col = c("lightblue", "lightgreen", "lightpink"))

# Add a horizontal line at the theoretical mean of the χ²-distribution

abline(h = ndf\_values, col = "red", lty = 2) # Theoretical means are equal to degrees of freedom

**Comment on the Figure**

* **Observation**: The boxplots represent the distribution of sample means for χ²-distributions with degrees of freedom 2, 4, and 10.
* As the degrees of freedom increase, the spread of the sample means decreases, indicating that higher degrees of freedom lead to less variability in the sample mean.
* The mean of the χ²-distribution is equal to its degrees of freedom (E[X]=ndf). The red dashed line indicates the theoretical mean, and you should see that the sample means are centered around this line as the number of degrees of freedom increases.

Question 2.7

Note: parts of this quesiton are also covered in Section 2.

1. Implement pseudo-random generation of Huber’s contamination model



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.

2.2 Bootstrapping

Question 2.8 (Bootstrap estimation of standard error)

Load the law school dataset in the bootstrap package, estimate the correlation between the twovariables in this dataset, and evaluate the bootstrap estimate of the standard error associated with this estimation.

# Load necessary libraries

if (!require(bootstrap)) {

install.packages("bootstrap") # Install bootstrap package if not installed

}

library(bootstrap)

# Load the law school dataset

data(lawschool)

# Estimate the correlation

correlation\_estimate <- cor(lawschool$lsat, lawschool$gpa)

cat("Correlation Estimate:", correlation\_estimate, "\n")

# Set the number of bootstrap samples

B <- 1000

# Function to compute correlation

cor\_func <- function(data, indices) {

d <- data[indices, ] # Resample the data

return(cor(d$lsat, d$gpa)) # Compute the correlation

}

# Perform bootstrap

set.seed(1) # Set seed for reproducibility

boot\_results <- boot(data = lawschool, statistic = cor\_func, R = B)

# Estimate standard error

bootstrap\_se <- sd(boot\_results$t)

cat("Bootstrap Standard Error:", bootstrap\_se, "\n")

Question 2.9 (Bootstrap linear regression estimates)

Consider R’s cars dataset.

(a) Obtain relevant regression parameter estimates for this dataset.

(b) Generate M = 10000 bootstrap estimates for these coefficients.

(c) Inspect the one-dimensional (i.e. marginal) distributions for all relevant bootstrap parameter estimates, and state your conclusions.

(d) Inspect the joint distribution for these sample parameter estimates, and state your conclusions

# Load the cars dataset

data(cars)

# Fit a linear regression model

lm\_model <- lm(dist ~ speed, data = cars)

# Obtain relevant regression parameter estimates

summary(lm\_model) # Display the summary of the model

# Set number of bootstrap samples

M <- 10000

# Create a matrix to store bootstrap estimates

boot\_estimates <- matrix(0, nrow = M, ncol = 2) # 2 coefficients: intercept and slope

# Set seed for reproducibility

set.seed(1)

# Bootstrap process

for (i in 1:M) {

# Generate bootstrap sample

sample\_indices <- sample(1:nrow(cars), replace = TRUE)

boot\_sample <- cars[sample\_indices, ]

# Fit model on bootstrap sample

boot\_model <- lm(dist ~ speed, data = boot\_sample)

# Store coefficients

boot\_estimates[i, ] <- coef(boot\_model)

}

# Load required library for visualization

if (!require(ggplot2)) {

install.packages("ggplot2") # Install ggplot2 if not installed

}

library(ggplot2)

# Convert boot\_estimates to a data frame

boot\_df <- as.data.frame(boot\_estimates)

colnames(boot\_df) <- c("Intercept", "Slope")

# Plot marginal distributions

par(mfrow = c(1, 2)) # Set up plotting area for 2 plots

# Intercept

hist(boot\_df$Intercept, main = "Bootstrap Distribution of Intercept",

xlab = "Intercept", breaks = 30, col = "lightblue")

# Slope

hist(boot\_df$Slope, main = "Bootstrap Distribution of Slope",

xlab = "Slope", breaks = 30, col = "lightgreen")

# Joint distribution plot

plot(boot\_df$Intercept, boot\_df$Slope,

main = "Joint Distribution of Bootstrap Estimates",

xlab = "Intercept", ylab = "Slope",

col = rgb(0.2, 0.6, 0.8, 0.5), pch = 19)

# Add a density contour

contour(density(boot\_df$Intercept, boot\_df$Slope), add = TRUE, lwd = 2)

Question 2.10 (Nonlinear estimation and bootstrapping)

Load the following data from R’s mtcars dataset:

x = mtcars$disp

y = mtcars$mpg

(a) Task: fit an exponential model

Yi = exp(θ1 + θ2Xi) + εi, i = 1, . . . , n, εi i.i.d.

to the sample y using R’s function nls(). Use initial values θ(0) = (3, −0.01).

Required:

(i) quote the coefficient estimates for this model fit;

(ii) explain, based on numerical assessment but without performing any further computa-

tions, whether this model appropriately describes the relationship between x and y;

(iii) provide a plot showing the model fit as a line going through the data points. Plot the

data points in black and the model fit in red.

(b) Name an alternative regression technique that would provide a nonlinear representation of

the relationship between x and y, without assuming a particular model.

(c) Task: set the pseudo-random seed to 1 (R instruction set.seed(1)) and compute B = 100

bootstrap estimates for the model fit of (a).

Required:

(i) quote the bootstrap means and standard deviations for estimators ˆθ1 and ˆθ2;  
(ii) quote the bootstrap estimate of the standard error for estimator ˆθ1;

(iii) quote a nonparametric 95% confidence intervals for model parameter θ1;

(iv) comment on the confidence interval found in (c.iii).

Round off your numerical answers to 4 decimal places where applicable.  
  
# Load necessary libraries

library(ggplot2)

# Load the mtcars dataset

data(mtcars)

# Define the variables

x <- mtcars$disp # Predictor

y <- mtcars$mpg # Response

# Fit the exponential model with standard variable names

model <- nls(y ~ exp(beta1 + beta2 \* x), start = list(beta1 = 3, beta2 = -0.01))

# Get the coefficient estimates

coef\_estimates <- coef(model)

cat("Coefficient Estimates:\n")

Coefficient Estimates:

cat("beta1:", round(coef\_estimates[1], 4), "\n")

beta1: 3.4988

cat("beta2:", round(coef\_estimates[2], 4), "\n")

beta2: -0.0023

# Predict values from the model

predicted\_y <- predict(model)

# Assessing the fit visually

plot\_data <- data.frame(x, y, predicted\_y)

# Plot the data points and fitted model

ggplot(plot\_data, aes(x = x, y = y)) +

geom\_point(color = "black") + # Data points

geom\_line(aes(y = predicted\_y), color = "red") + # Model fit

labs(title = "Exponential Model Fit to mtcars Data",

x = "Displacement (disp)",

+ y = "Miles Per Gallon (mpg)") +

+ theme\_minimal()

# Part (c) - Bootstrapping

set.seed(1) # Set seed for reproducibility

B <- 100 # Number of bootstrap samples

bootstrap\_results <- matrix(NA, nrow = B, ncol = 2)

# Bootstrap loop

for (b in 1:B) {

+ sample\_indices <- sample(1:nrow(mtcars), replace = TRUE)

+ x\_boot <- mtcars$disp[sample\_indices]

+ y\_boot <- mtcars$mpg[sample\_indices]

+

+ model\_boot <- nls(y\_boot ~ exp(beta1 + beta2 \* x\_boot), start = list(beta1 = 3, beta2 = -0.01))

+ bootstrap\_results[b, ] <- coef(model\_boot)

+ }

# Calculate means and standard deviations of bootstrap estimates

bootstrap\_means <- colMeans(bootstrap\_results)

bootstrap\_sds <- apply(bootstrap\_results, 2, sd)

# Output bootstrap results

cat("\nBootstrap Means:\n")

Bootstrap Means:

cat("ˆbeta1:", round(bootstrap\_means[1], 4), "\n")

ˆbeta1: 3.4945

cat("ˆbeta2:", round(bootstrap\_means[2], 4), "\n")

ˆbeta2: -0.0023

cat("\nBootstrap Standard Deviations:\n")

Bootstrap Standard Deviations:

cat("ˆbeta1 SD:", round(bootstrap\_sds[1], 4), "\n")

ˆbeta1 SD: 0.0592

cat("ˆbeta2 SD:", round(bootstrap\_sds[2], 4), "\n")

ˆbeta2 SD: 3e-04

# Standard error for estimator beta1

se\_beta1 <- bootstrap\_sds[1]

cat("\nBootstrap estimate of the standard error for estimator ˆbeta1:", round(se\_beta1, 4), "\n")

Bootstrap estimate of the standard error for estimator ˆbeta1: 0.0592

# Nonparametric 95% confidence intervals for beta1

ci\_beta1 <- quantile(bootstrap\_results[, 1], probs = c(0.025, 0.975))

cat("\n95% Confidence Interval for beta1:\n")

95% Confidence Interval for beta1:

cat("Lower:", round(ci\_beta1[1], 4), "\n")

Lower: 3.376

cat("Upper:", round(ci\_beta1[2], 4), "\n")

Upper: 3.5945

# Comment on the confidence interval

cat("\nComment on Confidence Interval:\n")

Comment on Confidence Interval:

if (ci\_beta1[1] < coef\_estimates[1] && ci\_beta1[2]> coef\_estimates[1]) {

+ cat("The confidence interval includes the original estimate of beta1, indicating that it is a plausible value based on the bootstrap samples.\n")

+ } else {

+ cat("The confidence interval does not include the original estimate of beta1, suggesting that the original estimate may not be well supported by the data.\n")

+ }

The confidence interval includes the original estimate of beta1, indicating that it is a plausible value based on the bootstrap samples.