hw4

陆靖磊 22300680221

2024-11-06

7.1

```r
sample_size <- 20
trim_levels <- 9
simulations <- 1000
mse_matrix <- matrix(0, nrow = sample_size / 2, ncol = 2)

calculate_trimmed_mse <- function(sample_size, simulations, trim_level) {
    trimmed_means <- numeric(simulations)
    for (iteration in seq_len(simulations)) {
        sorted_sample <- sort(rcauchy(sample_size))
        trimmed_mean_value <- mean(sorted_sample[(trim_level + 1):(sample_size - trim_level)])
        trimmed_means[iteration] <- trimmed_mean_value
    }
    mse_estimate <- mean(trimmed_means^2)
    mse_se <- sd(trimmed_means) / sqrt(simulations)
    return(c(mse_estimate, mse_se))
}

for (trim_level in 0:trim_levels) {
    mse_matrix[trim_level + 1, ] <- calculate_trimmed_mse(sample_size, simulations, trim_level)
}

mse_results <- as.data.frame(cbind(0:trim_levels, mse_matrix))
colnames(mse_results) <- c("Trim Level", "MSE Estimate", "Standard Error")
print(mse_results)
```

```
##    Trim Level MSE Estimate Standard Error
## 1           0  284.2535046     0.53331676
## 2           1    1.4527565     0.03810813
## 3           2    0.5116557     0.02262634
## 4           3    0.2249506     0.01499203
## 5           4    0.1709815     0.01308019
## 6           5    0.1802160     0.01343089
## 7           6    0.1391306     0.01179960
## 8           7    0.1385808     0.01176373
## 9           8    0.1292349     0.01136708
## 10          9    0.1492058     0.01221829
```

7.4

```r
simulate_ci <- function(n) {
  x <- rlnorm(n)
  y <- log(x)
  ybar <- mean(y)
  se <- sd(y) / sqrt(n)
  ci <- ybar + se * qnorm(c(0.025, 0.975))
  return(ci)
}

n <- 30
CIs <- replicate(10000, simulate_ci(n))
LCLs <- CIs[1, ]
UCLs <- CIs[2, ]
empirical_confidence_level <- (sum(LCLs < 0 & UCLs > 0)) / length(LCLs)
cat("Empirical confidence level:", empirical_confidence_level, "\n")
```

```
## Empirical confidence level: 0.9369
```

7.7

```r
# 偏度函数
sk <- function(x) {
  xbar <- mean(x)
  m3 <- mean((x - xbar)^3)
  m2 <- mean((x - xbar)^2)
  return(m3 / m2^(1.5))
}

m <- 10000
n <- 1000
p <- c(0.025, 0.05, 0.95, 0.975)

skstats <- replicate(m, expr = {
  x <- rnorm(n)
  sk(x)
})
# 计算 MC 方法的分位数
q1 <- quantile(skstats, p)

# 计算标准误
density_at_quantiles <- sapply(q1, function(q) dnorm(q, 0, sqrt(6 / n)))
v <- p * (1 - p) / (m * density_at_quantiles^2)
se <- sqrt(v)

# 计算两种近似方法的分位数
q2 <- qnorm(p, 0, sqrt(6 / n))
q3 <- qnorm(p, 0, sqrt(6 * (n - 2) / ((n + 1) * (n + 3))))

result <- data.frame(
  Sample = q1,
  StandardError = se,
  NormalExactVariance = q2,
  AsymptoticNormal = q3
```

```
)

print(result)
```

```
##          Sample StandardError NormalExactVariance AsymptoticNormal
## 2.5%  -0.1545809   0.002220424          -0.1518182       -0.1513636
## 5%    -0.1293687   0.001706938          -0.1274098       -0.1270283
## 95%    0.1249189   0.001553340           0.1274098        0.1270283
## 97.5%  0.1526236   0.002111906           0.1518182        0.1513636
```

7.10

```
Gini <- function(x) {
  n <- length(x)
  k <- 2 * (1:n) - n - 1
  x <- sort(x)
  g <- sum(k * x) / (n^2 * mean(x))
  return(g)
}

n <- 200
m <- 10000

generate_and_calculate_gini <- function(dist_func, dist_params) {
  replicate(m, Gini(do.call(dist_func, c(list(n), dist_params))))
}

g1 <- generate_and_calculate_gini(rlnorm, list())
g2 <- generate_and_calculate_gini(runif, list(min = 0, max = 1))
g3 <- generate_and_calculate_gini(rbinom, list(size = 1, prob = 0.1))

g <- data.frame(g1, g2, g3)
summary(g)
```

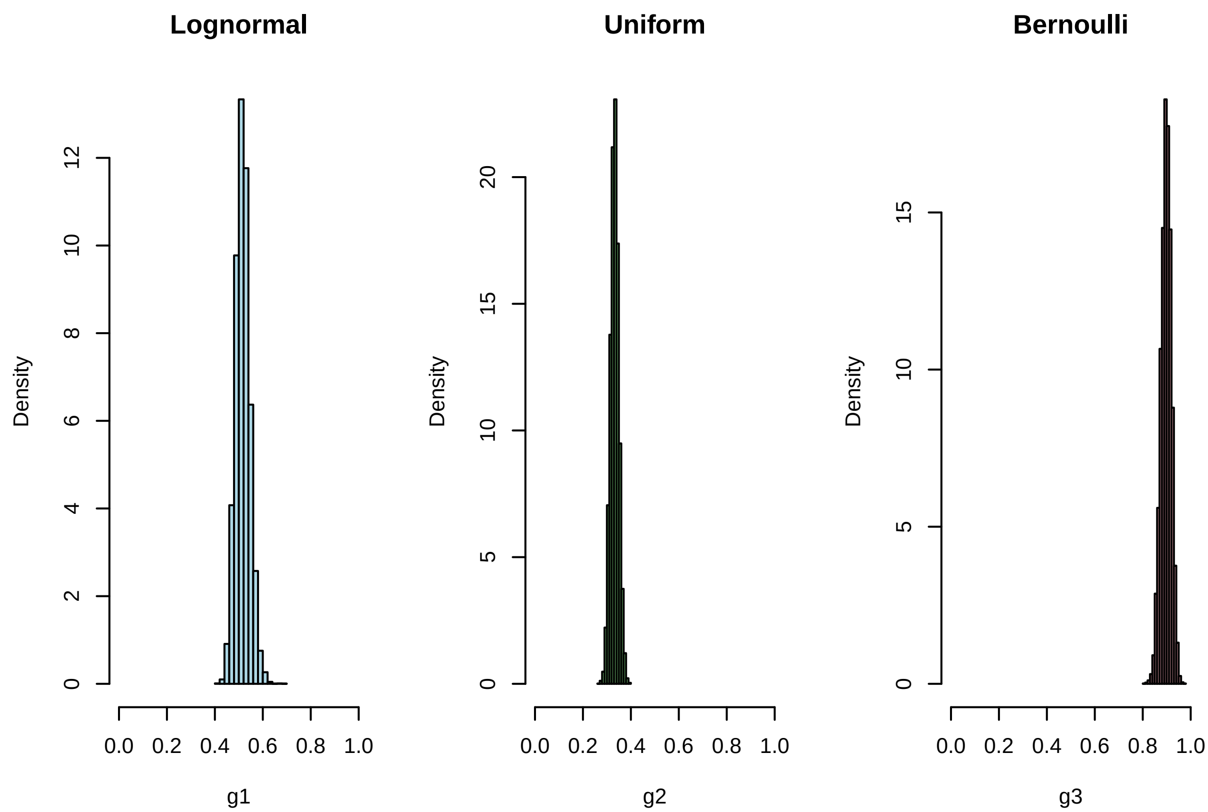
```
##        g1               g2               g3       
##  Min.   :0.4139   Min.   :0.2690   Min.   :0.805  
##  1st Qu.:0.4964   1st Qu.:0.3208   1st Qu.:0.885  
##  Median :0.5151   Median :0.3321   Median :0.900  
##  Mean   :0.5163   Mean   :0.3322   Mean   :0.900  
##  3rd Qu.:0.5350   3rd Qu.:0.3436   3rd Qu.:0.915  
##  Max.   :0.6916   Max.   :0.3988   Max.   :0.980
```

```
percentiles <- apply(g, MARGIN = 2, quantile, probs = seq(0.1, 0.9, by = 0.1))
print(percentiles)
```

```
##            g1        g2    g3
## 10% 0.4795709 0.3101003 0.875
## 20% 0.4921156 0.3180079 0.880
## 30% 0.5001681 0.3232204 0.890
## 40% 0.5078013 0.3278410 0.895
## 50% 0.5150546 0.3321024 0.900
```

```
## 60% 0.5224887 0.3364289 0.905
## 70% 0.5303592 0.3411227 0.910
## 80% 0.5400490 0.3465066 0.920
## 90% 0.5542725 0.3542982 0.925
```

```r
par(mfrow = c(1, 3))
hist(g1, prob = TRUE, main = "Lognormal", xlim = c(0, 1), col = "lightblue")
hist(g2, prob = TRUE, main = "Uniform", xlim = c(0, 1), col = "lightgreen")
hist(g3, prob = TRUE, main = "Bernoulli", xlim = c(0, 1), col = "lightpink")
```



P7.(A)

```r
alpha <- 0.05
n_simulations <- 10000
sample_size <- 100
type_I_error_count <- rep(0, 3)
distributions <- list(
  chi_squared = list(distribution = rchisq, df = 1),
  uniform = list(distribution = runif, min = 0, max = 2),
  exponential = list(distribution = rexp, rate = 1)
)

run_simulation <- function(distribution_info, sample_size, mu_0, alpha) {
  sample <- do.call(distribution_info$distribution, c(list(n = sample_size), distribution_info[-1]))
  test_result <- t.test(sample, mu = mu_0, alternative = "two.sided")
  if (test_result$p.value < alpha) {
```

```
    return(1)
  } else {
    return(0)
  }
}

for (i in 1:length(distributions)) {
  distribution_info <- distributions[[i]]

  if ("df" %in% names(distribution_info) && distribution_info$df == 1) {
    mu_0 <- 1

  } else if ("min" %in% names(distribution_info) && "max" %in% names(distribution_info) &&
             distribution_info$min == 0 && distribution_info$max == 2) {
    mu_0 <- 1

  } else if ("rate" %in% names(distribution_info) && distribution_info$rate == 1) {
    mu_0 <- 1
  }

  for (j in 1:n_simulations) {
    type_I_error_count[i] <- type_I_error_count[i] + run_simulation(distribution_info, sample_size, mu_(
  }

  empirical_type_I_error_rate <- type_I_error_count[i] / n_simulations
  cat("Empirical Type I error rate for", names(distributions)[i], ":", empirical_type_I_error_rate, "\n
}
```

```
## Empirical Type I error rate for chi_squared : 0.0656
## Empirical Type I error rate for uniform : 0.0508
## Empirical Type I error rate for exponential : 0.0577
```

P7.(C)

```
# 加载必要的包
library(mvtnorm)
library(MASS)

# 设置随机种子
set.seed(123)

# 生成样本
X <- matrix(rnorm(50*2), 50)

# 计算样本均值向量和协方差矩阵
mu <- colMeans(X)
Sigma <- cov(X)

# 计算偏度和峰度
n <- nrow(X)
p <- ncol(X)

# 计算偏度
```

```r
skewness <- sum(apply(X, 1, function(x) {
  (t(x - mu) %*% solve(Sigma) %*% (x - mu))^3
})) / n

# 调整偏度的计算以避免偏度的极端值
skewness <- skewness / (n * p)

# 计算峰度
kurtosis <- sum(apply(X, 1, function(x) {
  (t(x - mu) %*% solve(Sigma) %*% (x - mu))^2
})) / n

# 计算偏度和峰度的检验统计量
skewness_stat <- n * skewness / 6
kurtosis_stat <- (kurtosis - p * (p + 2)) / sqrt(8 * p * (p + 2) / n)

# 计算 p 值
skewness_p_value <- 1 - pchisq(skewness_stat, df = p * (p + 1) * (p + 2) / 6)
kurtosis_p_value <- 2 * (1 - pnorm(abs(kurtosis_stat)))

# 判断是否符合正态性
skewness_result <- ifelse(skewness_p_value > 0.05, "YES", "NO")
kurtosis_result <- ifelse(kurtosis_p_value > 0.05, "YES", "NO")
mv_normality_result <- ifelse(skewness_result == "YES" && kurtosis_result == "YES", "YES", "NO")

# 格式化输出
mv_test <- data.frame(
  Test = c("Skewness", "Kurtosis", "MV Normality"),
  Statistic = c(skewness_stat, kurtosis_stat, NA),
  p.value = c(skewness_p_value, kurtosis_p_value, NA),
  Result = c(skewness_result, kurtosis_result, mv_normality_result)
)

# 计算单变量 Shapiro-Wilk 检验
uv_shapiro <- apply(X, 2, function(x) {
  shapiro_test <- shapiro.test(x)
  data.frame(
    W = shapiro_test$statistic,
    p.value = shapiro_test$p.value,
    UV.Normality = ifelse(shapiro_test$p.value > 0.05, "Yes", "No")
  )
})

uv_shapiro <- do.call(rbind, uv_shapiro)
uv_shapiro$Variable <- paste0("V", 1:ncol(X))

# 重命名 W 为 W1 和 W2
uv_shapiro$W <- paste0("W", 1:ncol(X))

# 输出结果
cat("## $mv.test\n")
```

```
## ## $mv.test
```

```r
print(mv_test, row.names = FALSE)
```

```
##          Test Statistic   p.value Result
##      Skewness  2.587696 0.6290043    YES
##      Kurtosis -1.106370 0.2685664    YES
##  MV Normality        NA        NA    YES
```

```r
cat("\n## $uv.shapiro\n")
```

```
##
## ## $uv.shapiro
```

```r
print(uv_shapiro, row.names = FALSE)
```

```
##   W   p.value UV.Normality Variable
##  W1 0.9278568          Yes       V1
##  W2 0.9617732          Yes       V2
```