R Labs

▼ Lab 1

▼ Approximation with Central Limit Theorem

```
#################################
# R code for LAB1 - 08/10/2024  #
# This is the final version     #
#################################

## **Approximation with Central Limit Theorem (CLT): Example**

# Scenario:
# We're analyzing a water polo match where team X and team Y have probabilities p and q
# of scoring in each trial, respectively. We want to calculate the probability that team X wins.

# Set the given probabilities and number of trials for each team
p <- 0.5  # Probability that team X scores in each trial
q <- 0.7  # Probability that team Y scores in each trial
n <- m <- 20  # Number of trials (attempts) for both teams

# Compute the expected value (mean), variance, and standard deviation of the difference W = X - Y
mW <-  p * n - q * m  # Mean of W
varW <- n * p * (1 - p) + m * q * (1 - q)  # Variance of W (since X and Y are independent)
sdW <- sqrt(varW)  # Standard deviation of W

# Compute the probability that team X wins (i.e., W > 0) using the normal approximation
PWin_P <- pnorm(0, mean = mW, sd = sdW, lower.tail = FALSE)
PWin_P  # Display the probability that W > 0

# Visualization:

# Plot the normal approximation of W
curve(dnorm(x, mW, sdW), xlim = c(-12, 20), ylim = c(0, 0.3),
      xlab = "", ylab = "", cex.lab = 1.25)

# Plot the pmf of X (number of goals by team X)
points(0:n, dbinom(0:n, n, p), pch = 21, bg = 1, col = "blue")

# Plot the pmf of Y (number of goals by team Y)
points(0:m, dbinom(0:m, m, q), pch = 21, bg = 2)

# Draw a line at W = 0 to visualize the threshold for team X winning
segments(x0 = 0, y0 = -0.01, x1 = 0, y1 = dnorm(0, mW, sdW), lwd = 2)

# Add labels to the plot
text(14.25, 0.22, "Y", cex = 2, col = 2)         # Label for team Y's distribution
text(10, 0.2, "X", cex = 2, col = "blue")        # Label for team X's distribution
text(-4, 0.15, "X - Y", cex = 2, col = 1)        # Label for the difference distribution

# Compute the probability using continuity correction
PWin_P_cc <- pnorm(0.5, mean = mW, sd = sdW, lower.tail = FALSE)

# Compare probabilities with and without continuity correction
PWin_P_cc  # With continuity correction
PWin_P     # Without continuity correction

# Compute the exact probability using convolution (exact method)

# Probability mass functions for X and Y
Px <- dbinom(0:n, n, p)  # pmf of X
Py <- dbinom(0:m, m, q)  # pmf of Y

# Initialize the pmf of W (difference of two binomial variables)
Pw <- rep(NA, 2 * n + 1)  # W can take values from -n to n

# Counters for filling in Pw
count <- 1          # Counter for negative and zero differences
count2 <- 2 * n + 1 # Counter for positive differences
```

```
# Compute the pmf of W using convolution
for(i in 0:n){
  idx1 <- 1:(i + 1)
  idx2 <- (n + 1 - i):(n + 1)
  if(i == n){
    Pw[count] <- sum(Px[idx1] * Py[idx2])
  } else {
    Pw[count] <- sum(Px[idx1] * Py[idx2])
    Pw[count2] <- sum(Px[idx2] * Py[idx1])
  }
  count <- count + 1
  count2 <- count2 - 1
}

sum(Pw)  # Verify that the total probability sums to 1

# Compute the exact probability that team X wins (W > 0)
exact_PWin <- sum(Pw[(n + 2):length(Pw)])  # Sum probabilities where W > 0
exact_PWin

# Compare exact probability with approximations
PWin_P_cc  # Approximation with continuity correction
PWin_P     # Approximation without continuity correction

# Plot the exact pmf of W and the normal approximation
plot(-n:m, Pw, xlab = "w", ylab = "", col = "red", cex.axis = 1.5, cex.lab = 2)
curve(dnorm(x, mW, sdW), add = TRUE)  # Overlay the normal approximation

# Function to compute the pmf of W for varying n and m
PMFw <- function(n, m, p, q){
  Px <- dbinom(0:n, n, p = p)  # pmf of X
  Py <- dbinom(0:m, m, p = q)  # pmf of Y
  Pw <- rep(NA, 2 * n + 1)     # Initialize pmf of W

  count <- 1; count2 <- 2 * n + 1
  for(i in 0:n){
    idx1 <- 1:(i + 1)
    idx2 <- (n + 1 - i):(n + 1)
    if(i == n){
      Pw[count] <- sum(Px[idx1] * Py[idx2])
    } else {
      Pw[count] <- sum(Px[idx1] * Py[idx2])
      Pw[count2] <- sum(Px[idx2] * Py[idx1])
    }
    count <- count + 1
    count2 <- count2 - 1
  }
  return(Pw)
}

# Adjusting probabilities and trials for different scenarios
p <- 0.5
q <- 0.7

# Case with smaller number of trials (n = m = 5)
n <- m <- 5

# Recompute mean, variance, and standard deviation for the new n and m
mW <-  p * n - q * m
varW <- n * p * (1 - p) + m * q * (1 - q)
sdW <- sqrt(varW)

# Compute approximate probabilities
PWin_P <- pnorm(0, mean = mW, sd = sdW,  lower.tail = FALSE)
PWin_P_cc <- pnorm(0.5, mean = mW, sd = sdW,  lower.tail = FALSE)

# Compute exact probability using the PMFw function
Pw <- PMFw(n, m, p, q)
exact_PWin_small_n <- sum(Pw[(n + 2):length(Pw)])  # Exact P(W > 0)
```

```r
# Display the exact and approximate probabilities
exact_PWin_small_n  # Exact probability
PWin_P              # Approximation without continuity correction
PWin_P_cc           # Approximation with continuity correction

# Plot the pmf of W and normal approximation for smaller n
plot(-n:m, Pw, xlab = "w", col = "red", cex.axis = 1.5, cex = 2)
curve(dnorm(x, mW, sdW), add = TRUE)

# Case with larger number of trials (n = m = 100)
n <- m <- 100

# Recompute mean, variance, and standard deviation for the new n and m
mW <-  p * n - q * m
varW <- n * p * (1 - p) + m * q * (1 - q)
sdW <- sqrt(varW)

# Compute approximate probabilities
PWin_P <- pnorm(0, mean = mW, sd = sdW,  lower.tail = FALSE)
PWin_P_cc <- pnorm(0.5, mean = mW, sd = sdW,  lower.tail = FALSE)

# Compute exact probability using the PMFw function
Pw <- PMFw(n, m, p, q)
exact_PWin_large_n <- sum(Pw[(n + 2):length(Pw)])  # Exact P(W > 0)

# Display the exact and approximate probabilities
exact_PWin_large_n  # Exact probability
PWin_P              # Approximation without continuity correction
PWin_P_cc           # Approximation with continuity correction

# Plot the pmf of W and normal approximation for larger n
plot(-n:m, Pw, xlab = "w", col = "red", cex.axis = 1.5, cex = 2)
curve(dnorm(x, mW, sdW), add = TRUE)
```

Bivariate Normal Distribution

```r
## **Bivariate Normal Distribution**

# Function to compute the density of a bivariate normal distribution with zero means
# and unit variances, given correlation coefficient rho
NBiv <- function(x, y, rho){
  den <- (2 * pi * sqrt(1 - rho^2))
  num <- exp(-0.5 * (1 - rho^2)^(-1) * (x^2 + y^2 - 2 * rho * x * y))
  return(num / den)
}

# Generate grid points for x and y
xx <- yy <- seq(-3, 3, 0.1)

# Compute the density values over the grid for rho = 0.5
z <- outer(xx, yy, NBiv, rho = 0.5)

# Plotting the 3D surface of the bivariate normal density
par(mfrow = c(1, 2))
persp(xx, yy, z, xlab = "x", ylab = "y", zlab = "f(x,y)", cex.lab = 3)
persp(xx, yy, z, theta = 30, phi = 30, xlab ="x", ylab = "y", zlab = "f(x,y)", cex.lab = 3)

# Contour plots for different values of rho
rho_values <- c(-0.5, 0, 0.9)
z_list <- list()
for(j in 1:3) {
  z_list[[j]] <- outer(xx, yy, NBiv, rho = rho_values[j])
}

par(mfrow = c(1, 3))
for(j in 1:3) {
  contour(xx, yy, z_list[[j]], main = paste0("rho = ", rho_values[j]),
          cex.main = 3, cex.axis = 3, labcex = 2)
}
```

```
# Load the mvtnorm package for multivariate normal functions
# install.packages("mvtnorm")  # Uncomment if mvtnorm is not installed
library(mvtnorm)

# Biometric example: Modeling height and weight with bivariate normal distribution
# Given mean heights and weights, standard deviations, and correlation
mu_h <- 176         # Mean height in cm
mu_w <- 85.52       # Mean weight in kg
sd_h <- 7           # Standard deviation of height
sd_w <- 14.24       # Standard deviation of weight
rho <- 0.47         # Correlation between height and weight

# Construct the covariance matrix
cov_hw <- rho * sd_h * sd_w
Sigma <- matrix(c(sd_h^2, cov_hw, cov_hw, sd_w^2), 2, 2, byrow = TRUE)

# Generate samples from the bivariate normal distribution
set.seed(13)
X <- rmvnorm(1000, c(mu_h, mu_w), Sigma)

# Plot the marginal distributions of height and weight
par(mfrow = c(1, 2))
hist(X[, 1], prob = TRUE, breaks = 30, cex.lab = 2, main = "Height Distribution")
curve(dnorm(x, mu_h, sd_h), col = "red", add = TRUE)
hist(X[, 2], prob = TRUE, breaks = 30, main = "Weight Distribution")
curve(dnorm(x, mu_w, sd_w), col = "red", add = TRUE)

# Compute the conditional distribution of weight given height = 180 cm
x1_fix <- 180
mu_w_given_h <- mu_w + rho * (sd_w / sd_h) * (x1_fix - mu_h)
sd_w_given_h <- sqrt(sd_w^2 * (1 - rho^2))
c(sd_w, sd_w_given_h)  # Compare original and conditional standard deviations

# Compute the conditional distribution of height given weight = 50 kg
x2_fix <- 50
mu_h_given_w <- mu_h + rho * (sd_h / sd_w) * (x2_fix - mu_w)
sd_h_given_w <- sqrt(sd_h^2 * (1 - rho^2))
c(sd_h, sd_h_given_w)

# Plot the conditional distributions
par(mfrow = c(1, 2))
curve(dnorm(x, mu_w_given_h, sd_w_given_h), from = 50, to = 130,
      cex.lab = 2, cex.axis = 2, xlab = "Weight (kg)", ylab = "", main = "Weight | Height=180cm")
curve(dnorm(x, mu_h_given_w, sd_h_given_w), from = 140, to = 200,
      cex.lab = 2, cex.axis = 2, xlab = "Height (cm)", ylab = "", main = "Height | Weight=50kg")

# Scatterplot with contour lines of the bivariate normal density
xx <- seq(min(X[, 1]), max(X[, 1]), length.out = 500)
yy <- seq(min(X[, 2]), max(X[, 2]), length.out = 500)
zz <- outer(xx, yy, function(x, y) dmvnorm(cbind(x, y), mean = c(mu_h, mu_w), sigma = Sigma))

plot(X[, 1], X[, 2], xlab = "Height (cm)", ylab = "Weight (kg)", cex.axis = 2, cex.lab = 2, main = "Height v
contour(xx, yy, zz, add = TRUE, col = "red", nlevels = 20)
```

▼ **Inverse Sampling Method**

```
## **Inverse Sampling Method**

# Generating random numbers from the exponential distribution using inverse transform sampling

set.seed(13)  # Set seed for reproducibility
R <- 1000     # Number of random samples to generate
theta <- 2    # Rate parameter of the exponential distribution

# Generate R uniform random variables and apply the inverse CDF of the exponential distribution
x <- -log(runif(R)) / theta

# Plot the histogram of the generated data
hist(x, prob = TRUE, breaks = 30, cex.main = 2, cex.axis = 2, ylim = c(0, 2),
     main = "Histogram of Generated Exponential Data", xlab = "x")

# Overlay the theoretical exponential density
curve(dexp(x, rate = theta), from = 0, add = TRUE, lwd = 2, col = "red")
```

▼ Lab 2

▼ Distribution of the Sample Mean

```
#########################################
# R code - LAB 2 - Statistical Methods  #
# This is the final version             #
#########################################

## **Distribution of the Sample Mean**

# Objective:
# To investigate the distribution of the sample mean for different underlying distributions.

# Consider three cases:
# Case 1: Normal distribution N(0,1)
# Case 2: t-distribution with 3 degrees of freedom
# Case 3: Uniform distribution on (0,1)

set.seed(1234)  # Set a seed for reproducibility
R <- 1000       # Number of simulations (replications)
n <- 30         # Sample size

# Initialize an array to store the samples
# Dimensions: [distribution, sample_size, replication]
samples <- array(0, c(3, n, R))

# Generate samples for each distribution
for(i in 1:R){
  samples[1, , i] <- rnorm(n, mean = 0, sd = 1)  # Case 1
  samples[2, , i] <- rt(n, df = 3)               # Case 2
  samples[3, , i] <- runif(n, min = 0, max = 1)  # Case 3
}

# Compute the sample mean for each replication and distribution
sample_stat <- apply(samples, c(1, 3), mean)  # Dimensions: [distribution, replication]

# Alternative method using a loop (equivalent result)
sample_stat2 <- matrix(0, nrow = 3, ncol = R)
for(i in 1:R) {
  sample_stat2[, i] <- apply(samples[, , i], 1, mean)
}
max(abs(sample_stat - sample_stat2))  # Check that the methods yield the same results

# Visualize the distributions of the sample means using histograms
par(mfrow = c(1, 3))

# Histogram for Case 1: N(0,1)
hist(sample_stat[1, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = "N(0,1)", cex.main = 1.5)

# Histogram for Case 2: t(3)
hist(sample_stat[2, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = expression(t[3]), cex.main = 1.5)

# Histogram for Case 3: U(0,1)
hist(sample_stat[3, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = "U(0,1)", cex.main = 1.5)

# Overlay the theoretical normal distributions based on the Central Limit Theorem (CLT)
par(mfrow = c(1, 3))

# For N(0,1), the sample mean follows N(0, 1/n)
hist(sample_stat[1, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = "N(0,1)", cex.main = 1.5)
curve(dnorm(x, mean = 0, sd = sqrt(1 / n)), add = TRUE, col = "red", lwd = 2)
```

```r
# For t(3), the sample mean approximates N(0, 3/n) due to the finite variance
hist(sample_stat[2, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = expression(t[3]), cex.main = 1.5)
curve(dnorm(x, mean = 0, sd = sqrt(3 / n)), add = TRUE, col = "red", lwd = 2)

# For U(0,1), the sample mean approximates N(0.5, (1/12)/n)
hist(sample_stat[3, ], nclass = 30, probability = TRUE,
     xlab = "Sample Mean", main = "U(0,1)", cex.main = 1.5)
curve(dnorm(x, mean = 0.5, sd = sqrt((1 / 12) / n)), add = TRUE, col = "red", lwd = 2)

# Other graphical tools for assessment: ECDF vs CDF
par(mfrow = c(1, 3))

# Empirical CDF vs Theoretical CDF for N(0,1)
plot(ecdf(sample_stat[1, ]), xlab = "Sample Mean", main = "N(0,1)", cex.main = 1.5)
curve(pnorm(x, mean = 0, sd = sqrt(1 / n)), add = TRUE, col = "red", lty = 2)

# For t(3)
plot(ecdf(sample_stat[2, ]), xlab = "Sample Mean", main = expression(t[3]), cex.main = 1.5)
curve(pnorm(x, mean = 0, sd = sqrt(3 / n)), add = TRUE, col = "red", lty = 2)

# For U(0,1)
plot(ecdf(sample_stat[3, ]), xlab = "Sample Mean", main = "U(0,1)", cex.main = 1.5)
curve(pnorm(x, mean = 0.5, sd = sqrt((1 / 12) / n)), add = TRUE, col = "red", lty = 2)

# Quantile-Quantile Plots (QQ-Plots) to assess normality of sample means
par(mfrow = c(1, 3))

# QQ-Plot for Case 1: N(0,1)
qqnorm(sample_stat[1, ], main = "QQ-Plot N(0,1)")
abline(0, sqrt(1 / n), col = "red")  # Reference line

# QQ-Plot for Case 2: t(3)
qqnorm(sample_stat[2, ], main = expression("QQ-Plot t"[3]))
abline(0, sqrt(3 / n), col = "red")

# QQ-Plot for Case 3: U(0,1)
qqnorm(sample_stat[3, ], main = "QQ-Plot U(0,1)")
abline(0.5, sqrt((1 / 12) / n), col = "red")

## **Distribution of the Sample Variance under the Gaussian Case**

par(mfrow = c(1, 2))
sigma <- 1  # True standard deviation

# Extract sample variances for the normal distribution samples
sample_var <- apply(samples, c(1, 3), var)[1, ]

# Histogram of sample variances
hist(sample_var, nclass = 30, probability = TRUE,
     xlab = expression(s^2), main = "Sample Variance Distribution (N(0,1))", cex.main = 1.5)

# Overlay the theoretical chi-square distribution scaled by (n - 1)/sigma^2
curve((n - 1) / sigma^2 * dchisq((n - 1) * x / sigma^2, df = n - 1), add = TRUE, col = "red", lwd = 2)

# Empirical CDF vs Theoretical CDF of the sample variance
plot(ecdf(sample_var), xlab = expression(s^2), main = "ECDF of Sample Variance (N(0,1))", cex.main = 1.5)
curve(pchisq((n - 1) * x / sigma^2, df = n - 1), add = TRUE, col = "red", lwd = 2)
```

▼ Lab 3

▼ Point Estimation

```r
############################################
# Statistical Methods: Lab 3 - 22/10/2024  #
# This is the final version                #
############################################

# In this lab, we'll compare different estimators of the mean parameter and assess their properties.
```

```r
## **Point Estimation: Comparison of Four Estimators of the Mean Parameter**

# Initial settings
R <- 1000    # Number of replications
n <- 10      # Sample size
mu <- 2      # Population mean
sigma <- 2   # Population standard deviation

# Matrix to store estimates from four different estimators
# Columns correspond to different estimators
est <- matrix(0, nrow = R, ncol = 4)
label_est <- c("Mean", "Median", "(Min + Max)/2", "10% Trimmed Mean")

set.seed(1234)  # Set seed for reproducibility

for (i in 1:R) {
  x <- rnorm(n, mu, sigma)            # Generate sample from N(mu, sigma^2)
  est[i, 1] <- mean(x)                # Sample mean
  est[i, 2] <- median(x)              # Sample median
  est[i, 3] <- (min(x) + max(x)) / 2  # Mid-range
  est[i, 4] <- mean(x, trim = 0.1)    # 10% trimmed mean
}

# Visual comparison using boxplots
par(mfrow = c(1, 1), xaxt = "n")
boxplot(est, main = "Comparison of Four Estimators")
par(xaxt = "s")
axis(1, at = 1:4, labels = label_est)
abline(h = mu, lwd = 2, col = "blue")  # True mean

# Compute Bias of each estimator
bias <- apply(est, 2, mean) - mu
bias

# Compute Variance of each estimator
variance <- apply(est, 2, var)
variance

# Compute Mean Squared Error (MSE) of each estimator
MSE <- variance + bias^2
MSE

## **Assess Properties with Increasing Sample Size**

# Settings with different sample sizes
R <- 1000
n_values <- c(10, 200, 1000)  # Different sample sizes
mu <- 2
sigma <- 2

# Array to store estimates for each sample size
# Dimensions: [replication, estimator, sample_size]
est <- array(0, dim = c(R, 4, length(n_values)))
label_est <- c("Mean", "Median", "(Min + Max)/2", "10% Trimmed Mean")

set.seed(13)

for (j in 1:length(n_values)) {
  n <- n_values[j]
  for (i in 1:R) {
    x <- rnorm(n, mu, sigma)
    est[i, 1, j] <- mean(x)
    est[i, 2, j] <- median(x)
    est[i, 3, j] <- (max(x) + min(x)) / 2
    est[i, 4, j] <- mean(x, trim = 0.1)
  }
}

# Plot histograms for each estimator and sample size
for (k in 1:4) {
  par(mfrow = c(1, 3))
  for (j in 1:length(n_values)) {
    hist(est[, k, j], nclass = 30, probability = TRUE, xlab = "",
         xlim = c(-2, 6), main = paste(label_est[k], "- n =", n_values[j]), cex.main = 1.5)
    abline(v = mu, col = "blue", lwd = 2)  # True mean
  }
}

# Compute Variance, Bias, and MSE for each estimator and sample size
variance <- matrix(0, nrow = length(n_values), ncol = 4)
bias <- matrix(0, nrow = length(n_values), ncol = 4)

for (j in 1:length(n_values)) {
  variance[j, ] <- apply(est[, , j], 2, var)
  bias[j, ] <- apply(est[, , j], 2, mean) - mu
}

# Compute MSE
MSE <- variance + bias^2

# Set row and column names for clarity
colnames(bias) <- colnames(variance) <- colnames(MSE) <- label_est
rownames(bias) <- rownames(variance) <- rownames(MSE) <- paste("n =", n_values)

# Display results
variance
bias
MSE

# Observations:
# - As sample size increases, variance and MSE decrease for all estimators.
# - The sample mean tends to have the smallest variance and MSE among the estimators.
```

▼ Lab 4

▼ Interval estimation

```
###########################################
# Statistical Methods: Lab 4 - 25/10/2024 #
# This is the final version               #
###########################################

# Interval estimation ----

## Difference between two means ----

# Load the 'Anorexia.dat' dataset from the given URL into a data frame 'Anor'
Anor <- read.table("http://stat4ds.rwth-aachen.de/data/Anorexia.dat",
                   header = TRUE)

# Calculate the weight change (after - before) for the 'cognitive behavioral' therapy group ('cb')
cogbehav <- Anor$after[Anor$therapy == "cb"] - Anor$before[Anor$therapy == "cb"]

# Calculate the weight change (after - before) for the 'control' group ('c')
control <- Anor$after[Anor$therapy == "c"] - Anor$before[Anor$therapy == "c"]

# Perform a two-sample t-test comparing the mean weight changes between the 'cb' and 'c' groups
# Assume equal variances and use a 95% confidence level
res <- t.test(cogbehav, control, var.equal = TRUE, conf.level = 0.95)

# Extract the 95% confidence interval for the difference of means
res$conf.int

# Manually calculate the 95% confidence interval for the difference of means

# Calculate sample sizes for each group
n1 <- length(cogbehav)   # Number of observations in 'cb' group
n2 <- length(control)    # Number of observations in 'c' group

# Calculate sample variances for each group
var_cogbehav <- var(cogbehav)  # Variance of 'cb' group
```

```r
var_control <- var(control)    # Variance of 'c' group

# Calculate pooled variance (assuming equal variances)
s2 <- ((n1 - 1) * var_cogbehav + (n2 - 1) * var_control) / (n1 + n2 - 2)

# Compute the critical t-value for the given confidence level and degrees of freedom
t_crit <- qt(0.975, df = n1 + n2 - 2)  # Two-tailed test, so use 0.975

# Calculate the standard error of the difference between means
std_error <- sqrt(s2 * (1 / n1 + 1 / n2))

# Calculate the 95% confidence interval for the difference of means manually
CI <- (mean(cogbehav) - mean(control)) + c(-1, 1) * t_crit * std_error

# Display the manually calculated confidence interval
CI

## Difference between two proportions ----

# Define the number of successes and total trials for two groups
success <- c(315, 304)   # Number of successes in each group
total <- c(604, 597)     # Total number of trials in each group

# Perform a two-sample test for proportions
# 'correct = FALSE' disables Yates' continuity correction
res <- prop.test(success, total, conf.level = 0.95, correct = FALSE)

# Extract the 95% confidence interval for the difference in proportions
res$conf.int

# Manually calculate the 95% confidence interval for the difference in proportions

# Calculate sample proportions for each group
p1 <- success[1] / total[1]   # Proportion of successes in group 1
p2 <- success[2] / total[2]   # Proportion of successes in group 2

# Calculate the standard error of the difference between proportions
std_error_prop <- sqrt((p1 * (1 - p1)) / total[1] + (p2 * (1 - p2)) / total[2])

# Compute the critical z-value for the given confidence level
z_crit <- qnorm(0.975)  # Two-tailed test, so use 0.975

# Calculate the 95% confidence interval for the difference in proportions manually
CI_prop <- (p1 - p2) + c(-1, 1) * z_crit * std_error_prop

# Display the manually calculated confidence interval
CI_prop

# Hypothesis testing ----

## Test for the mean difference ----

# Two-sided two-sample t-test assuming equal variances
res.two <- t.test(cogbehav, control, var.equal = TRUE)
res.two  # Display the test results

# Manually calculate the test statistic for the difference between means
testStat <- (mean(cogbehav) - mean(control)) / sqrt(s2 * (1 / n1 + 1 / n2))

# Calculate the two-tailed p-value based on the test statistic
# Multiply by 2 because it's a two-sided test
p.value.two <- 2 * pt(testStat, df = n1 + n2 - 2, lower.tail = FALSE)

# Display the p-value
p.value.two

# Alternative calculation of the p-value
2 * (1 - pt(testStat, df = n1 + n2 - 2))

# One-sided two-sample t-test (alternative: mean of 'cogbehav' greater than 'control')
```

```
res.one <- t.test(cogbehav, control, var.equal = TRUE, alternative = "greater")
res.one  # Display the test results

# Manually calculate the one-sided p-value based on the test statistic
# Since the alternative is 'greater', we look at the upper tail
p.value.one <- pt(testStat, df = n1 + n2 - 2, lower.tail = FALSE)

# Display the p-value
p.value.one

# Load the 'RColorBrewer' package for color palettes
library(RColorBrewer)

# Select a color palette with 6 colors from 'YlOrRd' (Yellow-Orange-Red)
plotclr <- brewer.pal(6, "YlOrRd")

# Plot the t-distribution with degrees of freedom equal to 'n1 + n2 - 2'
curve(dt(x, df = n1 + n2 - 2), xlim = c(-5, 5), ylim = c(0, 0.4),
      main = "p-values and rejection region", col = "blue",
      lwd = 2, xlab = "t", ylab = expression(t[13]), yaxs = "i")

# Define the critical t-value for alpha = 0.05 (one-sided test)
t_crit_one_sided <- qt(0.95, df = n1 + n2 - 2)

# Define the x and y coordinates for shading the rejection region (upper tail)
cord.x <- c(t_crit_one_sided, seq(t_crit_one_sided, 5, 0.01), 5)
cord.y <- c(0, dt(seq(t_crit_one_sided, 5, 0.01), df = n1 + n2 - 2), 0)

# Shade the rejection region using the 'polygon' function
polygon(cord.x, cord.y, col = plotclr[3], border = NA)

# Add a vertical dashed line at the observed test statistic
abline(v = res.one$statistic, lty = 2, lwd = 2, col = "red")

# Add text annotations to indicate acceptance and rejection regions
text(0, 0.2, paste("Accept", expression(H[0])))
text(2.7, 0.08, paste("Reject", expression(H[0])))
text(as.double(res.one$statistic) - 0.15, 0.02, "t", col = "red", cex = 1.2)

# Note: In the plot above, the observed test statistic is close to the critical value.
# This means the result is marginal, and the graphical difference might be hard to detect.

## Test for equality of variance ----

# Using the built-in 'var.test' function (F-test for equality of variances)
var.test(cogbehav, control, alternative = "two.sided")

# Manually calculate the F-test statistic and p-value for equality of variances

# Calculate the ratio of sample variances (F-statistic)
ratiovar <- var_cogbehav / var_control  # F = s1^2 / s2^2

# Calculate the two-tailed p-value for the F-test
# The F-distribution is not symmetric, so adjust for both tails
pv_bi <- 2 * min(
  pf(ratiovar, df1 = n1 - 1, df2 = n2 - 1, lower.tail = FALSE),
  pf(ratiovar, df1 = n1 - 1, df2 = n2 - 1, lower.tail = TRUE)
)

# Display the p-value
pv_bi

## Test for independence ----

# Total sample size
n <- 760

# Observed frequencies in a 3x4 contingency table (regions vs blood types)
obs_freq <- matrix(c(50, 70, 30, 100,
                     114, 30, 10, 100,
```

```r
                     116, 27, 13, 100), nrow = 3, ncol = 4, byrow = TRUE)

# Assign column and row names for clarity
colnames(obs_freq) <- c("A", "B", "AB", "O")          # Blood types
rownames(obs_freq) <- c("South", "Central", "North")  # Regions

# Perform the chi-square test for independence
chisq.test(obs_freq)

# Calculate marginal proportions for columns (blood types)
mx <- colSums(obs_freq) / n
mx  # Column proportions

# Calculate marginal proportions for rows (regions)
my <- rowSums(obs_freq) / n
my  # Row proportions

# Note: The result of 'outer()' can be obtained using:
# - t(mx %*% t(my))
# - my %*% t(mx)
# - tcrossprod(my, mx)

# Calculate expected frequencies under the null hypothesis of independence
exp_freq <- outer(my, mx) * n
exp_freq  # Expected frequencies

# Calculate the chi-square test statistic manually
chi2 <- sum((obs_freq - exp_freq)^2 / exp_freq)

# Display the chi-square statistic
chi2

# Calculate the p-value using the chi-square distribution
# Degrees of freedom = (number of rows - 1) * (number of columns - 1)
df_chi2 <- (nrow(obs_freq) - 1) * (ncol(obs_freq) - 1)
p_value_chi2 <- pchisq(chi2, df = df_chi2, lower.tail = FALSE)

# Display the p-value
p_value_chi2

## Goodness of fit test ----

# Observed data: number of families with 0 to 5 children
child <- 0:5
fam <- c(52, 60, 55, 18, 8, 7)
total_families <- sum(fam)  # Total number of families (should be 200)

# Set the parameter lambda for the Poisson distribution
lambda <- 1.5

# Plot the observed relative frequencies
plot(child, fam / total_families, ylim = c(0, 0.35),
     xlab = "Number of Children", ylab = "Relative Frequency",
     main = "Observed vs Expected Frequencies")

# Calculate the expected probabilities under the Poisson distribution
# For categories 0 to 4, use 'dpois'
# For category 5 (5 or more children), use 'ppois' with 'lower.tail = FALSE'
expected_probs <- c(dpois(0:4, lambda), ppois(4, lambda, lower.tail = FALSE))

# Add lines representing the expected probabilities to the plot
segments(child, rep(0, length(child)), child, expected_probs, col = "red")

# Display the expected probabilities
expected_probs

# Calculate the expected frequencies by multiplying the expected probabilities by total families
exp <- expected_probs * total_families
round(exp, 4)  # Display the expected frequencies rounded to 4 decimal places
```

```r
# Calculate the chi-square statistic components for each category
chisq_el <- (fam - exp)^2 / exp
round(chisq_el, 4)  # Display the components of the chi-square statistic

# Calculate the total chi-square statistic
chisq.obs <- sum(chisq_el)
chisq.obs  # Display the chi-square statistic

# Calculate the p-value using the chi-square distribution
# Degrees of freedom = number of categories - number of estimated parameters - 1
# Since lambda is given (not estimated from data), number of estimated parameters = 0
df_gof <- length(fam) - 1  # df = 6 - 1 = 5

# Compute the p-value
p_value_gof <- pchisq(chisq.obs, df = df_gof, lower.tail = FALSE)

# Display the p-value
p_value_gof

# Alternatively, perform the goodness-of-fit test using 'chisq.test'
# Note: 'chisq.test' adjusts degrees of freedom if parameters are estimated from data
chisq.test(fam, p = expected_probs)
```