Assignment 3

Aman Sanwal, Arihant Sathpathy, Lipika Bagai, Ankita Singh, Gowthamy S

Question 1

METHOD 1

Here we performs kernel density estimation (KDE) and contour plotting for bivariate distributions to determine the number of modes (peaks). It explores four cases:

1. **Unimodal Distribution (Equal Variances)**: A bivariate normal distribution with equal variances.
2. **Unimodal Distribution (Different Variances)**: A bivariate normal distribution where one variable has a higher variance.
3. **Unimodal with Correlation and Shifted Mean**: A bivariate normal distribution with correlation and a nonzero mean.
4. **Bimodal Distribution**: A mixture of two bivariate normal distributions.

Unimodal Case 1: Equal Variance Bivariate Normal Distribution

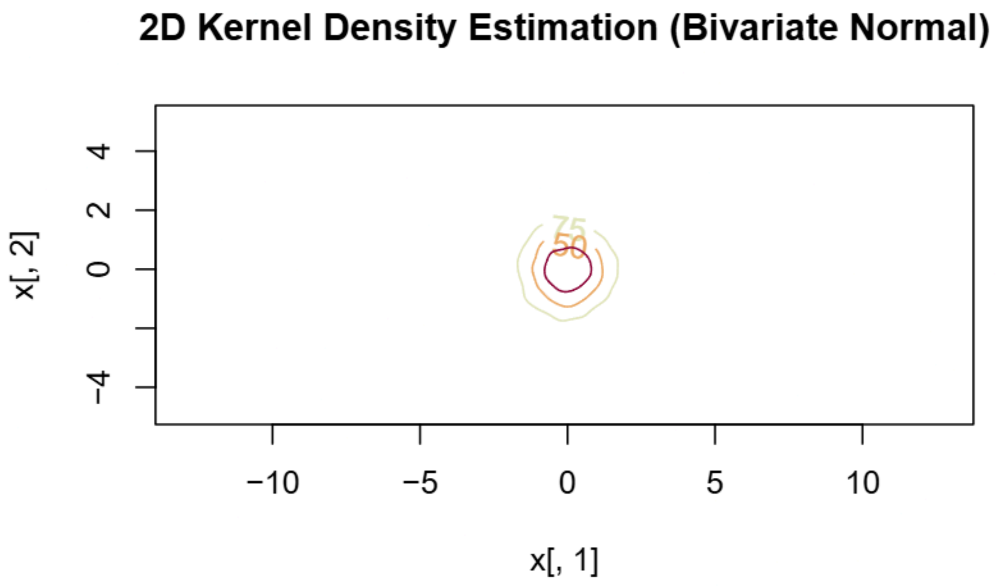
```r
library(MASS)
```

```
Warning: package 'MASS' was built under R version 4.4.3
```

```r
library(ks)
```

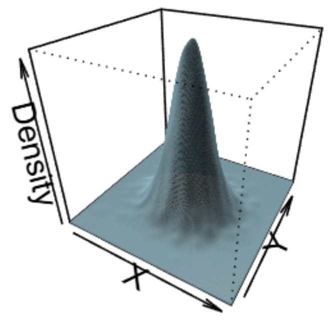
```
Warning: package 'ks' was built under R version 4.4.2
```

```r
set.seed(42)
n <- 10^4
mu <- c(0, 0)
sigma <- matrix(c(1, 0, 0, 1), nrow=2)
data <- mvrnorm(n, mu, sigma)
kde_est <- kde(data, gridsize = c(300, 300))
plot(kde_est, main="2D Kernel Density Estimation (Bivariate Normal)", asp = 1)
```

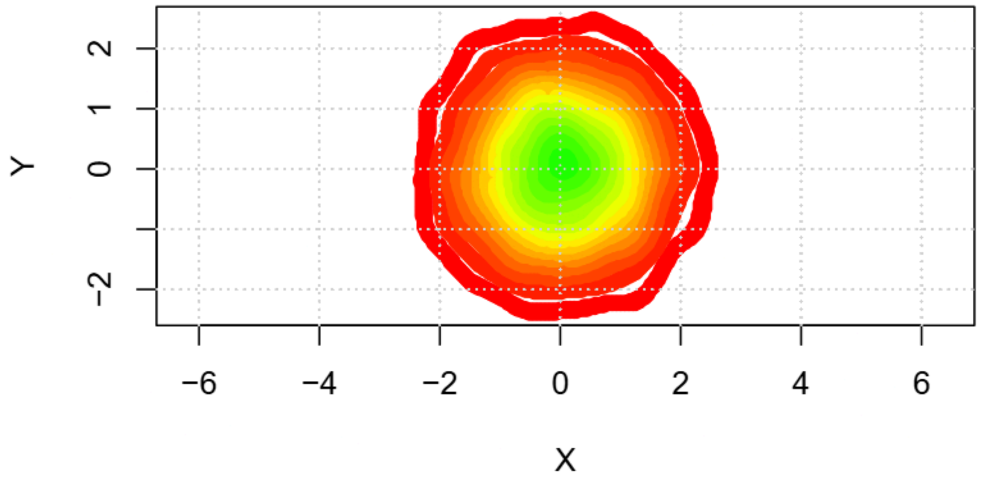


```r
persp(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate,
      theta=30, phi=30, col="lightblue", shade=0.5, border=NA,
      xlab="X", ylab="Y", zlab="Density", main="3D Kernel Density Estimate", asp = 1)
```

3D Kernel Density Estimate



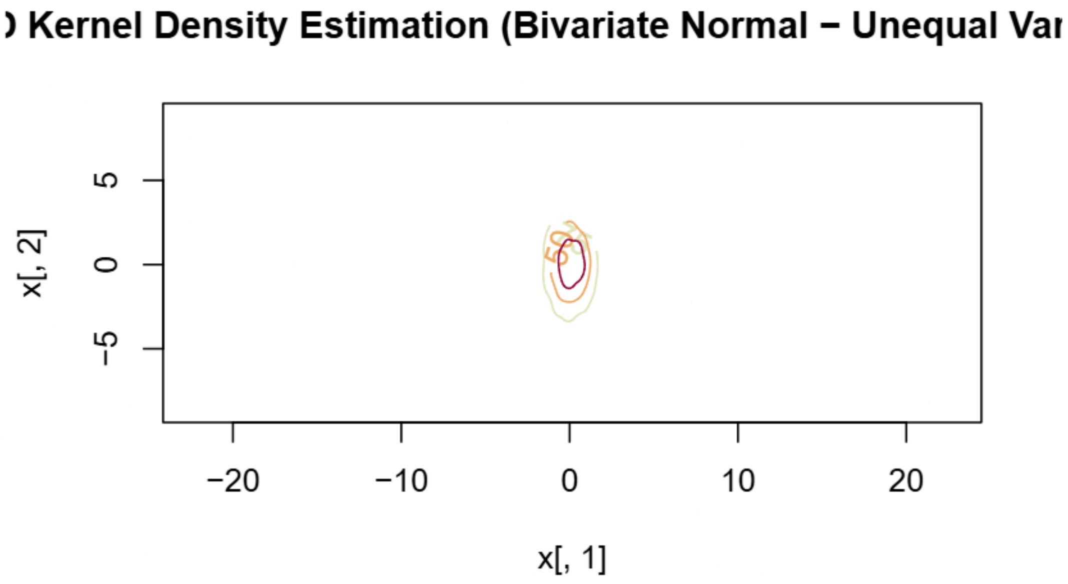
contour plots



Here we visualized density contours by plotting points where the values in a density_matrix are close to specified density levels. Instead of using the built-in contour() function, it manually identifies and plots these points to approximate contour lines.Here as we can observe we get approx circular contour plots centered at (0.0)

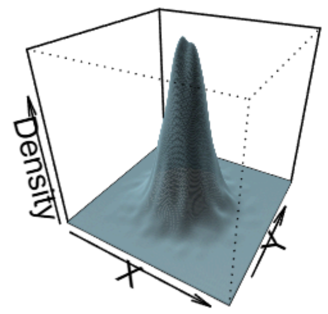
Unimodal Case 2: Unequal Variance Bivariate Normal Distribution

```r
sigma <- matrix(c(1, 0, 0, 4), nrow=2)
data <- mvrnorm(n, mu, sigma)
kde_est <- kde(data, gridsize = c(300, 300))
plot(kde_est, main="2D Kernel Density Estimation (Bivariate Normal - Unequal Variance)", asp
```

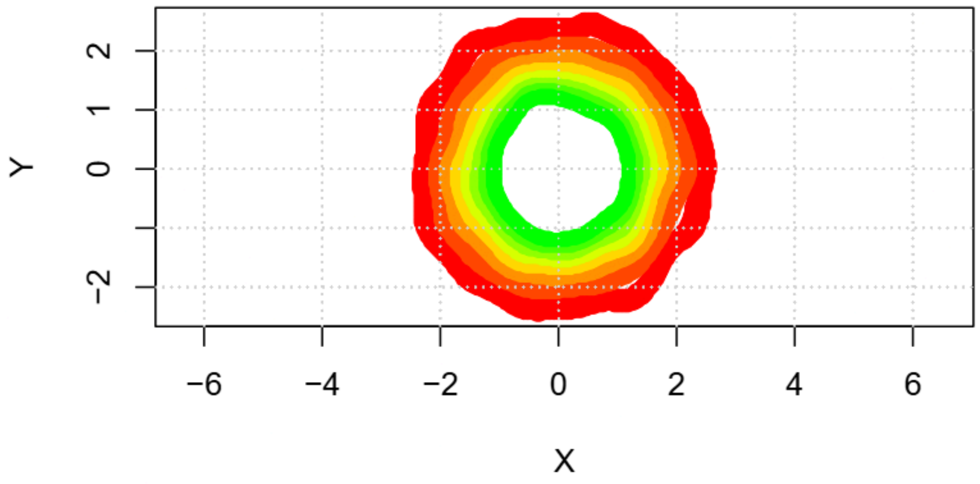


```r
persp(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate,
      theta=30, phi=30, col="lightblue", shade=0.5, border=NA,
      xlab="X", ylab="Y", zlab="Density", main="3D Kernel Density Estimate", asp = 1)
```

3D Kernel Density Estimate



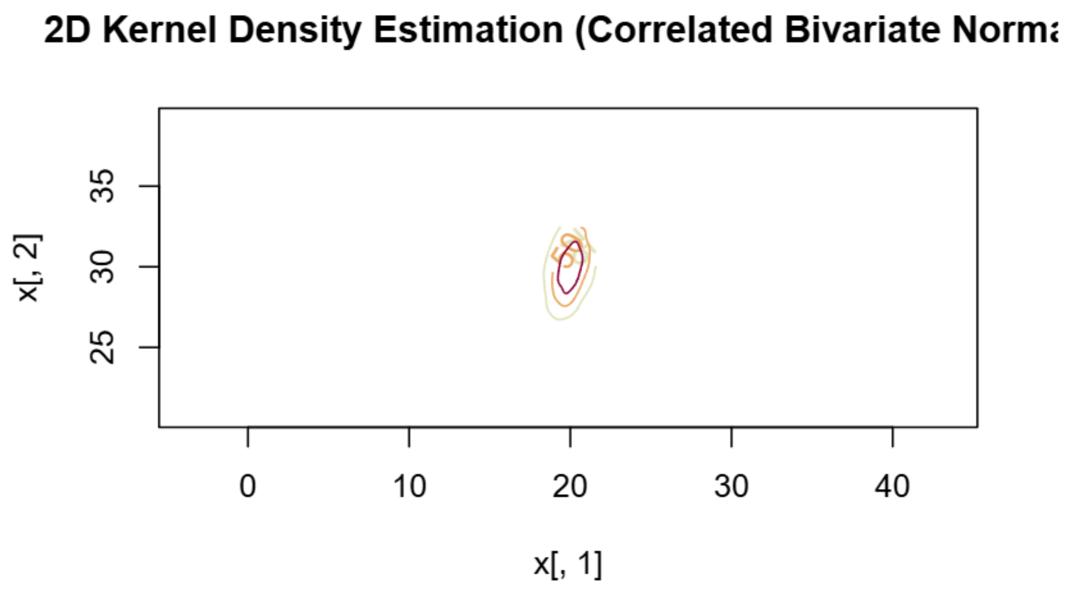
countour plots



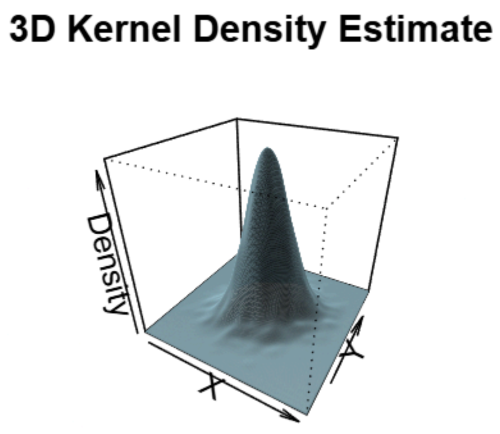
Here as we can observe we get approx ellipse contour plots centered at (0.0)

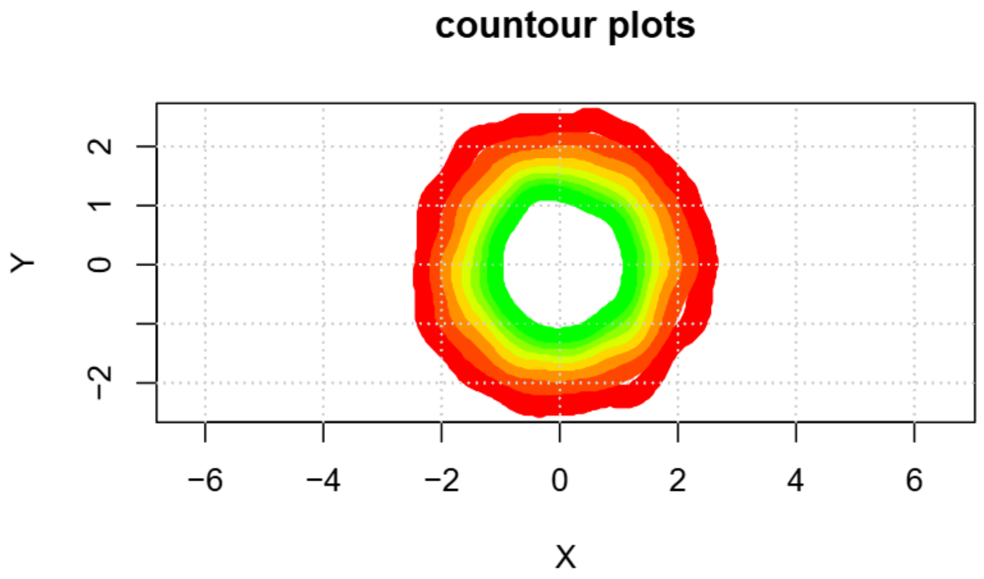
Unimodal Case 3: Correlated Bivariate Normal Distribution

```r
mu <- c(20, 30)
corr <- 0.7
sigma <- matrix(c(1, corr, corr, 4), nrow=2)
data <- mvrnorm(n, mu, sigma)
kde_est <- kde(data, gridsize = c(300, 300))
plot(kde_est, main="2D Kernel Density Estimation (Correlated Bivariate Normal)", asp = 1)
```



```r
persp(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate,
      theta=30, phi=30, col="lightblue", shade=0.5, border=NA,
      xlab="X", ylab="Y", zlab="Density", main="3D Kernel Density Estimate",asp = 1)
```



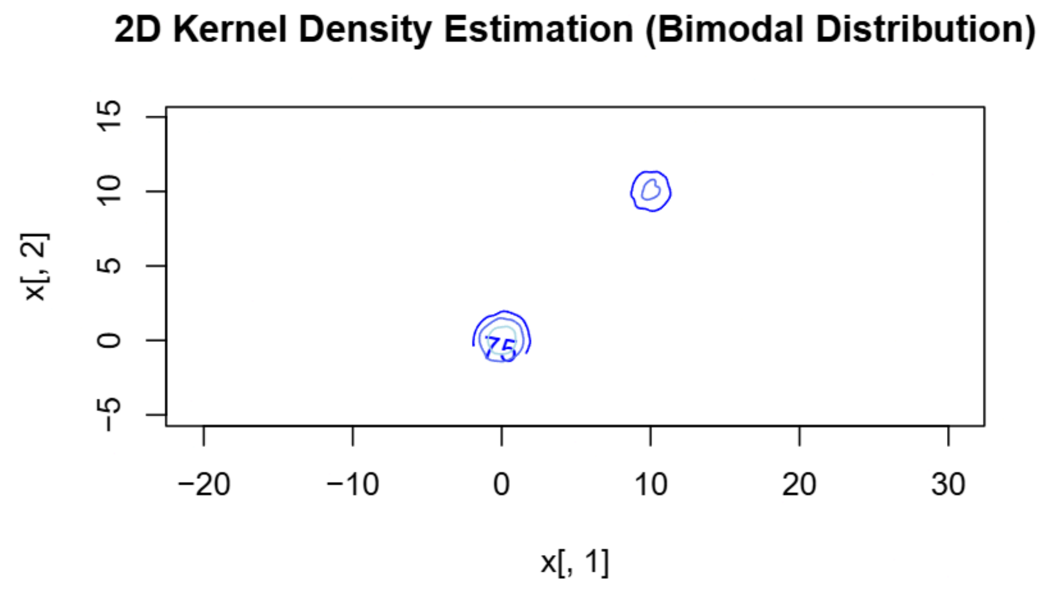


Here as we can observe we get approx ellipse centered at approx(20,30) , with its axis tilted as contour plots

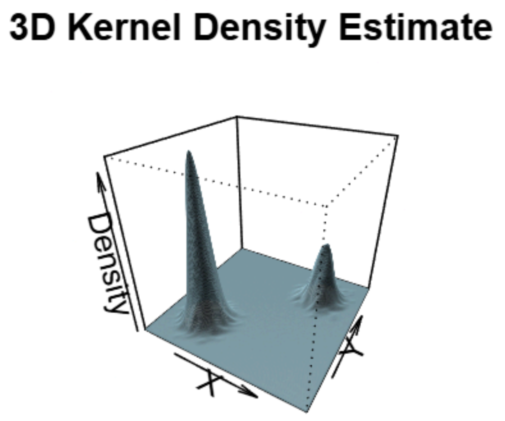
Bimodal Case: Mixture of Two Bivariate Normal Distributions

To visualize density contours, we identify and plot points where the estimated density is approximately equal to specific values. The density function considered here is a bimodal bivariate normal distribution, defined as a weighted mixture of two bivariate normal components: 70% centered at (0,0) and 30% centered at (10,10). Using a threshold-based selection, contour points corresponding to different density levels are plotted in varying colors, forming approximate contour lines. This approach provides a clear representation of the density structure, effectively capturing the bimodal nature of the distribution.

```r
mu1 <- c(0, 0)
mu2 <- c(10, 10)
sigma1 <- matrix(c(1, 0, 0, 1), nrow=2)
sigma2 <- matrix(c(1, 0, 0, 1), nrow=2)
data <- matrix(0, nrow = n, ncol = 2)
for(i in 1:n)
{ u <- runif(1, 0, 1)
  if(u <= 0.7)
  {
    data[i, ] <- mvrnorm(1, mu1, sigma1)
  }
 else { data[i, ] <- mvrnorm(1, mu2, sigma2) }
}
kde_est <- kde(data, gridsize = c(300, 300))
plot(kde_est, main="2D Kernel Density Estimation (Bimodal Distribution)",asp = 1, col.fun =
```

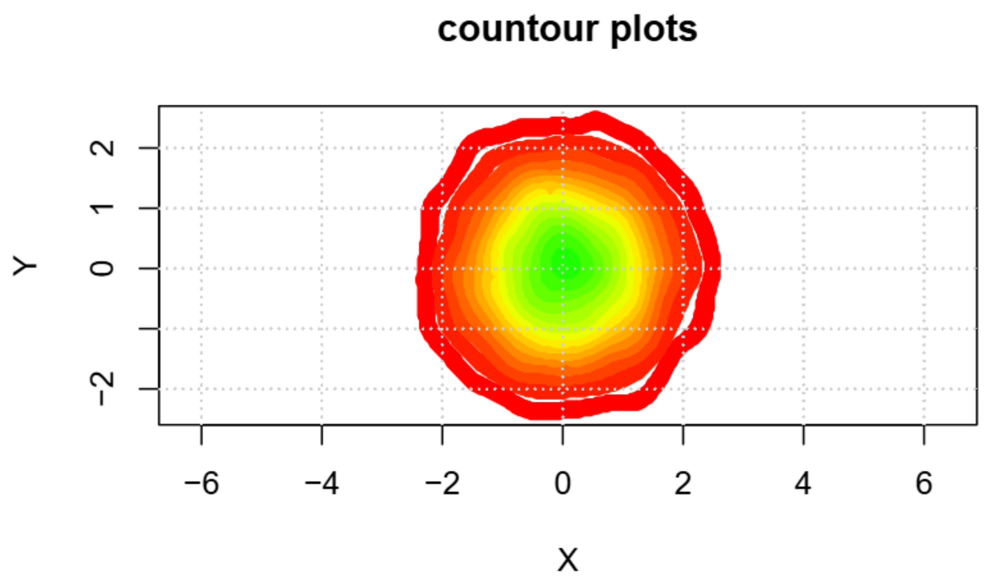


```
persp(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate, theta=30, phi=30
```



```r
thresh_err <- 0.001

plot_density_points <- function(density_matrix, x_eval, y_eval, val, err) {

  # Find indices where density is within the given range
  matches <- which(abs(density_matrix - val) < err, arr.ind = TRUE)

  # Extract corresponding x and y values
  x_match <- x_eval[matches[, 1]]  # Get x-values from first index
  y_match <- y_eval[matches[, 2]]  # Get y-values from second index

  # Plot the selected points
  plot(x_match, y_match, col = "red", pch = 16, xlab = "X", ylab = "Y",
       main ="countour plots", asp =1)

  # Add grid for better visualization
  grid()
}


plot_density_points(density_matrix, x_eval, y_eval, 0.01, thresh_err)



add_density_points <- function(density_matrix, x_eval, y_eval, val, err,co) {

  # Find indices where density is within the given range
  matches <- which(abs(density_matrix - val) < err, arr.ind = TRUE)

  # Extract corresponding x and y values
  x_match <- x_eval[matches[, 1]]  # Get x-values from first index
  y_match <- y_eval[matches[, 2]]  # Get y-values from second index

  # Plot the selected points
  points(x_match, y_match, col=co, pch = 16)

  # Add grid for better visualization
  grid()
}

my_colors <- colorRampPalette(c("red", "yellow", "green"))(16)
```

```
for(i in 2:16){
add_density_points(density_matrix, x_eval, y_eval, 0.01*i, thresh_err,my_colors[i])
}
```



The visualization method used in this analysis effectively captures the density structure of the bimodal bivariate normal distribution. By identifying and plotting points where the estimated density falls within a small threshold (thresh_err = 0.002), we approximate contour lines that reveal the distribution's shape. The results clearly depict two density peaks, corresponding to the two components of the mixture model centered at (0,0) and (10,10).

The use of a color gradient helps distinguish different density levels, while the fine-tuned error threshold ensures a smooth representation of the contours. This approach provides a flexible and interpretable way to visualize density distributions, especially in cases where traditional contour plotting methods may not be suitable.

METHOD 2

Introduction

Kernel Density Estimation (KDE) is a *non-parametric method* for estimating the probability density function of a random variable. This report explores KDE applied to both unimodal and bimodal bivariate normal distributions, using a grid-based approach to estimate density at different points in the data space. The goal is to analyze the resulting contour plots and understand how different distributions influence density visualization.

Kernel Density Estimation for Bivariate Normal Distributions

The density estimation is implemented using a **grid-based approach**, where:

- A **bivariate normal kernel** is used to smooth data points.
- A **bandwidth parameter** controls the level of smoothing.
- Density values are computed at grid points, forming a **contour plot**.
- Quantiles are used to define contour levels, highlighting density variations.

KDE Function Implementation

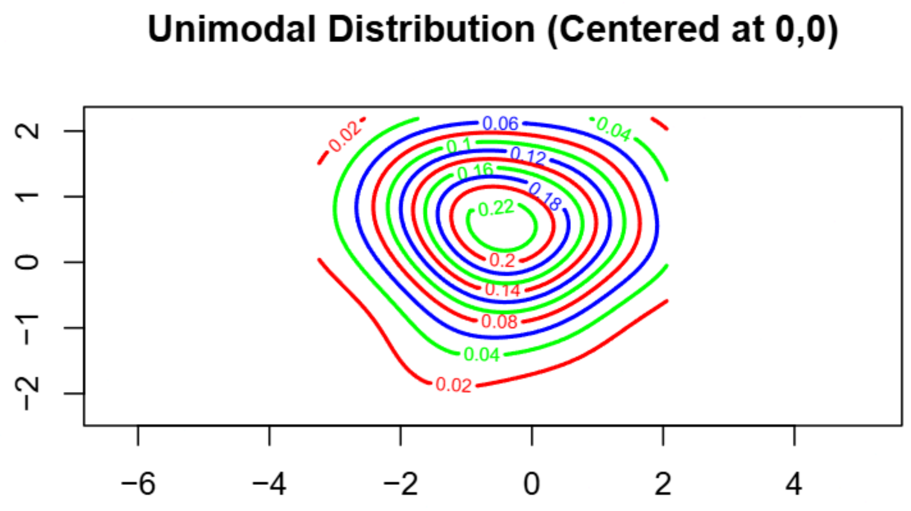
- `kde_biv_norm(x, mu, sigma)`: Computes the bivariate normal density at a given point `x`.
- `kde_biv(grid_x, grid_y, data, bandwidth)`: Estimates KDE by summing contributions from all data points, scaled by `bandwidth`.
- `plot_kde_contour(data, bandwidth, title, num_modes)`: Generates contour plots based on KDE estimates, differentiating between unimodal and bimodal distributions.

Unimodal Distribution 1

```
library(MASS)
library(mvtnorm)
```

```
Warning: package 'mvtnorm' was built under R version 4.4.2
```

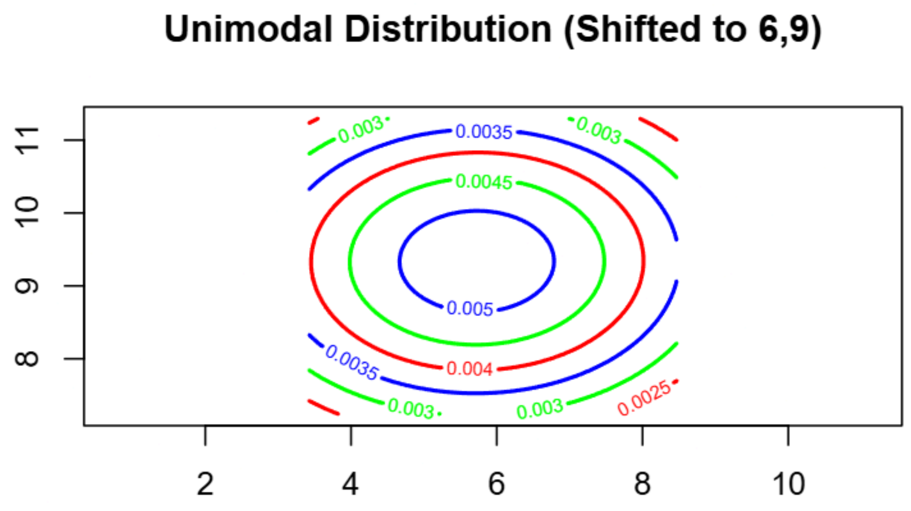
```
set.seed(123)

n <- 100  # Sample size

# Kernel Density Estimation Function
kde_biv <- function(grid_x, grid_y, data, bandwidth) {
  # Create grid
  grid_points <- expand.grid(grid_x, grid_y)
  # Initialize kde matrix
  kde_values <- numeric(nrow(grid_points))

  # Compute KDE using a loop over grid points
  for (i in 1:nrow(grid_points)) {
    kde_values[i] <- sum(dmvnorm(data, mean = as.numeric(grid_points[i, ]), sigma = bandwidtl
  }

  # Normalize KDE
  kde_values <- kde_values / (n * bandwidth)

  # Convert to matrix
  list(x = grid_x, y = grid_y, z = matrix(kde_values, nrow = length(grid_x), byrow = TRUE))
}

# Function to plot contours
plot_kde_contour <- function(data, bandwidth, title, num_modes) {
  x_vals <- seq(min(data[,1]), max(data[,1]), length.out = 100)
  y_vals <- seq(min(data[,2]), max(data[,2]), length.out = 100)

  kde_result <- kde_biv(x_vals, y_vals, data, bandwidth)

  contour(kde_result$x, kde_result$y, kde_result$z,
          main = title, col = c("blue", "red", "green"), lwd = 2, asp = 1)

  cat("Number of modes in", title, ":", num_modes, "\n")
}

# Generate Bivariate Normal Data
mu <- c(0, 0)
sigma <- matrix(c(1, 0, 0, 1), nrow = 2)
data <- mvrnorm(n, mu = mu, Sigma = sigma)

# Plot KDE
plot_kde_contour(data, bandwidth = 0.5, title = "Unimodal Distribution (Centered at 0,0)", nu
```



```
Number of modes in Unimodal Distribution (Centered at 0,0) : 1
```

The density is highest at the center (0,0), decreasing symmetrically outward.A single peak is observed, confirming unimodality.

```
# Unimodal Distribution 2
mu <- c(6,9)
plot_kde_contour <- function(data, bandwidth, title, num_modes) {
  x_vals <- seq(min(data[,1]), max(data[,1]), length.out = 100)
  y_vals <- seq(min(data[,2]), max(data[,2]), length.out = 100)

  kde_result <- kde_biv(x_vals, y_vals, data, bandwidth)

  contour(kde_result$x, kde_result$y, kde_result$z,
          main = title, col = c("blue", "red", "green"), lwd = 2, asp = 1)

  cat("Number of modes in", title, ":", num_modes, "\n")
}
data <- mvrnorm(n, mu = mu, Sigma = sigma)
plot_kde_contour(data, bandwidth = 5, title = "Unimodal Distribution (Shifted to 6,9)", num_
```

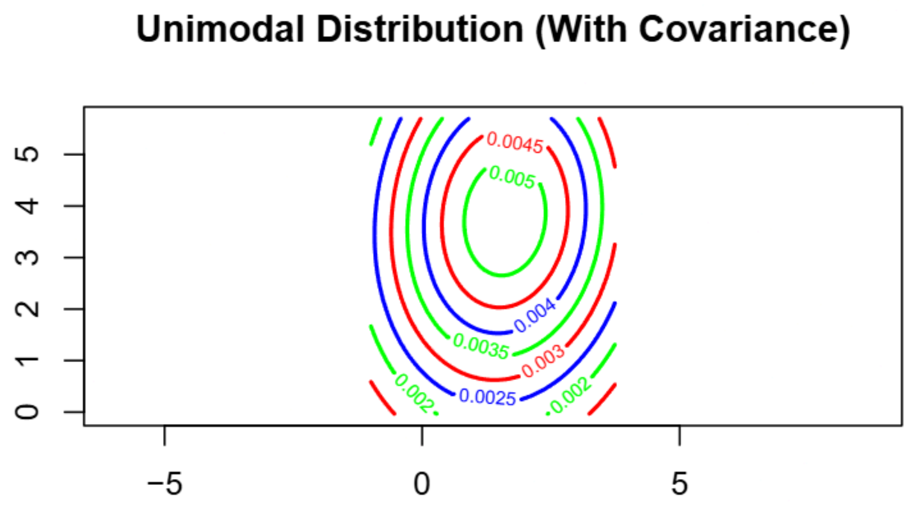


```
Number of modes in Unimodal Distribution (Shifted to 6,9) : 1
```

Similar structure to the previous case but with a new center at (6,9).

The contour shape remains similar, indicating symmetry and unimodality

```
# Unimodal Distribution 3 (With Covariance)
mu <- c(2,3)
cov <- 0.5
sigma <- matrix(c(1, cov, cov, 1), nrow = 2)
data <- mvrnorm(n, mu = mu, Sigma = sigma)
plot_kde_contour(data, bandwidth = 5, title = "Unimodal Distribution (With Covariance)", num_
```

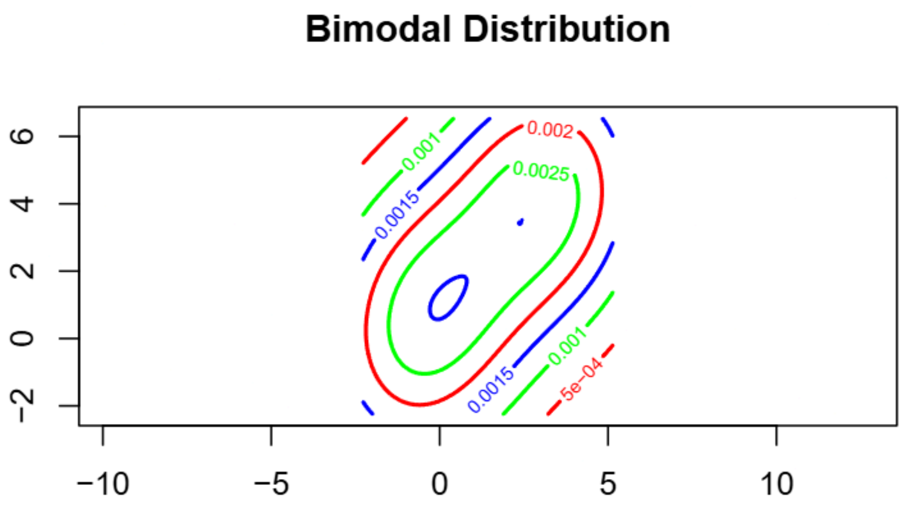


```
Number of modes in Unimodal Distribution (With Covariance) : 1
```

The introduction of covariance (0.5) skews the density.

Contours are elongated along the correlation direction, confirming the effect of dependency between variables.

```
# Bimodal Distribution 1
mu1 <- c(0,0)
mu2 <- c(3,4)
data <- rbind(mvrnorm(n/2, mu = mu1, Sigma = sigma),
              mvrnorm(n/2, mu = mu2, Sigma = sigma))
plot_kde_contour(data, bandwidth = 5, title = "Bimodal Distribution", num_modes = 2)
```



```
Number of modes in Bimodal Distribution : 2
```

Two distinct peaks are observed at (0,0) and (3,4), confirming bimodality.

Contours show two overlapping density regions, suggesting moderate interaction between the modes.

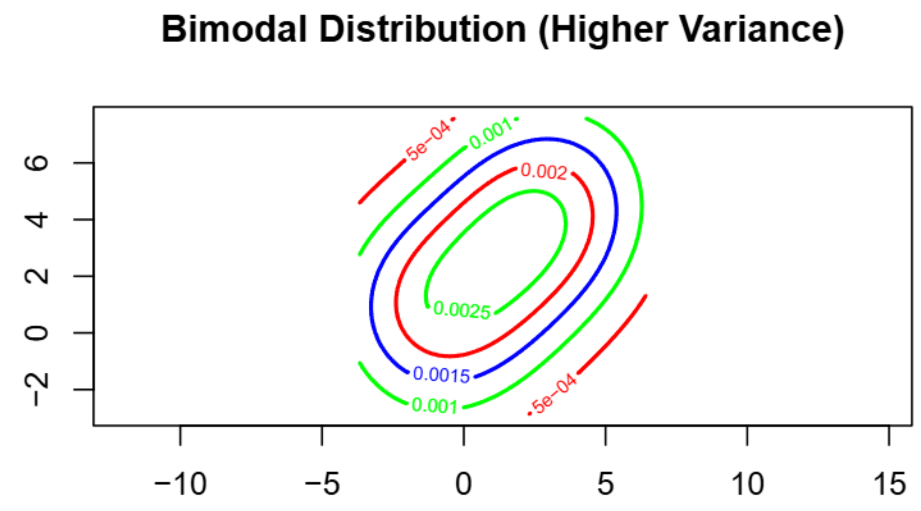
The bimodal nature of the data is evident from the two peaks in the contour plot.

The higher variance (2 in both directions) leads to a more spread-out distribution.

The correlation (cov = 0.5) slightly tilts the density contours, indicating some dependency between the two variables.

Bimodal Distribution 2 (With Different Variance)

```r
# Bimodal Distribution 2 (With Different Variance)
sigma <- matrix(c(2, cov, cov, 2), nrow = 2)
data <- rbind(mvrnorm(n/2, mu = mu1, Sigma = sigma),
              mvrnorm(n/2, mu = mu2, Sigma = sigma))
plot_kde_contour(data, bandwidth = 5, title = "Bimodal Distribution (Higher Variance)", num_
```



```
Number of modes in Bimodal Distribution (Higher Variance) : 2
```

Conclusion

As we can observe we get two different approx circles centered at (0,0) and (10,10) ; if we note carefully we can see that on increasing the height of plane of contour cut the circle centered at (10,10) vanishes after a time because we can see red and orange contours for it but not green and yellow ones which can be seen for the circle at (0,0) ; which makes sense as height of (10,10) peak is lesser than the one centered at (0,0) which can also be seen from the 3-D plot

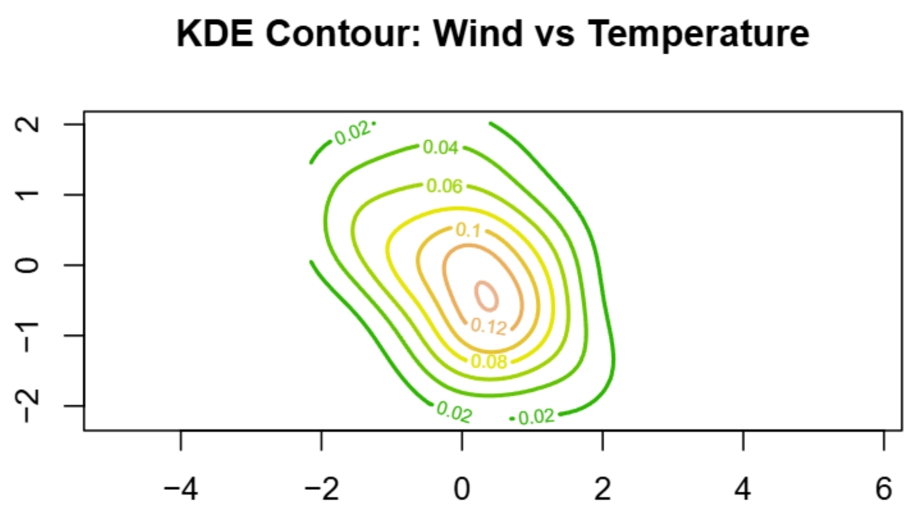
Note that both the density centers are quite far away that is 0,0 and 10,10 are quite far apart that the densities fx1 and fx2 do not affect each other .

Using Real Data to Plot Contours.

1.Unimodal data

```
#Unimodal data

library(MASS)
data(airquality)
```

```r
# Remove missing values
airquality <- na.omit(airquality)

# Standardize Wind and Temperature variables
airquality_std <- scale(airquality[, c("Wind", "Temp")])

# Define the KDE function
kde_biv <- function(grid_x, grid_y, data, bandwidth) {
  n <- nrow(data)
  kde_values <- matrix(0, nrow = length(grid_x), ncol = length(grid_y))

  for (i in 1:length(grid_x)) {
    for (j in 1:length(grid_y)) {
      x_point <- c(grid_x[i], grid_y[j])
      kernel_values <- apply((data - x_point) / bandwidth, 1, function(x) exp(-sum(x^2) / 2)
      kde_values[i, j] <- sum(kernel_values) / (n * bandwidth^2 * 2 * pi)
    }
  }

  list(x = grid_x, y = grid_y, z = kde_values)
}

# Function to count modes in KDE
count_modes <- function(kde_result) {
  z_matrix <- kde_result$z
  n_modes <- 0

  for (i in 2:(nrow(z_matrix) - 1)) {
    for (j in 2:(ncol(z_matrix) - 1)) {
      center <- z_matrix[i, j]
      neighbors <- c(
        z_matrix[i-1, j], z_matrix[i+1, j],
        z_matrix[i, j-1], z_matrix[i, j+1],
        z_matrix[i-1, j-1], z_matrix[i-1, j+1],
        z_matrix[i+1, j-1], z_matrix[i+1, j+1]
      )

      if (all(center > neighbors)) {
        n_modes <- n_modes + 1
      }
    }
  }
  return(n_modes)
}

# Function to plot KDE contour
plot_kde_contour <- function(data, bandwidth, title) {
  x_vals <- seq(min(data[,1]), max(data[,1]), length.out = 100)
  y_vals <- seq(min(data[,2]), max(data[,2]), length.out = 100)

  kde_result <- kde_biv(x_vals, y_vals, data, bandwidth)

  contour(kde_result$x, kde_result$y, kde_result$z,
          main = title, col = terrain.colors(10), lwd = 2,asp = 1)



  num_modes <- count_modes(kde_result)
  cat("Number of modes in", title, ":", num_modes, "\n")
}

# Convert to matrix
data_matrix <- as.matrix(airquality_std)

# Plot contour and count modes
plot_kde_contour(data_matrix, bandwidth = 0.5, title = "KDE Contour: Wind vs Temperature")
```



```
Number of modes in KDE Contour: Wind vs Temperature : 1
```

```
#here the number of modes is 1
```

First, we standardizes the data to ensure a fair comparison and removes any missing values. The `kde_biv` function computes the density estimate using a Gaussian kernel, while `count_modes` identifies the number of local maxima in the density function. The `plot_kde_contour` function visualizes the estimated density with a contour plot, overlaying the original data points for reference.

2.Bimodal data

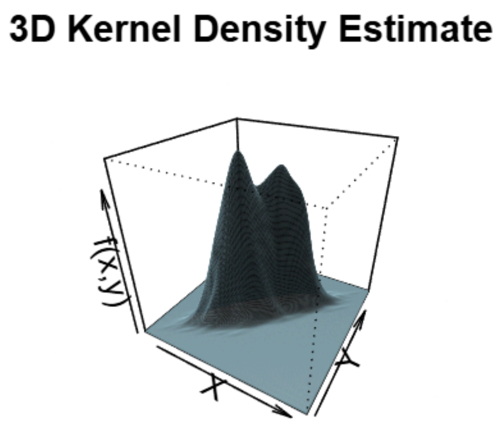
```
library(MASS)
library(ks)
library(readr)
```

```
Warning: package 'readr' was built under R version 4.4.2
```

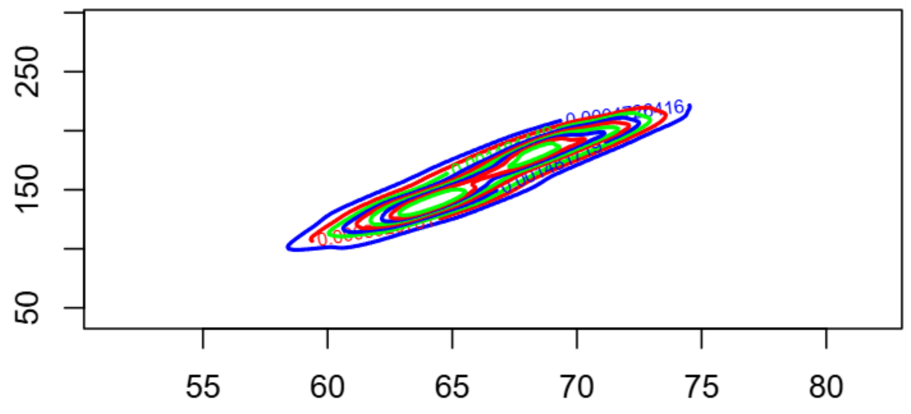
```
df <- read_csv("weight-height.csv")
```

```
Rows: 10000 Columns: 3
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): Gender
dbl (2): Height, Weight

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
data<- matrix(0, nrow = 10000, ncol = 2)

data[,1] <- df$Height
data[,2] <- df$Weight


kde_est <- kde(x = data)  # Perform 2D KDE
kde_est <- kde(data, gridsize = c(300, 300))

evalu<-kde_est$eval.points

x_eval<-evalu[1]
y_eval<-evalu[2]
val<- max(kde_est$estimate)
density_matrix <- kde_est$estimate

options(max.print = 100000)  # Increase limit to 100,000 values


contour_levels <- kde_est$cont


# Perspective 3D Plot
persp(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate,
      theta=30, phi=30, col="lightblue", shade=0.5, border=NA,
      xlab="X", ylab="Y", zlab="f(x,y)", main="3D Kernel Density Estimate")
```



```r
x_eval <- unlist(x_eval)  # Convert list to numeric vector

y_eval <- unlist(y_eval)

selected_levels <- quantile(kde_est$cont, probs = seq(0.1, 0.9, by = 0.1))

contour(kde_est$eval.points[[1]], kde_est$eval.points[[2]], kde_est$estimate,
        levels = selected_levels,
        col = c("blue", "red", "green"), lwd = 2)
```



We can observe the two modes appearing for male and female respectively , however since the peaks are very close hence they are a bit merged and not completely separated also since there is positive correlation between weight and in general as can be expected hence we can #see that the ellipse is slanted

QUESTION 2

Introduction

Monte Carlo methods provide a powerful tool for estimating complex integrals, solving numerical problems, and approximating mathematical constants. This report explores the use of Monte Carlo integration to estimate a given integral and evaluate its rate of convergence. Additionally, the method is used for estimating the mathematical constant using different approaches.

The objectives of this report include:

- Estimating a given integral using Monte Carlo methods.

- Analyzing the rate of convergence of the estimator.

- Comparing two different estimators for .

- Evaluating the variance of the estimators and their efficiency.

Monte Carlo Estimation of Integral

The integral to be estimated is given by:

$$I = \int_0^1 \frac{x^8(1-x)^8(25+816x^2)}{3164(1+x^2)}\,dx$$

This integral is evaluated using Monte Carlo integration by sampling from a uniform distribution (U(0,1)).

```r
# Function for the integral
f <- function(x) {
  return((x^8 * (1 - x)^8 * (25 + 816 * x^2)) / (3164 * (1 + x^2)))
}

# Monte Carlo estimation
set.seed(123)
n <- 1e3
x_samples <- runif(n, 0, 1)
estimated_integral <- mean(f(x_samples))

cat("Estimated value of the integral:", estimated_integral, "\n")
```

```
Estimated value of the integral: 2.66593e-07
```

Rate of Convergence Analysis

The rate of convergence of the Monte Carlo estimator is analyzed using the delta method. The approach involves adjusting a parameter (*delta*) to examine how quickly the estimate converges to its true value.

```r
integral_estimator <- function(n) {
  x <- runif(n, 0, 1)
  mean(f(x))
}

rate_of_convergence <- function(n) {
  del <- 0.1
  while(TRUE) {
    est <- integral_estimator(n)
    if (is.infinite(n^del * abs(est))) {
      return(del)
    } else {
      del <- del + 0.1
    }
  }
}

n_values <- c(100, 1000, 10000, 100000)
delta_values <- sapply(n_values, rate_of_convergence)

results <- data.frame(n = n_values, delta = delta_values)
print(results)
```

```
      n delta
1 1e+02 154.2
2 1e+03 102.8
3 1e+04  77.1
4 1e+05  61.7
```

Estimation of

Two different Monte Carlo estimators for are considered: 1. **Estimator 1**: Based on the given integral:

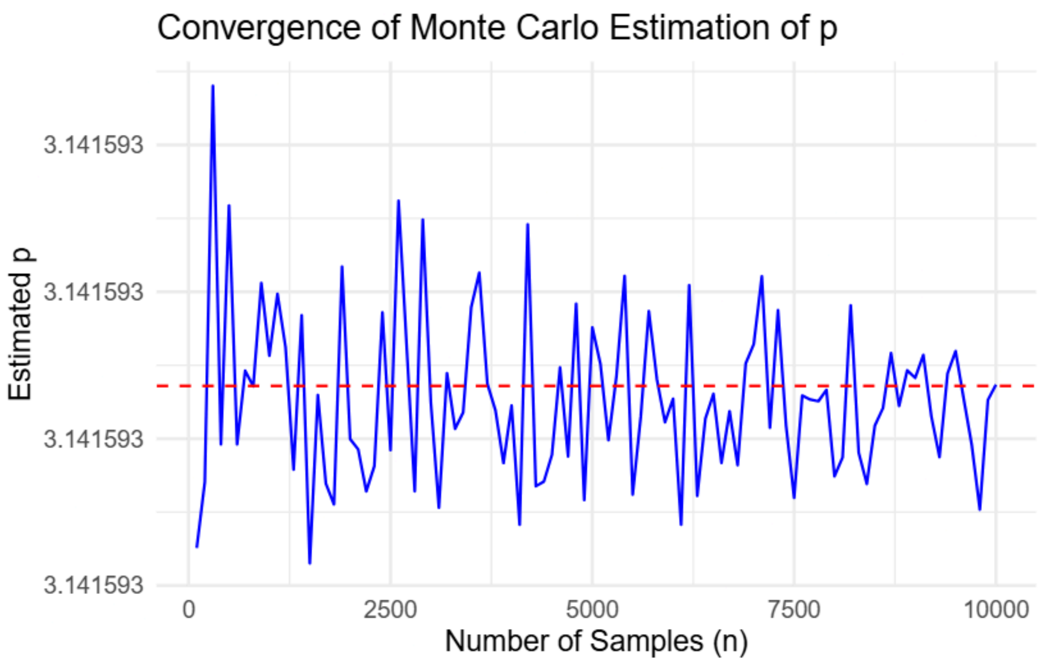
$$\pi \cdot 1 = \frac{355}{113} - \hat{I}$$

2. Estimator 2: Based on the standard Monte Carlo integration method:

$$\pi_2 = \mathbb{E}\left[\frac{4}{1+X^2}\right]$$

```
library(ggplot2)

estimate_pi_1 <- function(n) {
  355/113 - integral_estimator(n)
```

```r
}

n_values <- seq(100, 10000, by = 100)
pi_estimates <- sapply(n_values, estimate_pi_1)

# Plot convergence of pi estimates
pi_df <- data.frame(n = n_values, pi_est = pi_estimates)

ggplot(pi_df, aes(x = n, y = pi_est)) +
  geom_line(color = "blue") +
  geom_hline(yintercept = pi, linetype = "dashed", color = "red") +
  labs(title = "Convergence of Monte Carlo Estimation of  ",
       x = "Number of Samples (n)",
       y = "Estimated  ") +
  theme_minimal()
```



Comparison with Unit Circle Method

Another common Monte Carlo method for estimating   involves randomly sampling points inside a unit square and computing the fraction of points that fall inside the unit circle.

```
estimate_pi_standard <- function(N) {
  x <- runif(N, -1, 1)
  y <- runif(N, -1, 1)
  mean(x^2 + y^2 <= 1) * 4
}

N_vals <- c(1000, 10000, 100000)
results <- data.frame(N = N_vals, Integral_Estimate = NA, Pi_Estimate = NA, Pi_Standard = NA)

for (i in seq_along(N_vals)) {
  N <- N_vals[i]
  I_hat <- integral_estimator(N)
  pi_corrected <- 355 / 113 - I_hat
  pi_standard <- estimate_pi_standard(N)
  results[i, 2] <- I_hat
  results[i, 3] <- pi_corrected
  results[i, 4] <- pi_standard
}

print(results)
```

```
      N Integral_Estimate Pi_Estimate Pi_Standard
1 1e+03      2.676215e-07    3.141593     3.13200
2 1e+04      2.624096e-07    3.141593     3.15360
3 1e+05      2.675352e-07    3.141593     3.14092
```

Variance Comparison

To determine which estimator is more efficient, the variance of each estimator is computed and compared.

```
num_trials <- 100
var_corrected <- var(replicate(num_trials, 355 / 113 - integral_estimator(10000)))
var_standard <- var(replicate(num_trials, estimate_pi_standard(10000)))

cat("\nVariance of Corrected Estimator:", var_corrected, "\n")
```

```
Variance of Corrected Estimator: 1.259707e-17
```

```
cat("Variance of Standard Monte Carlo Estimator:", var_standard, "\n")
```

```
Variance of Standard Monte Carlo Estimator: 0.0002209282
```

Conclusion

In this report, Monte Carlo integration was used to estimate a given integral, analyze the rate of convergence, and apply different estimators for . The results demonstrate that:

- The integral estimation method provides an efficient way to approximate values numerically.

- The rate of convergence can be analyzed using the delta method.

- The variance of the corrected estimator for   is significantly lower than that of the unit circle method, making it a more efficient approach.

- The convergence plot clearly illustrates that as the number of samples increases, the estimated value of   stabilizes around its true value.