Homework 1 - Monte Carlo Methods

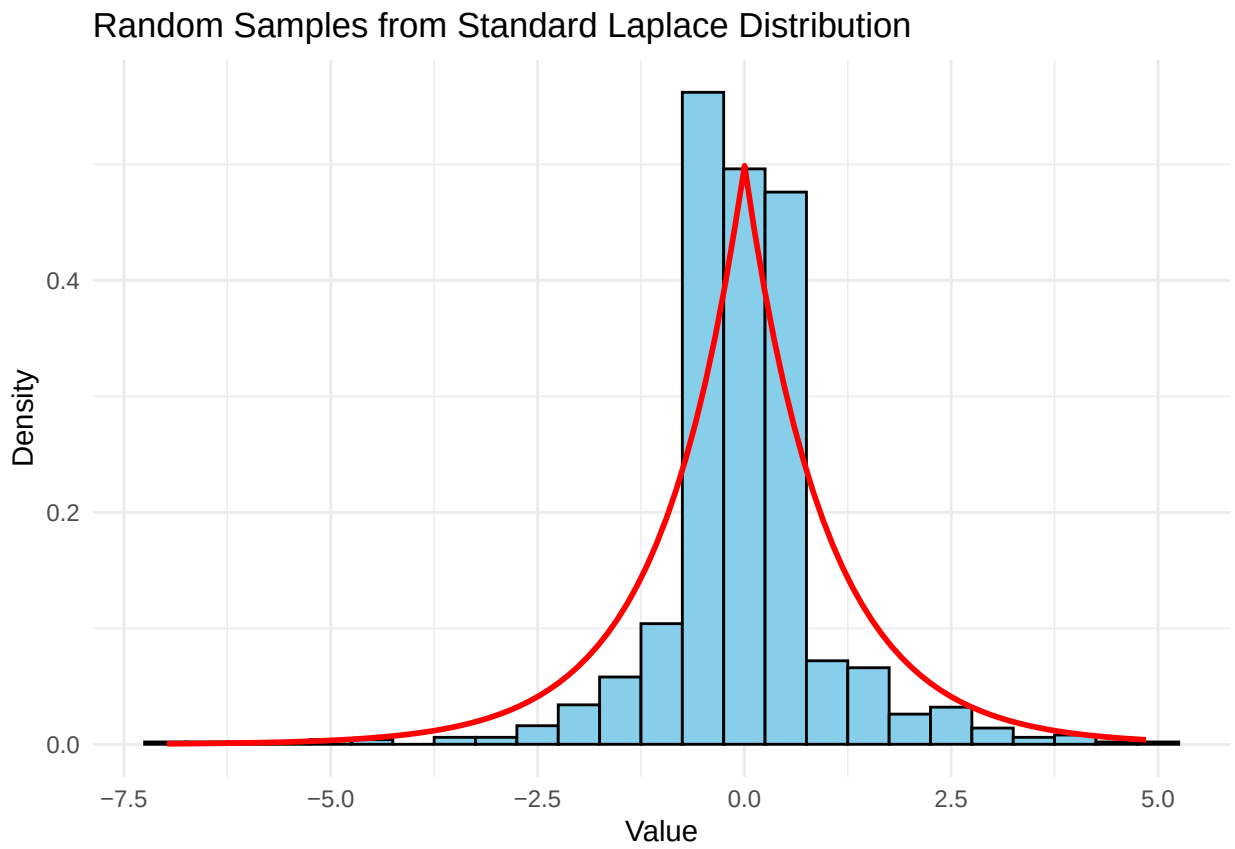
Zhuodiao Kuang(zk2275)

Problem 1

The standard Laplace distribution has density $f(x) = 0.5e^{-|x|}, x \in (-\infty, \infty)$. Please provide an algorithm that uses the inverse transformation method to generate a random sample from this distribution. Use the $U(0, 1)$ random number generator in R, write a R-function to implement the algorithm. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the standard Laplace distribution.)

Answer1

```r
# Function to generate random samples from the standard Laplace distribution
generate_laplace_samples <- function(n) {
  u <- runif(n)  # Generate n random values from U(0, 1)
  sign_val <- ifelse(runif(n) > 0.5, 1, -1)  # Randomly choose the sign
  x <- sign_val * log(2 * u)  # Apply the inverse CDF
  return(x)
}

set.seed(123)
laplace_samples <- generate_laplace_samples(1000)

# Create a data frame for visualization
df <- data.frame(x = laplace_samples)

# Plot true density function and simulation results
library(ggplot2)
ggplot(df, aes(x)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black",
                 aes(y = after_stat(density))) +
  stat_function(fun = function(x) 0.5 * exp(-abs(x)), color = "red",
                linewidth = 1) +
  labs(title = "Random Samples from Standard Laplace Distribution",
       x = "Value", y = "Density") +
  theme_minimal()
```



Problem 2

Use the inverse transformation method to derive an algorithm for generating a Pareto random number with $U \sim U(0,1)$, where the Pareto random number has a probability density function

$$f(x;\alpha,\gamma) = \frac{\gamma\alpha^{\gamma}}{x^{\gamma+1}} I\{x \geq \alpha\}$$

with two parameters $\alpha > 0$ and $\gamma > 0$. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

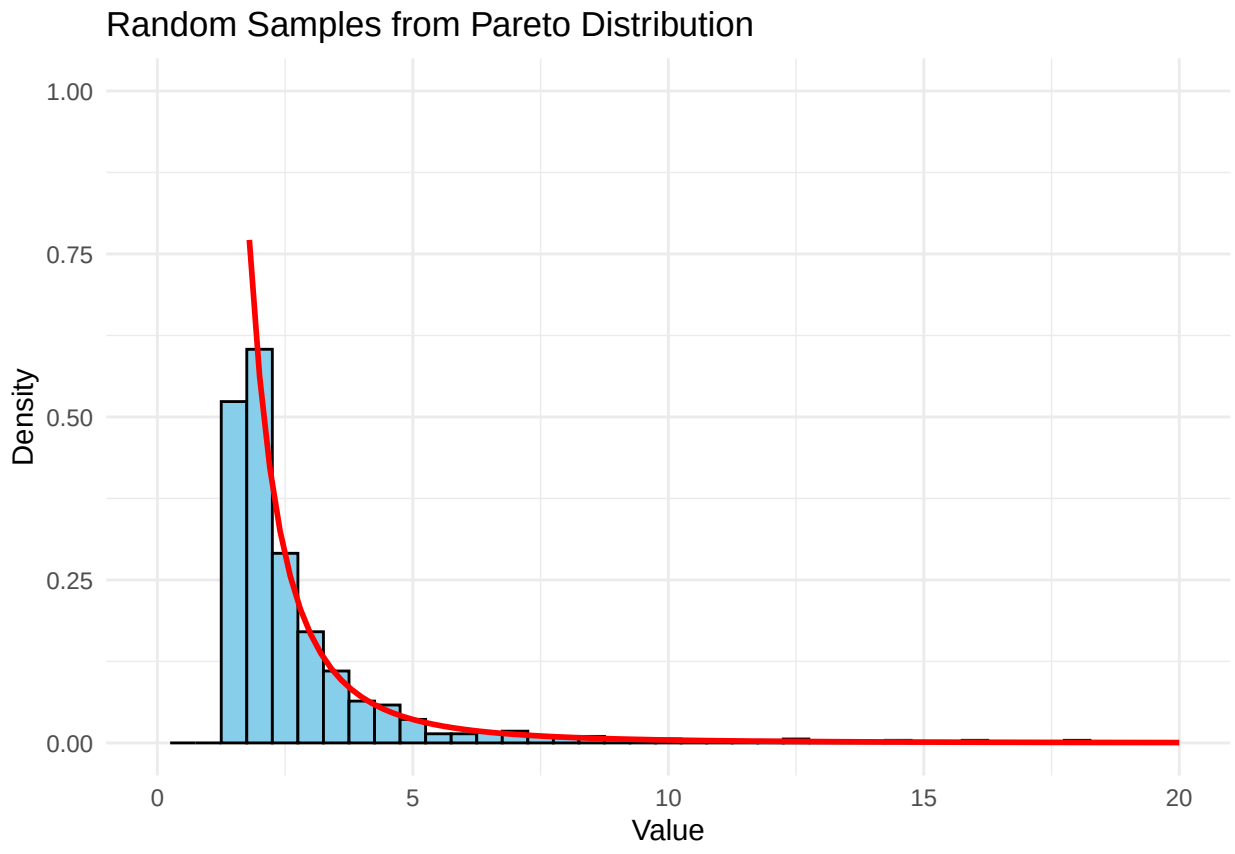
Answer2

```r
# Function to generate random samples from the Pareto distribution
generate_pareto_samples <- function(n, alpha, gamma) {
  u <- runif(n)  # Generate n random values from U(0, 1)
  x <- alpha * (1 / (1 - u))^(1 / gamma)  # Apply the inverse CDF
  return(x)
}

# Example usage:
set.seed(123)  # For reproducibility
```

```r
alpha_val <- 1.5
gamma_val <- 2
pareto_samples <- generate_pareto_samples(1000, alpha_val, gamma_val)

# Visualization:
library(ggplot2)
ggplot(data.frame(x = pareto_samples), aes(x)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black",
                 aes(y = after_stat(density))) +
  stat_function(fun = function(x)
    gamma_val*alpha_val^gamma_val/(x^(gamma_val+1)),
    color = "red", linewidth = 1) +
  ylim(0,1)+
  xlim(0,20)+
  labs(title = "Random Samples from Pareto Distribution",
       x = "Value", y = "Density") +
  theme_minimal()
```

```
## Warning: Removed 3 rows containing non-finite values (‘stat_bin()‘).
```

```
## Warning: Removed 2 rows containing missing values (‘geom_bar()‘).
```



Problem 3

Construct an algorithm for using the acceptance/rejection method to generate 100 pseudorandom variable from the pdf

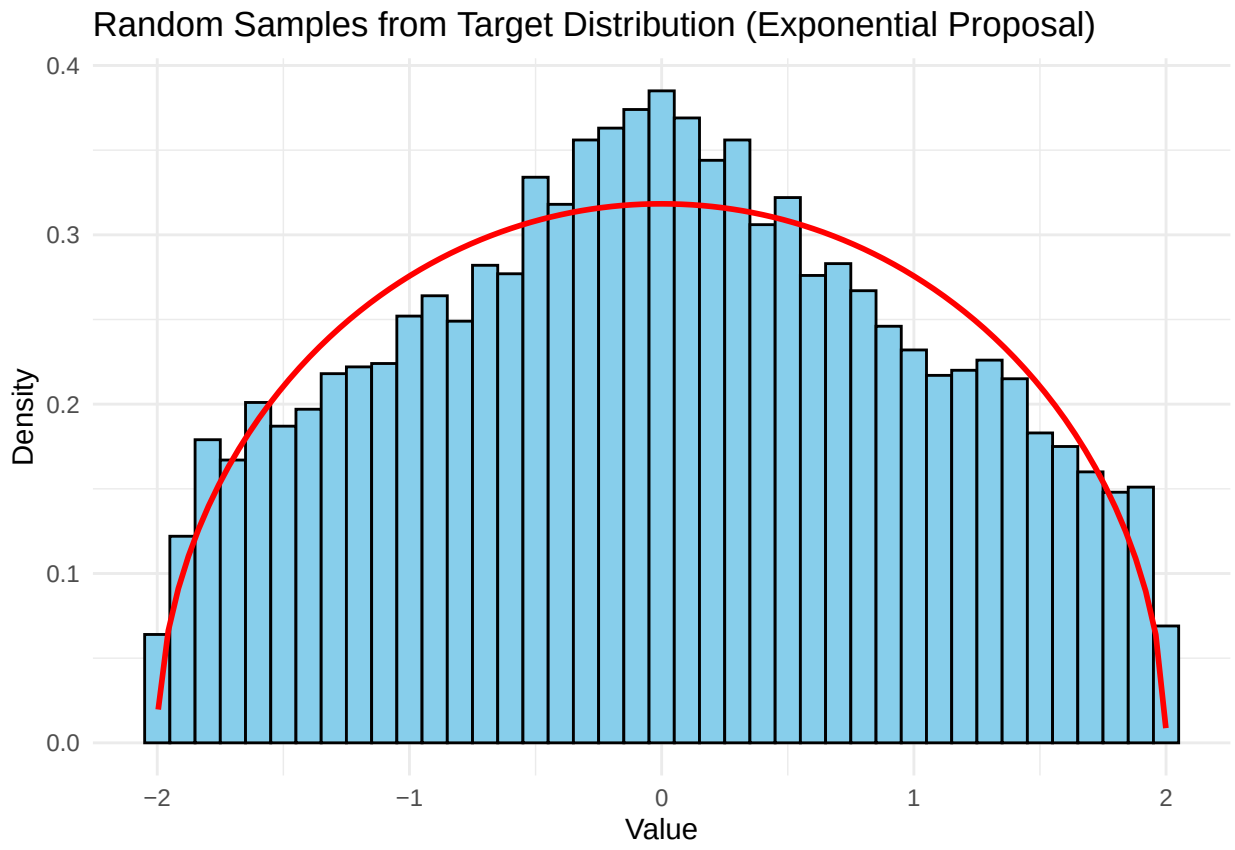
$$f(x) = \frac{2}{\pi\beta^2}\sqrt{\beta^2 - x^2},\ -\beta \le x \le \beta.$$

The simplest choice for $g(x)$ is the $U(-\beta, \beta)$ distribution but other choices are possible as well. Use visualization tools to validate your algorithm (i.e., illustrate whether the random numbers generated from your function truely follows the target distribution.)

Answer3

```r
# Function to generate pseudorandom variables using acceptance-rejection method
generate_pareto_samples_exponential <- function(n, beta) {
  lambda <- 1 / beta  # Exponential parameter
  M <- 2 / (pi * beta)  # Upper bound for the acceptance ratio
  accepted_samples <- numeric(n)

   for (i in 1:n) {
    repeat {
      Sign <- rbinom(1,1,0.5)
      if(Sign == 1){Sign = 1}
      else{Sign = -1}
      X <- Sign*rexp(1, rate = lambda)  # Generate sample from Exp(lambda)
      if (abs(X) > beta) {
        next  # Reject if X > beta
      }

      f_X <- 2 / (pi * beta^2) * sqrt(beta^2 - X^2)  # Target pdf value at X
      g_X <- lambda/2 * exp(-abs(lambda * X))  # Exponential pdf value at X

      R <- f_X / (M * g_X)  # Acceptance ratio
      U <- runif(1)

      if (U <= R) {
        accepted_samples[i] <- X
        break  # Accept the sample
      }
    }
  }

  return(accepted_samples)
}




set.seed(211)
beta = 2 # example beta
accepted_samples_exp <- generate_pareto_samples_exponential(10000, beta)
```

```r
# Visualization:
ggplot(data.frame(x = accepted_samples_exp), aes(x)) +
  geom_histogram(binwidth = 0.1, fill = "skyblue",
                 color = "black",aes(y = after_stat(density))) +
  stat_function(fun = function(x)
    2*sqrt(beta^2-x^2)/(pi*beta^2),
                color = "red", linewidth = 1) +
  labs(title = "Random Samples from Target Distribution (Exponential Proposal)",
       x = "Value", y = "Density") +
  theme_minimal()
```



Problem 4

Suppose you are tasked with estimating the expected annual health care costs for a population of patients. The given distribution represents the historical data on annual health care expenses for this population, and it follows a right-skewed distribution resembling the Gamma distribution with $\alpha = 3$ and $\beta = 2$.

The pdf of the Gamma distribution with shape parameter α (alpha) and rate parameter β (beta) is given by:

$$f(x; \alpha, \beta) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}$$

Where: - x is the random variable. - α (alpha) is the shape parameter. - β (beta) is the rate parameter. - $\Gamma(\alpha)$ is the gamma function.

4.1 Please Monte Carlo simulation to sample from this Gamma distribution and calculate the expected annual costs.

4.2 Evaluate the variance of your estimator to assess its efficiency.

Answer4

```r
set.seed(212)
# Parameters for the target Gamma distribution
alpha <- 3
lambda <- beta <- 2
C <- 10


# Function to sample from the proposal distribution (Exponential)
sample_proposal <- function() {
  rexp(1, rate=lambda)
}

# Function to calculate the PDF of the target Gamma distribution
gamma_pdf <- function(x, alpha, beta) {
  (beta^alpha * x^(alpha-1) * exp(-beta*x)) / gamma(alpha)
}

# Function to calculate the PDF of the proposal Exponential distribution
exp_pdf <- function(x, lambda) {
  lambda * exp(-lambda*x)
}

# Acceptance-rejection sampling function
samples <- n_samples <- 1000

# For variance
K <- 100
expected_costs <- c(1:K)


for (i in 1:K) {
  accepted <- 0
  while (accepted < n_samples) {
  x_proposal <- sample_proposal()
  u <- runif(1)

  # Calculate the acceptance probability
  f <- gamma_pdf(x_proposal, alpha, beta)
  g <- exp_pdf(x_proposal, lambda)
  if (u < (f / (C * g))) {
    accepted <- accepted + 1
    samples[accepted] <- x_proposal
                        }
  }
  # Calculate the expected annual costs (mean of the accepted samples)
```

```
expected_costs[i] <- mean(samples)
}




# Calculate the variance of the estimator (variance of the accepted samples)
variance_estimator <- var(expected_costs)

# Print the results
cat("Expected annual health care costs:", expected_costs[1], "\n")
```

```
## Expected annual health care costs: 1.397773
```

```
cat("Variance of the estimator:", variance_estimator, "\n")
```

```
## Variance of the estimator: 0.0004224218
```

Problem 5

Show that in estimating $\theta = E\{\sqrt{1-U^2}\}$ it is better to use U^2 rather than U as the control variate, where $U \sim U(0,1)$. To do this, use simulation to approximate the necessary covariances. In addition, implement your algorithms in R.

Answer5

Estimation

```
estimate_theta_U <- function(n) {
    U <- runif(n)
    theta_U <- mean(sqrt(1 - U^2) -
    cov(U, sqrt(1 - U^2)) / var(U) * (U - mean(U)))
    variance_U <- var(sqrt(1 - U^2) -
    cov(U, sqrt(1 - U^2)) / var(U) * (U - mean(U)))
    return(list(theta = theta_U, variance = variance_U))
}

estimate_theta_U2 <- function(n) {
    U <- runif(n)
    theta_U2 <- mean(sqrt(1 - U^2) -
    cov(U^2, sqrt(1 - U^2)) / var(U^2) * (U^2 - mean(U^2)))
    variance_U2 <- var(sqrt(1 - U^2) -
     cov(U^2, sqrt(1 - U^2))/ var(U^2) * (U^2 - mean(U^2)))
    return(list(theta = theta_U2, variance = variance_U2))
}

estimate_theta_U(1000)
```

```
## $theta
## [1] 0.793523
##
## $variance
## [1] 0.007181739
```

```
estimate_theta_U2(1000)
```

```
## $theta
## [1] 0.7859443
##
## $variance
## [1] 0.001646453
```

Problem 6

Obtain a Monte Carlo estimate of

$$\int_1^{\infty} \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$$

by importance sampling and evaluate its variance. Write a R function to implement your procedure.

Answer6

The first step is to select an appropriate importance distribution that is both easily sampled from and as close as possible to the shape of the integrand. The standard normal distribution is a good choice for the importance distribution, but since our integral is from 1 to ∞, we might want to use a different distribution with a similar tail behavior for $x > 1$, or a truncated normal distribution starting from 1.

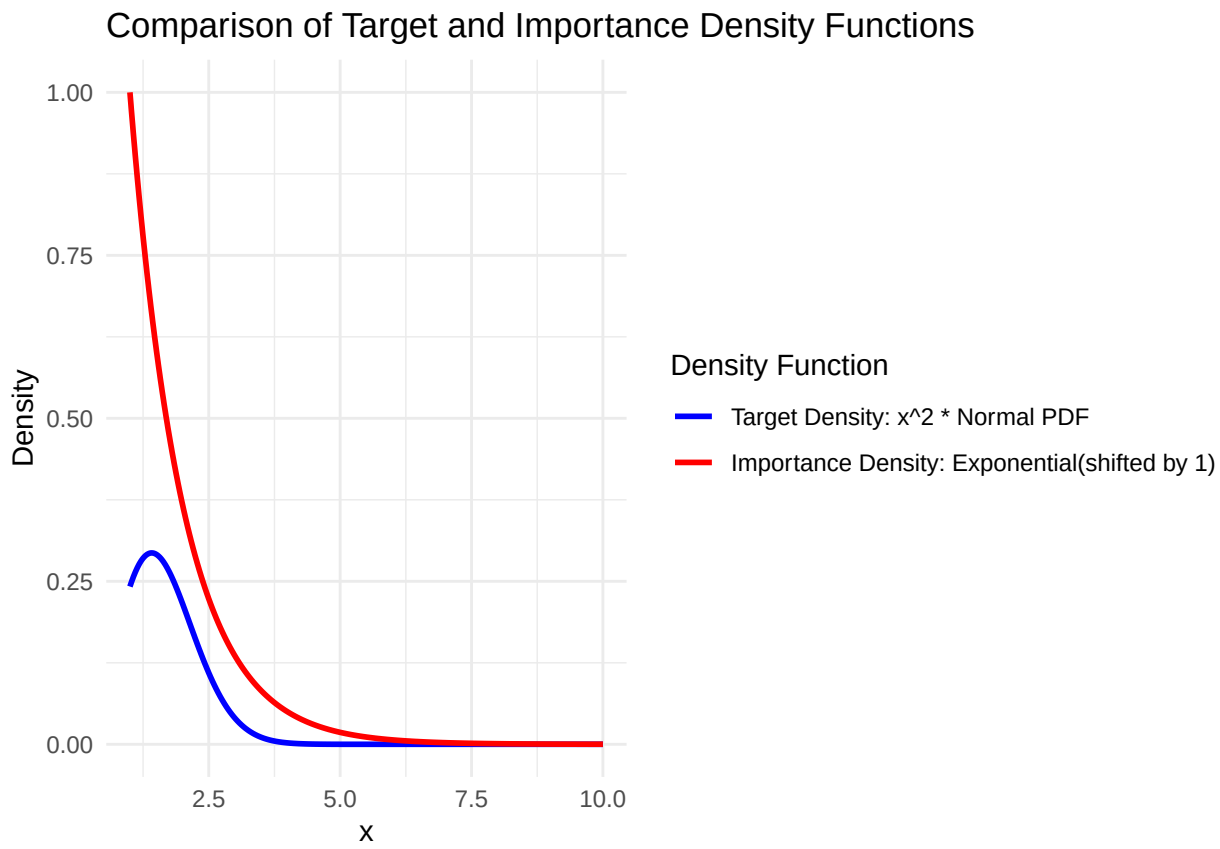
```
# Load necessary library
library(ggplot2)

# Define the target density function: x^2 * normal PDF
target_density <- function(x) {
  (x^2 / sqrt(2 * pi)) * exp(-0.5 * x^2)
}

# Define the importance density function: exponential PDF shifted by 1
lambda <- 1
importance_density <- function(x) {
  lambda * exp(-lambda * (x - 1))
}

# Create a sequence of x values starting from 1
x_values <- seq(1, 10, length.out = 1000)

# Compute the target and importance density values
target_values <- target_density(x_values)
importance_values <- importance_density(x_values)

# Create a data frame for plotting
```

```r
df <- data.frame(x = x_values,
                 Target = target_values,
                 Importance = importance_values)

# Melt the data frame for plotting with ggplot2
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.3.2
```

```r
df_melted <- melt(df, id.vars = 'x',
                  variable.name = 'DensityType',
                  value.name = 'Density')

# Plot the densities with legends
ggplot(data = df_melted, aes(x = x, y = Density, color = DensityType)) +
  geom_line(size = 1) +
  labs(title = "Comparison of Target and Importance Density Functions",
       x = "x", y = "Density") +
  theme_minimal() +
  scale_color_manual(values = c("Target" = "blue", "Importance" = "red"),
                     name = "Density Function",
                     labels = c("Target Density: x^2 * Normal PDF",
                                "Importance Density: Exponential(shifted by 1)")) +
  theme(legend.position = "right")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
# Display the plot
ggsave("density_comparison_plot_with_legends.png", width = 8, height = 6)
```

To guarantee that the variance of the importance sampling estimate is minimized, it might be preferable to use a distribution that naturally incorporates the x^2 factor in its probability density function, given the nature of the integrand, which is a component of the Gaussian function weighted by x^2. The tail of a Chi-squared distribution with an appropriate number of degrees of freedom or a different distribution that more closely resembles the behavior of the function for $x > 1$ might be used for this purpose.

```
importance_sampling_estimate <- function(n, lambda=1) {
  # Generate samples from the importance distribution (exponential distribution)
  samples <- rexp(n, rate=lambda) + 1 # Shift by 1 to start from 1

  # Compute the target function values
  target_values <- (samples^2 / sqrt(2*pi)) * exp(-0.5 * samples^2)

  # Compute the importance density values
  importance_values <- dexp(samples-1, rate=lambda)

  # Compute weights
  weights <- target_values / importance_values

  # Estimate the integral
  estimate <- mean(weights)
```

```
  # Compute the variance of the estimate
  variance <- var(weights) / n

  return(list(estimate=estimate, variance=variance))
}

# Example of using the function with 10,000 samples
result <- importance_sampling_estimate(10000)
print(result)
```

```
## $estimate
## [1] 0.4020425
##
## $variance
## [1] 2.498056e-06
```

Problem 7

Sample Size

```
dnorm(56,40,sqrt(10))
```

```
## [1] 3.482898e-07
```

```
# Number of policyholders to simulate
n <- 1e6
```

Data Generation

```
set.seed(212)
# Parameters for age distribution
age_mean <- 40
age_sd <- sqrt(10)  # Standard deviation (square root of variance)

# Function to sample ages
sample_ages <- function(n) {
  rnorm(n, mean=age_mean, sd=age_sd)
}

# Acceptance-rejection sampling for Gamma distribution
sample_gamma_ar <- function(alpha, beta, n) {
  C <- 10  # Assume a constant, for demonstration purposes
  lambda <- beta  # Proposal parameter

  samples <- numeric(n)
  accepted <- 0

  while (accepted < n) {
```

```
    x_proposal <- rexp(1, rate=lambda)
    u <- runif(1)
    f <- (beta^alpha * x_proposal^(alpha-1) * exp(-beta*x_proposal)) / gamma(alpha)
    g <- lambda * exp(-lambda*x_proposal)

    if (u < (f / (C * g))) {
      accepted <- accepted + 1
      samples[accepted] <- x_proposal
    }
  }

  return(samples)
}

# Number of policyholders to simulate
n_policyholders <- n

# Sample ages
ages <- sample_ages(n_policyholders)

# Initialize vector for health expenses
health_expenses <- numeric(n_policyholders)

# Assign health expenses based on age group
for (i in 1:n_policyholders) {
  if (ages[i] >= 18 && ages[i] <= 35) {
    # Young adult
    health_expenses[i] <- sample_gamma_ar(2, 1, 1)
  } else if (ages[i] > 35 && ages[i] <= 55) {
    # Middle-aged adult
    health_expenses[i] <- sample_gamma_ar(3, 1.5, 1)
  } else if (ages[i] > 55) {
    # Senior adult
    health_expenses[i] <- sample_gamma_ar(4, 2, 1)
  }
}

# Calculate expected premiums
expected_premium_overall <- mean(health_expenses)
expected_premium_young <- mean(health_expenses[ages >= 18 & ages <= 35])
expected_premium_middle <- mean(health_expenses[ages > 35 & ages <= 55])
expected_premium_senior <- mean(health_expenses[ages > 55])

# Print the results
cat("Expected Annual Health Insurance Premiums:\n")
```

```
## Expected Annual Health Insurance Premiums:
```

```
cat("Overall:", expected_premium_overall, "\n")
```

```
## Overall: 1.846172
```

```r
cat("Young Adults (18-35):", expected_premium_young, "\n")
```

```
## Young Adults (18-35): 2.001294
```

```r
cat("Middle-Aged Adults (36-55):", expected_premium_middle, "\n")
```

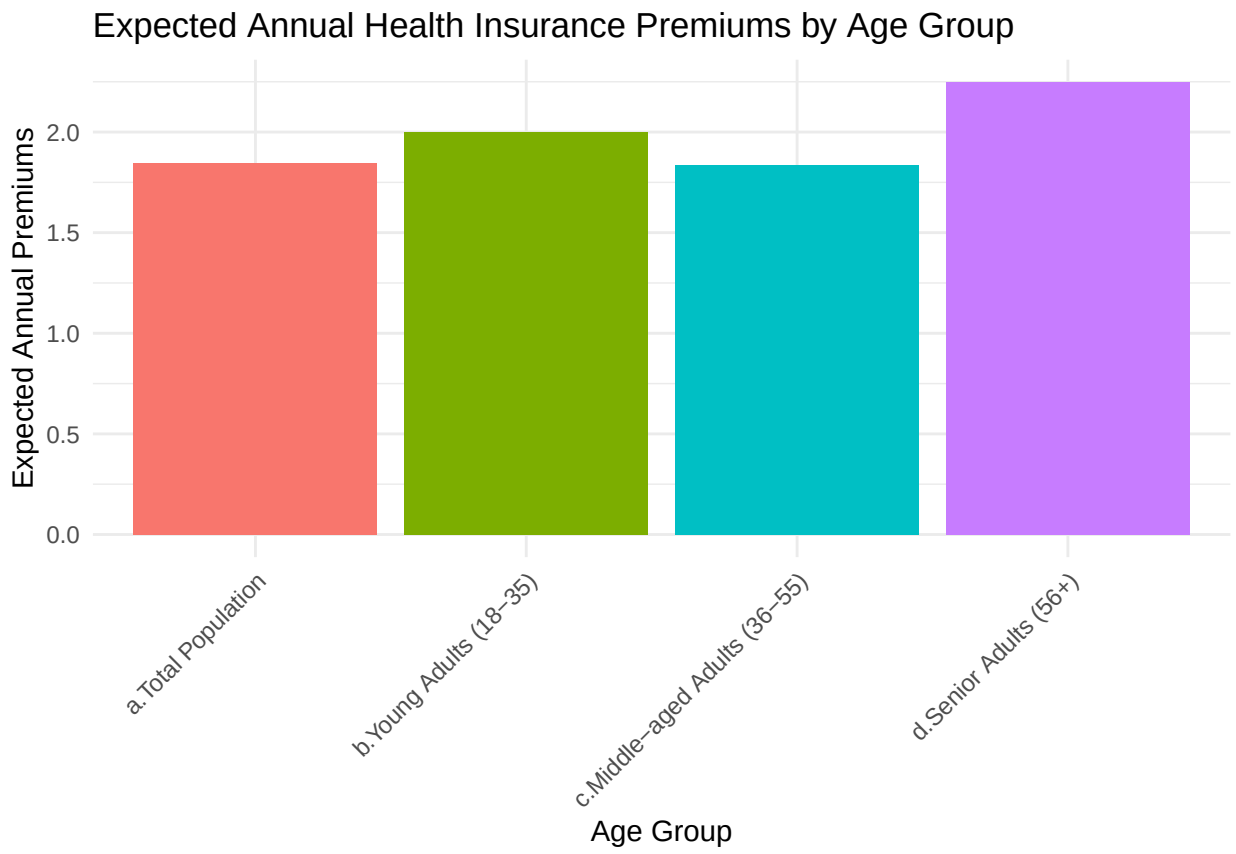
```
## Middle-Aged Adults (36-55): 1.836791
```

```r
cat("Senior Adults (56+):", expected_premium_senior, "\n")
```

```
## Senior Adults (56+): 2.247047
```

Visualization

```r
# Create a data frame for plotting
premiums_df <- data.frame(
  AgeGroup = c("a.Total Population",
               "b.Young Adults (18-35)",
               "c.Middle-aged Adults (36-55)",
               "d.Senior Adults (56+)"),
  ExpectedPremiums = c(expected_premium_overall,
                       expected_premium_young,
                       expected_premium_middle,
                       expected_premium_senior)
)


premiums_df|>
  arrange(AgeGroup)|>
  ggplot( aes(x = AgeGroup, y = ExpectedPremiums, fill = AgeGroup)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  theme_minimal() +
  labs(title = "Expected Annual Health Insurance Premiums by Age Group",
       x = "Age Group", y = "Expected Annual Premiums") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
premiums_df
```

```
##                       AgeGroup ExpectedPremiums
## 1           a.Total Population         1.846172
## 2       b.Young Adults (18-35)         2.001294
## 3 c.Middle-aged Adults (36-55)         1.836791
## 4        d.Senior Adults (56+)         2.247047
```